Graduate School ETD Form 9
(01/07)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis Acceptance

This is to certify that the thesis prepared

By Maja Pusara

Entitled An Examination of User Behavior for User Re-Authentication

Complies with University regulations and meets the standards of the Graduate School for originality and quality

For the degree of Doctor of Philosophy

Final examining committee members

Carla E. Brodley, Co-Chair, Chair

Robert L. Givan

Eugene H. Spafford, Co-Chair

Silvio Micali

Cristina Nita-Rotaru

Christopher W. Clifton

Approved by Major Professor(s): Carla E. Brodley

Eugene H. Spafford

Approved by Head of Graduate Program: Mark J. T. Smith

Date of Graduate Program Head's Approval: 7/3/07

AN EXAMINATION OF USER BEHAVIOR FOR USER RE-AUTHENTICATION

A Thesis

Submitted to the Faculty

of

Purdue University

by

Maja Pusara

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 2007

Purdue University

West Lafayette, Indiana

I dedicate this dissertation to my parents who stood by me every step of the way and whose belief in me gave me the strength and courage to persevere.

ACKNOWLEDGMENTS

I would like to acknowledge Professors Carla E. Brodley and Eugene H. Spafford for going above and beyond their roles as advisors and Professor Cristina Nita–Rotaru for being one of my role models. I would like to acknowledge Doctoral Advisory Committee members for their insightful guidance and support and Dr. Tom Goldring of the NSA for his belief in the project and financial support. I would also like to acknowledge Professors Carla E. Brodley, Cristina Nita–Rotaru, Judith Stafford and Chris Powers for helping me conduct a class–wide data collection without which the research presented in this document would not have been possible. Finally, I would like to acknowledge my parents for their love and support.

TABLE OF CONTENTS

Page

Page

Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

Figure Page

SYMBOLS

U	a single user
M	a single machine
N	number of users
W	window size
f	frequency

ABBREVIATIONS

GUI	Graphical User Interface
HIDS	Host based Intrusion Detection System
FSS	Feature Subset Selection

ABSTRACT

Pusara, Maja. Ph.D., Purdue University, August, 2007. An Examination of User Behavior for User Re-Authentication. Major Professors: Carla E. Brodley and Eugene H. Spafford.

The research presented addresses the problem of insider threat mitigation and detection in computer security. The underlying hypothesis is that one can successfully model user behavior on the basis of user's inputs and on GUI changes as a response to those inputs. In particular, the goal is to determine to what extent users can be characterized by their mouse movements, keystroke dynamics and GUI events. The implemented system raises an alarm when the current behavior of user U, deviates sufficiently from learned "normal" behavior of user U. A closed–setting deployment scenario is assumed and a supervised learning algorithm is applied to discriminate among N users. The strength of the keystroke, mouse and GUI classifier, individually and in combination, is investigated on a set of 61 users. The results obtained show that by combining all three sources of information one can differentiate these individuals with a false positive rate of 14.47% and a false negative rate of 1.78% with the detection time of 2.20 minutes (if user utilized I/O devices). Experiments with a second dataset of 73 users illustrate that even if all users are given an identical task – in this case filling in a form word-by-word from a template – they can still be discriminated. The scalability of the system is examined to determine how well the approach scales to more users or to a scenario where one cannot know the behavior of all users in advance and therefore wants to discriminate between whether it is or is not the valid user. Finally, the accuracy and the computational efficiency of the system are investigated in the presence of a limited amount of data for the profile of user.

1. INTRODUCTION

An integral part of computer security is user authentication, which seeks to confirm the identity of a user for the purpose of granting individual users access to their respective accounts. Authentication can be achieved by something the user *knows* (e.g., access passwords, PIN codes), something the user *owns* (e.g., access tokens, ID badges, PC cards, smart cards [1], wireless identification agents [2]) or something the user *is* (e.g., a fingerprint [3], a palm print [4], a voice sample [5], an iris pattern [6], which are referred to as biometrics [7]). User *re-authentication* refers to the process of continuous authentication of a user for the duration of the user's login session.

Typically, authentication is performed once at the start of each user session. Detecting whether the current user of a computer system was still the valid user who initially authenticated or whether the authentication itself had been compromised (e.g., via a weak or stolen password [8]) has long been an issue of concern in the privacy and security communities. User re-authentication has been addressed by the research community in two ways 1) indirectly by profiling the operating system and its applications and 2) directly by profiling a valid user. The majority of prior research efforts have focused on the indirect approach – building a model of what constituted "normal" software operation. These intrusion detection systems could be used to determine the authenticity of a current user based on his/her library [9] and system call invocation [10–20], call-stack data operation [21] or program trace analysis [22–41]. Direct approaches to user re–authentication have investigated the authenticity of a current user based on his/her command line input data [42–53], keystroke dynamics [54–58] or mouse activity [41, 59, 60].

The classical intrusion detection systems [14,22,33,39] while successful in flagging system–based intrusions (e.g., intrusions that occur in memory, operating system or

applications), lacked the ability to detect intrusions by impostors or masqueraders. Consider a situation in which a trusted user's password had been stolen and an intruder had access to the trusted user's data. This particular intrusion would have evaded detection by the classical intrusion detection systems, because the behavior of an operating system or an application could have very well continued to be "normal." To address this problem we propose a biometric user re–authentication system that uses keystroke dynamics, mouse movements[1] and Graphical User Interface (GUI) events to identify a valid user and detect a masquerader or an intruder.

Before proceeding further it is important to discuss two deployment scenarios for a user re–authentication system. In an open–setting scenario, exemplified by a public library or an Internet cafe, users are allowed and, in fact, encouraged to use computer systems at will. In such a setting it is virtually impossible to collect the data from all users (i.e., the data is *not* labeled) and an unsupervised learning method [61] may be used to build a model of valid user behavior. In unsupervised learning, *only* the valid user's data is available to build a profile of his/her behavior. Data from other users is assumed not to be obtainable. In such an environment, it is difficult to draw a boundary between the "normal" behavior of a valid user and the "anomalous" behavior of an intruder (i.e., *self* versus *not self* discrimination). If the boundary is too tight, the valid user's instances can be misclassified as those belonging to an intruder – this is referred to as the false positive or the false alarm rate. If the boundary is too loose some intruder's instances can be misidentified as those belonging to the valid user – this is referred to as the false negative rate. This is a well-known design trade-off in most stochastic models. If the detection of intruders is of primary importance, a model of valid user behavior is built with tight bounds thereby causing a potentially high rate of false alarms. If we wish to limit the number of false alarms so as not to ask a user to authenticate him/herself repeatedly,

[1]The proposed system can be implemented with different kinds of pointing devices (e.g., a mouse, touchpad, joystick, etc.).

a model of user behavior is built with loose bounds thereby allowing some intruders to go undetected.

In the closed–setting scenario admittance to the premises is restricted and access to the hosts is strictly monitored (e.g., a financial institution or a government building). In such a setting, it is viable to assume that data can be collected from the entire personnel present at the site and that an intruder is likely to come from "within" (this form of vulnerability is known as the "insider threat" in the computer security community). In the closed-setting scenario supervised learning [61] can be applied to learn a model of normal user behavior. In supervised learning, data is available from both a trusted user and an intruder(s) *before* a profile of normal behavior is generated. As a result, the boundary between the "normal" behavior of the valid user and the "anomalous" behavior of an intruder produces lower false positive and false negative rates than in unsupervised learning. Our goal is to determine which one of the employees is currently using a workstation. To this end, we assume a closed–setting scenario and apply supervised learning algorithms to build a profile of normal user behavior.

In the absence of a user re–authentication process, a computer system is more susceptible to *attack* or *misuse*. Research showed that although hackers, viruses and other external hazards received considerable media attention, there is a significant threat presented by insiders who might be employees, temporary workers and/or consultants. Each year different branches of the government announce their findings on cyber–security. According to the 2006 CSI/FBI Computer Crime and Security Survey, unauthorized access was the second greatest source of financial loss [62]. The recently released comprehensive report analyzing insider threats to banking and finance sector generated by the Secret Service's National Threat Assessment Center (NTAC) and the Software Engineering Institute's CERT Coordination Center informed the public of the following findings [63]:

1. "Most of the incidents (83%) were executed physically from within the insider's organization and took place during normal business hours."

2. "The impact of nearly all insider incidents in the banking and finance sector was a financial loss for the victim organization: in 30% of the cases the financial loss exceeded $500,000. Many victim organizations incurred harm to multiple aspects of the organization."

We propose a user re–authentication system that employs machine learning techniques for authentication using behavioral biometrics. Specifically, we address the scenario in which we wish to detect if a trusted user or an intruder has gained access to a user account. Our system is developed in view of the following research objectives:

Computer Security Objective: To design and implement an on–line, scalable user re–authentication system founded on the data collected from user's inputs (mouse and keyboard) and Graphical User Interface (GUI) events for the purpose of detecting an attacker.

Machine Learning Objective: To design an accurate, classification system for detecting outliers (e.g., intruders) in high dimensional temporal sequence data when the amount of data per user is limited.

We conjecture that the proposed system will be able to accomplish the following:

1. Detect insiders pretending to be other insiders;
2. Detect outsiders pretending to be insiders;
3. Discriminate users in a pair–wise sense;
4. Determine the sensitivity of user profiles on different hardware configurations;
5. Discriminate users when they are behaving in an identical manner;
6. Determine the degree of system's scalability and computational efficiency;
7. Determine the strength of each data source (e.g., the mouse, keystrokes and GUI) individually and in combination;

8. Exploit granularity of the data to obtain a comprehensive feature space;

9. Reduce the candidate feature space to a subset of most predictive features; and

10. Improve the accuracy measure when the amount of data per user dataset is limited;

We began the dissertation with a description of the related work. We introduced the data sources used and explained the feature extraction process. We postulated that in our candidate feature space there was some subset of most uncorrelated and discriminatory features and validated our conjecture empirically.

In Chapter 4 we studied the strength of each biometric source individually and in combination on a set of 61 users. Initially, we constructed a classifier for each pair of users to gain insight into which of our biometric sources was most discriminatory. We then investigated the ability of our system to detect insiders pretending to be other insiders. We determined the accuracy of the system when a profile of "normal" user behavior was built after seeing a valid user's dataset and the remaining $N - 1$ intruders' datasets. The results showed that 1) by combining all three sources of information (mouse, keyboard and GUI events) one could differentiate individuals and detect intrusions from the inside; 2) features extracted from the mouse wheel, mouse movements, control keys and GUI combo box and icon events were the best indicators of user's behavior; and 3) more data was needed per user to improve the performance. In this chapter we also studied feature space reduction as one possible method to improve the run-time efficiency while maintaining accuracy.

In Chapter 5 we examined the performance of the system when a limited amount of data was available per user profile. We used smoothing functions such as entropy and majority vote applied over the m most recent classifications to improve the accuracy measurement. *Smoothing* refers to a process used to reduce the noise, in our case, minor, transient changes in the current user's behavior. Our results showed that smoothing over several windows resulted in higher accuracy than simply

increasing the window size, W, of the data points. Furthermore, we observed that as m increased the detection time increased.

In Chapter 6 we studied 1) the classifier's ability to detect outsiders pretending to be insiders; 2) whether we could track a valid user as he/she was using different computer and I/O system configurations; 3) whether we could discriminate users when they were given an identical task to perform; and 4) the scalability and computational efficiency of our user re-authentication system. The following paragraphs outline each of these experiments and describe their significance.

In Chapter 4 we assumed a closed–setting scenario (data must be collected from all users) and applied a supervised learning algorithm to build a profile of normal user behavior. In Chapter 6 we concentrated on a scenario where one could not know the behavior of all users in advance (e.g., temporary workers or visitors) and therefore wished to detect an outsider pretending to be an insider. We considered the following two cases:

1. We had a 3-user dataset (Alice, Bob and Carol). We built a classifier C_1 to discriminate Alice from Bob. We wished to use C_1 to discriminate Alice from Carol.

2. We had an N–user dataset, where N was large. We built a classifier C_2 to discriminate Alice from Bob, Dave, Ed and potentially hundreds of other users. We wished to use C_2 to discriminate Alice from (previously unobserved) Carol.

We conjectured that the classifier C_2 would produce lower error rates when discriminating Alice from Carol than C_1. To validate our conjecture we tested the behavior of a supervised classifier on a previously *unseen* user dataset as N grew large. We also computed the amount of time it took a classifier to raise an alarm when an intrusion occurred.

Our next objective was to examine the ability of the proposed system to track a valid user as he/she was using different computer and I/O configurations. We

observed that user profiles were highly sensitive to hardware configurations and concluded that a separate user profile should be built for each hardware configuration.

We then studied whether we could discriminate users when they were behaving in an identical manner. To this end, we collected a dataset by a group of 73 users who were given an identical task to perform. The users were asked to fill out an electronic copy of a travel expense sheet word-by-word from a template. The accuracy of the combined data source was tested when a profile of "normal" user behavior was built after seeing a valid user's dataset and the remaining $N-1$ intruders' datasets. The results indicated that 1) there was still a strong signal of user identity; and 2) datasets dense with I/O events produced more accurate models of user behavior.

The scalability of the system was investigated on the larger of the two datasets (i.e., on the 73-user dataset). The accuracy of the combined data source was tested again when a profile of "normal" user behavior was built after seeing a valid user's dataset and the remaining $N-1$ intruders' datasets on three subsets that we created by randomly picking 20, 40 and 60 users. The tests showed that the system produced consistent and nearly constant levels of false positive, false negative and overall error rates as the number of users increased from 20 to 40 to 60 to 73, thereby demonstrating empirically the scalability of the system. However, we concluded that to truly test the scalability of a system hundreds and even thousands of user datasets were needed.

As a final point, we examined the computational efficiency of the proposed system and determined a computational bottleneck to be the computation of features used to build a valid user's profile.

The remainder of this document is structured as follows: Chapter 2 discusses the related work in both behavioral authentication and re–authentication. Chapter 3 describes the data sources and the features extracted from each data source. Chapter 4 explains the design and implementation of our user re–authentication system, investigates the performance of each biometric source empirically and conducts an in-depth analysis of the feature space. Chapter 5 examines seven smoothing functions with

the goal of improving the performance of the system when the amount of data per user is limited. Chapter 6 relaxes the closed-setting scenario constraints to study the performance of a supervised biometric classifier on an unlabeled dataset. Additionally, the issues of the system's scalability and computational efficiency are addressed in detail. Chapter 7 summarizes our approach to behavioral re–authentication and discusses its significance in a real–world setting.

2. RELATED WORK

This chapter reviews two related bodies of work in authentication; those used for the initial authentication of a user to a resource (host) and those used to continuously monitor the user after authentication. The chapter concludes with a discussion of the machine learning techniques used for learning models of normal behavior.

2.1 User Authentication in Computer Security

User authentication, commonly referred to as personal identification, is a technique whereby a user registers some secret "identity–verifying" information in advance, and then on each attempt to access the system (login) this information is used for authentication. The "identity–verifying" information falls into one of the following categories [64]:

1. *Biometrics*: fingerprints, retina pattern, face pattern, voice print and handprint
2. *Objects*: ID card, credit card and smart card
3. *Knowledge*: password, PIN and digital signature
4. *Actions*: signature and patterns of behavior

Biometrics encompass the *physiological* properties of a person – they represent what person *is* in a physiological sense. A drawback of biometrics is the high cost of their implementation. Additionally, users may not accept them because of perceptions of invasions of privacy or even physical danger (e.g., retina scan). There is a stability–sensitive distinction between biometrics; those that cannot be changed such as fingerprints and retina patterns [3, 6] and those that can be altered such as

voice print or handprint [4, 5]. For example, a cold can distort a biometric based on a voice print.

The most popular forms of authentication are objects and knowledge. Objects or possessions are identity–verifying information that a person *has*, whereas knowledge is what person *knows*. Objects and knowledge have their advantages over other methods. They require minimum or no additional hardware to implement and are unobtrusive to the user. Unfortunately, different types of cards can be lost, stolen, and passwords can be guessed or overheard. In addition, hackers can break many poorly chosen passwords with apparent ease [65].

The final category is the actions that represent what person *does* and *how* she does it. Actions model patterns of user behavior and hence they are often referred to as a behavioral modeling approach. This is the approach used in this research on user re–authentication and therefore we describe these methods in detail next.

Behavioral modeling can be used as stand–alone authentication systems or in combination with other authentication mechanisms as a method of password hardening [66]. The behavioral patterns that arise from the user input and interaction with the receiving machine are sources of data for user authentication. These input patterns are generated via input devices such as the keyboard and mouse.

2.1.1 Keystroke Dynamics

Perhaps the best known method of authentication was based on keystroke dynamics. By analyzing users' typing patterns, it was possible to identify users based on certain habitual typing rhythm patterns [67]. The motivation to use keystroke dynamics was derived from handwriting recognition systems, which used correlation analysis of writing–hand movements, pressure measurements on the signature surface, and power spectrum estimation of written scripts to identify individuals [68–74].

Early studies focused on determining whether users could be discriminated based on latency between two character sequences (digraphs). In 1980, Gaines *et al.* [75]

conducted an experiment with seven secretaries, each of whom was given the task of retyping the same three paragraphs at two different times four months apart. They recorded the digraph latency timings and computed the means of those digraph times whose rate of occurrence was at least ten. Then they compared latencies between two sessions to determine if the means and the variances were the same at both sessions. Although the population sample was too small to draw any conclusions, the results were promising, but unacceptable for authentication purposes.

Similar experiments were conducted by Leggett and Williams with 17 programmers instructed to type the same text consisting of approximately one thousand words [76]. A user was "accepted" if more than 60.0% of the comparisons between the test signature and the mean reference latencies were within 0.5 standard deviations of the mean reference digraph latency. Leggett and Williams reported a false positive rate of 5.5% and false negative rate of 5.0%. The main drawback of this study was the amount of text participants needed to enter for authentication.

To compute the degree of similarity between the current signature and stored records of latency some authentication methods used the Mahalanobis distance [77] and the Euclidean distance [78]. Both approaches made a use of two additional data sources: 1) the keystroke pressures and 2) the time it took to type a predefined number of words as their attributes. However, the results for both of these methods were unavailable, because the experiments were conducted for commercial ventures.

Joyce and Gupta [64] also examined latency in an experiment with 33 users who were instructed to type a quadruplet (login name, password, first and last name) eight times for the purpose of training and another five times for the purpose of testing. The *mean reference signature* for each user was computed as a collection of individual signatures, one for each string of the quadruplet, by calculating the mean and standard deviation of the eight values for each digraph latency and discarding any outliers (datum greater than three standard deviations above the mean). A user was considered him/herself if the comparison between the test signature and the *mean reference signature* was within 1.5 standard deviations. The authors reported

false positive rate of 16.36% and false negative rate of 0.25%. Even though this method used less than forty characters for authentication its high false positive rate made it infeasible in practice.

Bleha *et al.* examined whether latency could be applied to distinguish between samples of legal users and imposters. They employed a Bayes classifier and a minimum distance classifier [79]. Fourteen valid and 25 invalid users were instructed to type their first and last names and a password. To create each valid user's profile, time duration between successive keystrokes was collected on two occasions and the two entries were then combined into a single entry by choosing the shorter of the two corresponding time intervals between keystrokes (a.k.a. the shuffling technique). For each of the classifiers, the normalization procedure was applied to accommodate the variation in the name lengths and a different threshold value was used. An entry was rejected if the threshold values were exceeded for both of the classifiers thereby producing an unacceptable false positive rate of 8.1% and false negative rate of 2.8%.

Brown and Rogers were the first to collect keystroke duration in addition to keystroke latency [80]. Authentication was done in three ways, using the Euclidean distance between the raw keystroke data, backpropagation, and partially connected backpropagation neural networks. Two groups of users (21 in the first group and 25 in the second group) were tested on an input of their own first and last name (approximately 15–16 characters in length). The best results were achieved with a partially connected backpropagation neural network with a 0.0% false negative rate and a false positive rate of 12.0% on the first 21 users and 21.2% false positive rate on the other 25.

A similar line of research in terms of physical measurements (keystroke duration and latency) was found in Obaidat and Sadoun, in which fifteen users provided their login name 225 times each day over a period of eight weeks [81, 82]. Fifteen individuals then attacked each of the 15 users 15 times. It was unclear whether all 15 of the attackers were the same set of users trying to log into each other's accounts or a new group of users. The results reported were zero false positive and

zero false negative rates. The results were encouraging and it would be interesting to repeat the same experiment for the purpose of re–examining false negative rate when attackers were allowed to practice typing the legal users' login names before their data was used for evaluation.

Monrose and Rubin conducted experiments with 31 user (originally they had 42 participants, but some of the data had to be eliminated because of timing errors) and were the first to investigate the use of "Free–style" (i.e., non–structured) text as opposed to constant chosen phrases such as usernames [83, 84]. Their users were not restricted to a single machine; rather they were given the freedom to collect data at their own convenience with their own machines. Both keystroke latency and duration were collected. They reported results with three different similarity measures in their experiments: Euclidean distance, non–weighted probability and weighted probability. Classification was done by K–Nearest Neighbor hierarchical clustering [85]. They reported an identification rate (i.e., how well one identified him/herself) of approximately 90.0%.

Monrose *et al.* also studied password hardening where a hash of password and attributes of keystroke dynamics was stored in the profile [66]. Experimental evaluation produced a perfect false negative rate and 48.4% false positive rate. The high false positive rate was not surprising considering that normal behavior was modeled on short character strings (user name and password).

Bergadano *et al.* [57] used the degree of disorder in trigraph latencies as a dissimilarity metric. The classification was done statistically by classifying an unknown instance as belonging to the user with the smallest *mean* distance between the unknown instance and the user's profile of normal behavior. They evaluated their approach on 154 individuals achieving a false positive rate of 4.0% and a false negative rate of 0.01%. This performance was reached using the same sampling text (683 characters) for all individuals and allowing typing errors.

The most recent research in keystroke dynamics (duration and latency) was by Sang *et al.* in 2004 in which they used one–class Support Vector Machines (SVMs)

to classify ten users with the false positive rate of 0.02% and false negative rate of 0.1% [86]. Their approach outperformed two–class SVMs, back–propagation neural network and Levenberg–Marquardt neural network [87].

2.1.2 Mouse Dynamics

Behavioral modeling based on mouse dynamics has received far less attention than its counterpart keystroke dynamics. However, recently a study was conducted by Everitt and McOwan on a Java–based internet biometric authentication system [88]. They combined two distinct tests to ensure authenticity, a typing style test and a mouse–based signature test, achieving an overall false negative rate of 4.4% and false positive rate of 1.0%. A group of 41 participants was tested and they were instructed to provide username, password and signature forty times. The authentication was done remotely and each user had her own machine and her own mouse or pointing device. Latency and hold times were analyzed for keystrokes, and Euclidean distance and angle were analyzed for actual mouse signatures. Similarity measure was the *chromosome fitness rank* used in genetic algorithms [89]. The *fitness* for a relationship between two points in space was given by the standard deviation equation of a function f that calculated either the distance or angle between the points for a particular signature. Relationships were then ranked in order of fitness and the ten fittest angle and ten fittest distance relationships were used for signature representation. Classification was done with three different back–propagation neural networks. Results reported for the mouse–based authentication system only were 16.3% false positive rate and on average 3–4% false negative rate. The mouse–based authentication system was clearly outperformed by the combined mouse– and key–based authentication system. Some of the details of the system were omitted in the paper because the project was a commercial venture.

2.2 Behavioral Re–Authentication in Computer Security

User re–authentication, or what is sometimes referred to as continuous authentication, is primarily achieved by behavioral modeling. One could envision a biometric system with similar capabilities, but this kind of system would be exceedingly impractical, expensive and, most importantly, annoying to users. Consequently, the primary focus of the research community in the field of re–authentication has been behavioral modeling.

Anderson was first to recognize the need for modeling host behavior in computer security [90]. Alongside the development and wider use of computer networks as well as the increased need for shared access of information, security, protection and privacy of both information and hosts, have become overwhelmingly important [91]. To this end, a multitude of data sources, descriptive features and classifiers have been proposed and used, each improving upon the accuracy of its predecessors. At the time of writing this document, all research efforts in the area have used one of the following data sources for software behavioral modeling: system call traces [10–20], command–line input [42–53], keystroke dynamics [54–56, 58], mouse dynamics [41, 60], various audit logs [22–41], call–stack data [21] and computer usage statistics which overlapped with some of the other sources. Preference for a particular data source was somewhat subjective, yet for a specific purpose certain sources have been known to outperform other sources, for an example, to model the behavior of a host computer (e.g., server) system call and program–execution traces have been a good choice.

2.2.1 System Calls and Call Stack Information in Program Profiling

In the context of this work we believed that it was important to consider program profiling approaches in addition to user profiling approaches, because there was a pronounced overlap in the data sources and techniques used to model normal behavior in general. We limited ourselves to the intrusion detection systems employed

on the hosts and moreover, we did not consider those host–based IDSs that used the network traffic data for modeling system's normality. Reason for this exclusion was that models stemming from the network traffic were inherently subject to a great degree of external factors that distinguished them from other more traditional host–based IDSs.

System call analysis for the purpose of program re–authentication has received the most attention. This interest in system call analysis stemmed from the fact that most intrusion detections (e.g., buffer overflows, etc.) could be detected by observing sequences of system calls. To study evolution of these systems, we examined both static and dynamic analysis of system calls. Static analysis has bordered on secure software engineering and operating systems design and has been characterized by specifications of normal program behavior at compile or link time. Conversely, dynamic analysis has been performed at run–time, when the program counter and the stack information were available.

The callgraph model of Wagner *et al.* analyzed system calls via static analysis of the program code [10]. A non–deterministic finite automaton (NDFA) was used to model system call sequences. This model is innately non-deterministic because the branching information cannot be predicted statically. If all non-deterministic paths are blocked at some point, the system signals an anomaly. There were no false alarms because all paths were considered in the model. However, the model did produce some impossible paths that were able to potentially impede the detection process by failing to detect intrusions. Unlike some other behavioral models that express their results in terms of false positive and false negative rates, behavioral models based on system call traces express results in terms of *performance* (runtime overhead) and *precision* (robustness of detection against a targeted attack). Wagner *et al.* reported overhead of 40.0 seconds per transaction for half of the programs in their experiments.

Giffin *et al.* also used static analysis, but applied it to binary executables rather than source code [11]. They developed many optimization and obfuscation tech-

niques to improve precision and efficiency. Specifically, "inserting null calls" helped reduce the amount of non-determinism thereby solving the impossible paths problem and increasing the precision. However, this technique required rewriting of the executables and renaming the name space, which could be cumbersome in certain situations. They reported a 0.5% overhead for the unoptimized, non–deterministic finite state automaton and 13% overhead at moderate levels of optimization while simultaneously reaching a detection rate of 74%.

The method proposed by Sekar *et al.* [12] modeled program behavior with a finite state automaton (FSA) at runtime. System calls were captured at the user level and therefore did not require any changes to the kernel. Runtime monitoring for intrusion detection was done by examining the states of the FSA model of normal program behavior. If there was no transition between the current state and the new state labeled with the system call name that was intercepted, the new state was considered anomalous and an anomaly count was incremented. When the anomaly count exceeded a threshold, an intrusion was flagged. Their method associated different *weights* with different kinds of anomalies, so instead of incrementing the anomaly count by one, it was incremented by the weight associated with the anomaly observed. The runtime overhead caused by system call interception was 100–250% and the false positive rate was 0.0% after 10,000 system calls used in training. Sekar and Uppuluri later devised an efficient runtime detection system at the kernel level [13]. They used interposition techniques[1] at the operating system's system call interface to enable the kernel–resident state machines to observe application process behavior. This resulted in an overhead of less than 1.5% for the *ftpd*, *telnet* and *http* benchmarks.

Forrest *et al.*'s approach moved a sliding overlapping window of length n across a symbolic audit data stream thereby creating a series of *n–grams*, each containing a series of n consecutive system calls [14–16]. The distinction between normal and

[1]System call interposition is a method for regulating and monitoring application behavior. It intercepts system calls issued by a user–mode application program and it either allows or denies them based on rules and policy.

anomalous traces was done by a simple comparison in which a new trace was compared against all of the traces in the profile's database. If only an exact match of sequence length n was found, was the trace considered normal. Subsequent extensions of this work considered some alternate classification models HMMs [92] and a manually constructed FSA [16].

Unlike the study of Forrest *et al.* which analyzed fixed–length sequences, Wespi *et al.* used the Teiresias [93] algorithms to construct variable–length sequences of system calls [17, 18]. The pattern matching algorithm was designed to match the sequences exactly (i.e., they could not be matched as ordinary expressions using wildcards). The algorithm looked for the exact match first and then for the best partial match. If no pattern in the profile matched the beginning of the string, then the first event was counted as *uncovered* and removed. The algorithm then searched recursively for other patterns that exactly covered the continuation of the string up to a given depth D or to the end of the string whichever came first. For each string the algorithm extracted the number of system call events that were *uncovered*. The decision whether to raise an alarm was based on the length of the *uncovered* sequences (i.e., the greater the length of the *uncovered* sequences the more likely it was that an intrusion occurred). They compared their algorithm with the algorithm proposed in [14]. The results were remarkably similar for both algorithms: variable–length and fixed–length, achieved 0.0% false positive and 20.0% false negative rate when the fixed–length of three was used in [14]; and [14] achieved 0.03% false positive and 10.0% false negative rate when the fixed–length was set at four. It was not clear that there was a compelling reason for implementing a variable–length sequence matching algorithm considering that its accuracy was comparable to the fixed–length algorithm and its efficiency was lower because of the algorithm's complexity.

The generic software wrappers proposed by Ko *et al.* were designed to enforce access control and intrusion detection checks triggered by events during process execution [19]. They were implemented in the kernel to analyze system call data. They combined signature-based intrusion detection technique with sequence- and

behavior-based intrusion detection techniques. Although software wrappers are a powerful mechanism they are difficult to port, incur system slow-down, and are somewhat cumbersome to manage and configure.

BlueBox [20] was an example of an intrusion detection system based on system call introspection.[2] It was a policy driven technique similar to sandboxing,[3] which theoretically improved upon both anomaly detection and misuse detection techniques. Unlike misuse detection, sandboxing was capable of detecting novel attacks and unlike anomaly detection it was not prone to a high false alarm rate. However, its disadvantages were that it was version–specific, hence it needed to be reconfigured with each software update; and it was not capable of detecting in–memory attacks. BlueBox operated via system call handler wrappers. This kind of implementation alleviated some of the problems faced by typical in–kernel implementations. BlueBox achieved a perfect detection rate on the system calls data set at the cost of a 10.0% slow–down during the runtime.

Oleg *et al.* examined call stack data for the purpose of intrusion detection [21]. They generated the so–called "VtPath" by using the return addresses from the stack. They were able to detect the following anomalies: stack, return address, system call and virtual path anomalies. They solved the impossible path exploit[4] and they were able to treat dynamically loadable libraries (DLLs) like statically linked functions.[5] The false positive rate was virtually zero when one million system calls were used for training purposes. Runtime overhead was 150 milliseconds per process.

[2]Introspection is an approach typically used for inspecting a virtual machine from the outside for the purpose of analyzing the software running inside.

[3]Sandboxing is a technique that adds a layer of protection between an exploited program and the computer. It confines the normal behavior of a program to a certain domain, so that when the program tries to perform an action outside of this domain for no apparent reason, this action is treated as suspicious behavior.

[4]The impossible path exploit is another name for the impossible path problem. It is a path from one system call to the next that goes undetected because of the non–deterministic nature of Control Flow Model (CFG) of normal program behavior.

[5]They were able to treat DLLs, which are normally linked during the runtime, as if they were present in the source code or loaded at the compile time.

2.2.2 Command–Line Input Data

Lane and Brodley were the first to experimentally investigate intrusion detection based on a behavioral model of user command–line input data [42–47]. In the instance based learning framework (IBL) they collected fixed–length sequences of a user's commands as a profile. To determine if a current sequence was from the same user they applied a similarity measure that counted the matches with the adjacency bias and a polynomial upper bound. Experiments were conducted with eight users and results reported varied by user as to false positive and false negative rates with an average accuracy of 70.0%. Unlike system calls data, for which there were a pre–specified number of sequences of system calls, the sheer number of user–driven command line input data sequences was capable of potentially exhausting system resources. For the purpose of data reduction they applied a version of a greedy clustering algorithm, which reduced the size of the user model by 70.0% with only a small loss in accuracy.

Computer intrusion detection based on Bayes Factors for comparing command transition probabilities was implemented in [48]. In this context, a Bayes Factor[6] statistic was used to test the null hypothesis that the observed command transition probabilities came from the profiled user's transition matrix. The Bayes Factor tests based on the observation of a block of 100 commands had a false alarm rate of 6.6% while detecting about 78% of blocks from simulated intrusions performed on fifty users each contributing 15,000 commands. To address the problem of concept drift (i.e., how user behavior changes over time) the authors implemented exponentially weighted discount mechanism that needed to be intrusion-free for a certain run-size before updating could take place. The work was extended in [49] by implementing five different statistical models Uniqueness, Bayes one-step Markov, Hybrid multi-step Markov, Compression, Sequence-Match [42] and IPAM. The best results were

[6]Bayes Factor statistic is the posterior probability of the null hypothesis when the prior probability of the null hypothesis is one-half.

obtained with Bayes one-step Markov with 6.7% false positive rate and 30.7% false negative rate evaluated on the command-line data provided by seventy users.

ADMIT was a real–time host–based IDS that created user profiles using semi-incremental clustering techniques [50]. The *Longest Common Subsequence* (LCS) was used as a similarity metric when comparing new sequences to the existing user profiles. LCS gave the length of the longest subsequence of tokens that the two sequences had in common. Authors used the past sequences to determine if the current sequence was only noise or if it was a true change from profile. They called this process *sequence rating* and they used a number of possible rating metrics LAST_n, WEIGHTED and DECAYED_WEIGHTS to do it. K–means algorithm was used for dynamic clustering. The system was evaluated on the same data set as [42] producing a false positive rate of 15.3% and a false negative rate of 19.7%.

Fawcett and Provost applied their fraud detection algorithm to intrusion detection to discriminate among different users based on their command-line input data [51]. They applied the *DC-1* algorithm, which used a discrimination method called *change detection*. Change detection modeled the transitions that occurred from normal to anomalous activity for each individual user. Change detection was effective when there was little commonality within anomalous activity and significant differences between normal and anomalous activity. The *DC-1* system was evaluated on 77 users and a total of 8,000 login sessions. Although the system performed better than random, it was defeated by a simple Chi–square profiling technique.

Coull *et al.* used bioinformatics techniques for a pair–wise semi–global sequence alignment[7] to characterize the similarity between the monitored session and the past user's behavior [52]. The pair-wise sequence alignment algorithm was a variation of the classic Smith-Waterman algorithm [94]. The *scoring scheme* was designed so that matches positively influenced the score of the alignment and so that matches were preferred to gaps. Command-line data was used to build a profile for the user. Empirical evaluation was conducted with seventy participants. The method proved

[7]Semi-global alignment allows gaps at the ends.

to be superior to Bayes 1-step Markov, Naive Bayes (with and without updating), Hybrid Markov, IPAM, Uniqueness, Sequence Matching [42] and Compression. It achieved a false positive rate of 7.7% and a false negative rate of 24.0%.

Leu and Yang [53] ran statistical tests to see if they could recognize a current user as being him/herself. To do this, they first assigned each command a score, which was an integer representing the degree of "normalcy" (i.e., the higher the score, the more anomalous a command was). Then they assigned a *frequency* weight to each sequence of commands so that the weight was dependent upon the rate of occurrence of each sequence. The most frequent sequences of commands were included in the user's profile. The recognition was done by comparing a new instance with the profile of current user's behavior. The results reported had a true positive rate of 72.43% on 772 participants who executed a total of 90,000 commands.

2.2.3 Keystroke Dynamics

While keystroke dynamics has been extensively applied for authentication, less attention has been paid to their use for re–authentication. In [54] 30 subjects entered the same reference text of 2,200 characters twice as a measure of their normal behavior. Two different texts of 574 and 389 characters were also collected to be used as intruder behavior. Statistical analysis was performed on digraph latencies and thresholds were set so that the system would correctly accept both smaller text samples when compared against the reference profile of the user who provided them. The false negative rate was 15.0% with 26.0% of the impostors detected in the first forty keystrokes of the testing samples and most of the impostors (i.e., $> 50\%$) detected within 160 keystrokes.

Dowland *et al.* [55] ran statistical analysis and different data mining algorithms on the users' data sets. They achieved 50.0% correct classification rate on four users. The experiment was then refined in [56] to include some application specific information for PowerPoint, Word, Messenger and Internet Explorer. The best true

acceptance rate was measured to be 60.0% on eight users who collected the data over the period of three months.

Bergadano *et al.* [58] created a system that measured the time between the depression of the first key and the depression of the nth key for each *n-graph*[8]. The distance between two typing samples was expressed as the sum of the absolute values of the distances of each *n–graph* of the second sample with respect to the position of the same *n–graph* in the first sample. This distance was normalized by the maximum number of *n–graph* occurrences in the two samples. If more than one *n–graph* distance was of interest, the cumulative sum of the individual normalized distances was computed. Two different texts, each 300 characters long were used in the experiments on forty users (trusted users) instructed to type both of the text samples for an overall total of 137 samples of the first text and 137 samples of the second text. Another ninety users (impostors) were instructed to type the second text only once. To classify an unknown instance X, the mean distance between X and each user's profile was computed as the mean of the distances between X and each sample in the profile. An instance X was said to belong to user U if the mean distance value between X and U was the *m-th* smallest mean distance among all legal users. The *5-th* smallest distance produced 12.5% false negative rate and zero false positive rate. The authors improved their false negative rate by implementing a supervised learning scheme that computed the mean and the standard deviation of the distances between every sample in U's profile and every sample in someone else's profile. This lowered the false negative rate to 5.36% while maintaining a false positive rate of zero. They also evaluated the performance of their system on varying text lengths, specifically, the text of lengths 150, 75 and 38 characters and obtained the following rates FA=5.84% FN=7.5%, FA=9.49% FN=7.5% and FA=32.11% FN=7.5% respectively.

[8] *N–graph* denotes n characters in a sequence. For an example, digraph means two characters in a sequence.

2.2.4 Mouse Dynamics and GUI Events

Data sources used to model a server's behavior are often inadequate for user profiling. These sources best describe the behavior of programs, kernels and operating systems, and they fail to capture the individuality and uniqueness of human behavior. Graphical User Interface (GUI) events and mouse dynamics model user behavior more directly. However, little work as of yet has used mouse dynamics and GUI events with the exception of Goecks and Shavlik [60] and Pusara and Brodley [95].

Goecks and Shavlik described a set of potential features for predicting user behavior in the World Wide Web (WWW) environment [60]. The feature set included counts of the number of hyperlinks clicked on by the user, the number of scrolling events and the amount of mouse activity in a time period. The actions of resizing a window and the usage of the Edit menu and/or a scroll bar were termed *scrolling events* by the authors. The *mouse activity* was the usage of the menu, in this case the Main menu, and just the fact that a link was highlighted when the cursor was above it. The authors never recorded the mouse movements or even mouse events other than clicks on the hyperlinks. Their fundamental goal was, given the HTML text of a web page, to predict the amount of normal user actions on the page. They used a fully connected, three-layer neural network as a learning algorithm. Empirical results were designed to measure the accuracy in predicting their three chosen user actions. One of the authors collected the data by visiting some 200 web pages, clicking on one-fifth of the hyperlinks on each web page and producing a total of forty scrolling-related and seventy mouse-related events. The results obtained had a root mean squared error of 7.0%, 5.0% and 13.0% for the hyperlinks clicked, scrolling activity and mouse activity, respectively.

In [95] user re–authentication system via mouse movements was designed and implemented. We showed that users could be distinguished based on the way they operated their computer mouse device. To create a model of normal behavior cursor

coordinates and mouse events (e.g., clicks, double clicks, etc.) were recorded. Statistical measurements of distance, angle and speed were computed and mouse events were counted. Decision tree classifier was used to obtain the final results. Empirical evaluation was conducted on eighteen users producing a false positive rate of 0.43% and a false negative rate of 1.75%.

2.2.5 Audit Log Data

In 1985 Denning and Neumann proposed an architectural prototype of an intrusion detection system which jumpstarted the development of IDSs as we know them to be today [22]. Denning and Neumann envisioned many aspects that later became the foundation of modern IDSs (e.g., the structure of an audit record). They were the first to define different vulnerabilities/threats and propose profile structure and profile templates.

Lunt and a team of developers at SRI International were the first to actually implement many of the ideas proposed by Denning and Neumann [23–27]. They came up with the Intrusion Detection Expert System (IDES), followed by an improved version Next-generation IDES (NIDES). Both IDES and NIDES observed user behavior on a monitored computer system and adaptively learned what was normal for individual users, groups, remote hosts, and the overall system behavior. Observed behavior was flagged as a potential intrusion if it deviated significantly from the expected behavior or if it triggered a rule in the expert-system rule base. The normal behavior was statistically modeled with multivariate distributions. Known anomalies (e.g., worms and viruses) were modeled by a set of rules stored in the expert database. Because of the vast amount of space occupied by observed audit data, they only kept the frequency counts, the means and the covariance of the data in the history of profiles. To address the issue of periodic activity updates, they implemented exponential decay aging on the history data whereby giving a half-life

to each measure in the profiles. Winkler and Page proposed an architecture similar to the one found in IDES [28].

The concept of expert rules characterizing either program or user *normal* behavior over time became widely accepted in the early 1990's. In [29] sequential rules describing a user's behavior over time were examined. Attributes measured were the event type, name of the executable file, object name, object type, privileges, status of an execution and process ID. Sequential rules were then extracted from these attributes and the percentage of "cover" was examined (i.e., how well the rules covered new sequences of attributes). The authors reported that 9.5% of the rules covered 63.5% of the events observed, but what one must keep in mind is that there was a subset of the events observed that were not covered by any rule, the so-called "unknown events," which were then forwarded to the sys admin for further investigation. Consequently, true detection and false positive rates were not reported in the paper.

Debar *et al.* [30] used recurrent neural networks with backpropagation for user identification. The authors collected a variety of audit log data such as login/out signals, user commands, timestamps, memory and I/O usage, key hit and latency. They reported a 6.12% error in identifying a single user (i.e., the experiment had only one user). Another neural network approach on audit log data emerged recently [31]. In this paper, a radial basis function (RBF) network was used for intrusion detection and it produced accuracy of 74.0% when detecting two attacks (*format* and *ffb*).

In the tradition of its predecessors, [32] proposed software architecture and a rule-based language for universal audit trail analysis, without conducting any empirical evaluation of the prototype. The main contribution of the paper was the rule–based language, RUSSEL, which was tailor–made for the analysis of sequential files in a single pass.

Helman and Liepins [33] investigated statistical foundations of audit trail analysis for the detection of computer misuse. They modeled computer transactions as generated by two stationary stochastic processes, the legitimate (normal) process N and

the misuse process M. They formally demonstrated that the accuracy was bounded by a function of the difference of the densities of the two processes. When the distributions were unknown, as was often the case in practice, the authors suggested a *frequentist* approach to estimating the normal behavior (i.e., the most frequent user actions were considered normal) and random approach to estimating anomalies (i.e., the occurrence of anomalous events was completely random in nature). This random assumption for anomalies is rarely if ever true in practice. To construct the model, the authors used a supervised expert rule–base approach. They also reduced the dimensionality of their feature space by attribute projection and value aggregation. The system was evaluated on the audit command data set collected from eight users. It contained 30,000 transactions. The reported results had a 5.0% false positive rate and 5.0-7.0% false negative rate which was high considering that the data was labeled (i.e., supervised-learning model) and the number of users was small.

Circa 2000, a group of publications came from UCSB presenting the State Transition Analysis Technique (STAT) tool suite which included USTAT (User STAT), NSTAT (remote Network-host STAT), NetSTAT (Network STAT) and WinSTAT (Windows STAT) [34]. The approach supported reusability, portability and extendibility. It was conceived as a misuse detection method to describe computer penetrations as sequences of actions that an attacker performed to compromise the security of a computer system. The attack scenarios were graphically represented by the state transition diagrams.

Markov chain model of normal behavior was attempted in [35]. Audit data of the Sun Solaris system containing a total of 284 different event types (i.e., system calls) was used for the empirical evaluation of the system. Fifteen attacks were added to the "clean" data set for testing purposes: password guessing, symbolic links to gain root privileges, gaining unauthorized access remotely, etc. All 284 event types were represented by one of the states in the Markov chain. The construct the model of normal behavior Ye computed the transitional probabilities between the states. The model was evaluated on 1613 data instances and it achieved a perfect detection

rate with a zero false positive rate on the fifteen attacks above mentioned. These results were surprising considering that the likelihood of every one of the event types occurring in the set of 1613 data instances was low. Furthermore, it was always possible to invoke a rare event type as part of the normal behavior. Such rare events could then induce false alarms. Attacks such as password guessing and remote access were easily detectable with this model because they invoked only a small subset of different event types. It would have been interesting to measure the performance of the proposed system on some other attacks including the obfuscated attacks. Ye [36] also implemented a scalable clustering technique for intrusion signature detection. In this work he compared the results obtained by supervised clustering versus those obtained by the decision tree algorithm C4.5. To construct a model of normal behavior in his supervised clustering method he used the distance–based similarity metric. He had thirty different attributes (i.e., features) in the data set, each attribute representing the frequency count of a particular system call. The purpose of the empirical evaluation was to determine the degree to which the thirty system calls could be identified. The results were perfect (i.e., FP=FN=0%) for the clustering method while the decision tree produced a false negative rate of 5.0%. No practical application was known to us that could benefit from identification of system calls.

Most recent architecture for audit log data analysis was proposed by [37–39]. Lee *et al.* suggested the use of adaptive learning algorithms to facilitate model construction, to perform incremental updates and to improve usability. They also suggested the use of unsupervised anomaly detection algorithms to reduce the reliance on labeled data. The key ideas were to use data mining techniques (the association rules algorithm and the frequent episodes algorithm) to discover consistent and useful patterns of system features that described program and user behavior, and use the set of relevant system features to compute (inductively learned) classifiers that could recognize anomalies and known intrusions. The authors used the RIPPER algorithm, an inductive rule learner, to compute the detection rules for various attacks. They also introduced the concept of *artificial anomaly generation* whereby anomalies were

"man-made" by *near-miss* instances that were close to the known data, but were not in the training data. To meet the challenges of both efficient learning (mining) and real-time detection, they proposed an agent-based architecture for intrusion detection systems where the learning agents continuously compute and provide the updated models to the detection agents.

Shavlik and Shavlik [41] proposed an algorithm that created a model representing "each particular computer's range of normal behavior. [41]" Parameters that determined when an alarm should be raised, because of abnormal activity, were set on a per computer basis. A total of 200 different measurements including the number of bytes transferred, CPU load and the number of programs currently running were recorded. For each of the measurements the following features were obtained: actual value measured, average of the previous 10 and the previous 100 values, difference between current and previous value, difference between current and average of last 10, difference between current and average of last 100 and difference between averages of previous 10 and previous 100. Classification was done by taking weighted votes from a pool of individual features and continually adjusting the weights to improve accuracy. The method was evaluated on a set of sixteen users who collected the data over a period of fifty days. At the window length of 1200 seconds true detection rate was measured to be 95.0%, the false positive rate was one per day and the CPU cycle usage was approximately 1.0%. From the paper, it was not clear if the system was designed to model the behavior of users or programs. The authors claimed that it was capable of modeling both, but this might not have been the case considering that users and programs differ in many ways (i.e., unlike users, programs are version specific or operating system specific and unlike programs, user behavior is not bounded by a finite set of possible actions).

2.2.6 Subverting Classification

When considering the weaknesses of any security mechanism we needed to examine the worst–case scenario, which in the case of behavioral modeling was an attacker having access to the data sources, learning algorithm and a classifier. Given this knowledge, could an attacker working in probabilistic polynomial time subvert the classification of the system and remain undetected? Any model that is computer generated (i.e., a generative model) is vulnerable to a clever attacker, because there is always a way to reverse engineer it. For an example, Wagner and Soto investigated mimicry attacks on system call IDSs [96]. They concluded that an attacker could evade the detection simply by inserting sequences of no–op instructions into attack sequence of system calls. Keystroke and mouse dynamics IDSs were somewhat more difficult to subvert because it was virtually impossible for an attacker to reproduce the same duration and latency timings for the keystrokes or the same distribution of the mouse clicks and mouse movements for the mouse by simply using the input devices. For the attacker to be successful in this case, an additional PC device was required to transmit the attack sequence from the serial port of the attacker's PC to the serial port of the target PC. This inherently assumed that there was an algorithm for converting the attacker's signature into the trusted user's signature running on the attacker's PC. The best way to prevent an attacker from subverting the system would be to ensure the privacy of the model of normal behavior. One way that this might be achieved is by encrypting the model, which in turn would make it difficult for the attacker to break the system but it also might make it harder for us to use the model.

2.3 Machine Learning Approaches in Behavioral Modeling

There are two machine learning approaches that are commonly applied to anomaly detection: supervised and unsupervised learning. Which was chosen depended on whether data was available for each user that could possibly use a particular host.

In such cases, supervised learning has been used, otherwise, unsupervised learning was applied. Note that in some cases data could be available for some users, but not for all and we investigated this scenario in Chapter 6.

Smoothing filter functions have found a wide area of application in the continuous time series domain [97]. In this document, we explored the applicability of smoothing filter functions to discrete, temporal sequence data (sometimes referred to as event sequence data). *Smoothing* refers to a process of reducing noise in a dataset, in our case, reducing minor, transient changes in user behavior. In Chapter 5 we showed that smoothing could effectively lower the number of false alarms. We differ the discussion of the related work in this area until Chapter 5.

2.3.1 Supervised Learning

Consider a machine that receives some sequence of inputs $x_1, x_2, x_3, .., x_n$ where the x_i are a sensory input, in this case user's keystroke, mouse or GUI events. This is the *data* that is collected for each user. In supervised learning for each input sequence there is a corresponding output sequence $y_1, y_2, y_3, .., y_n$ labeling which of the N users is currently using a workstation (e.g., labeling the current user as either a trusted user or an intruder) [61]. The goal in supervised learning is to produce the correct output (i.e., new label) given a new input. In this dissertation, we wished to determine if the current user of a computer system was the trusted user or an intruder, where an intruder was an employee whose dataset was collected in advance and whose data was available to us *a priori* to building a model of the trusted user behavior.

There are a multitude of supervised learning techniques available for building models of normal behavior when data is available for all users of interest. Neural networks (NNs) [30, 31, 80, 88], decision trees (DTs) [95], support vector machines (SVMs) [86], Finite State Automaton Machines (FSAM) [10–13], supervised sequence matching [52] and supervised statistical models [23–27, 58] are well known

supervised learning methods. In comparison to unsupervised learning methods, these techniques produced lower error rates and fewer false alarms, which was not surprising considering the training set had labeled data (i.e., it had instances of both normal and anomalous behaviors). Among the various techniques, SVMs have been shown to outperform others, but only when the dataset did not have skewed class-distributions and the number of irrelevant features (i.e., attributes) was not overwhelming [98,99]. The keystroke authentication approach in [86] that used one–class SVMs outperformed other methods with a false positive rate of 0.02% and a false negative rate of 0.1%.

2.3.2 Unsupervised Learning

Consider again a machine that receives a sensory input as a sequence of inputs $x_1, x_2, x_3, .., x_n$ – in the domain of our user re-authentication system these inputs are user's keystroke, mouse or GUI events. Unlike supervised learning, no corresponding sequence of outputs is present *a priori* to building a model of normal user behavior [61]. Only the valid user data is available, and the intruder(s) data is either unobtainable or does not exist. In such an environment, it is difficult to draw a boundary between the "normal" behavior of a valid user and the "anomalous" behavior of an intruder. If the boundary is too tight, the valid user's instances can be misclassified as those belonging to an intruder – this is referred to as the false positive or the false alarm rate. If the boundary is too loose some intruder's instances can be misidentified as those belonging to the valid user – this is referred to as the false negative rate. This is a well-known design trade-off in most stochastic systems. If the detection of intruders is of primary importance, a model of valid user behavior is built with tight bounds thereby causing a potentially high rate of false alarms. If we wished to limit the number of false alarms so as not to ask a user to authenticate him/herself repeatedly, a model of user behavior should be built with loose bounds thereby allowing some intruders to go undetected.

Many unsupervised learning techniques have been applied to anomaly detection: clustering [36, 50, 77–80, 83, 84], sequence matching [14–18, 42–44, 46, 47, 53], Markov Models (MMs) [35, 48, 49], statistical modeling [23–27, 33, 41, 54–56, 60, 64, 66, 75, 76, 79, 81] and expert rules [29, 32, 34, 37–39] are the most prominent choices.

2.3.3 Multi–modal Data Analysis

Multi–modal data analysis refers to situations in which there is more than one data source. The sources could be combined at the data level [100], at the feature level [95, 101] and at the classifier level [102–104]. Combination at the data level meant that all of the incoming data was merged into a single data file before any kind of processing was performed on it. A single set of features was then extracted from the combined data file without regard to each individual data source. This approach did not produce optimal results when each data source had unique characteristics not reflected in the obtained feature set.

Combination at the feature level meant that the feature/attribute values extracted on a per data–source basis were combined into a single feature file which was then forwarded to the classifier. Combination at the classifier level meant that a separate classifier was generated for each of the data sources and some kind of a voting scheme was implemented to produce the final classification decision.

In this document we constructed a *combined* classifier by merging features extracted from each data source (e.g., keystrokes, mouse and GUI) into a single feature vector (see Chapter 3). The optimal combination of different data sources is an active area of research in many scientific fields, but it is outside the scope of this thesis.

3. DATA SOURCES AND FEATURE EXTRACTION

User re–authentication has been tackled by the research community in two ways 1) indirectly by profiling the operating system and/or applications and 2) directly by profiling a valid user. Our research efforts took the direct approach to user re–authentication. We investigated the authenticity of a current user based on his/her keystroke dynamics, mouse activity[1] and GUI events. This chapter gives an overview of the user re–authentication process, describes the data collected and features extracted for each data source.

3.1 Overview of the User Re-Authentication Process

The goal of a user re–authentication system in an operational setting is to detect and flag anomalies in the behavior of a current user. To accomplish this task, the user re–authentication system first collects *clean* data from each user. The *clean* data is considered to be a collection of data points that are known to belong to a particular user who behaved in accordance with the computer security policy during the data collection process. Collecting the clean data can necessitate significant resources and be expensive in practice. When the data collection is completed, a profile of normal user behavior is built for each user. This profile is then used to compare the current user's behavior with that of a valid user. Any significant difference between the two profiles should signal an intrusion to a system administrator.

The overview of the system's operation is shown in Figure 3.1. Without any loss of generality let us assume that the user on the left was a trusted user and that the user on the right was an insider trying to access some confidential information that

[1]The proposed system can be implemented with different kinds of pointing devices (e.g., a mouse, touchpad, joystick, etc.).

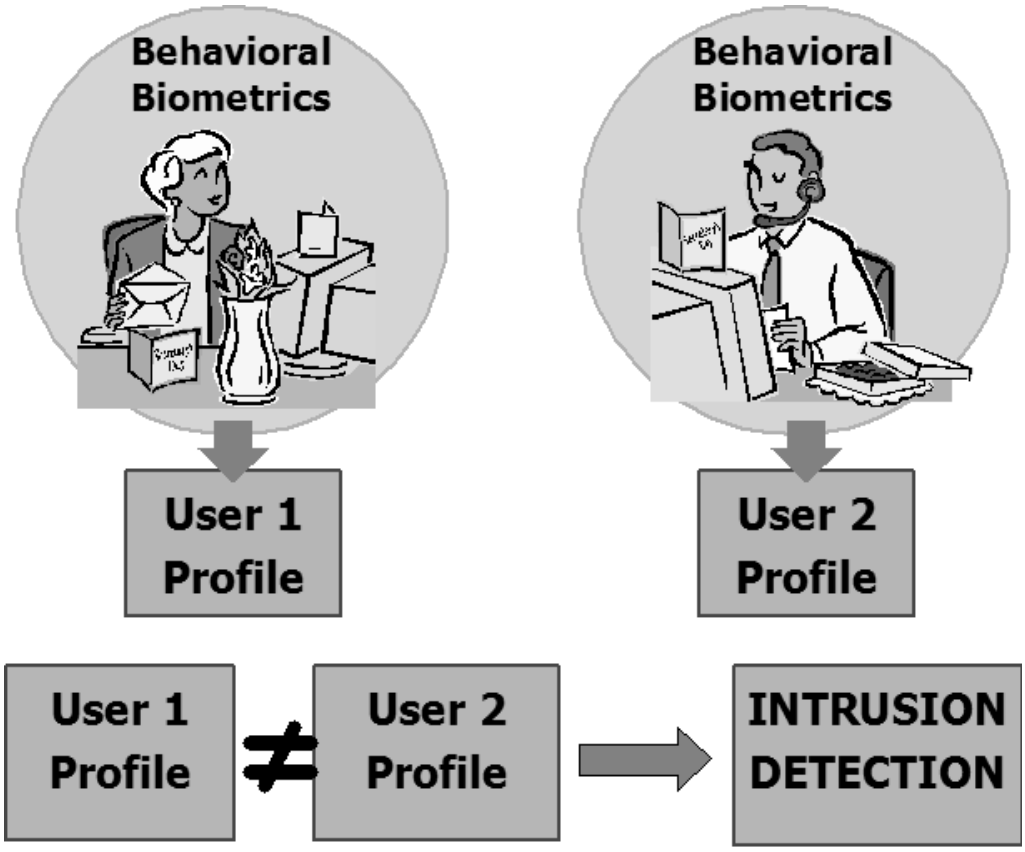


Fig. 3.1. Overview of the user re–authentication system operation.

was either stored on our trusted user's computer or was remotely accessed through her account. During the data collection phase, the system collected data from both employees and created two profiles of each user's normal behavior. When the intruder was using our trusted user's computer to access the confidential information, the system was collecting his mouse movements, keystrokes and GUI events and was comparing his behavior to that of the valid user. Mismatch between the two profiles raised an alarm and alerted the system administrator to a possible intrusion.

3.2 The Data Collection Process

Borland's Delphi 7.0 was used to program the data collection algorithm for the Windows operating system environment. The executable and the library file used

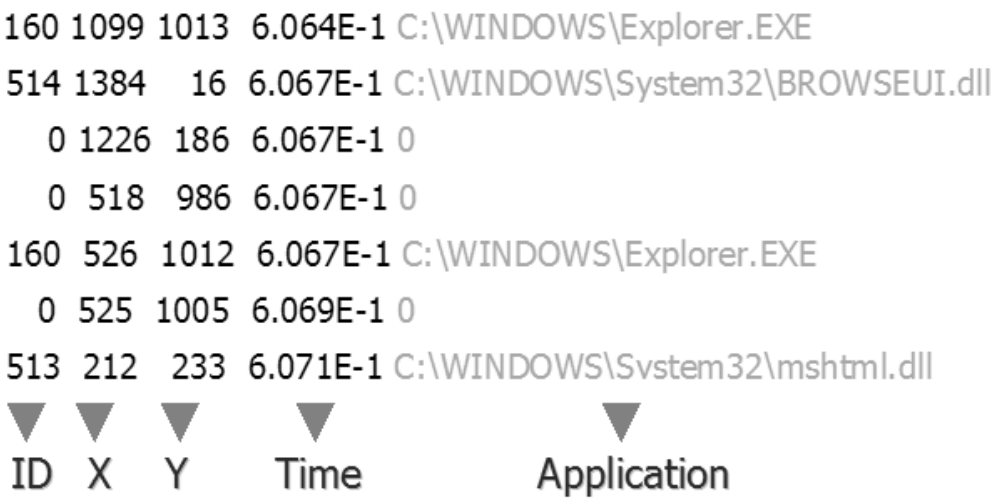


Fig. 3.2. Excerpt from User 10 *raw* data file.

to collect the data are shown in Appendices A and B. The $SetWindowsHookEx$ procedure was used to capture Windows API calls (i.e., events) on the return path from an application to the kernel. Three $SetWindowsHookEx$ procedures were employed: 1) *WH_MOUSE* for the mouse data; 2) *WH_KEYBOARD* for the keystroke data; and 3) *WH_CALLWNDPROCRET* for the GUI data.

The data collection was conducted in the computer labs of Tufts University and Purdue University. The computers used in the data collection were Dell Workstations.

3.3 Data Sources and Features Extracted for each Source

We considered three sources of data for user re–authentication: keystrokes, mouse movements and GUI events. For each user, we recorded his/her keystroke, mouse and GUI data points in the order in which they were induced. For each data point we recorded the identification number of the event, the X and Y screen coordinates, the time when it happened and the application in which the data point occurred. Each

collected data instance we referred to as the *raw* data instance and each collected data file we referred to as the *raw* data file.

An excerpt from user–10's raw data file is shown in Figure 3.2. The first column shows a unique identification number of each event (this was the number assigned to a particular event by the Windows operating system), the second and the third columns show the cursor's screen coordinates, the fourth column shows the system time when the event occurred and the last column shows the application in which the event occurred. We recorded the application information to ensure that users followed the data collection instructions and used the specified software. We did not use the application information as a feature, because we wished to determine the strength of each data source for the duration of the entire user login session, which in a practical setting entailed switching among several different applications. We did, however, test our user re–authentication system when users were given an identical task to perform (see Chapter 6).

We computed features by examining a *window of W* data points at a time. In this section we describe *all* candidate features. Our goal is to determine a predictive feature subset. Thus, we first designed a comprehensive candidate feature space, that we subsequently reduced to a subset of most discriminative features in Chapter 4.

3.3.1 Mouse Data

To capture a user's mouse behavior we recorded all Windows mouse events. In Figure 3.3, the "ID" column lists unique mouse–event identification numbers and the "Message Name" column displays the name of each mouse event defined by the Windows operating system. Upon a closer examination of the different mouse events, we grouped them according to their type into a hierarchy, because we conjectured that a hierarchical view of the data would enable feature extraction at varying levels of granularity.

ID	Message Name
513	WM_LBUTTONDOWN
514	WM_LBUTTONUP
515	WM_LBUTTONDBLCLK
516	WM_RBUTTONDOWN
517	WM_RBUTTONUP
518	WM_RBUTTONDBLCLK
519	WM_MBUTTONDOWN
520	WM_MBUTTONUP
521	WM_MBUTTONDBLCLK
522	WM_MOUSEWHEEL

ID	Message Name
161	WM_NCLBUTTONDOWN
162	WM_NCLBUTTONUP
163	WM_NCLBUTTONDBLCLK
164	WM_NCRBUTTONDOWN
165	WM_NCRBUTTONUP
166	WM_NCRBUTTONDBLCLK
167	WM_NCMBUTTONDOWN
168	WM_NCMBUTTONUP
169	WM_NCMBUTTONDBLCLK
160	WM_NCMOUSEMOVE

Fig. 3.3. Mouse events in Windows.

Figure 3.4 shows the mouse hierarchy. At the top level of the mouse hierarchy were *all* mouse data points. At the next level, the mouse data was split into "mouse events" and "client" and "non–client" (NC) area "mouse movements" (the client area was defined to be the area of the application window below the menu and toolbars). The client area mouse movements were *not* collected at the rate at which they were induced because this rate was prohibitively high (approximately several hundred movements per second); consequently, capturing all of them would have created a volume of data that was intractable. To mediate this problem, client–area mouse movements were rate limited by being recorded every 100msec *if and only if* the mouse position changed. We set the time interval to be 100msec because we speculated that the time it took human beings "to do things" was measured in seconds or even minutes. Although a 100msec interval was a long time from a computer processor's perspective, it was a relatively short time for a human, who might or might not have moved a mouse even by a pixel in this period.

The "mouse events" data was divided into mouse "wheel" movements and "clicks" and the "click" data was further divided into "single" and "double" clicks. Both

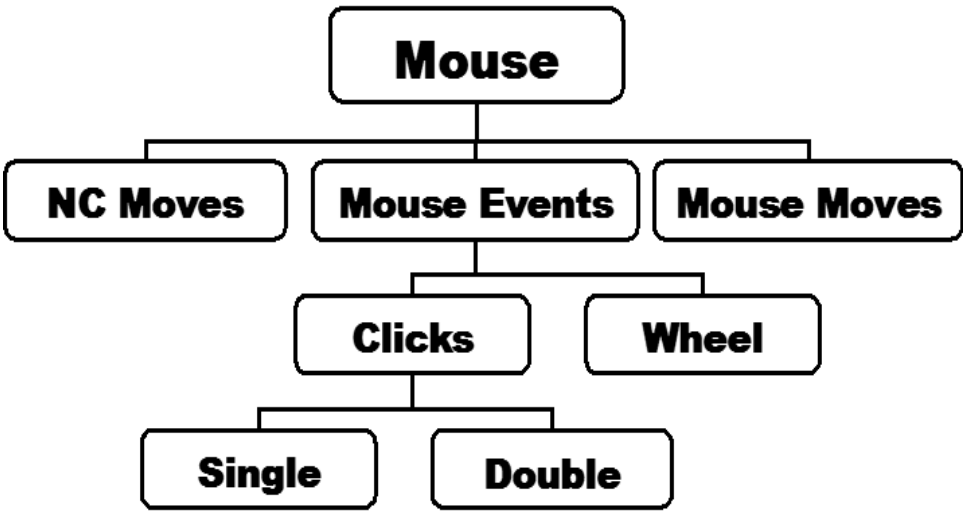


Fig. 3.4. Mouse Feature Hierarchy.

client and non-client area clicks were combined in the "clicks" category. Moreover, client and non-client area left, right and middle button single clicks were combined into the "single" category and client and non-client area left, right and middle button double clicks were combined into the "double" category. This was because single and double clicks were the most infrequent mouse events. Consider a user searching for a specific document on the World Wide Web. He/she used a keyboard to type in a search string, a mouse wheel or a scroll-bar to navigate through the search engine results, left mouse click to make a selection and likely spent the rest of the time reading the selected document. Observing his/her mouse behavior over time, it was clear that the amount of click-data was negligible in comparison to the mouse and wheel movement data. Consequently, we did not get better performance from higher granularity of the click–data.

3.3.2 Mouse Features

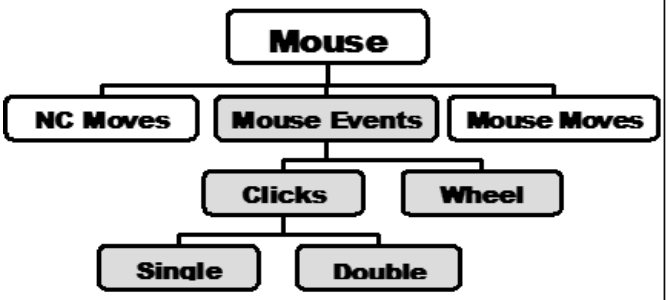
For each data point (mouse, keystrokes and GUI events) we recorded the cursor's screen coordinates X and Y and the system time when the event occurred. The client area mouse movements were rate sampled every 100msec and recorded *if and only if* the screen cursor moved. This difference resulted in differences in how features were extracted from client and non-client area mouse movements. We next describe the features extracted from the mouse data.

Mouse event features: Over a window of W data points we counted the number of events in each mouse event category and subcategory. The other features were extracted from examining either two subsequent mouse events or mouse events separated by k data points. The parameter k, called the *frequency*, was customized for each user and each type of events. We discussed it's selection in Section 3.3.7. We computed the mean, standard deviation and skewness (i.e., the third moment) of: 1) distance, 2) speed=$\frac{distance(P_i,P_j)}{time(P_i,P_j)}$, where $P_i = (x_i, y_i)$ and $P_j = (x_j, y_j)$ were two data points, 3) angle of orientation, 4) X, 5) Y coordinates and 6) *n–graph* duration,[2] where $n \in [1, 8]$. For example of $k = 3$, then to compute the distance between the ith and $i + 3$rd data point we calculated distance=$\sqrt{(x_{i+3} - x_i)^2 + (y_{i+3} - y_i)^2}$ (see Figure 3.6).

We chose a range of $[1, 8]$ for n in the n-graph duration, because research in keystroke dynamics achieved accurate results when monitoring the duration between the first to fourth keystroke [57]. Our value of n went to eight because each mouse event induced two separate events during the data collection (i.e., a *button–up* and a *button–down* event). This created 200 features. On the top row of Figure 3.5, the table on the right lists the features extracted from each shaded category of mouse event data shown on the left.

NC moves features: Over a window of W data points we counted the number of NC moves and compute the mean, standard deviation and skewness of: 1) distance,

[2]The n–graph duration is defined as the elapsed time between the first and the nth data point [57].



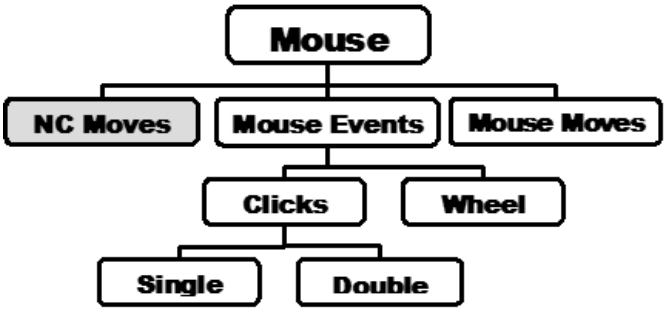
For each category and subcategory of **Mouse Events** compute

Number of Mouse events

Mean, **Std. Dev.**, **Skewness** of:

- Distance
- Angle of orientation
- Speed
- X-coordinate
- Y-coordinate
- n-graph, $n \in [1,8]$

over a window of W data points



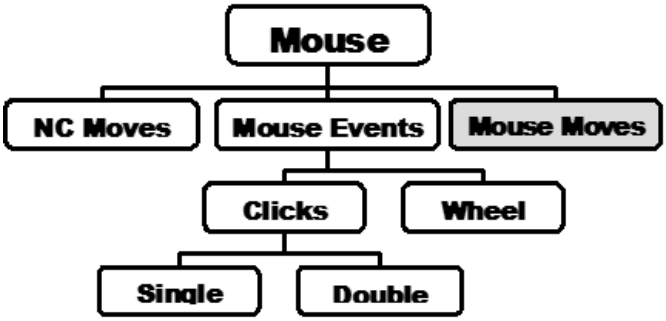
For **NC Moves** compute

Number of NC moves

Mean, **Std. Dev.**, **Skewness** of:

- Distance
- Angle of orientation
- Speed
- X-coordinate
- Y-coordinate
- n-graph, $n \in [1,8]$

over a window of W data points



For **Mouse Moves** compute

Number of Mouse moves

Mean, **Std. Dev.**, **Skewness** of:

- Distance
- Angle of orientation
- Speed
- X-coordinate
- Y-coordinate
- n-graph, $n \in [1,8]$

over a window of W data points

Fig. 3.5. Mouse Features: The top figure shows features extracted from the mouse events; the center figure show features extracted from the NC movements; and the bottom figure shows features extracted from the mouse movements.

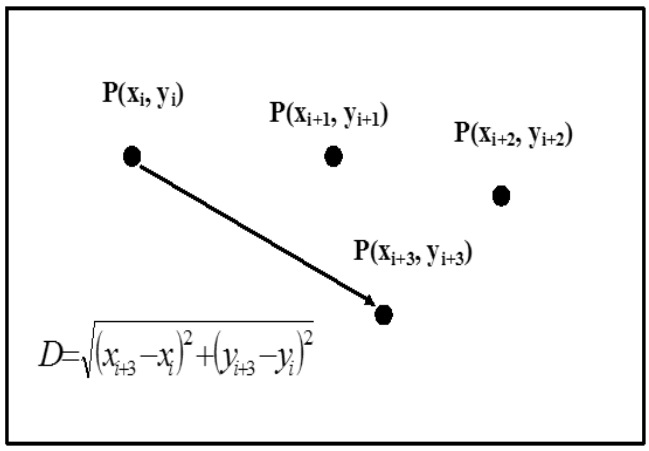


Fig. 3.6. Frequency example for $k = 3$. Distance is computed between the ith and $i + 3$rd data point.

2) speed 3) angle of orientation, 4) X, 5) Y coordinates and 6) *n–graph* duration, where $n \in [1, 8]$, between either two subsequent NC moves or NC moves separated by k data points. This created 40 features. The center row of Figure 3.5 lists the features extracted from the shaded "NC moves" category of mouse data.

Client–area mouse moves features: Over a window of W data points we counted the number of mouse movements and compute the mean, standard deviation and skewness of: 1) distance, 2) speed 3) angle of orientation, 4) X, 5) Y coordinates and 6) *n–graph* duration, where $n \in [1, 8]$, between either two subsequent mouse moves or mouse moves separated by k data points. This created 40 features. The bottom row of Figure 3.5 gives the features extracted from the shaded "Mouse moves" category of mouse data.

3.3.3 Keystroke Data

There were only two keystroke events in the Windows operating system environment: WM_KEYUP and WM_KEYDOWN. Each one of these events carried a unique identifier describing the exact keyboard button that was pressed or depressed. To build an accurate model of normal user keystroke behavior we recorded

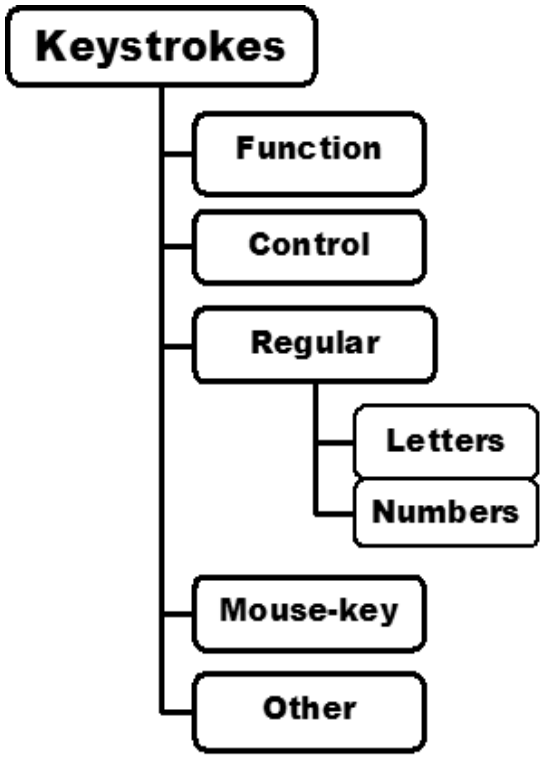


Fig. 3.7. Keystroke Feature Hierarchy.

each keystroke event. Similar to the mouse data, we grouped keystrokes according to their type into a hierarchy, which is shown in Figure 3.7.

The top level of the hierarchy contained all keystroke data points. At the next level the keyboard data was split into "function" keys (e.g., F1–F12), "control" keys (e.g., Control, Alt and Delete), "regular" keys (i.e., letters of the alphabet and numbers), "mouse" keys (i.e., keys used by expert users to navigate through a computer system without the use of a mouse device, such were Page Up/Down, Tab, arrow keys, etc.) and "other" keys (e.g., punctuation keys, Pause/Break, PrtSc/SysRq, etc.). Finally, the "regular" keys were divided into "letters" and "numbers."

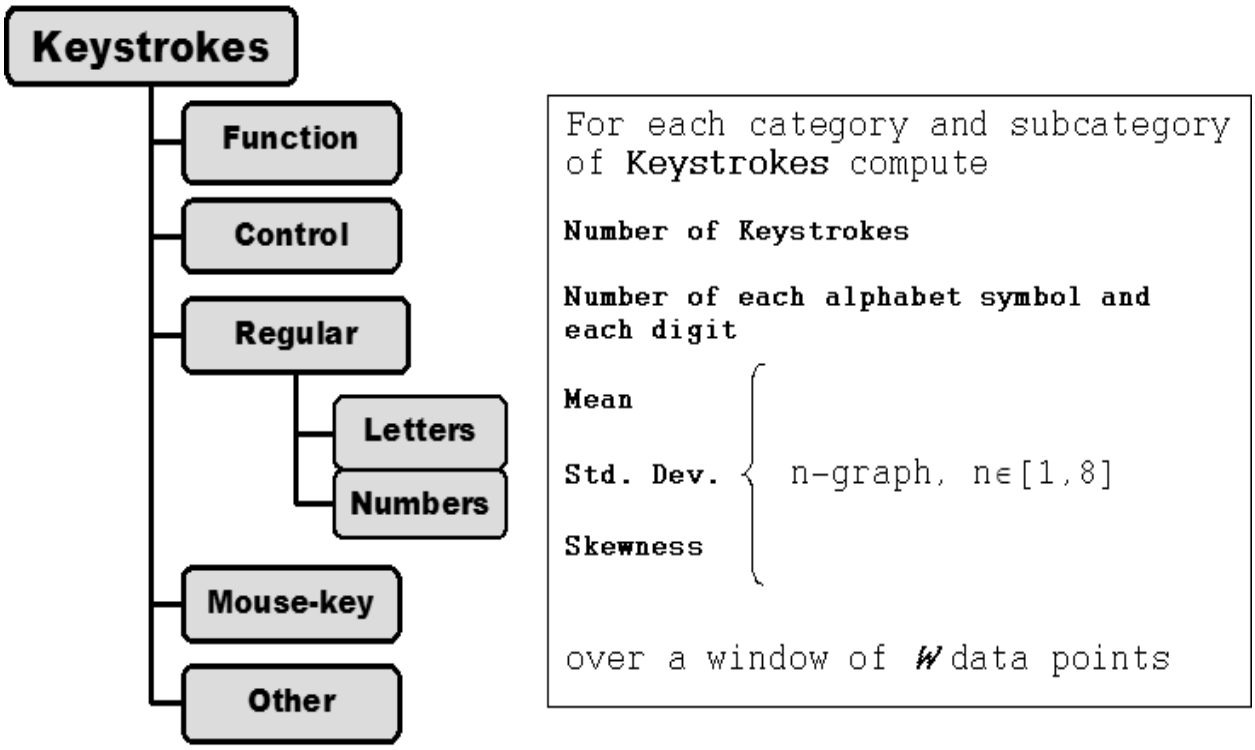


Fig. 3.8. Keystroke Features.

3.3.4 Keystroke Features

Although we recorded the cursor's X and Y screen coordinates for each keystroke event we did not use this information to compute features for the keystroke data.

Keystroke features: Over a window of W data points we counted the number of events in each keystroke category and we computed the mean, standard deviation and skewness of the *n–graph* duration between consecutive keystrokes in the category where $n \in [1, 8]$. Each keystroke event also induced two separate events during the data collection (i.e., a *key–up* and a *key–down* event). Because keystroke events were less frequent than mouse events we computed the n–graph durations between two subsequent events only, i.e., $k = 1$ for the keystroke data. We also counted the number of occurrence of each alphabet letter and each numeral thereby obtaining 26 alphabet features (i.e., A(a)–Z(z)) and ten (i.e., 0–9) numeric features. This created

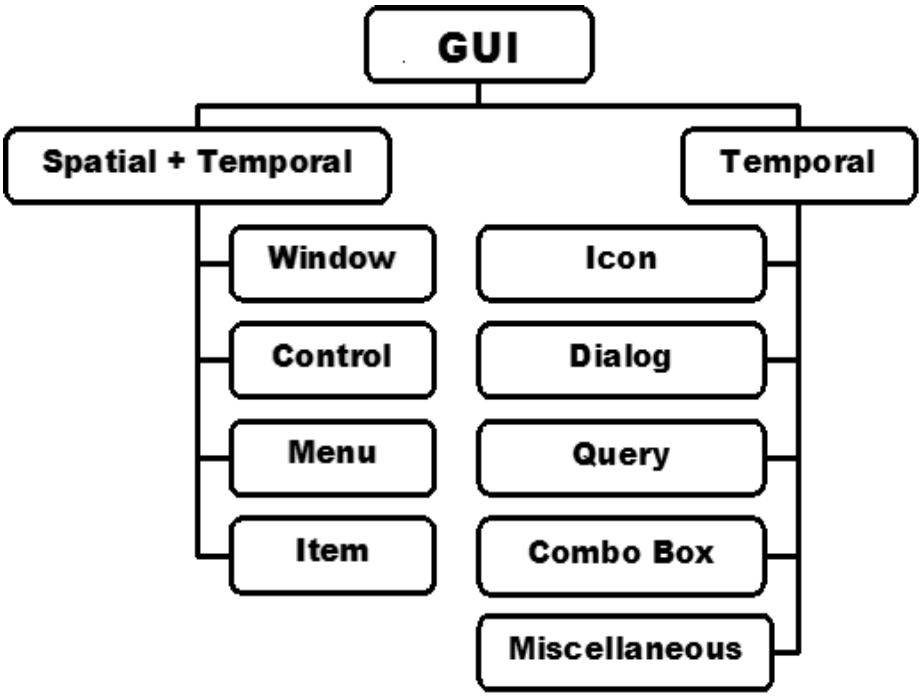


Fig. 3.9. GUI Feature Hierarchy.

236 features. Figure 3.8 lists the features extracted from each shaded keystroke category.

3.3.5 GUI Data

We collected data induced by 138 GUI events (see Appendix C for a complete list). Similar to the mouse and keystroke data, we grouped GUI events according to their *function* into a hierarchy, which is shown in Figure 3.9. The top level of the hierarchy contained *all* GUI events. At the next level the GUI data was split into the "spatial and temporal" and "temporal" *only* data because some of the GUI events were purely temporal in nature and some were both temporal and spatial. The "temporal and spatial" data was split into "window" (i.e., scroll bar, minimize, maximize, restore, move, etc.), "control" (i.e., application and process control, open/close, etc.),

Name	Description	Locality
Clipboard	Functions and messages for data transfer	T
Control	Drawing control	T & S
Dialog	Dialog box notifications	T
Icon	Icon painting notifications	T
Item	List/combo box, menu, button	T & S
Menu	Menu notifications	T & S
Miscellaneous	Power, close app, language change, etc.	T
Query	User queries	T
Window	Window notifications	T & S

Fig. 3.10. Temporal *only* and temporal+spatial GUI data.

"menu" (e.g., open, select, navigate, close) and "item" (e.g., list, button, etc.) events and the "temporal" data was split into "icon," "dialog," "query," "combo box" (e.g., open/close, select, move, resize, etc.) and "miscellaneous" events (e.g., power up/down, language change, background color change, etc.).

3.3.6 GUI Features

GUI events encompassed those events that carried both spatial and temporal information and others that carried temporal information only. We listed the different types of GUI events in Figure 3.10. The first, second and third columns of Figure 3.10 list a type of the GUI event, description and whether a particular event was temporal (T) or both temporal and spatial (T & S), respectively.

Spatial features: Over a window of W data points we counted the number of events in each spatial category and we computed the mean, standard deviation and skewness of: 1) distance, 2) speed, 3) angle of orientation, 4) X, 5) Y coordinates and 6) *n–graph* duration, where $n \in [1, 8]$, between either two subsequent spatial events or events separated by k data points. This created 200 features. The top sub–

Table 3.1
A list of frequent events.

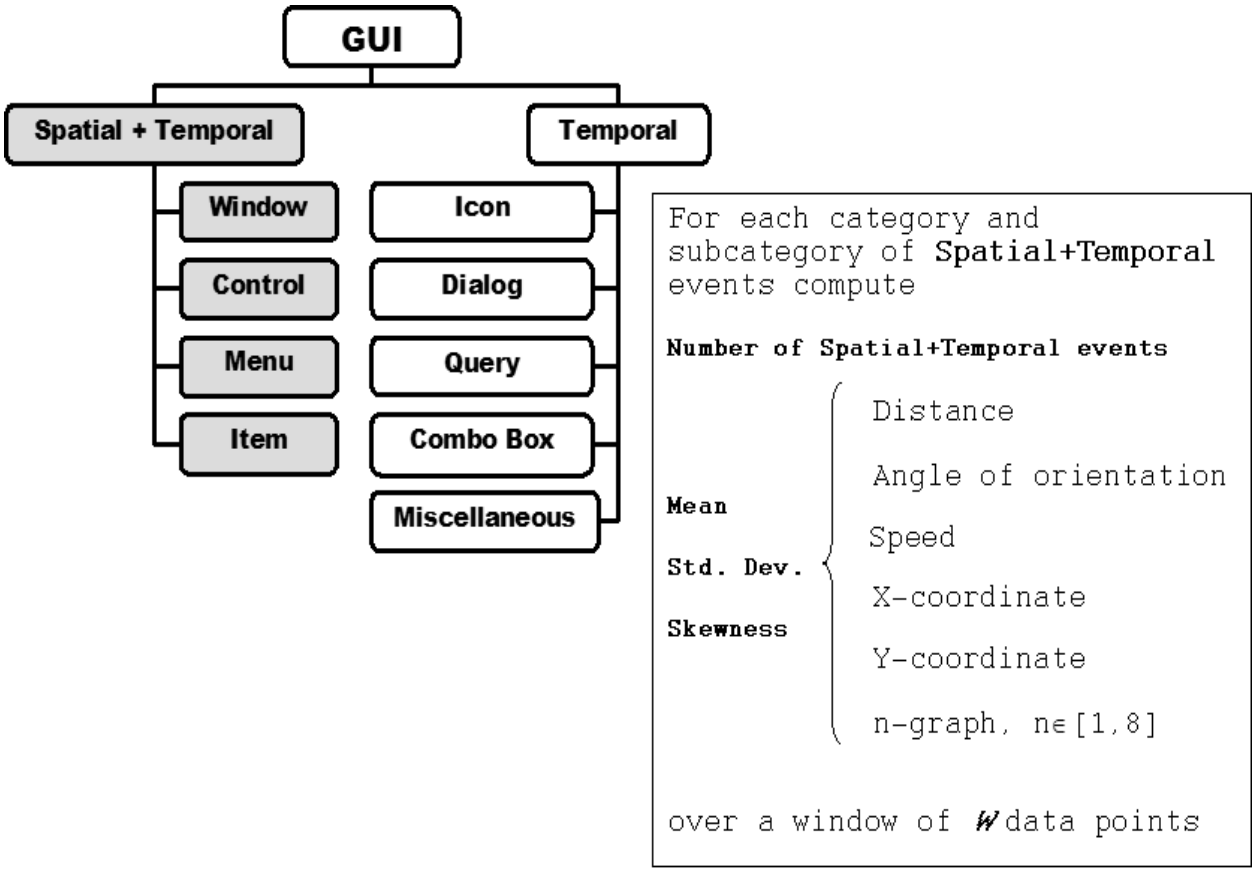
#	**Event Description**
1	*All* mouse events
2	Mouse movements
3	Mouse wheel movements
4	NC mouse movements
5	*All* GUI events
6	Spatial+Temporal
7	Temporal
8	Window
9	Dialog

figure of Figure 3.11 lists the features extracted from the shaded "Spatial+Temporal" category of GUI data.

Temporal features: Over a window of W data points we counted the number of events in each temporal category and we computed the mean, standard deviation and skewness of the *n–graph* duration between consecutive temporal events where $n \in [1, 8]$. Similarly to the mouse data, we computed the n–graph durations either between two subsequent temporal events or between two events separated by k data points. This created 240 features. The bottom sub–figure of Figure 3.11 lists the features extracted from the shaded "Temporal" category of GUI data.

3.3.7 Summary of the Feature Space

Ideally, the window size W is specified for each user and the frequency k is specified for each user and each subcategory of events in the data hierarchies of Figures 3.4 to 3.9 for the mouse, keystroke and GUI, respectively. We examined



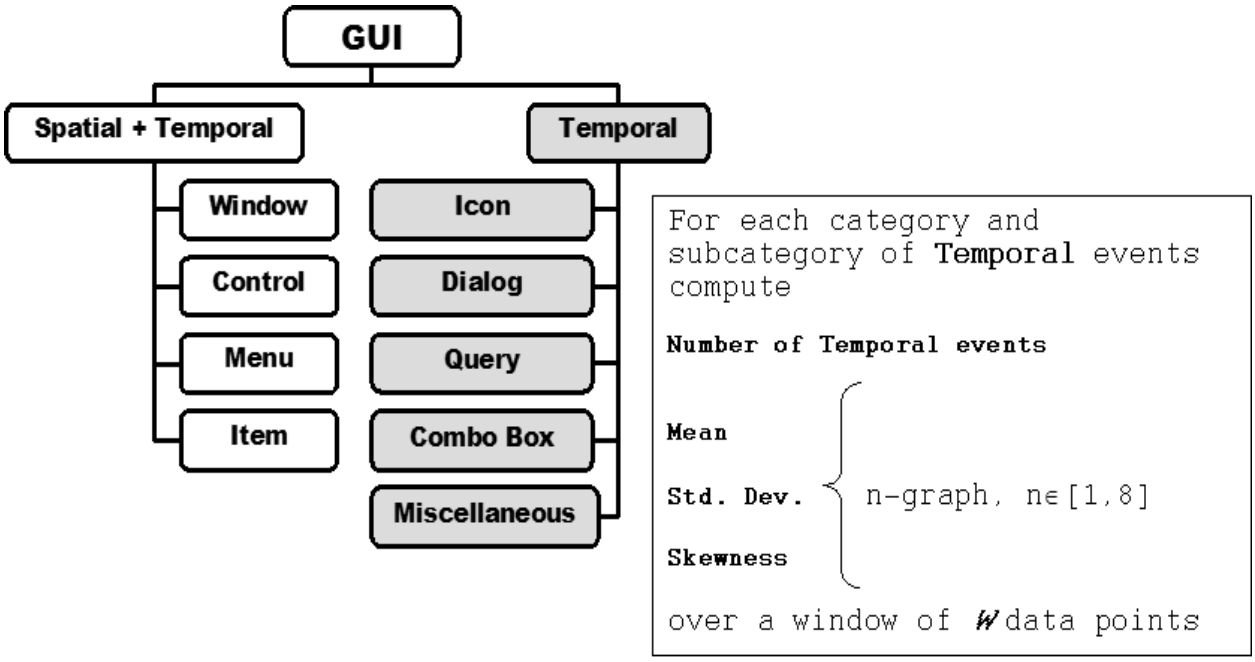


Fig. 3.11. GUI Features: The top figure shows features extracted from the spatial and temporal events; and the bottom figure shows features extracted from the temporal *only* events.

the following candidate values $\{100, 300, 500, 1000\}$ for W and $\{1, 5, 10, 15, 20\}$ for k. In the experiments in Chapter 4 we set W to be 500[3] data points and k to be 8 for frequent events (see Table 3.1) and 1 otherwise. We deemed frequent those events that were induced at least ten times per second and infrequent otherwise. In Chapter 5 we revisited the selection of W and investigated the performance of our user re–authentication system for different values of W.

A complete list of features can be found in Appendix D. The size of the *candidate* feature space was 956. We postulated that some subset of the most uncorrelated features was the best discriminator of a user's behavior and validated our supposition in Sections 4.3.2 and 4.3.3. The most uncorrelated features in our dataset were those features generated from the *leaves* of each tree–hierarchy (see Figures 3.4 to 3.9). We also examined the strength of each data source (i.e., mouse, keystrokes and GUI) individually and in combination in Chapter 4.

Furthermore, we studied the conjecture that grouping data points according to their type improved performance. We tested this conjecture on the mouse data. We selected the mouse dataset because its hierarchy had the most levels – four, whereas the GUI and keystrokes each had three. In Chapter 4 we first removed the features associated with the fourth level of the mouse hierarchy and re–ran the Anomaly Detection experiment (see Section 4.3.2) to obtain new measurements of the mouse classifier performance. Then we removed the features associated with both the fourth and the third level of the mouse hierarchy and once again re–ran the Anomaly Detection experiment to obtain a new set of results. We showed that grouping mouse points by function outperformed the non–hierarchized data set.

[3] $W = 500$ was equivalent to a fifty second time period if the user utilized I/O devices.

4. EMPIRICAL ANALYSIS OF BIOMETRIC SOURCES IN USER RE–AUTHENTICATION

To determine whether the current user's behavior was anomalous or not one must first build a model of normal behavior. Building the model required an initial training phase, during which the user's data was collected, model parameters were selected and the final model was produced. This model was then used to continually monitor the user's account. If the current behavior of a user deviated significantly from the model of normal behavior, the system flagged this behavior as anomalous and did one or all of the following: alerted the system administrator, asked the user to authenticate again and/or closed the current login session. In this chapter we describe the experimental methodology of a user re–authentication system and present results that illustrate the strength of each data source, individually and in combination, when identifying a user.

4.1 Building a Model of Normal Behavior

The problem of user re–authentication could be regarded as either a supervised or an unsupervised learning problem. The choice depended on whether public access to hosts was restricted or not. In Chapter 1 we stated that in a closed setting, one could collect data from all employees, and then apply a supervised learning algorithm to obtain a classifier able to discriminate each employee from the others. There were two drawbacks to this approach: 1) the data for all users needed to be collected to perform the classification - in a practical setting this could be expensive in terms of resources; and 2) a highly skewed class distribution would make it difficult for many learning algorithms to output a classification that was not the majority class

prediction (i.e., the "non–uniform" class distribution problem), which occurred in our case when the number of classes, (i.e., users), grew large [105].

If collecting data from all individuals who had physical access was not feasible, then re–authentication might have been viewed as an unsupervised learning problem. In this case, the data would have been collected for each individual user whose behavior we were trying to model. During the user re–authentication process, a "normal" user's profile, generated from the data during the training phase, was compared against the current user's behavior (i.e., self versus not–self comparison). If there was a significant difference between the two, the current behavior was classified "anomalous." Unlike the supervised learning approach, we would not have been given the intruder's profile a priori to detect his/her presence. The difficulty with unsupervised anomaly detection is that the false positive rate of such approaches is often unacceptably high [106].

In our experiments we first assumed a closed–setting scenario and applied a supervised learning algorithm to evaluate the performance of our system. We then examined the ability of a supervised classifier to detect previously unseen intruders in Chapter 6, with the objective of demonstrating that we were able to preserve high accuracy measures of supervised learning approaches while being able to detect previously unseen users.

4.2 Experimental Methodology

We begin this section with the introduction of the learning algorithm used for feature subset selection (FSS) [107] and classification. We then provide the description of the data set and discuss two anomaly detection implementation schemes. We present a series of experiments designed to investigate the applicability of each biometric source, individually and in combination, to user re–authentication. For each experiment we explain its purpose, present and discuss the results, and draw conclusions.

Table 4.1
A contingency table.

True Value	**Classified "+"**	**Classified "-"**
pos	TP	FP
neg	FN	TN

4.2.1 Feature Subset Selection and Classification

We applied a supervised learning algorithm to determine whether we could discriminate the users. We assumed a closed–setting scenario in which data could be readily obtained from all of the employees. The intruders were likely to come from "within." We used a readily–available decision tree algorithm WEKA j48 [108] to perform feature subset selection and classify the instances as either belonging or NOT belonging to a valid user. We also used Support Vector Machines (SVMs), SVMlight [109] and LibSVM [110] packages, to build a model of normal user behavior. SVMs, through the SVMlight [109] and LibSVM [110] were known to produce good results on many problems [111], but the results obtained on our datasets were inferior to those obtained with the decision tree algorithm. This was because our datasets had highly skewed class distributions (i.e., the ratio between the minority and majority classes exceeded 1:60 in our experiments) and contained many redundant features [98, 99].

To better understand the biases exhibited by different classifiers we considered their mechanisms for feature selection and the effect of over-sampling (i.e., randomly sampling from a class with a sampling frequency significantly higher than twice the highest frequency of the class being sampled). We used a contingency table (see Table 4.1) to categorize features into three groups: 1) positive features $\frac{tp}{pos} > \frac{fn}{neg}$, 2) negative features $\frac{tp}{pos} < \frac{fn}{neg}$, and 3) neutral features $\frac{tp}{pos} = \frac{fn}{neg}$. A decision tree classifier had an embedded feature selection mechanism that led to its selection of

positive features to branch. Over–sampling increased the complexity of the tree and allowed for many negative features to be used in the built trees. Therefore, decision trees were sensitive to sampling but insensitive to feature selection [99].

Feature selection had a significant impact on SVMs particularly when the number of features was large [112]. Positive (minority class) instances tended to reside far away from the boundary when the data was skewed. Over–sampling had no effect because no new data was generated and hence the support vectors did not change. Over–sampling, by increasing the error penalty of the minority class, resulted in a change in the margin if there were some misclassifications during training. If no error occurred in the minority class during training, over–sampling was ineffective. This was especially true when the number of features was large as was the case with our dataset [99].

To restore the balance between the minority and majority classes in our datasets we randomly over–sampled from the minority class. We then used the decision tree classifier to build a model of normal user behavior from the balanced dataset. Another reason we chose to use decision trees and not SVMs was because they provided a comprehensible representation of their classification decisions. The regression trees [113] were not applicable in our domain because of their quantized output (our task was to determine the identity of a current user). Although techniques such as boosting [114, 115] might have obtained higher classification accuracy their running time was prohibitive for the large scale experiments reported in this paper. In an operational setting they should be tested.

For details regarding the specifics of WEKA j48 the reader is referred to [108]. Here are provided only the key aspects of the algorithm related to decision tree estimation, particularly as it pertains to feature selection. A decision tree is a predictive model; that is a mapping from observations about an item to conclusions about its target value [116]. In a decision tree *leaves* represent classifications and *branches* represent conjunctions of features that lead to those classifications. In Figure 4.1 we

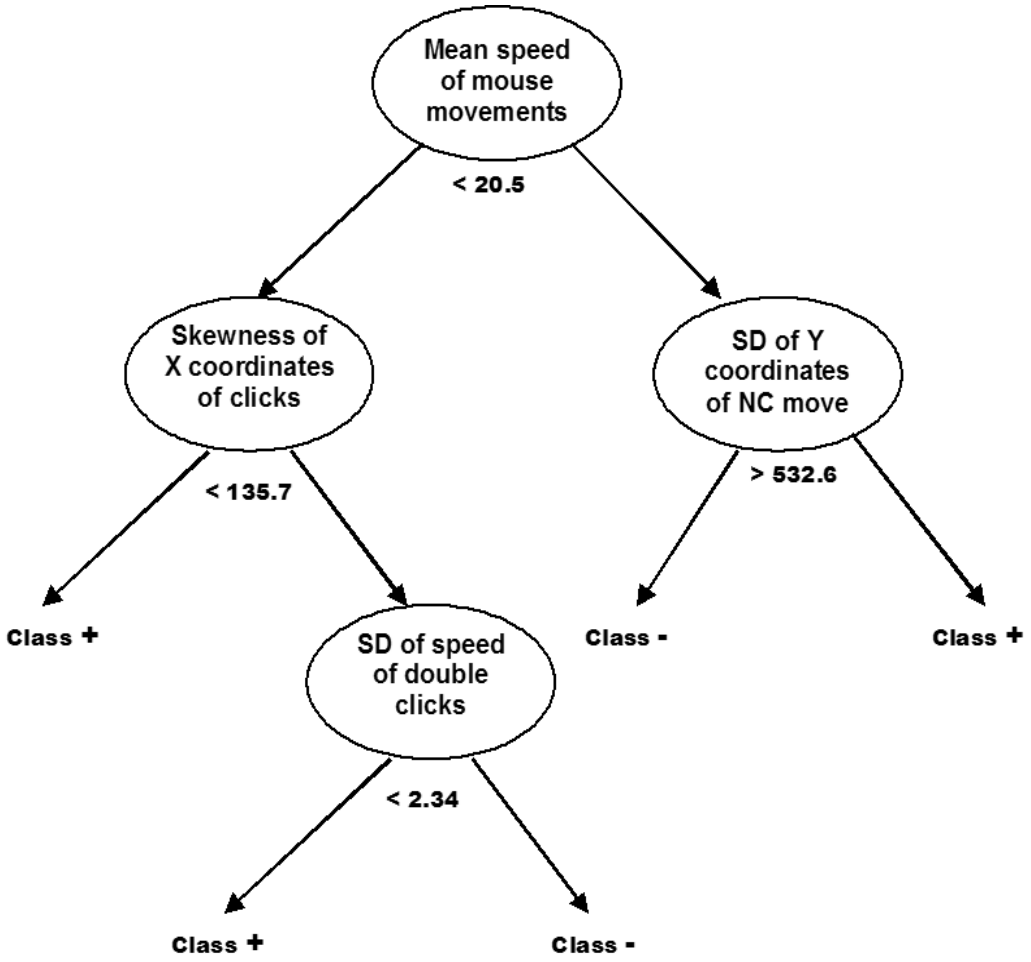


Fig. 4.1. A decision tree.

showed a simple example of a decision tree (e.g., this tree had three internal nodes and five leaves).

The most important element of the decision tree estimation algorithm is the method used to estimate splits at each internal node of the tree. To do this, WEKA j48 used a metric called the *information gain ratio* that measured the reduction in entropy in the data produced by a split. In this framework, the test at each node within a tree was selected based on splits of the training data that maximized the reduction in entropy[1] of the descendant nodes. Using this criterion, the training

[1] $Entropy = -\sum_x P(x) * log_2[P(x)]$, where P(x) is the probability density function of a random variable X when $X = x$.

data was recursively split such that the gain ratio was maximized at each node of the tree. This procedure continued until each leaf node contained only examples of a single class or no gain in information was given by further testing. The result was a large, complex tree that overfitted the training data. If the training data contained errors, then overfitting the tree to the data in this manner could have led to poor performance when classifying previously unseen data. Therefore, the tree needed to be pruned back to reduce classification errors when data outside of the training set was classified. To address this problem WEKA j48 used confidence–based pruning, which pruned any rule that did not satisfy a confidence threshold. Furthermore, any rule that was a subset of the pruned rule was also pruned. The details could be found in [117].

When using the decision tree to classify new examples, WEKA j48 supplied both a class label and a confidence value for its prediction. The confidence value was a decimal number ranging from zero to one – one meaning the highest confidence – and it was given for each classified instance. The confidence values were computed by first estimating the success probability (probability of a positive outcome) of each prediction and then calculating the corresponding z–scores of a normal distribution with the zero mean and the unit variance [118]. The assumption that the success probabilities were distributed *normally* was justified by the Central Limit Theorem (CLT).

4.2.2 Data Set I

A group of 61 volunteers was asked to participate in the data collection process. The group consisted of undergraduate and graduate students majoring in computer science at either Purdue or Tufts University. Volunteers were instructed to use a Windows machine and to behave as they would normally in any other situation. They were given a reading assignment followed by a set of twenty questions about the material they just read (see Appendix E for more detail). Users were specifically

instructed to read the assignment off the screen as opposed to reading it from a printout in order for us to record as many events as possible while they navigated between the reading material and a text editor. They were also given a set of web pages to look at and answer yet another set of questions.

Volunteers collected the data for 4.10 hours on average; the standard deviation was 4.52 hours.[2] They had ten days to complete the assignment and some of them chose to do it in multiple sittings (e.g., they answered five questions, left their workstation for a couple of minutes, hours or even overnight and then returned to finish the remainder). When we computed the time it took each volunteer to complete the assignment we chose not to subtract the "pause" time, because the "pause" time is a part of natural and normal user behavior. Instead, to mediate any artifacts induced by the elapsed time measurements we computed the number of raw data instances produced by each volunteer. In Chapter 3 we defined raw data instances as the original data points collected by each user. Each raw data instance had the unique identification number of the event, the X and Y screen coordinates, the time when the event happened and the application in which the event occurred (see Figure 3.2). The average and the standard deviation of the number of raw data instances per dataset was 92,068 and 55,712, respectively. Regarding the amount of storage space needed per user, the average number of raw data instances translated to approximately 4.5MB of space.

In Chapter 3 we postulated that grouping data points according to their type improved performance. In this chapter we empirically validated our conjecture on the mouse data because it had the most levels of hierarchy - four, while keystrokes and GUI each had three. From the 61–user dataset we extracted the mouse data points and computed the feature vectors (a feature vector instance was equivalent to a classification instance) from the mouse hierarchy in Figure 3.4. After computing the feature vectors for each user we observed that some users had significantly fewer

[2]The average time a user spent on the data collection did not follow the normal distribution because it had a heavy right tail. It did however follow the gamma distribution.

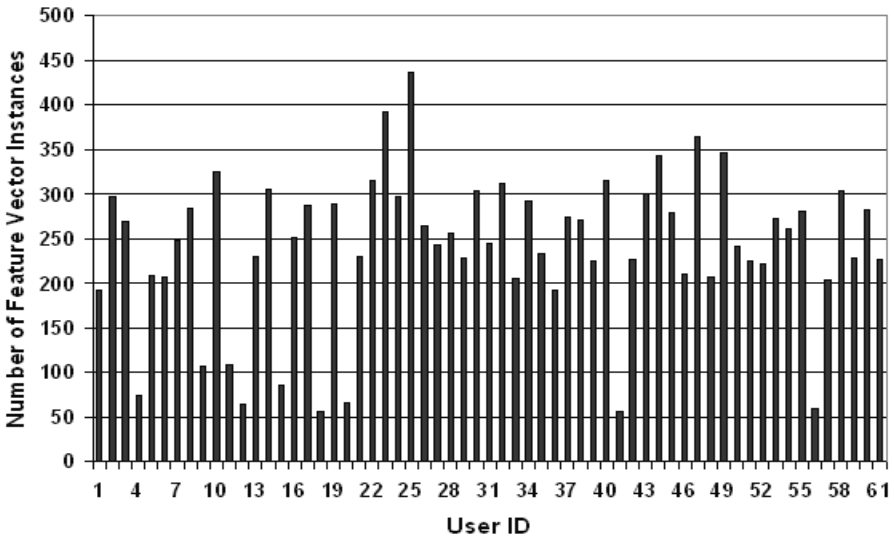


Fig. 4.2. Number of mouse feature vector instances per user. Nine users have fewer than 150 instances in their data sets.

vector instances in their data sets than others (recall that each feature vector was a summary over a window of W raw data points). A classifier building a model of valid user behavior could fail to generate an accurate profile when the number of instances of one of the classes was small. This classifier would either produce a one–node decision tree that classified every instance as the majority class or produce a larger decision tree in which the conjunctions of features at the tree nodes would not be able to achieve meaningful class separation. To address this issue we decided to set a threshold on the number of feature vector instances per user and to omit those users whose dataset fell below this threshold for the experiments involving the mouse data *only* (see Sections 4.3.3 and 4.3.4). Our rationale was that we had insufficient mouse data to properly analyze whether we could discriminate these users from the others. We set the threshold at 150 because in the experiments we used 10% of the data for testing (90% for training) and we wanted to have at least 15 test instances per user on which to base our results. After examining each user data set in Figure 4.2 we observed that nine users produced at least one standard deviation below the

```
procedure Compute_FB_rate(length:integer; class_label,true_label: TDynamicArray);
var
count_alarm, count_N, FBells, i: integer;
FB_rate: double;
begin
    count_alarm:=0;  FBells:=0; count_N:=0;

    for i:=1 to length do
    begin
        // if_statement when the true label is normal
        if(true_label[i]='N') then
        begin
            count_N:=count_N+1;
            if(class_label[i]='A') then
                count_alarm:=count_alarm+1
            else if((class_label[i]='N') and (count_alarm>0)) then
            begin
                FBells:=FBells+1;
                count_alarm:=0;
            end;
        end;
        // if_statement when the true label is anomalous and count_alarm>0
        // this adds the last false bell
        if((true_label[i]='A') and (count_alarm>0)) then
            FBells:=FBells+1;

    end; // end of for loop

    FB_rate:=FBells/count_N;
end;
```

Fig. 4.3. Algorithm to compute FB rate.

average amount of data. Thus we excluded users 4, 9, 11, 12, 15, 18, 20, 41 and 56 from the data set, leaving us with 52 users in the *mouse* dataset.

4.2.3 Implementation Schemes

We reported results for two implementation schemes. Our first scheme classified one feature vector instance at a time. If the instance belonged to the profile of a valid user, we serviced the user's request. If the instance did not belong to the profile of a valid user, we did one or all of the following: alerted the system administrator, asked user to authenticate again and/or closed the current login session. Such a simplistic implementation scheme might induce a high rate of false alarms. Consider a user

Table 4.2
Performance metrics.

Performance Metric	**Definition**
False Positive Rate	FP/(#positive instances)
False Bell Rate	FB/(#positive instances)
False Negative Rate	FN/(#negative instances)
Error Rate	(FP+FN)/(#instances)
Bell Count	Number of *bells* raised on the test data

having lunch and operating the keyboard and the mouse device at the same time. His/her current behavior might differ from the profile of normal behavior and this could prompt the user re–authentication system to raise an alarm(s). We conjectured that some minor, transient changes in user's behavior might be *smoothed–out* when the behavior was observed over several classification instances.

To this end, we implemented a smoothing filter on top of the basic scheme. We looked at a window[3] of $n \in \{1 : 11\}$ feature vector instances at a time and if $m \in \{1, n\}$ of those instances belonged to a valid user, we serviced the user's request, otherwise we raised an alarm. The values chosen for n and m were *user specific* and empirically derived on the training datasets by minimizing the *criterion* function: $min \sum_{i=1}^{N}(c_1 * FP(u_i) + c_2 FN(u_i))$; $c_1, c_2 \in R^+$ where u was a user and $FP(u_i)$ and $FN(u_i)$ were the false positive and false negative rates of user u_i, $i \in [1, N]$, respectively. We reported results in Section 4.3 with $c_1 = c_2 = 1$. The choice of c_1 and c_2 in practice is deployment specific.

In addition to measuring the false positive and false negative rate we also measured the *false bell rate*, which we defined to be the number of *bells* divided by the number of positive instances (see algorithm to compute false bell rate in Figure 4.3).

[3]We chose a range of 1 to 11 in this chapter because some users have only 15 test instances in their datasets. In Chapter 6 we ran experiments on the combined data source *only*, which had more data per user thus allowing us to expand the smoothing range.

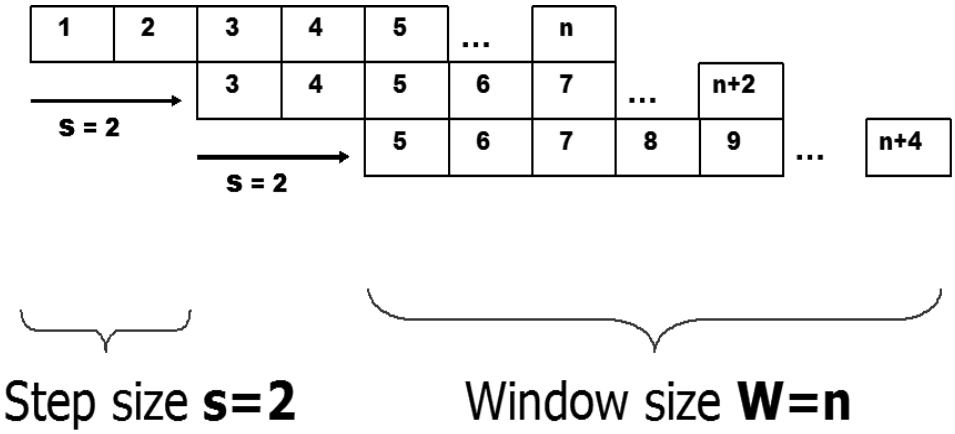


Fig. 4.4. Overlapping windows of size n with the step size of $s = 2$.

A single *bell* was defined as a sequence of contiguous alarms without the interrupting "normal" window. In a practical setting, when the alarm is raised for the first time, the bell is turned on and kept on until it is serviced by a system administrator. Consecutive alarms are observed as being a part of the original bell. By counting the number of bells and dividing this count by the number of positive instances we obtained the false bell rate (i.e., FB rate). A high false bell rate was an indicator of high variability and inconsistency in user behavior.

In our experiments we reported the false bell, false positive, false negative and error[4] rate as well as the actual number of bells sounded for both implementation schemes (see Table 4.2 for a complete list of performance metrics). For the combined data source we also examined the tendencies of error rates as the smoothing window m and the detection time increased. For each experiment we generated the receiver operating characteristics (ROC) plots.

4.3 Empirical Evaluation

Our experiments in this chapter were designed to address our first objective:

[4]Error rate was defined as the sum of the false positives and false negatives divided by the total number of classified instances.

> *Computer Security Objective*: To design and implement an on–line, scalable user re–authentication system founded on the data collected from user's inputs (mouse and keyboard) and Graphical User Interface (GUI) events for the purpose of detecting an attacker.

We carried out a detailed analysis of the scalability of our system in Chapter 6. For each experiment we reported the results for the basic and smoothing filter implementation schemes. In Section 3.3.7 we stated that each classification instance was computed over a window of W data points where $W = 500$. In our experiments, we overlapped windows so that a new window was obtained from the previous $W - s$ and subsequent s points, where $s = stepsize$ ($s = 50$ in our experiments). Figure 4.4 shows overlapping windows of n data points with the *stepsize* of 2. We overlapped windows to reduce the *time–to–alarm* (e.g., the time it takes to detect an intruder). We classified an instance every s points following the classification of the first instance. For a stepsize of $s = 50$ this was roughly equivalent to a five–second detection time period, if the user was active. We recorded mouse movements every 100msec, so 50 data points corresponded approximately to a five-second time interval.

To evaluate our methods we used a ten–fold cross–validation (CV). For each of the ten runs, 90% of the user data set was used for training and the remaining 10% was used for testing – a different 10% was used for testing in each run. The results reported were averaged over the inner eight cross–validation folds. We omitted the results obtained from the first and last 10% of the data because our volunteers used this time to familiarize themselves with the data collection software and to wrap–up the data collection process, respectively, making their behavior different than the remainder of the data collection. A comparison of the feature vector values in each data file obtained for each user revealed higher variance in user behavior in the first and last 10% of the data.

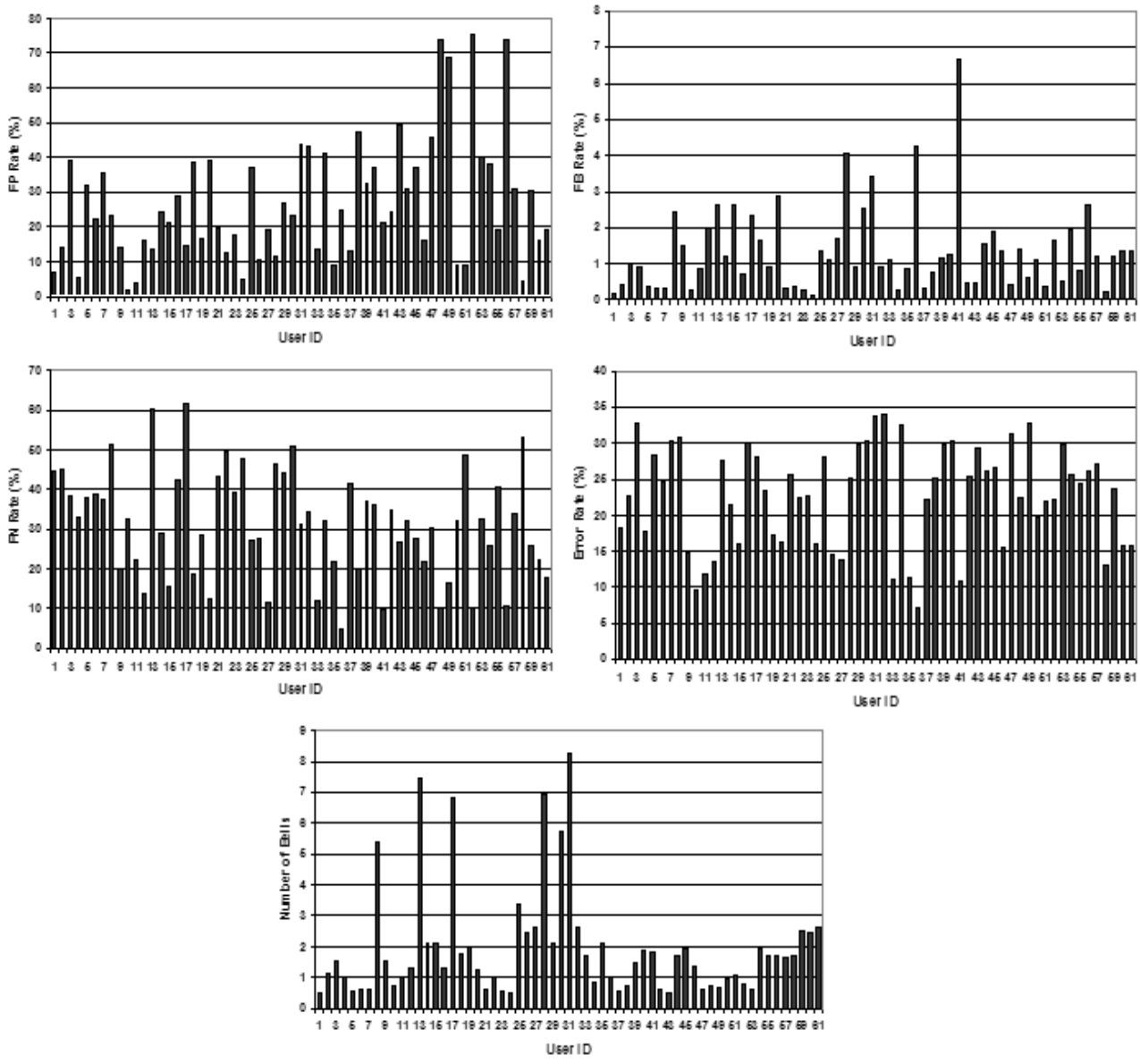


Fig. 4.5. Upper left figure shows the false positive rates; upper right shows the false bell rates; center left shows the false negative rates; center right figure shows the overall error rates, and bottom figure shows the number of false bells from the keystroke data for all 61 users in the Pairwise Detection experiment.

4.3.1 Experiment I: Pairwise Discrimination

Our first experiment was designed to give us an initial insight into the strength of each data source (the keystroke, mouse, GUI and combined data source) as a user

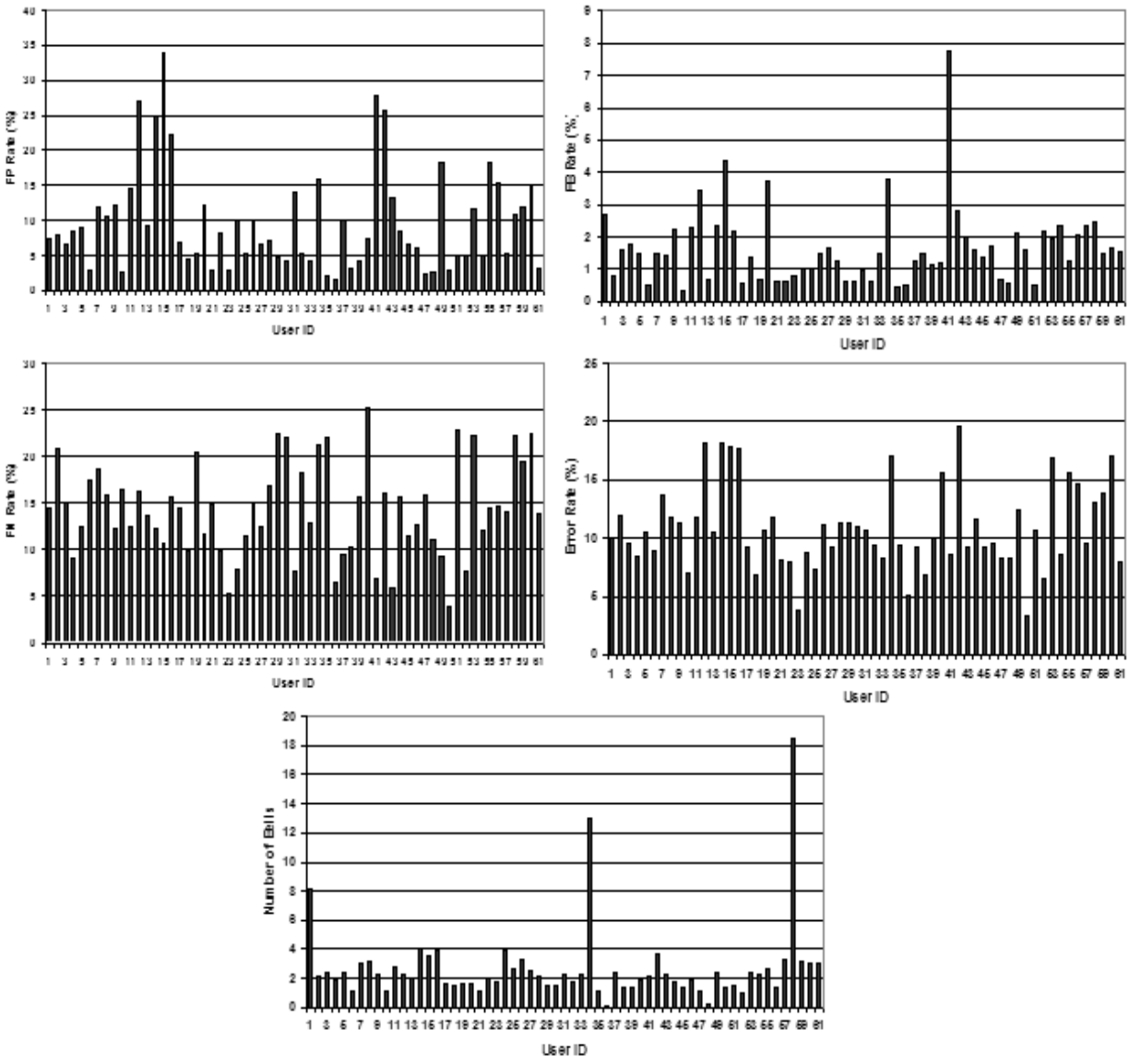


Fig. 4.6. Upper left figure shows the false positive rates; upper right shows the false bell rates; center left shows the false negative rates; center right figure shows the overall error rates, and bottom figure shows the number of false bells from the mouse data for all 61 users in the Pairwise Detection experiment.

re–authentication tool. We wanted to answer the question of whether there was a signal of "normalcy" per user and if so, to what extent. To determine if we could

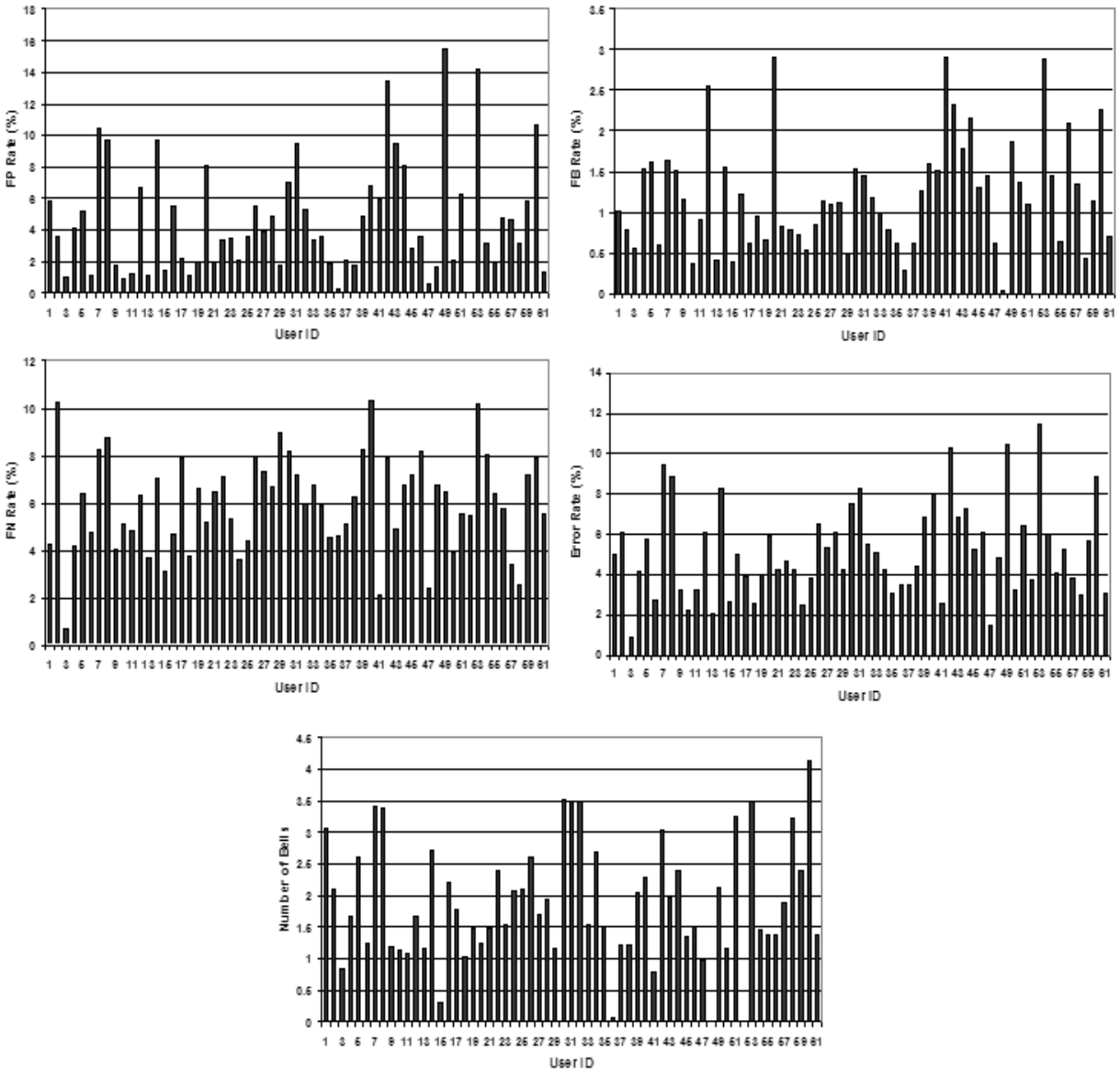


Fig. 4.7. Upper left figure shows the false positive rates; upper right shows the false bell rates; center left shows the false negative rates; center right figure shows the overall error rates, and bottom figure shows the number of false bells from the GUI data for all 61 users in the Pairwise Detection experiment.

distinguish users from one another we built a binary classifier for each pair of users. The results for each user are shown in Figures 4.5 to 4.8 for the keystroke, mouse,

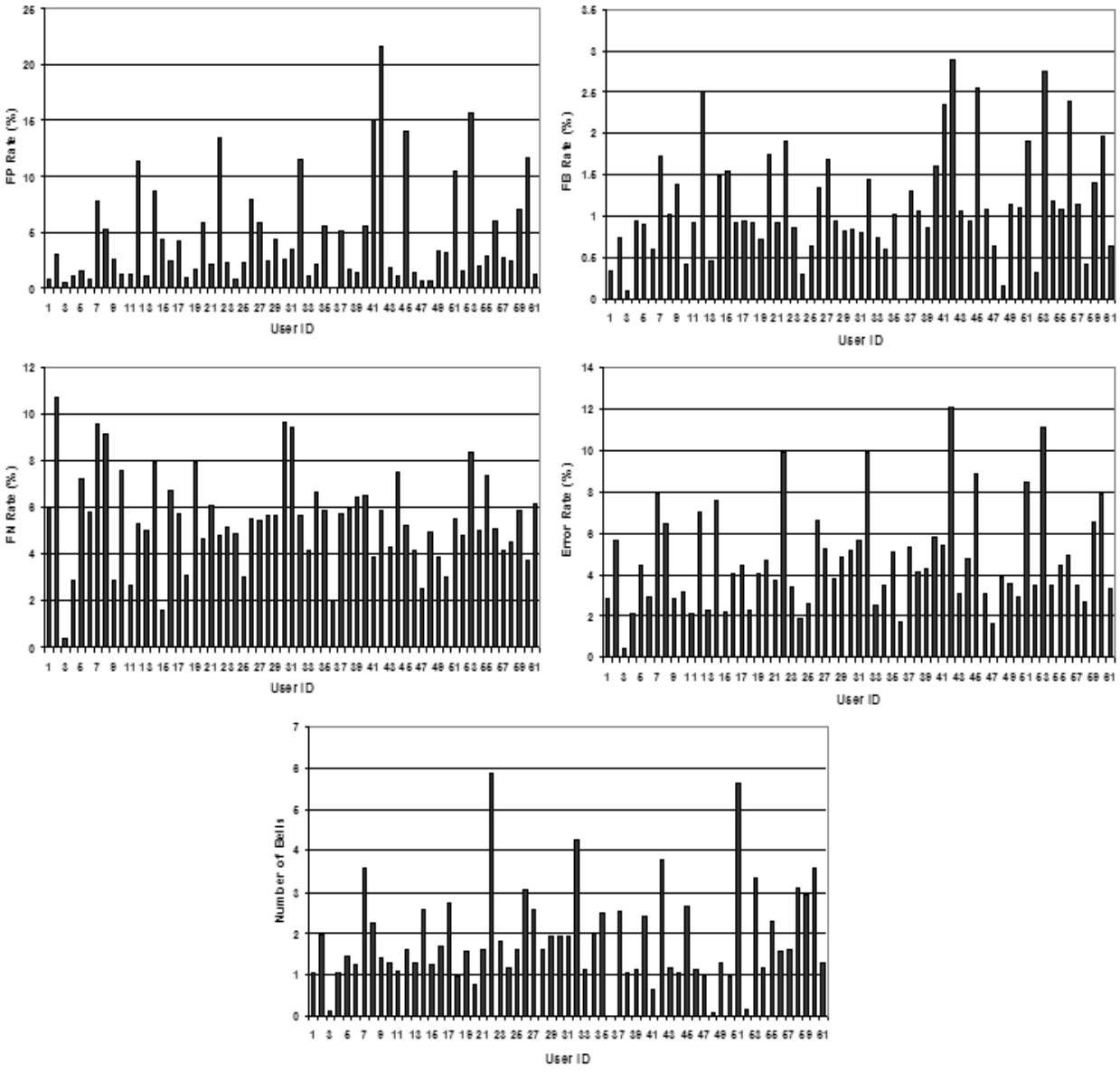


Fig. 4.8. Upper left figure shows the false positive rates; upper right shows the false bell rates; center left shows the false negative rates; center right figure shows the overall error rates, and bottom figure shows the number of false bells from the combined data for all 61 users in the Pairwise Detection experiment.

GUI and combined data sources, respectively. Table 4.3 summarizes the results obtained for three implementation schemes for each data source over all 61 users.

Table 4.3
The average and the standard deviation values of FP, FB, FN and Error Rates and the Bell Count over all 61 users for the basic and smoothing implementation schemes in the Pairwise Detection experiment. Tables (a), (b), (c) and (d) show results from the keystroke, mouse, GUI and combined data, respectively.

Scheme	**FP Rate**	**FB Rate**	**FN Rate**	**Error Rate**	**Bell Count**
Basic	31.85±18.93%	2.51±1.95%	33.49±18.19%	25.03±6.10%	3.73±3.0
Smoothing	26.78±17.47%	1.33±1.17%	31.23±13.52%	22.74±7.07%	1.93±1.77

(a)

Scheme	**FP Rate**	**FB Rate**	**FN Rate**	**Error Rate**	**Bell Count**
Basic	13.0±8.60%	3.29±1.64%	14.62±4.25%	12.58±4.12%	5.82±3.60
Smoothing	9.66±7.18%	1.65±1.19%	14.28±5.0%	10.87±3.63%	2.68±2.73

(b)

Scheme	**FP Rate**	**FB Rate**	**FN Rate**	**Error Rate**	**Bell Count**
Basic	5.60±4.23%	1.63±1.04%	7.26±2.08%	6.20±2.62%	2.83±2.10
Smoothing	4.57±3.57%	1.18±0.68%	5.99±2.04%	5.14±2.30%	1.88±0.95

(c)

Scheme	**FP Rate**	**FB Rate**	**FN Rate**	**Error Rate**	**Bell Count**
Basic	5.12±5.05%	1.32±0.93%	6.76±2.03%	5.48±2.63%	2.30±1.95
Smoothing	4.52±4.58%	1.13±0.65%	5.45±2.05%	4.64±2.43%	1.86±1.18

(d)

The second and third row of each sub–table in Table 4.3 show the results produced by the basic and smoothing implementation schemes, respectively.

The lowest error rates were generated by the combined classifier demonstrating that for these users a combination of different sources carried a stronger signal of normalcy than each data source individually. The average false positive, false bell,

false negative and error rates of the combined classifier over all 61 users were 4.52%, 1.13%, 5.45% and 4.64%, respectively. The average bell count was 1.86 bells. The ranking by accuracy for the individual data sources was 1) GUI, 2) mouse and 3) keystroke. This was not surprising considering that the average number of raw data instances in the 61–user dataset was 67,700.25, 11,510.54 and 1674.06 for the GUI, mouse and keystroke data, respectively. The amount of the GUI data was significantly higher than that of the other two sources thus generating more accurate results. The amount of keystroke data was so small that feature vectors computed for the keystroke classifier over a window of $W = 500$ data points were zero–vectors for most users.

Closer examination of different implementation schemes (see Table 4.3) revealed that the smoothing scheme outperformed the basic implementation scheme. This was not surprising considering that the smoothing filter function averaged out minor, transient changes in user behavior thereby effectively reducing the number of false alarms. The results obtained supported our initial conjecture that there was a signal of "normalcy" per user.

4.3.2 Experiment II: Anomaly Detection

In this experiment our goal was to determine if we could detect an insider impersonating another insider. Specifically, we studied if a *binary* classifier could build an accurate model of normal user behavior after seeing the behavior of a valid user A and the behavior of the remaining $N-1$ users in the role of intruders. The obtained results are shown in Figures 4.9 to 4.12 for the keystroke, mouse, GUI and combined data sources, respectively. Figure 4.13 shows the results from Figure 4.12 in a rate–descending order. The bar graphs of Figure 4.13 show the false positive, false bell, false negative and error rates, respectively, in the descending order for each of the 61 users. The corresponding pie charts of Figure 4.13 show the percentage distribution of users across each respective error–rate range. Table 4.4 summarizes the results

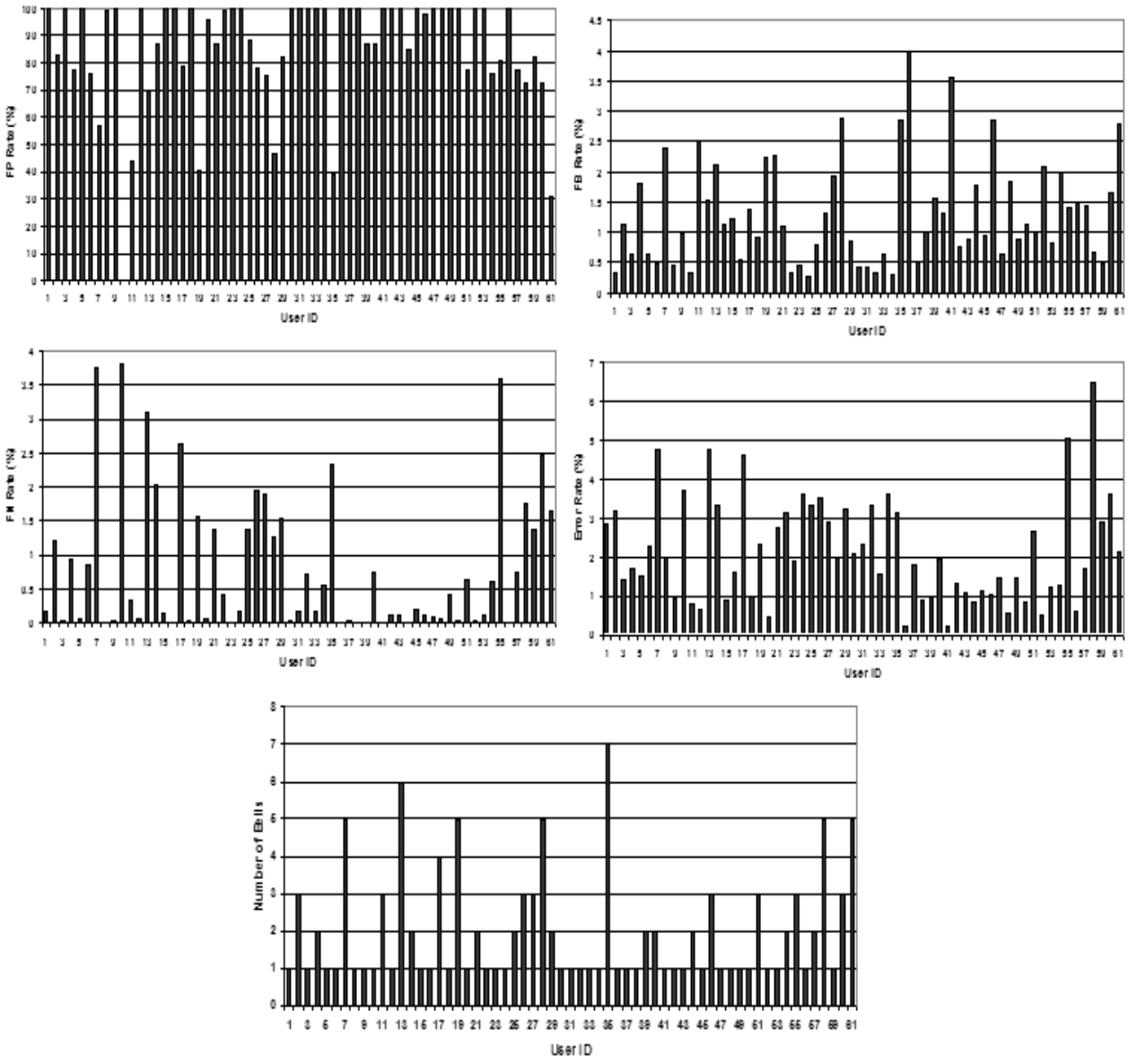


Fig. 4.9. The top left figure shows the false positive rates; the top right shows the false bell rates; center left shows the false negative rates; center right shows the error rates; and the bottom figure shows the bell count from the keystroke data for each of the 61 users in the Anomaly Detection experiment.

obtained for three implementation schemes for each data source over all 61 users. The second and third row of each sub–table in Table 4.4 show the results produced

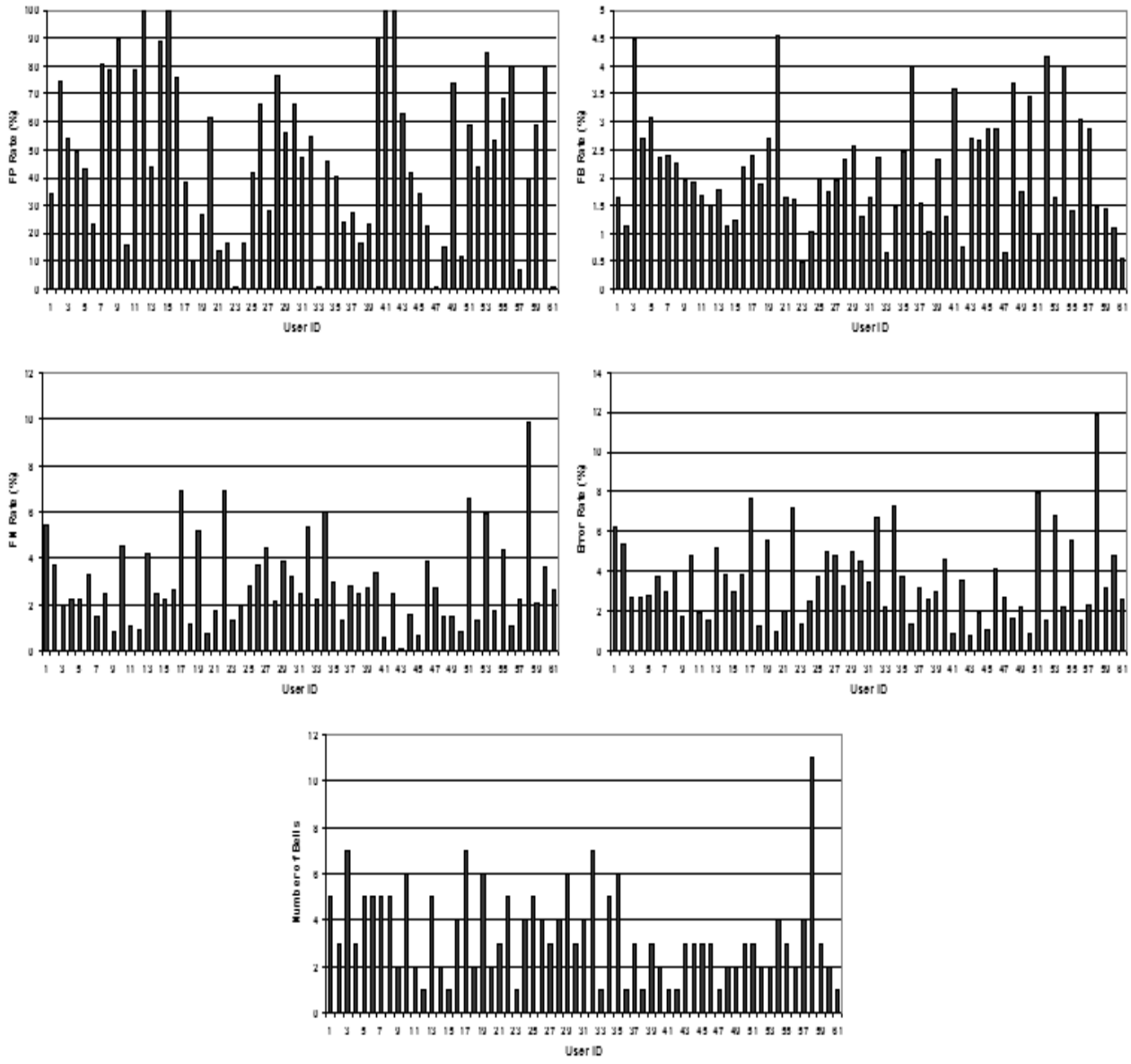


Fig. 4.10. The top left figure shows the false positive rates; the top right shows the false bell rates; center left shows the false negative rates; center right shows the error rates; and the bottom figure shows the bell count from the mouse data for each of the 61 users in the Anomaly Detection experiment.

by the basic and smoothing implementation schemes, respectively. We examined the performance of each data source in more detail. Our goal was to understand the

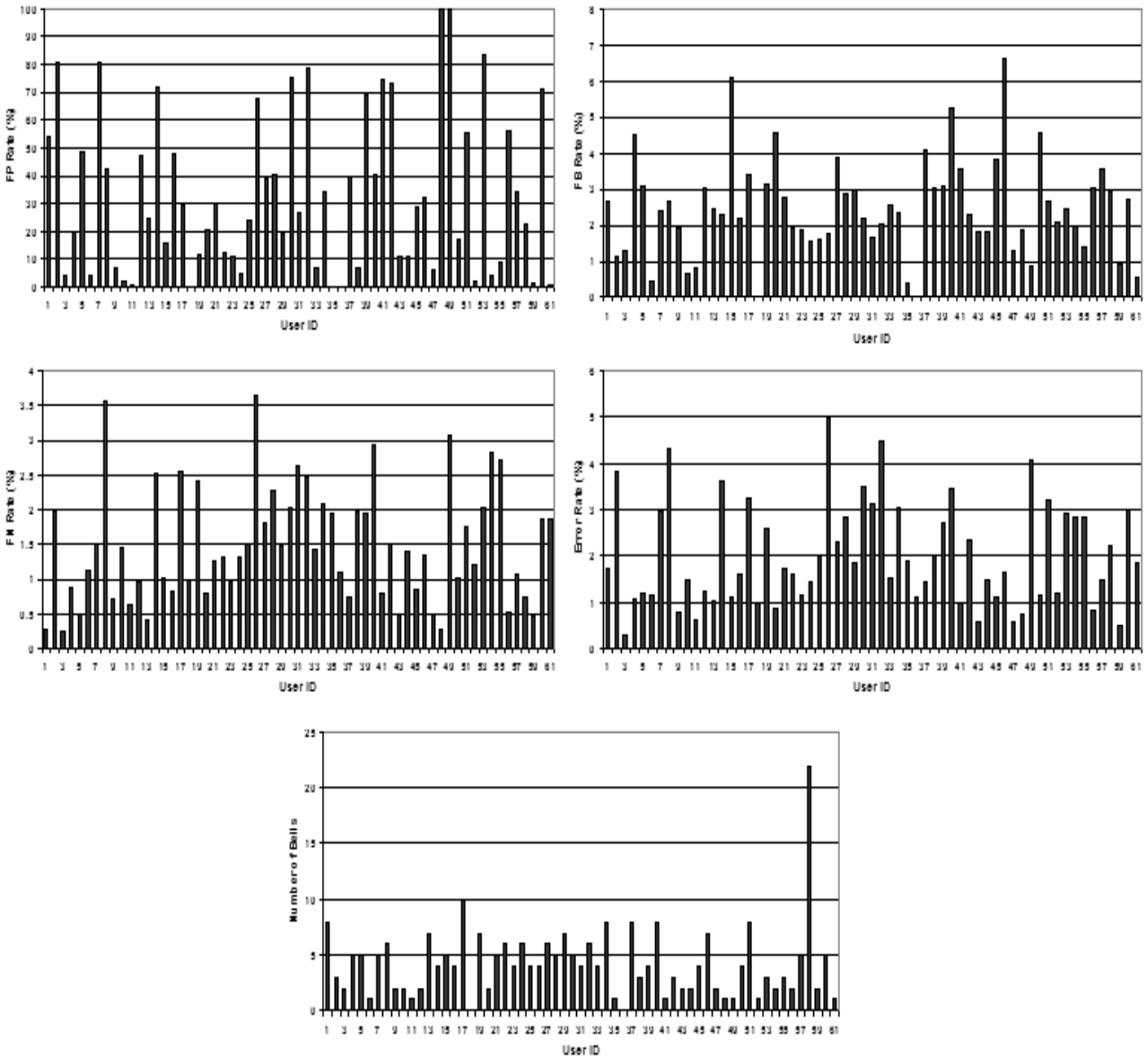


Fig. 4.11. The top left figure shows the false positive rates; the top right shows the false bell rates; center left shows the false negative rates; center right shows the error rates; and the bottom figure shows the bell count from the GUI data for each of the 61 users in the Anomaly Detection experiment.

reasons behind high false positive, false bell and false negative rates for some users

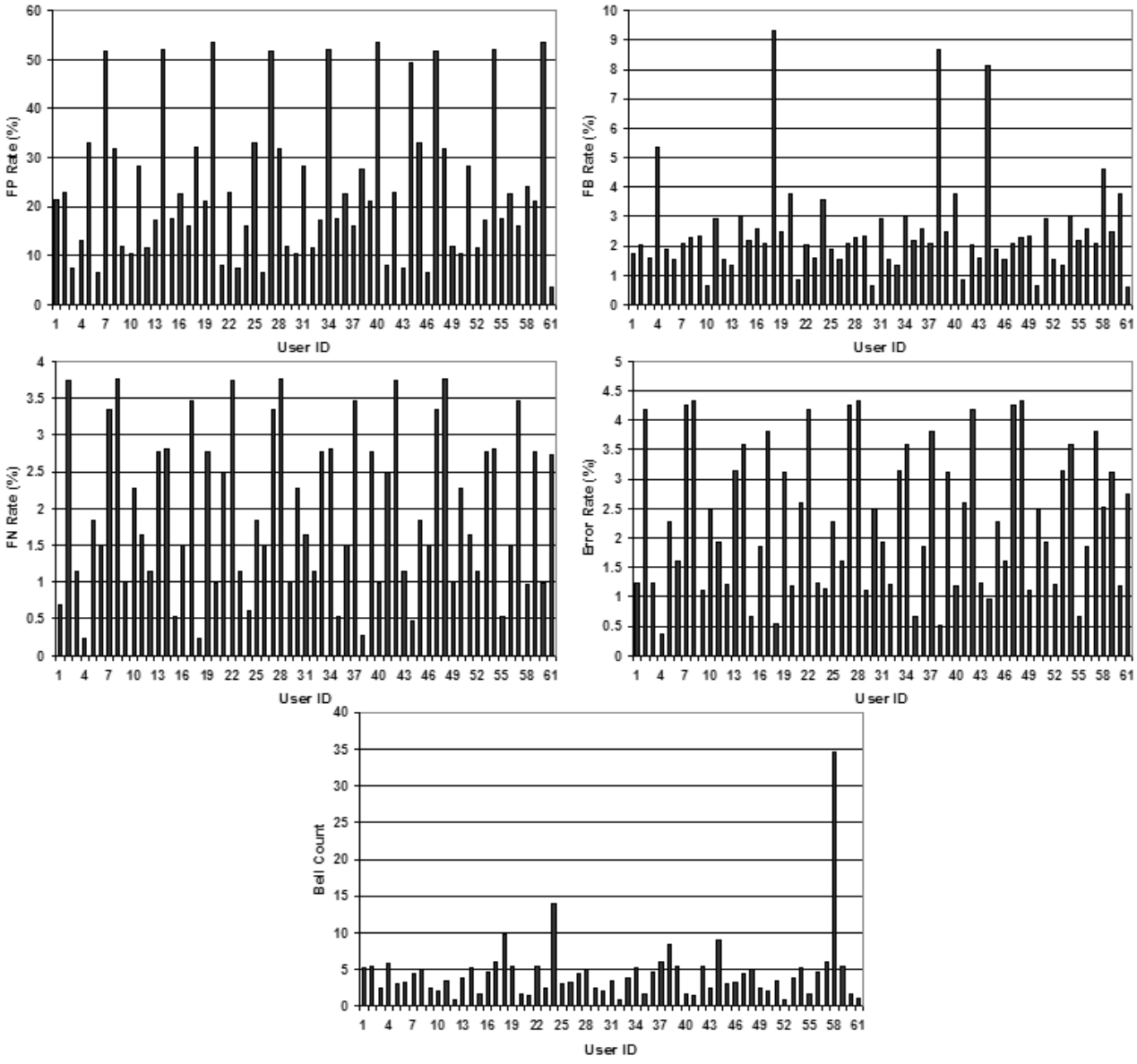


Fig. 4.12. The top left figure shows the false positive rates; the top right shows the false bell rates; center left shows the false negative rates; center right shows the error rates; and the bottom figure shows the bell count from the combined data for each of the 61 users in the Anomaly Detection experiment.

and to investigate the characteristics of decision trees generated for each data source to determine which features were best in discriminating among different users.

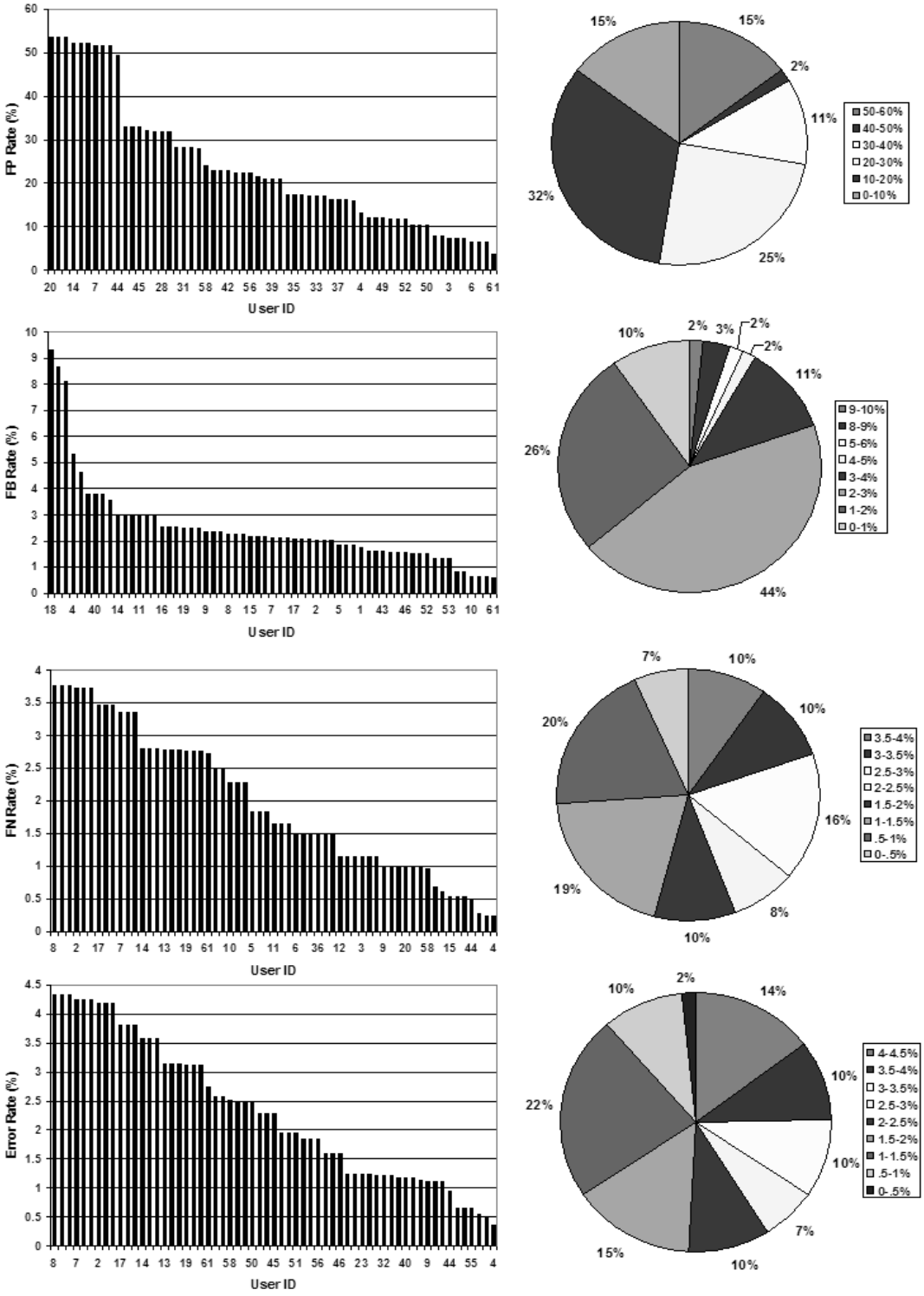


Fig. 4.13. The bar graphs show the error rates in the descending order for each of the 61 users. The pie charts show the percentage distribution of users across each error–rate range. From the top, the bar graphs and the corresponding pie charts representing FP, FB, FN and Error rates obtained from the combined data in the Anomaly Detection experiment are shown.

Table 4.4
The average and the standard deviation values of FP, FB, FN and Error Rates and the Bell Count over all 61 users for the basic and smoothing implementation schemes in the Anomaly Detection experiment. Tables (a), (b), (c) and (d) show results from the keystroke, mouse, GUI and combined data, respectively.

Scheme	**FP Rate**	**FB Rate**	**FN Rate**	**Error Rate**	**Bell Count**
Basic	94.57±9.90%	1.94±1.84%	0.27±0.27%	1.78±1.02%	3.62±5.12
Smoothing	85.77±20.98%	1.30±0.87%	0.81±1.04%	2.14±1.35%	1.98±1.49

(a)

Scheme	**FP Rate**	**FB Rate**	**FN Rate**	**Error Rate**	**Bell Count**
Basic	75.08±18.84%	5.99±3.01%	0.84±0.47%	2.04±1.17%	10.82±8.09
Smoothing	48.41±29.24%	2.10±0.99%	2.91±1.88%	3.59±2.15%	3.41±1.97

(b)

Scheme	**FP Rate**	**FB Rate**	**FN Rate**	**Error Rate**	**Bell Count**
Basic	53.42±27.18%	6.65±2.82%	0.59±0.33%	1.41±0.81%	11.59±7.53
Smoothing	33.36±28.88%	2.45±1.36%	1.49±0.84%	2.0±1.12%	4.26±3.30

(c)

Scheme	**FP Rate**	**FB Rate**	**FN Rate**	**Error Rate**	**Bell Count**
Basic	45.70±17.30%	6.28±1.70%	0.54±0.27%	1.21±0.60%	10.85±5.76
Smoothing	23.37±15.97%	1.76±0.94%	1.50±1.40%	1.77±1.01%	2.66±1.82

(d)

Keystroke Data Source

The keystroke data source produced the average false positive, false bell, false negative and error rates of 85.77%, 1.30%, 0.81% and 2.14%, respectively and a bell count of 1.98 bells. It had the highest false positive rate among all four data sources

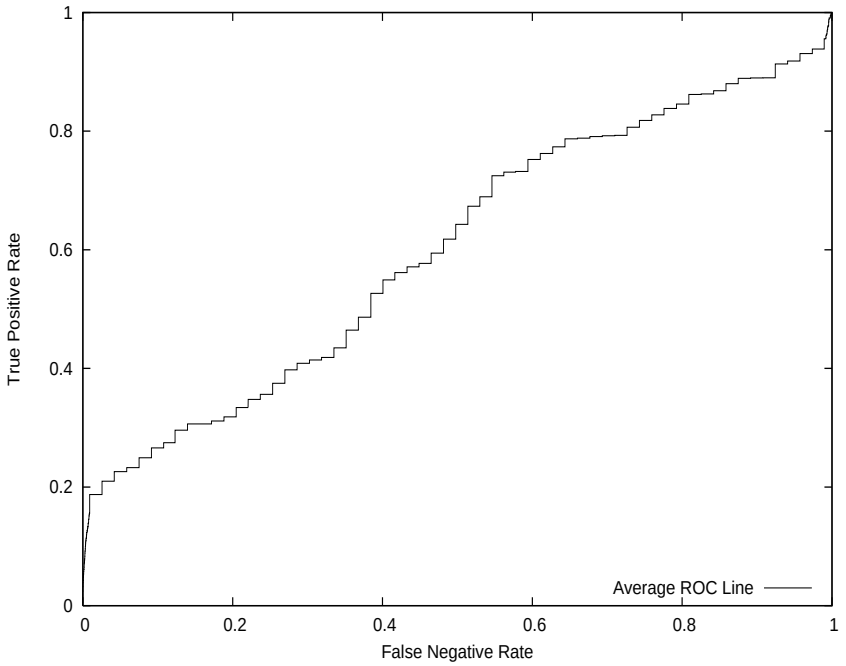


Fig. 4.14. ROC graph for the keystroke data source in the Anomaly Detection experiment.

(e.g., 31 users had FP=100%). The ROC curve for the *smoothed* keystroke classifier is shown in Figure 4.14. The area under the ROC curve was 0.60 which indicated that the performance of the keystroke classifier although poor was still better than *random guessing* (e.g., random guessing has equal chances of correctly detecting an intruder when intrusion occurred or correctly identifying a current user - it produces the area under the ROC curve of 0.5).

As discussed in the pairwise experiment the amount of keystroke data was so small in the 61-user dataset that most users had zero-vectors in their datasets. Because the amount of data per user was limited to that extent, the classifier was unable to build an accurate model of user behavior. Consequently, even small variations in human activities caused it to misclassify a new instance as belonging to an intruder and raise an alarm. The final outcome was a high false positive and a low false negative rate. The false bell rate of the keystroke classifier was low which suggested that most alarms occurred in a sequence as one *long* alarm.

Table 4.5
The ten most significant keystroke features over all 61 users in the Anomaly Detection experiment.

#	**Feature Description**
1.	Mean of the 1–graph of **letters**
2.	Standard deviation of the 8–graph of **mouse–keys**
3.	Mean of the 7–graph of **numbers**
4.	Mean of the 8–graph of **all keystrokes**
5.	The third moment of the 1–graph of **other keys**
6.	Number of the **control keys**
7.	Standard deviation of the 2–graph of **other keys**
8.	Standard deviation of the 6–graph of **other keys**
9.	Skewness of the 1–graph of **mouse–keys**
10.	Mean of the 3–graph of **all keystrokes**

Analysis of error rates: Considering the discussion in the above paragraph we found it interesting to discover why user 10 had a false positive rate of *only* 0.32% for the keystroke data. The user with the next lowest false positive rate of 30.73% was user 61. Closer examination of the user 10's dataset revealed that this user collected twice the average number of raw data instances (e.g., 3120) over a period of 25.1 hours. The most distinguishing features for this user was the percentage of all keystroke points (the top level of the keystroke hierarchy in Figure 3.7) and the number of "other" keystrokes which led us to conclude that this user utilized the keyboard for quick multimedia launch (i.e., opening a web browser, e-mail account, etc. directly from the keyboard). This behavior in addition to the assigned data collection task described in Section 4.2.2 made user 10 distinguishable from the other sixty users.

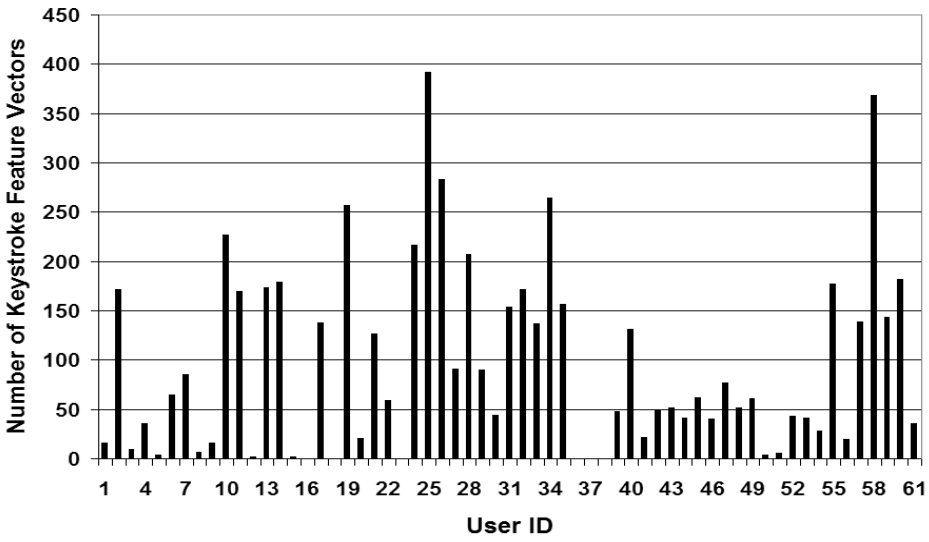


Fig. 4.15. Number of keystroke feature vector instances per user. *Only* seventeen users have at least 150 instances in their data sets.

Analysis of features and decision trees: We examined what characteristics and what attributes discriminated users the best. Accordingly, we analyzed the size of the decision trees produced by the keystroke classifier and the features used to describe users. The average number of unique features per tree was 57 and the average size of the trees was 257 nodes (excluding leaves), which suggested that some features appeared more than once in the same tree, but with a different range test $F_i > a$, where F_i was a feature and a was a value in its obscured range. This was an important result from an efficiency standpoint because the number of features computed for each user was directly proportional to the time it took to classify a current user. To obtain an understanding of which attributes best captured a user's behavior, we examined the ten most significant keystroke features over all 61 users (see Table 4.5). We obtained these features by observing a root node of each decision tree and selecting those root nodes that appeared in the largest number of trees. The decision trees of 35 users had as a root node one of these ten most significant features.

Table 4.6
The average and the standard deviation values of FP, FB, FN and Error Rates and the Bell Count over all 17 users for the basic and smoothing implementation schemes in the Anomaly Detection experiment for the reduced keystroke data source.

Scheme	**FP Rate**	**FB Rate**	**FN Rate**	**Error Rate**	**Bell Count**
Basic	65.45±26.73%	7.70±2.82%	2.47±1.99%	6.08±2.70%	1.70±0.77
Smoothing	63.82±28.30%	7.70±2.82%	2.36±2.03%	5.90±2.82%	1.70±0.77

In Chapter 3 we postulated that features computed from the *leaves* of the keystroke feature hierarchy in Figure 3.7 were uncorrelated and most discriminatory. This experiment showed that eight of the ten most significant features were computed from the *leaves* of the keystroke hierarchy: mouse–keys, control and other keys, and letters and numbers. The other features were computed from the regular keys. Closer inspection of these ten features suggested that mean and standard deviation were the most reliable statistical measurements, and that the 8–graph duration was the best "raw" measurement for modeling user behavior.

Results on the reduced keystroke dataset: To test the strength of the keystroke data source in more detail, we repeated the Anomaly Detection experiment on a reduced dataset. We investigated the number of keystroke feature vectors produced for each user in the 61–user dataset (see Figure 4.15) and eliminated those users who had fewer than 150 vector instances. This left us with *only* 17 users in the reduced dataset. New results are shown in Figure 4.16. Table 4.6 summarizes the error rates over all 17 users for the basic and smoothing implementation schemes. The ROC curve is shown in Figure 4.17. The area under the ROC curve was 0.63 which was better than the results obtained from the keystroke data on the 61–user dataset.

The reduced keystroke dataset produced the average false positive, false bell, false negative and error rates of 63.82%, 7.70%, 2.36% and 5.90%, respectively, and

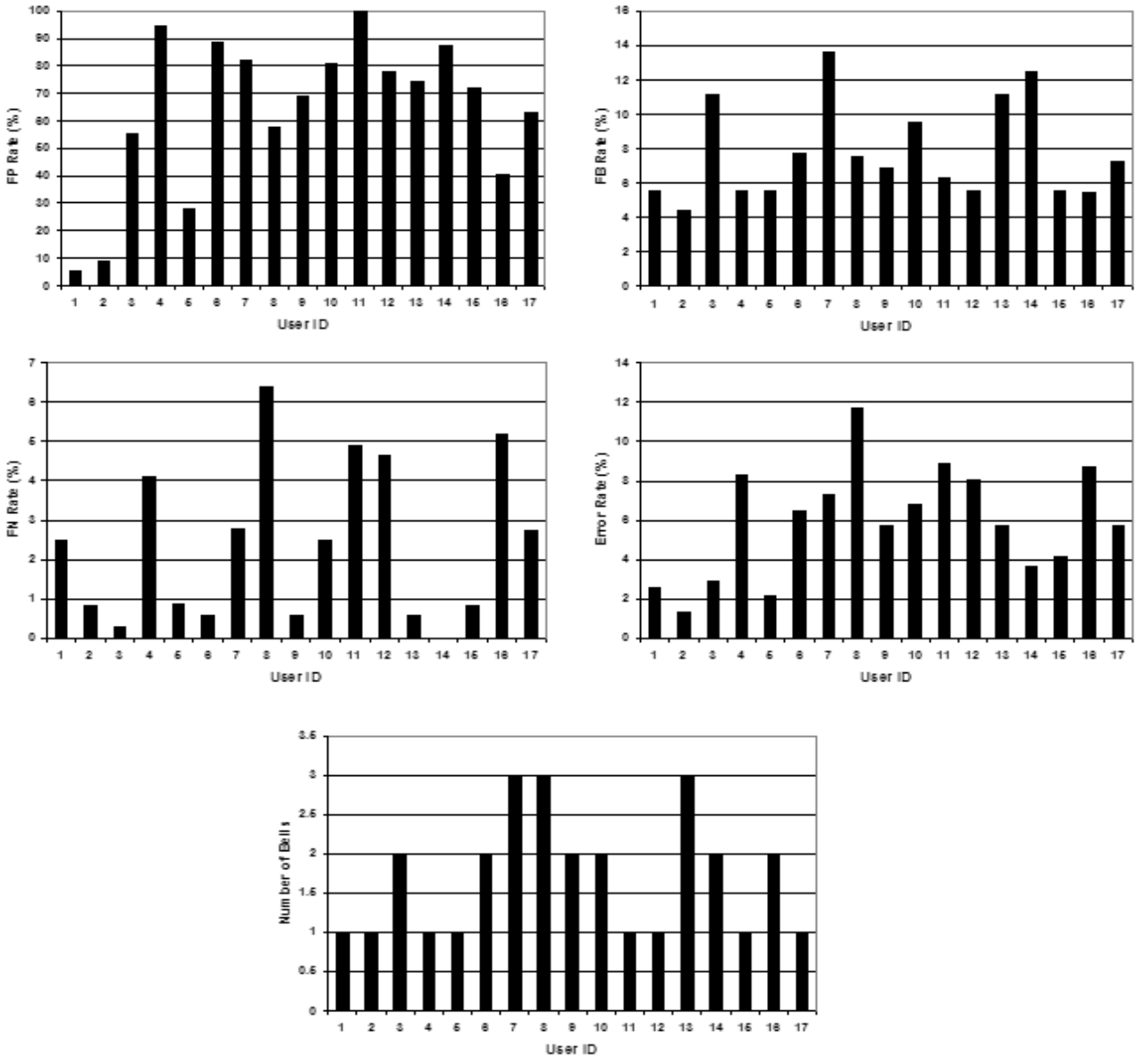


Fig. 4.16. The top left figure shows the false positive rates; the top right shows the false bell rates; center left shows the false negative rates; center right shows the error rates; and the bottom figure shows the bell count on the reduced keystroke data for each of the 17 users in the Anomaly Detection experiment.

a bell count of 1.70 bells. The false positive rate went from 85.77% for the 61–user dataset down to 63.82% for the 17–user dataset. At the same time false bell, false

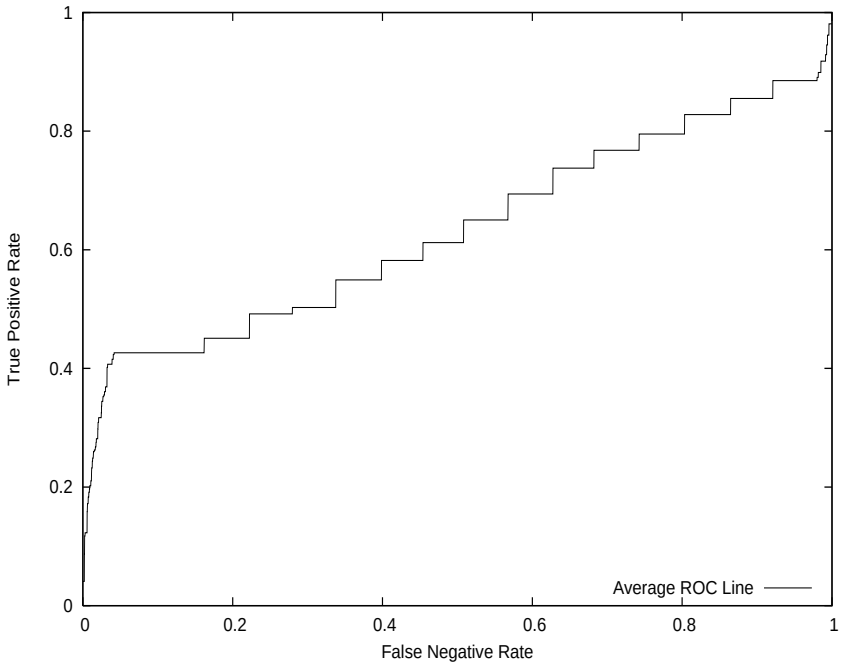


Fig. 4.17. ROC graph for the reduced keystroke data source in the Anomaly Detection experiment.

negative and error rates increased and the bell count remained the same. There was a trade-off between the false positive and false negative rates and it depended on: 1) the number of users in the dataset (e.g., as the number of users increased the false positive rate increased and the false negative rate decreased), and 2) the amount of data per user (e.g., although each of the 17 users had at least 150 feature vector instances in their datasets, *additional* data was needed per user to further lower the false positive and false negative rates). Only four users had below 50% false positive rate and two users had above 5% false negative rate.

The average tree size was 79 internal nodes (excluding leaves) for the decision trees of the reduced keystroke dataset. The average number of unique features per tree was 28.

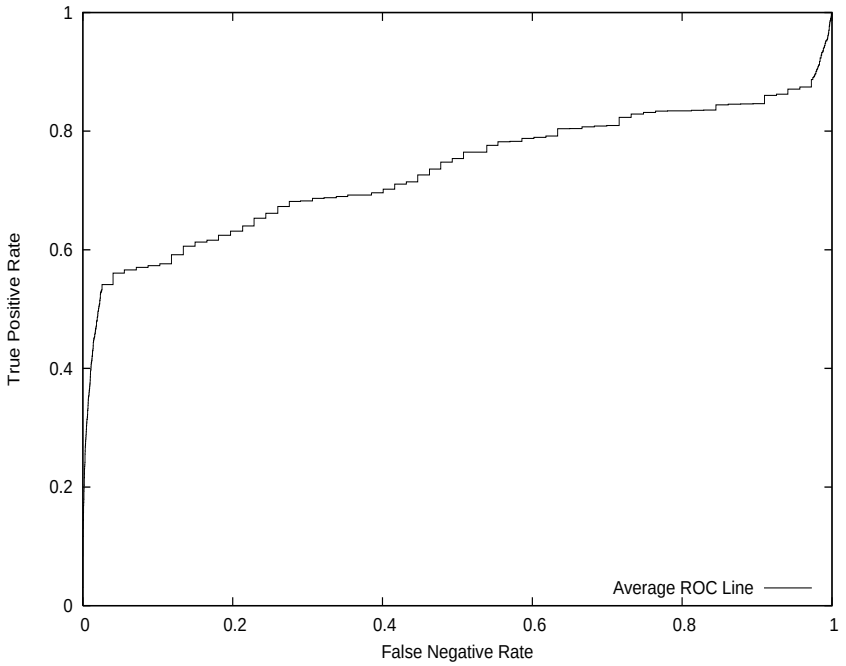


Fig. 4.18. ROC graph for the mouse data source in the Anomaly Detection experiment.

Mouse Data Source

On average, the mouse classifiers produced false positive, false bell, false negative and error rates of 48.41%, 2.10%, 2.91% and 3.59%, respectively, and a bell count of 3.41 bells. The ROC curve for the mouse data is shown in Figure 4.18. The area under the ROC curve was 0.73. The performance of the mouse data source was significantly better than that of the keystroke classifier, which was not surprising for two reasons 1) there was more mouse than keystroke data in the 61-user dataset and 2) each mouse data point had both spatial and temporal information about a user's behavior whereas keystroke data had only temporal information. Both the GUI and the combined classifier outperformed the mouse classifier.

Analysis of high FB and FN rates: We examined individual users' raw data files, feature vectors and decision trees generated by the mouse classifier to determine why some users produced high error rates. Our observations are summarized in Tables

Table 4.7
Users with false bell rate above 4.0% for the mouse data source in the Anomaly Detection experiment.

ID	FB Rate	Elapsed Time	SD(Moves)	SD(Events)
3	4.49%	**1.63** hrs	34.33	29.74
20	4.54%	**0.23** hrs	51.03	**37.87**
52	4.17%	**0.97** hrs	**235.95**	34.26
AVG	2.1%	4.1 hrs	55.25	26.04

4.7 and 4.8. Table 4.7 shows users with a false bell rate above 4.0%. The first column of the table shows the User ID, the second column reports the false bell rate for the specific user; the third column reports the time it took user to complete (or not complete) the data collection process; and the fourth and fifth column show the standard deviation measurements of the number of mouse movements and the number of *all* mouse events (e.g., wheel, NC moves, clicks, etc.), respectively. The last row of the table shows the average values across all 61 users. Table entries printed in bold highlight those values that are at least one standard deviation above or below the average.

Examination of Table 4.7 revealed that all three users spent less than two hours collecting the data. Users 20 and 52 spent *only* 13.8 and 58.2 minutes on the data collection. Moreover, users 20 and 52 exhibited a high variance in their behavior, particularly with regard to the mouse movements and mouse events, respectively. Short data collection time and inconsistent behavior produced high false bell rates for these users.

Table 4.8 shows users with a false negative rate above 6.0%. The first column of the table shows the User ID, the second column has the user's false negative rate, the third column reports the time it took user to complete the data collection process; and the fourth and fifth columns show the average number of all mouse events and

Table 4.8
Users with false negative rate above 6.0% for the mouse data source in the Anomaly Detection experiment.

ID	FN Rate	Elapsed Time	#Events	#Wheel
17	6.9%	**2.63** hrs	16.78	**0.0**
22	6.94%	**2.48** hrs	20.31	**0.17**
34	6.03%	**0.83** hrs	14.77	2.17
51	6.6%	**0.73** hrs	**6.03**	2.75
58	9.84%	13.94 hrs	**7.61**	1.28
AVG	2.91%	4.1 hrs	20.45	1.89

mouse wheel movements, respectively. Examination of Table 4.8 revealed users 34 and 51 spent less than one-quarter of the average time collecting the data points. Users 17 and 22 induced virtually no mouse wheel movements and users 51 and 58 produced approximately one-third of the average number of all mouse events. An interesting case was user 58 who spent 13.94 hours collecting the data and induced the highest false negative rate of all 61 users. We previously concluded that a short data collection time and a limited amount of data per user might lead to poor classifier performance. The performance of user 58 suggested that in addition to a longer collection time and larger datasets, the mouse classifier needed mouse event data (e.g., single and double clicks, wheel movements, etc.) to build a more accurate model of normal user behavior. The only event data user 58 produced within one standard deviation of the average was mouse wheel data. Therefore, passive user behavior and the absence of event data might induce high error rates too.

Analysis of features and decision trees: Similarly to the keystroke data we analyzed the size of the decision trees produced by the mouse classifier and the features used to describe users. The average number of unique features per tree was 111, while the average size of the trees was 625 nodes (excluding leaves). The

Table 4.9
The ten most significant mouse features over all 61 users in the Anomaly Detection experiment.

#	**Feature Description**
1.	Number of **all mouse events**
2.	Mean of Y coordinates of **mouse movements**
3.	Skewness of Y coordinates of **mouse movements**
4.	Skewness of Y coordinates of **NC movements**
5.	Standard deviation of Y coordinates of **NC movements**
6.	Mean of the 6–graph of **mouse movements**
7.	Mean of the 4–graph of **all clicks**
8.	Standard deviation of the 6–graph of **wheel movements**
9.	Skewness of the 7–graph of **wheel movements**
10.	Mean of the speed of **double clicks**

average size of the trees was interesting because it was larger than that constructed from each of the other three data sources (including the combined classifier). Usually large training datasets create large decision trees [119], but this was not true for the mouse data source. One possible explanation for the size of the mouse decision trees was a higher variance of user mouse behavior. The space of possible outcomes of the raw mouse data (e.g., user's X and Y screen coordinates) was higher than that of keystrokes and temporal GUI events.

Eight of the ten most significant mouse features were computed from the *leaves* of the mouse hierarchy: NC and mouse moves, wheel movements and double clicks thereby supporting our claim that the most discriminatory features also the most uncorrelated features (see Table 4.9). The other two features were computed from the clicks and mouse events categories. The decision trees for 39 users had as a root node one of these ten features. The mean and skewness were the most reliable

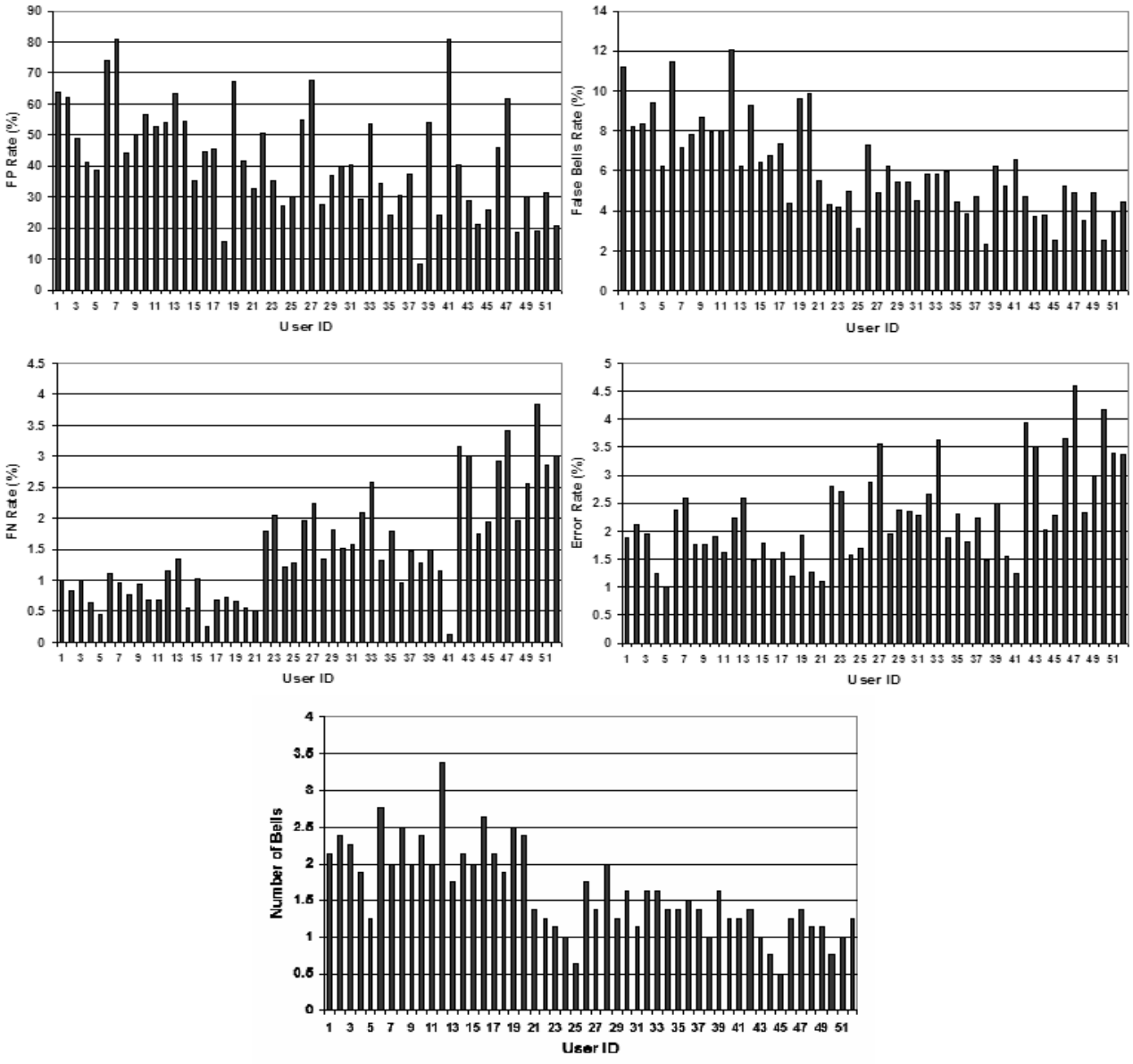


Fig. 4.19. The top left figure shows the false positive rates; the top right shows the false bell rates; center left shows the false negative rates; center right shows the error rates; and the bottom figure shows the bell count on the reduced mouse data for each of the 52 users in the Anomaly Detection experiment.

statistical measurements, and Y coordinates of the screen cursor were the best "raw" measurements for modeling mouse behavior.

Table 4.10
The average and the standard deviation values of FP, FB, FN and Error Rates and the Bell Count over all 52 users for the basic and smoothing implementation schemes in the Anomaly Detection experiment for the reduced mouse data source.

Scheme	**FP Rate**	**FB Rate**	**FN Rate**	**Error Rate**	**Bell Count**
Basic	61.43±28.45%	8.96±4.97%	0.89±0.60%	2.03±0.94%	2.29±1.16
Smoothing	43.66±31.93%	5.26±3.06%	2.04±1.39%	2.84±1.48%	1.40±0.82

Results on the reduced mouse dataset: Similar to the keystrokes we repeated the Anomaly Detection experiment on a reduced mouse dataset. In Section 4.2.2 we eliminated nine users from the 61–user mouse dataset which left us with 52 users (see Figure 4.2). The results on the reduced mouse dataset are shown in Figure 4.19. Table 4.10 summarizes the error rates over all 52 users for the basic and smoothing implementation schemes. The ROC curve is shown in Figure 4.20. The area under the ROC curve was 0.78 which was better than the results obtained on the 61–user mouse dataset.

The reduced mouse dataset produced the average false positive, false bell, false negative and error rates of 43.66%, 5.26%, 2.04% and 2.84%, respectively, and a bell count of 1.40 bells. The false positive, false negative and error rates went from 48.41%, 2.91% and 3.59% for the 61–user dataset down to 43.66%, 2.04% and 2.84% for the 52–user dataset. The bell count decreased from 3.41 to 1.40 bells. Only the false bell rate increased from 2.10% to 5.26% which suggested that false alarms on the 52–user dataset were sporadic as opposed to consecutive. The average tree size was 95 internal nodes (excluding leaves) and the average number of unique features per tree was 33.

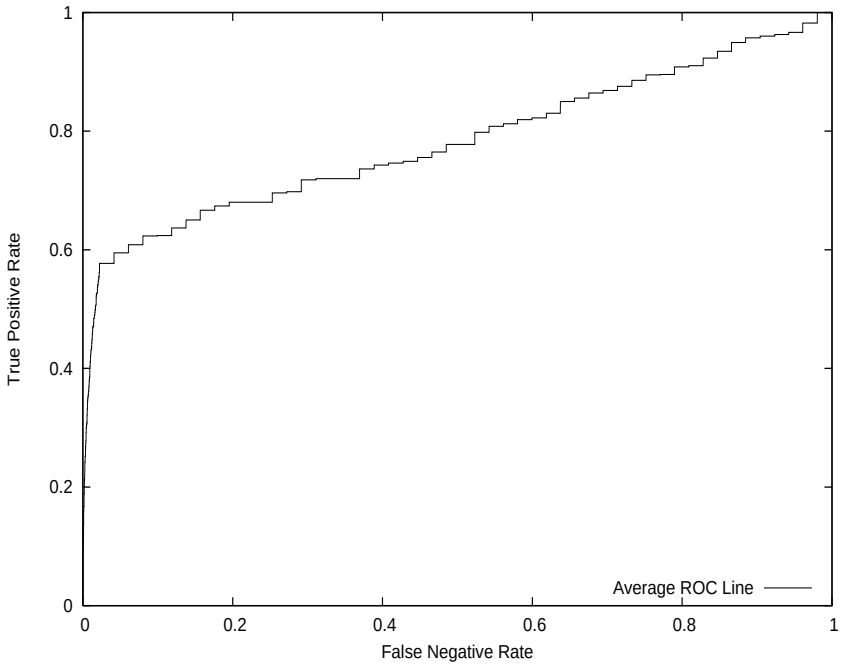


Fig. 4.20. ROC graph for the reduced mouse data source in the Anomaly Detection experiment.

GUI Data Source

Among the individual data sources, the GUI classifier performed the best. It produced the average false positive, false bell, false negative and error rates and a bell count of 33.36%, 2.45%, 1.49%, 2.0% and 4.26 bells, respectively. The ROC curve for the GUI data is shown in Figure 4.21. The area under the ROC curve was 0.80. 83.7% of the 61-user dataset was composed of the GUI data.

Analysis of high FB and FN rates: Similarly to the mouse classifier we examined the behavior of those individual users with high false bell or false negative rates. Table 4.11 shows users with false bell rate above 5.0%. The first column of the table shows User ID, second column has the false bell rate for the specific user; the third column has the time it took user to complete the data collection process; and the fourth, fifth and sixth columns show the average number of "item" and "miscellaneous" points and the standard deviation of "miscellaneous" points,

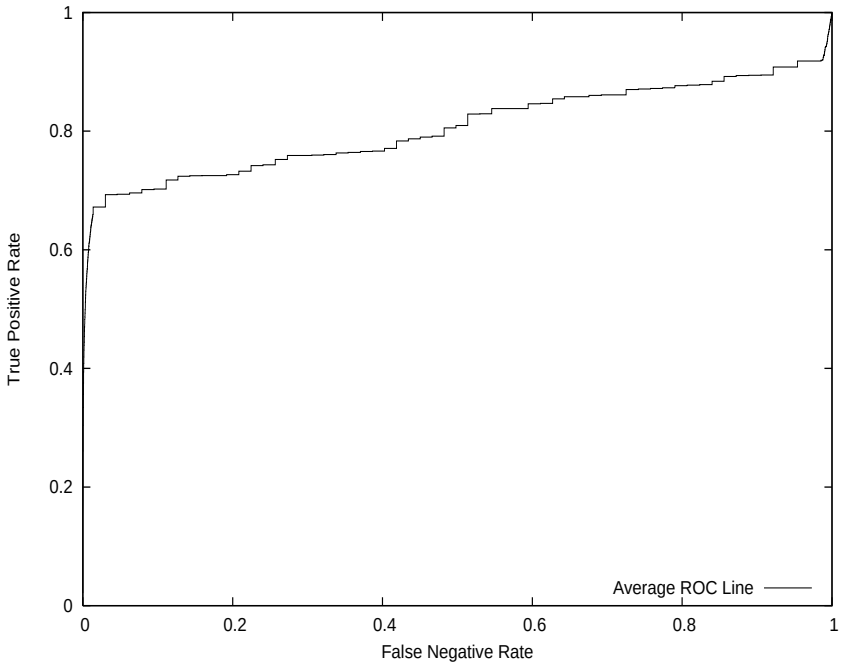


Fig. 4.21. ROC graph for the GUI data source in the Anomaly Detection experiment.

respectively. Users 15, 40 and 46 had the highest false bell rate on the GUI dataset. We mentioned previously that a short data collection time, a small amount of data per user and a high variance in user behavior resulted in the classifier incorrectly classifying a valid user as an intruder. These conclusions were valid for the GUI data also. All three users spent less than one-third of the average time collecting the data. A small number of "item" events was induced by users 40 and 46, and also "miscellaneous" events by user 40 (see GUI feature hierarchy in Figure 3.9). Finally, user 15 *only* sporadically induced "miscellaneous" events which resulted in a high variance in his/her behavior with respect to this type of GUI data.

Table 4.12 shows users with false negative rate above 3.0%. The first column of the table shows User ID, second column has the user's FN rate, third column has the elapsed time of the data collection; and the fourth, fifth, sixth and seventh columns show the average number of "dialog" events, and the standard deviation of *all* GUI, "temporal" and "dialog" events, respectively. Users 26 and 49 collected the data

Table 4.11
Users with false bell rate above 5.0% for the GUI data source in the Anomaly Detection experiment.

ID	FB	Time	#Item	#Miscellaneous	SD(Miscellaneous)
15	6.10%	**0.70** hrs	21.0	24.93	**30.72**
40	5.26%	**1.21** hrs	**5.43**	**1.17**	2.63
46	6.67%	**1.60** hrs	**8.52**	4.82	8.81
AVG	2.45%	4.1 hrs	22.20	6.18	9.84

Table 4.12
Users with false negative rate above 3.0% for the GUI data source in the Anomaly Detection experiment.

ID	FN	Time	#Dialog	SD(GUI)	SD(Temporal)	SD(Dialog)
8	3.57%	18.43 hrs	**6.90**	85.59	45.91	14.07
26	3.66%	**1.44** hrs	31.68	181.16	77.66	**75.05**
49	3.07%	**0.81** hrs	**0.05**	**308.12**	**156.58**	0.36
AVG	1.49%	4.1 hrs	18.46	161.93	80.19	30.58

for under an hour and a half thereby producing a high variance in their behavior in regard to *all* GUI and "temporal" events for user 49, and "dialog" events for user 26 (see GUI feature hierarchy in Figure 3.9). Moreover, user 49 induced virtually zero "dialog" events. User 8 was an interesting case because he/she spent 18.43 hours collecting the data. Similarly to user 58 who produced a high false negative rate on the mouse data, user 8 was an example of a passive user who only sporadically interacted with the I/O devices.

Analysis of features and decision trees: The GUI classifier used on average 103 unique features per tree and generated on average 442-node decision trees (excluding leaves). High tree-size to feature ratio was interesting from an efficiency standpoint.

Table 4.13
The ten most significant GUI features over all 61 users in the Anomaly Detection experiment.

#	**Feature Description**
1.	Number of **temporal events**
2.	Standard deviation of X coordinates of **window events**
3.	Skewness of X coordinates of **spatial events**
4.	Skewness of the 7–graph of **icon events**
5.	Mean of the 8–graph of **item events**
6.	Mean of the 3–graph of **query events**
7.	Standard deviation of the 7–graph of **dialog events**
8.	Mean of the 8–graph of **spatial events**
9.	Mean of the angle of orientation of **control events**
10.	Skewness of the 1–graph of **temporal events**

Six of the ten most significant features were computed from the *leaves* of the GUI hierarchy: window, icon, query, dialog, control and item events (see table 4.13). The other most significant features were computed for the temporal and spatial event categories. We assessed that the decision trees of 51 users had as a root node one of these ten most significant features. Both mean and skewness were the most reliable statistical measurements; and 7- and 8-graph durations and the X coordinates of the screen cursor were the best "raw" measurements for modeling user behavior.

Combined Data Sources

The combined classifier had the lowest error rates. When smoothed over $m \in [1, 11]$, the time to alarm (TTA) was 50 seconds ($m = 11$) and the average false positive, false bell, false negative and error rates and a bell count of the combined classifier were 23.37%, 1.76%, 1.50%, 1.77% and 2.66 bells, respectively. The ROC

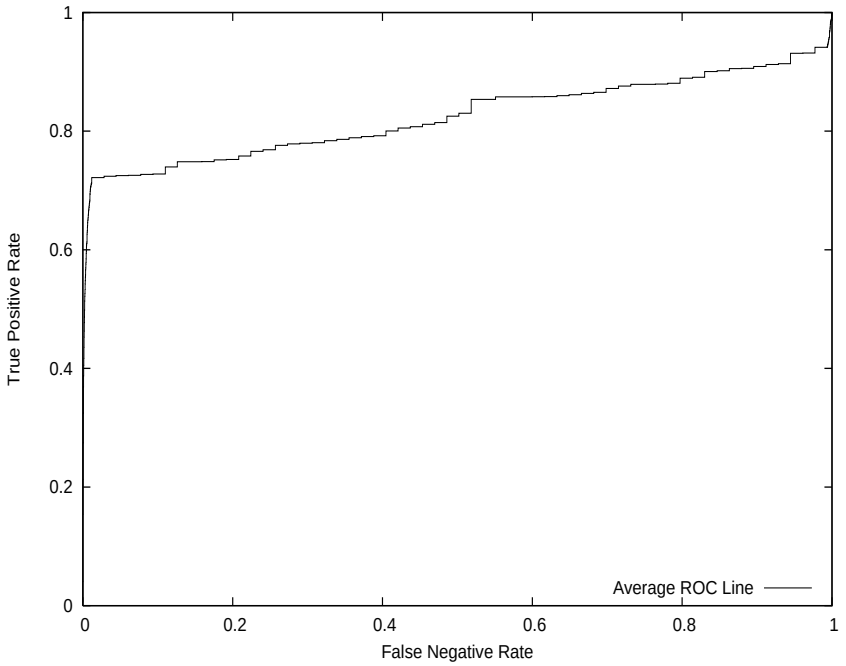


Fig. 4.22. ROC graph for the combined data source in the Anomaly Detection experiment.

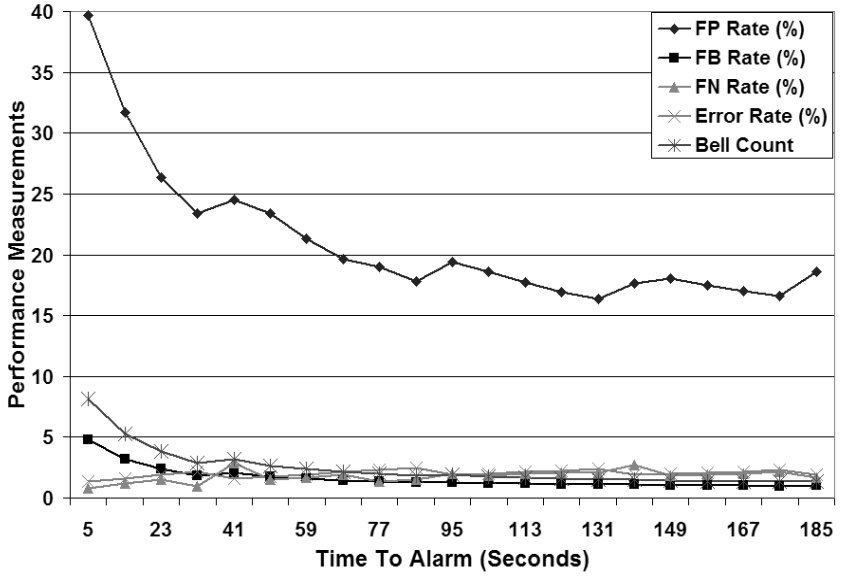


Fig. 4.23. Performance measurements versus the time to alarm for the combined data source in the Anomaly Detection experiment.

curve for the combined data is shown in Figure 4.22. The area under the ROC curve was 0.82. It was evident that for this dataset a combination of different biometric sources improved accuracy. Figure 4.23 shows the error rates as the time to alarm increased from 5 to 185 seconds. The lowest false positive, false bell, false negative and error rates of 16.34%, 1.09%, 2.1% and 2.40%, respectively, and a bell count of 1.50 bells were obtained when the detection time was 131 seconds and the results were smoothed over $m = 29$ classification instances.[5] We next analyzed the performance of the combined classifier in more detail.

Analysis of high FB and FN rates: Table 4.14 shows users with a false bell rate above 5.0%. The first column of the table shows User ID, second column has the false bell rate for the specific user; the third column has the time it took user to complete the data collection process; and the fourth and fifth columns show the average number of all mouse points and all keystroke points, respectively. Detailed examination of Table 4.14 revealed that the number of mouse points was very low for users 4 and 18. Although these users moved a mouse to complete the assignment, they induced very few mouse events which made it difficult for the classifier to distinguish them from the rest. Similarly, users 4, 18 and 38 had only a few keystroke points. Finally, users 18, 38 and 44 spent less than an hour collecting the data which was one–quarter of the average data collection time. Although the combined classifier combined the keystrokes, mouse and GUI data, it needed to have enough of the *combined* data to build an accurate profile of normal behavior.

Table 4.15 shows users with false negative rate above 3.5%. The first column of the table shows User ID, second column has the user's false negative rate, third column has the elapsed time of the data collection; and the fourth and fifth columns show the average number of mouse movements and keystroke points, respectively. Analysis of the Table 4.15 entries revealed that users 22, 28, 42 and 48 spent less than two and a half hours collecting the data which was about half the average

[5]Higher values of m increased the average error rates, because of the limited number of classification instances per user.

Table 4.14
Users with false bell rate above 5.0% for the combined data source in the Anomaly Detection experiment.

ID	**FB Rate**	**Elapsed Time**	**#*All* Mouse**	**#Keystrokes**
4	5.34%	8.88 hrs	**37.4**	**18.41**
18	9.32%	**0.38** hrs	**31.0**	**2.0**
38	8.67%	**0.87** hrs	141.6	**1.3**
44	8.11%	**0.90** hrs	156.9	22.8
AVG	1.76%	4.1 hrs	86.0	28.4

Table 4.15
Users with false negative rate above 3.5% for the combined data source in the Anomaly Detection experiment.

ID	**FN Rate**	**Elapsed Time**	**#Mouse Moves**	**#Keystrokes**
2	3.73%	3.35 hrs	**37.3**	34.2
8	3.76%	18.43 hrs	44.6	**3.8**
22	3.74%	**2.48** hrs	**32.2**	**11.2**
28	3.76%	**1.33** hrs	57.8	63.3
42	3.73%	**0.72** hrs	81.5	23.1
48	3.75%	**0.72** hrs	188.3	58.6
AVG	1.50%	4.1 hrs	65.5	28.4

time. Furthermore, users 2 and 22 had less than two–thirds of the average number of mouse movements; and users 8 and 22 had well below the average number of keystroke points. All these factors contributed to the failure of the combined classifier to generalize well. We concluded that the classifier overfit the data and created a *too broad* a profile of the valid user's behavior. Subsequently, when an intrusion occurred the intruder's behavior was misidentified as that of the valid user and an

Table 4.16
The ten most significant features over all 61 users in the Anomaly Detection experiment.

#	**Feature Description**
1.	Number of **item events**
2.	Number of **combo box events**
3.	Mean of Y coordinates of **mouse wheel**
4.	Skewness of Y coordinates of **mouse moves**
5.	Number of **icon events**
6.	Number of **regular keys**
7.	Mean of Y coordinates of **mouse events**
8.	Number of **mouse events**
9.	Number of **control keys**
10.	Mean of Y coordinates of **double clicks**

alarm was not raised. This suggested that more *active* data (i.e., event data) per user dataset was needed to improve the overall accuracy. We conjectured that this was particularly true as the number of users to be discriminated increased. We validate our conjecture in Chapter 6.

Analysis of features and decision trees: Finally, we analyzed the size of the decision trees produced by the combined classifier and the features used to describe users. The average number of unique features per tree was 150, while the average size of the trees was 372 nodes (excluding leaves). Seven of the ten most significant features were computed from the *leaves* of the hierarchies in Figures 3.4 to 3.9: item, combo box, mouse wheel, mouse movements, icon, control keys and double clicks. The other most significant features were computed from the regular keys and mouse events (see Table 4.16). The decision trees of 47 users had as a root node one of these ten most significant features. The number of events in a particular category

was the prevailing measurement, and the Y screen coordinates of the cursor were the best "raw" measurements for modeling user behavior.

Discussion

We concluded the Anomaly Detection experiment with a brief discussion of our findings. Empirical results suggested the advantage of using a combination of data sources in user re–authentication as opposed to using each source as a standalone classifier. The results also showed benefit from applying a smoothing filter function on top of the basic anomaly implementation scheme. Closer examination of the data sets of those users who produced either a high false bell or false negative rate revealed that these users had spent less time doing the data collection and/or had under–utilized their I/O devices. We inferred that a longer data collection time and more raw data instances per user are needed to build a more accurate profile of user behavior.

We assumed a closed–setting scenario (i.e., data could be collected from all users) and applied a supervised learning algorithm to learn a profile of normal user behavior. As we shall see in Chapter 6, we relaxed the closed–setting scenario constraints and investigated:

1. The performance of the combined classifier on a previously *unseen* intruder dataset (e.g., the detection of an outsider pretending to be an insider);

2. The ability to track a valid user across different computer and I/O configurations;

3. The degree of distinguishability among users when they were given an identical task to perform; and

4. Whether the approach was scalable, i.e., did the accuracy measurement remain fairly consistent as the number of users increased;

Table 4.17
The ten most correlated features in the 280–feature space.

#	**Feature Description**
1.	Skewness of the 2–graph of **all mouse events**
2.	Skewness of the 3–graph of **all mouse events**
3.	Skewness of the 4–graph of **all mouse events**
4.	Skewness of the 5–graph of **all mouse events**
5.	Skewness of the 6–graph of **all mouse events**
6.	Skewness of the 7–graph of **all mouse events**
7.	Skewness of the 8–graph of **all mouse events**
8.	Standard deviation of the distance of **mouse movements**
9.	Standard deviation of the distance of **double clicks**
10.	Standard deviation of the angle of **wheel movements**

4.3.3 Experiment III: Reducing the Feature Space

In this experiment we addressed the issue of feature space reduction. It was well known that a classifier was only as good as the features it used [107]. The importance of feature space reduction was amplified when the feature set was high–dimensional as was the case with our system. In Chapter 3 we conjectured that there existed some subset of most uncorrelated and discriminatory features in our candidate feature space. In this section we introduced a heuristic approach to feature space reduction. We evaluated the performance of classifiers constructed from the reduced feature set on the mouse data *only*. We used the mouse dataset because its feature space had higher granularity than keystroke and GUI data.

We wished to determine how many features could be eliminated from the dataset before the classification error rates began to rise. To accomplish this task, we employed the following heuristic:

Table 4.18
The ten least correlated features in the 280–feature space.

#	**Feature Description**
1.	Mean of X coordinates of **single clicks**
2.	Mean of X coordinates of **double clicks**
3.	Standard deviation of X coordinates of **NC mouse movements**
4.	Standard deviation of Y coordinates of **NC mouse movements**
5.	Skewness of Y coordinates of **NC mouse movements**
6.	Mean of speed of **mouse movements**
7.	Standard deviation of the speed of **mouse movements**
8.	Skewness of the speed of **mouse movements**
9.	Mean of Y coordinates of **mouse movements**
10.	Standard deviation of the speed of **NC mouse movements**

1. We computed the pairwise correlation between each two candidate mouse features. There were 280 of them.

2. We counted the number of features each feature was correlated with a correlation factor of 0.8[6] and above.[7] Features extracted from the mouse events had a correlation factor of 1.0 with features extracted from the mouse wheel and clicks. We showed the ten most (i.e., with the highest correlation coefficient) and the ten least (i.e., with the lowest correlation coefficient) correlated features in Tables 4.17 and 4.18, respectively.

3. We selected a subset of those features which had a correlation factor of 0.8 and above with at most n other features.

[6]We set the threshold at 0.8 because we wanted to reduce the feature space to the extent of observing increased error rates on a reduced feature space during classification.

[7]We experimented with higher threshold values in step (3).

4. We repeated the Anomaly Detection experiment on the reduced feature subsets to determine the performance of the classifiers.

Correlation matrix: For each pair of features, (f_i, f_j), we computed the correlation coefficient $r(f_i, f_j)$ for each feature f_i. Then, we counted the number of features $f_j (i \neq j)$ such that $r(f_i, f_j) \geq 0.8$. If this count, n_i for feature f_i was $\leq cutoff$, then we retained the feature, otherwise we eliminated it. We evaluated values for cutoff in the set $[10, 15, 25]$ which produced subsets of size 134, 113 and 89, respectively. The 134–, 113– and 89–feature subset represented a 52%, 60% and 68% reduction in the feature space size, respectively.

We compared the performance of using the full 280, and of the 134, 113 and 89 feature subsets using the same experimental setup as in the Anomaly Detection experiment (see Section 4.3.2). Our hypothesis was that the absence of noisy or irrelevant features would reduce errors due to classification overfitting and improve the overall accuracy. Furthermore, we anticipated a decrease in the amount of time required to train a mouse classifier.

Anomaly detection: To obtain a model of normal user behavior we trained a *binary* mouse classifier on a valid user dataset and the remaining $N-1$ user datasets in the role of intruders. We used all four feature subsets: 1) 280–, 2) 134–, 3) 113– and 4) 89–feature subset to determine the overall accuracy of the system. The results shown in Figure 4.24 were obtained by the 134-feature subset. The average false positive, false bell, false negative and error rates were 40.06%, 4.84%, 2.20% and 2.93%, respectively, and a bell count was 1.29 bells over all 52 users. Table 4.19 summarizes the results obtained for each mouse feature subset. The first row of Table 4.19 shows the results obtained by the two implementation schemes (e.g., basic and smoothing) from the 280–feature space. The second, third and fourth rows of Table 4.19 show the results obtained from the 134–, 113– and 89–feature subset, respectively. In Section 4.3.2 Figure 4.19 showed results for each user obtained from the 280–feature space.

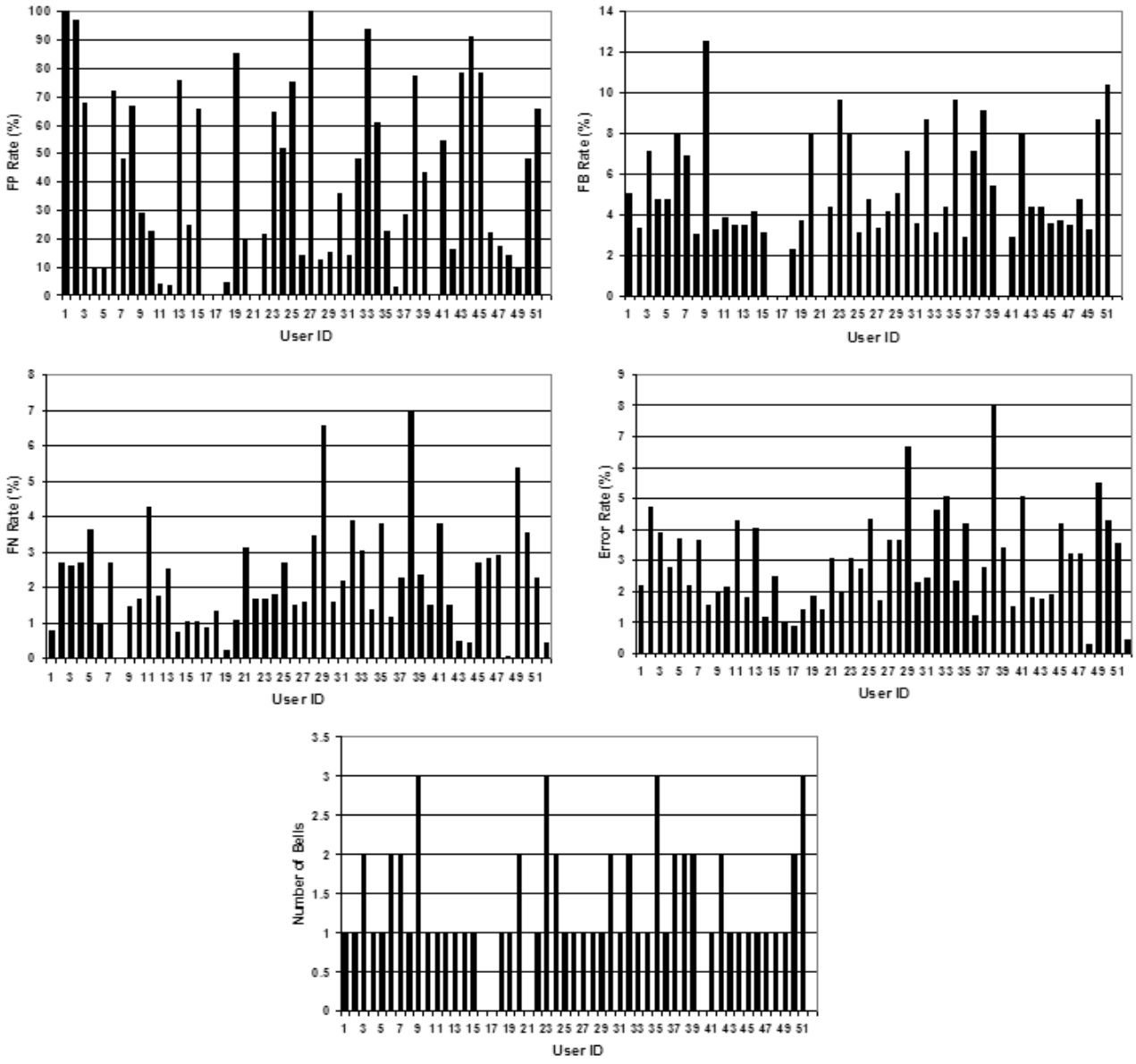


Fig. 4.24. Upper left figure shows the false positive rates; upper right shows the false bell rates; center left shows the false negative rates, center right shows the overall error rates and the bottom figure shows the Bell Count on the 134–feature subset for all 52 users in the Feature Space Reduction experiment.

Both 134- and 113–feature subsets outperformed the 280-feature subset thereby supporting our claim that many mouse features were highly correlated and hence

Table 4.19
The average and the standard deviation values of FP, FB, FN and Error Rates and the Bell Count over all 52 users for the basic and smoothing implementation schemes in the Feature Space Reduction experiment. Tables (a), (b), (c) and (d) show results for feature subsets of 280 (i.e., the full feature space), 134, 113 and 89 features, respectively.

Scheme	**FP Rate**	**FB Rate**	**FN Rate**	**Error Rate**	**Bell Count**
Basic	61.43±28.45%	8.96±4.97%	0.89±0.60%	2.03±0.94%	2.29±1.16
Smoothing	43.66±31.93%	5.26±3.06%	2.04±1.39%	2.84±1.48%	1.40±0.82

(a)

Scheme	**FP Rate**	**FB Rate**	**FN Rate**	**Error Rate**	**Bell Count**
Basic	57.33±28.85%	9.22±5.55%	0.83±0.58%	1.91±0.93%	2.38±1.46
Smoothing	40.06±32.20%	4.84±2.84%	2.20±1.49%	2.93±1.56%	1.29±0.75

(b)

Scheme	**FP Rate**	**FB Rate**	**FN Rate**	**Error Rate**	**Bell Count**
Basic	60.32±27.73%	9.66±4.97%	0.83±0.52%	1.96±0.88%	2.52±1.34
Smoothing	42.30±31.40%	5.24±3.02%	2.08±1.34%	2.85±1.44%	1.40±0.82

(c)

Scheme	**FP Rate**	**FB Rate**	**FN Rate**	**Error Rate**	**Bell Count**
Basic	62.23±28.20%	9.64±5.03%	0.93±0.64%	2.08±1.0%	2.48±1.21
Smoothing	43.80±32.98%	4.93±3.44%	2.30±1.47%	3.09±1.57%	1.23±0.73

(d)

redundant. The 89-feature subset obtained slightly higher error rates, but this was not surprising considering its size was one–third of the candidate feature space. Areas under the ROC curves for the 280–, 134–, 113– and 89–feature subsets were 0.78, 0.79, 0.77 and 0.73, respectively. We showed the ROC curve for the 134–feature subset in Figure 4.25. In Figure 4.26 we plotted the average error rates for each feature

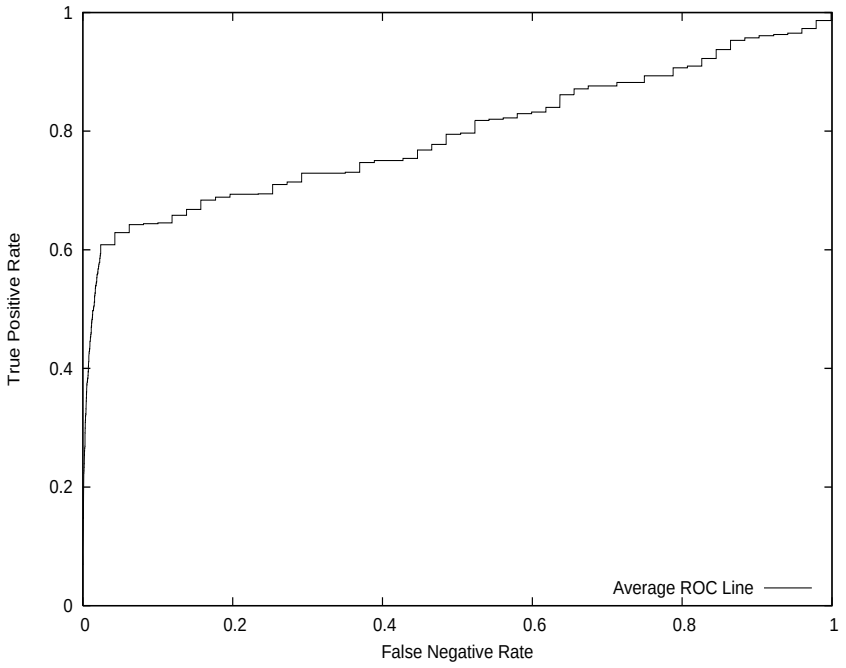


Fig. 4.25. ROC graph for the 134–feature mouse subset in the Feature Space Reduction experiment.

subset as the number of features increased from 89 to 280. The performance of the classifiers constructed on the reduced feature subsets was nearly identical to that of the complete feature space classifier. We concluded that the performance was similar, but that the speed–up achieved during the training phase was significant when we performed the feature space reduction (recall that the runtime of the decision trees was dependent on the number of features in a dataset).

Analysis of features and decision trees: To gain more insight into the reduced feature subset performance we analyzed the size of the decision trees and the features used to describe users for all four mouse feature subsets. We summarized the average Tree Size and the number of Unique Features per User (i.e., the features that appeared at least once in one of the decision trees for a specific user) obtained over all 52 users in Table 4.20. The average size of the decision trees was virtually identical for all four classifiers. In comparison to the results obtained on the 61-user dataset, the 52-user mouse classifier produced six times smaller decision trees with

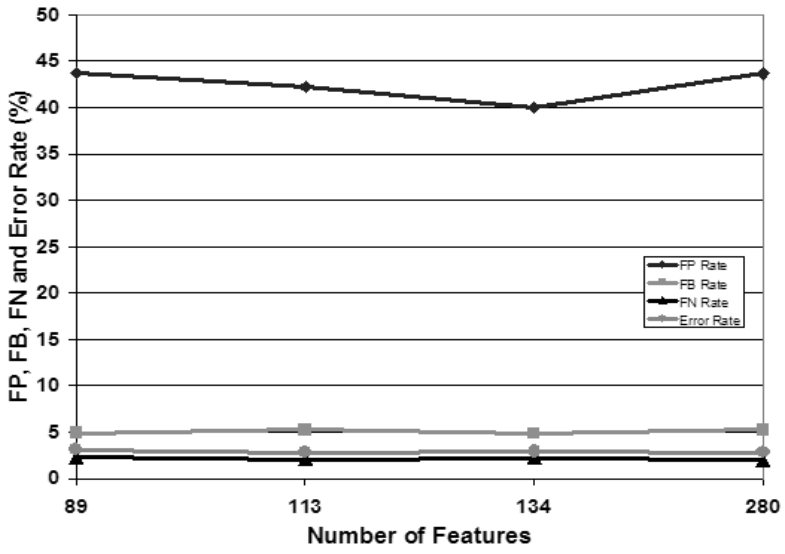


Fig. 4.26. The average FP, FB, FN and Error rates and the Bell Count for each feature subset over all 52 users in the Feature Space Reduction experiment.

one–third the number of features per tree, because we eliminated those users with insufficient amount of training data per dataset. From Table 4.20 we saw that the number of unique features per tree was also nearly identical for all four classifiers which validated our choice of feature space reduction heuristic. In environments where efficiency was more important than accuracy it might be desirable to reduce the number of features even further. One way to do it was by a careful examination of those decision tree nodes where a *tie* between two or more features with the same entropy value occurred and then selecting the most uncorrelated one to be the node feature.

4.3.4 Experiment IV: Evaluation of the Feature Hierarchy

Our last experiment described in this Chapter was designed to examine the "usefulness" of grouping mouse data into the mouse hierarchy shown in Figure 3.4. First, we *omitted* features associated with the fourth layer of the hierarchy and then re-

Table 4.20
The average Tree Size and the number of Unique Features per User over all 52 users in the Feature Space Reduction experiment.

Feature Subset	**280 Features**	**134 Feature**	**113 Features**	**89 Feature**
Tree Size	94.70	96.62	98.5	104.27
Unique Features per User	33.27	31.50	31.0	30.40

Table 4.21
The Feature Hierarchy experiment results.

Feature Set	**Levels 1, 2, 3 & 4**	**Levels 1, 2 & 3**	**Levels 1 & 2**	**Levels 3 & 4**
FP Rate	43.66%	44.84%	49.07%	50.26%
FB Rate	5.26%	6.34%	6.91%	3.05%
FN Rate	2.23%	1.48%	1.84%	2.92%
Error Rate	3.02%	2.31%	2.75%	3.77%
Bell Count	1.40	1.69	1.85	0.79

peated the Anomaly Detection experiment using the remaining 200 features. The smoothed results averaged over all 52 users are shown in the third column of Table 4.21. Then we *omitted* features associated with both the third and the fourth layers of the hierarchy, which left us with 120 features, and obtained the smoothed results shown in the forth column of Table 4.21. Finally, we *omitted* features associated with the top two layers of the hierarchy, which left us with 120 features, and obtained the smoothed results shown in the fifth column of Table 4.21. The difference between these results and the results obtained using the complete feature set shown in the first column of Table 4.19 became statistically significant[8] when we removed

[8]We used the t-test with alpha=0.05 to evaluate the statistical significance.

both lower layers or both upper layers of the hierarchy. We concluded that grouping mouse data into the mouse hierarchy improved the overall accuracy, but it also increased the time it took to generate a profile of valid user behavior. This is a well–known design tradeoff that could influence the choice of feature space in practice. In an environment where efficiency constraints are present our recommendation is to implement the most uncorrelated features first and add additional features if the measured performance is unsatisfactory.

4.4 Summary and Conclusions

In this Chapter we investigated the applicability of keystroke dynamics, mouse movements and GUI events, individually and in combination, to continuous authentication of users for the duration of their login sessions. We assumed a closed–setting deployment scenario where data could be readily obtained from all users. We collected data from 61 volunteers. We also extracted the mouse *only* data from the 61-user dataset to obtain a 52-user dataset which we then used to study the feature space in more detail.

The first experiment was designed to give us an initial insight into the strength of each classifier as a user re–authentication tool. We built a binary classifier to examine the degree of discrimination between each pair of users. The obtained results were shown in Table 4.3. Our next step was to test if each classifier could build an accurate model of normal user behavior after seeing the behavior of a valid user A and the behavior of the remaining $N-1$ users in the role of intruders. The results indicated that the combined classifier outperformed each individual classifier (see Table 4.4). The decision trees generated in this experiment for each classifier supported our hypothesis that most discriminating features were computed from the leaves of the feature hierarchies in Figures 3.4 to 3.9. The Feature Space Reduction experiment further illustrated this point on the 52-user mouse dataset by employing a heuristic-driven approach to select some subset of the most uncorrelated features. Finally,

the fourth experiment was designed to investigate the usefulness of our feature space based on the mouse hierarchy from Figure 3.4. Removing features associated with the two lower and two upper levels of hierarchy decreased the performance significantly, and we concluded that in an environment where efficiency constraints were present the most uncorrelated features should be implemented first. Additional features might be added if the measured performance was unsatisfactory.

5. BOOSTING PERFORMANCE WHEN THE AMOUNT OF DATA IS LIMITED PER USER

In this chapter we investigated the performance of a user re–authentication system when the amount of data per user dataset was limited. This situation arises in practice when the data collection resources are insufficient and/or we wish to build a profile of a user whose presence at a computer workstation is sporadic at best.

During the classification process, we assumed (without any loss of generality) that a classifier outputted a binary string of class–labels, "N"'s and "A"'s, where "N" stood for the *normal* and "A" for the *anomalous* behavior of a current user, for each classification instance. Given n raw data instances we wished to determine if it was better (e.g., more accurate and more computationally efficient) to let $\|W\| = n$ or let $\|W\| = \frac{n}{m}$ and applied a smoothing filter to the m classifications. In Chapter 2 we defined *smoothing* as a process of reducing noise in a dataset, in our case, reducing minor, transient changes in user behavior.

Accuracy considerations: We conjectured that a classifier constructed over a smaller window size W and smoothed over m classification instances was more accurate when discriminating among different users than a classifier constructed over windows of size $m * W$. We based this conjecture on the following observation: as we increased the window size W and computed a classification instance over W, we reduced the granularity of behavioral patterns present in that window. So, when we built a classifier over a smaller W and smoothed m such windows, we preserved the patterns better, because we were effectively averaging over $m + W$ instead of $m * W$ instances.

Computational efficiency considerations: Additional benefit from constructing classifiers over smaller window sizes was increased efficiency. The time it took to

compute features increased with the window size. For a window size of W, the time it took to compute the features was $O(\|W\|^3)$ (skewness is an $O(n^3)$ algorithm). Doubling the window size would increase the time it took to compute the features by a factor of eight. In our experiments we implemented overlapping windows (see Figure 4.4). As a result, the features were computed every 50 data points following the computation of the first feature vector instance.

The remainder of this chapter is organized as follows: Section 5.1 discusses the related work. Section 5.2 describes experimental methodology. Seven smoothing filter functions and an optimization criterion used to select a threshold for each smoothing function are formally defined. Section 5.3 explains the experiments and discusses the results. Section 5.4 summarizes and concludes this chapter.

5.1 Related Work

Given m binary $\in [A, N]$, sequential outputs our goal was to make a prediction for the entire sequence of "A"s and "N"s. To this end, we applied a smoothing filter function over m. In the published literature, the similarity metrics were most closely related to smoothing filter functions, but there were significant differences. Similarity metrics were used to *classify* a sequence of attributes (which did not need to be binary attributes) by computing the degree of "similarity" between the current sequence and an existing profile of normal behavior whereas the smoothing filter functions were applied *after* the current sequence was classified by a classifier to *smooth-out* any minor, transient changes in user behavior.

5.1.1 Similarity Metrics

Similarity metrics were used in different areas of machine learning, e.g., in clustering, instance-based learning, pattern recognition, etc. Before we describe the various similarity metrics used in computer security, we first introduced some notation that was common to the methods described below (see Table 5.1) [49].

Table 5.1
Notation used to describe Uniqueness, Bayes 1-Step Markov, Hybrid Multi–step Markov, Compression and IPAM methods.

Notation	**Meaning**
C	Training data
c	Test data
C_{ut}	tth sequence of user u in C
N_{ujk}	Number of times user u used the sequence (j, k) in C
N_{uk}	Number of times user u used sequence k in C
N_u	Length of user u's training data sequence
n_{ujk}, n_{uk}, n_u	As above for test data
x_{ub}	Score for user u in a sequence b
U	Total number of users
U_k	Number of users who have used sequence k in C
K	Total number of distinct sequences
T	Number of commands in a test data sequence

Degree of disorder: Bergadano, Gunetti and Picardi used a distance metric, $d(A1, A2)$, to compute the similarity between two N-element arrays $A1$ and $A2$ [57]. They computed $d(A1, A2)$ by first measuring the *degree of disorder* defined as the sum of the distances between the position of each element in $A1$ and the position of the same element in $A2$. They then normalized the degree of disorder by dividing it by the value of the maximum disorder of a N–element array, thereby making it possible to compare the disorder of arrays of different sizes. They defined the maximum disorder of an array of N elements as $\frac{N^2}{2}$, if N was even; and $\frac{(N^2-1)}{2}$, otherwise. They classified the current instance X as belonging to the user u with the smallest mean distance $md(u, X)$, where $md(u, X) = \frac{1}{M}\sum_{i=1}^{M} d(u_i, X)$ and M was the number of arrays in the user u's profile. The authors made further refinements to the

classification rule by introducing the mean of the distances of the arrays forming the model of user u_1's behavior (denoted by $m(u_1)$) and classifying X as belonging to u_1 when the following held: $md(u_1, X) < m(u_1) + |k(md(u_2, X) - m(u_1))|$, where $k \in [0, 1]$. In our experiments we used Shannon entropy and the weighted entropy to measure the disorder of n classification instances and learn a profile of normal user behavior.

Sequence match: Lane and Brodley [42] defined the similarity function, $Sim(X, Y)$, between sequences $X = (x_0, x_1, ..., x_{l-1})$ and $Y = (y_0, y_1, ..., y_{l-1})$ as $Sim(X, Y) = \sum_{i=0}^{l-1} w(X, Y, i)$, where $w(X, Y, i) = 0$, if $i < 0$ or $x_i \neq y_i$; and $w(X, Y, i) = 1 + w(X, Y, i - 1)$, if $x_i = y_i$. Note that $w(X, Y, i) = 0$ for $i < 0$ which resulted in $w(X, Y, 0)$ being well defined when $x_0 = y_0$. The authors also defined the converse measure, distance, to be $Dist(X, Y) = Sim_{max} - Sim(X, Y)$, where $Sim_{max} = Sim(X, X) = \sum_{i=1}^{l} i = \frac{l(l+1)}{2}$. A user profile was a collection of sequences D and the similarity between the profile and a newly observed sequence X was defined to be $Sim_D(X) = max_{Y \in D}\{Sim(Y, X)\}$. The sequence X was classified as normal if $Sim_D(X) \geq r$, $r \in [0, 1]$; and anomalous, otherwise. The "streaks" smoothing function described in Section 5.2.1 captured the same idea as Lane and Brodley's sequence matching function.

Uniqueness: The "uniqueness" approach was based on the idea that rare patterns of behavior might indicate the presence of an intruder. Schonlau *et al* defined a test statistic, which built on the notion of unpopular and uniquely used sequences as [49]: $x_u = \frac{1}{n_u} \sum_{k=1}^{K} W_{uk}(1 - \frac{U_k}{U})n_{uk}$, where the weights $W_{uk} = \frac{-v_{uk}}{v_k}$ if user u's training data contained sequence k and $W_{uk} = 1$ otherwise. They defined $v_{uk} = \frac{N_{uk}}{N_u}$ and $v_k = \sum_u v_{uk}$. The fraction $(1 - \frac{U_k}{U})$ acted as a uniqueness index. It was 0 if all users had used this sequence before and 1 if none of the users had used it before. The weights W_{uk} determined if the uniqueness index should be added or subtracted depending on whether the sequence was seen before or not. If $x_u \geq t$, where t was a threshold, the current user was an intruder and if $x_u < t$, the current user was a valid user. The authors assigned the same threshold of 0.2319, derived empirically,

to all users. This approach required a *history* table containing *all* sequences seen for each user in the training data. The table look–up slowed down the classification process during testing and hence, degraded the computational efficiency of a user re–authentication system.

Bayes One–Step Markov: The Bayes one–step Markov approach considered the one–step transitions from one sequence to the next in addition to the sequence frequencies [49]. It used a Bayes factor statistic to test the null hypothesis that the observed one–step sequence transition probabilities were consistent with a historical transition matrix. A Dirichlet distribution was assumed when modeling probabilities. Our goal was to *smooth–out* transient user behavior *after* a particular instance had been classified by a classifier. Each sequence that we observed was a binary string of labels, "A"s and "N"s, where "A" stood for *anomalous* and "N" for *normal* behavior of the current user. Unlike command–line input data used in [49] which had a finite dictionary, our input had 2^n possibilities, where n was the length of a sequence under observation. A historical transition matrix produced by the Bayes one–step Markov approach grew exponentially in size with n. As a result, the smoothing filter functions we described in this chapter were more efficient than the Bayes one–step Markov approach.

Hybrid Multi–step Markov: This method was based on a multi–step Markov chain and on an independence model [49]. The high dimensionality inherent in a multi–step Markov chain was overcome by: 1) restricting attention to a subset of the most frequent sequences (with the remaining sequences represented under a single sequence termed "other") and 2) using a *mixture transition distribution* (MTD) approach to model the transition probabilities [49]. In a case when the test data contained many sequences unobserved in the training data, a simple independence model was used instead of the Markov model. The independence model estimated the probabilities from a contingency table of users instead of sequences. The authors designed an approach that automatically toggled between the Markov and the independence model. Similarly to the Bayes one–step Markov method, this method

lacked efficient computation during testing (it is an $O(S^2T)$ algorithm, where S was a finite number of states over a time sequence given by T) and we did not implement it.

Compression: The hypothesis behind compression approaches was that the test data appended to historical training data compressed more readily when the test data stems from a valid user rather than an intruder [49]. Schonlau *et al* defined a score x to be the number of additional bytes needed to compress test data when appended to the training data as $x = compress(\{C, c\}) - compress(C)$, where $\{C, c\}$ was the test data appended to the training data and $compress()$ was a function that gave the number of bytes of the compressed data. The authors used the UNIX tool "compress" to implement their approach. This approach might be effective on a smaller training dataset, but training data could be large in practice. In our experiments we had on average 450MB of data per user. Compressing this much data to classify each instance was computationally prohibitive.

IPAM: Davison and Hirsh developed the Incremental Probabilistic Action Modeling (IPAM) algorithm [120]. It was based on a one–step sequence transition probabilities estimated on the training data. The estimated probabilities were updated over time using an exponential updating scheme. Upon arrival of a new sequence all existing transition probabilities were aged by multiplying them with α, and $1 - \alpha$ was added to the most recent addition (α was empirically set at 0.9). Given a sequence, it was then possible to predict the next sequence by choosing the one with the highest transitional probability. The fraction of valid predictions of the test data formed a score. If the score was below a threshold an alarm was raised. This method also lacked efficient computation during testing for the same reasons mentioned above and we did not implement it.

To investigate the performance of a user re–authentication system when the amount of data per user dataset was limited, we implemented: 1) majority vote, 2) weighted majority vote, 3) entropy, 4) weighted entropy, 5) information gain, 6) weighted information gain and 7) streaks, as a smoothing filter function.

5.2 Experimental Methodology

In this section we investigated the performance of a user re–authentication system when the amount of data was limited per user dataset. Specifically, we studied if a *smoothed* classifier constructed over a smaller window size could produce better or equal accuracy measurements than an *unsmoothed* classifier constructed over a larger window size. To this end, we varied window sizes $W \in [100, 300, 500, 1000]$ and we determined the effectiveness of: 1) majority vote, 2) weighted majority vote, 3) entropy, 4) weighted entropy, 5) information gain, 6) weighted information gain and 7) streaks, as a smoothing filter function. We formally defined each smoothing function and the optimization criterion equation used for threshold selection. Our goal was to determine if there existed some behavioral patterns in the classifier's binary output which could be used to achieve comparable performance for different values of W.

The empirical results obtained in Chapter 4 suggested an advantage of using a combination of data sources to perform user re–authentication as opposed to using each source as a standalone classifier. To this end, we ran experiments in this chapter on the combined data source *only*. We used the 61–user dataset described in Section 4.2.2.

We reported the FP, FB, FN and Error rates as well as the actual number of bells sounded for the basic and smoothing implementation schemes in our experiments (see Table 4.2 for a complete list of performance metrics).

To evaluate our methods we used a ten–fold cross–validation (CV). In each of the ten CV runs we used a randomly-selected 70% of the user dataset for training, 15% for tuning and the remaining 15% for testing. Recall that in the experiments of Chapter 4 we selected the smoothing parameters on the training data. Here, we created a separate tuning set for the parameter-selection to obtain more robust parameter-values (i.e., class–labels produced by a classifier on the training data were

more accurate than those on the tuning or testing data). The results reported were averaged over ten cross–validation folds.

5.2.1 Smoothing Filter Functions

We began by introducing the notation used throughout this section to define the smoothing filter functions. Let S be the set of all classification instances and let $N = \|S\|$. We define $S_m \subseteq S$, $m \in \{1, M\}$. Furthermore, let n_m^+ and n_m^- be the number of valid user and intruder instances in S_m, respectively, and let c_m^+ and c_m^- be the sum of probabilities of valid user and intruder instances in S_m, respectively. We denote the current classification instance as m_j, $j \in [1, m]$ [121].

1. **Majority Vote:** We label the current classification instance in S_m as "N," if $n_m^+ \geq t$, where $t \in [1, m]$, and "A" otherwise.

2. **Weighted Majority Vote:** We label the current classification instance in S_m as "N," if $c_m^+ \geq t$, where $t \in [0, m]$, and "A" otherwise.

3. **Entropy:** Let $p_m^+ = \frac{n_m^+}{n_m^+ + n_m^-}$ be the probability of valid user instances in S_m and let $p_m^- = \frac{n_m^-}{n_m^+ + n_m^-}$ be the probability of intruder instances in S_m. We compute the entropy of S_m as $H(S_m) = -p_m^+ * \log p_m^+ - p_m^- * \log p_m^-$. We label the current classification instance in S_m as "N," if $H(S_m) \geq t$, where $t \in [0, 1]$, and "A" otherwise.

4. **Weighted Entropy:** Let $p_m^+ = \frac{c_m^+}{c_m^+ + c_m^-}$ be the probability of valid user instances in S_m and let $p_m^- = \frac{c_m^-}{c_m^+ + c_m^-}$ be the probability of intruder instances in S_m. We compute the entropy of S_m as $H(S_m) = -p_m^+ * \log p_m^+ - p_m^- * \log p_m^-$. We label the current classification instance in S_m as "N," if $H(S_m) \geq t$, where $t \in [0, 1]$, and "A" otherwise.

5. **Information Gain:** Let p_m^+, p_m^- and $H(S_m)$ be as defined in (3). We define the information gain between the current and the kth previous instance

to be $I(S_m) = H(S_m) - H(S_{m-k})$, where $k \in [2, m]$. We label the current classification instance in S_m as "N," if $I(S_m) \geq |t|$, where $t \in [-1, 1]$, and "A" otherwise.

6. **Weighted Information Gain:** Let p_m^+, p_m^- and $H(S_m)$ be as defined in (4). We define the information gain between the current and the kth previous instance to be $I(S_m) = H(S_m) - H(S_{m-k})$, where $k \in [2, m]$. We label the current classification instance in S_m as "N," if $I(S_m) \geq |t|$, where $t \in [-1, 1]$, and "A" otherwise.

7. **Streaks:** We define L_m^+, the streak function counting the number of valid user "streaks" as

$$L_m^+ = \begin{cases} 1 + L_m & \text{if } m_j = m_{j-1} \\ 0 & \text{otherwise} \end{cases}$$

We label the current classification instance in S_m as "N," if $L_m^+ \geq t$, where $t \in [1, m]$, and "A" otherwise.

5.2.2 The Optimization Criterion

Similar to Section 4.2.3 we implemented each smoothing filter function over a window $m \in [1, 17]$. We increased the upper limit of m from 11 to 17 because we increased the size of our testing set from 10% to 15% and we used the combined data source only. We empirically selected m for each smoothing function and each user so as to optimize the following criterion: $min \sum_{i=1}^{N}(c_1 * FP(u_i) + c_2 FN(u_i))$; $c_1, c_2 \in R^+$ where u was a user and $FP(u_i)$ and $FN(u_i)$ were the false positive and false negative rates of user u_i, $i \in [1, N]$, respectively. We reported results with $c_1 = c_2 = 1$ which assigned equal weight to both error rates.

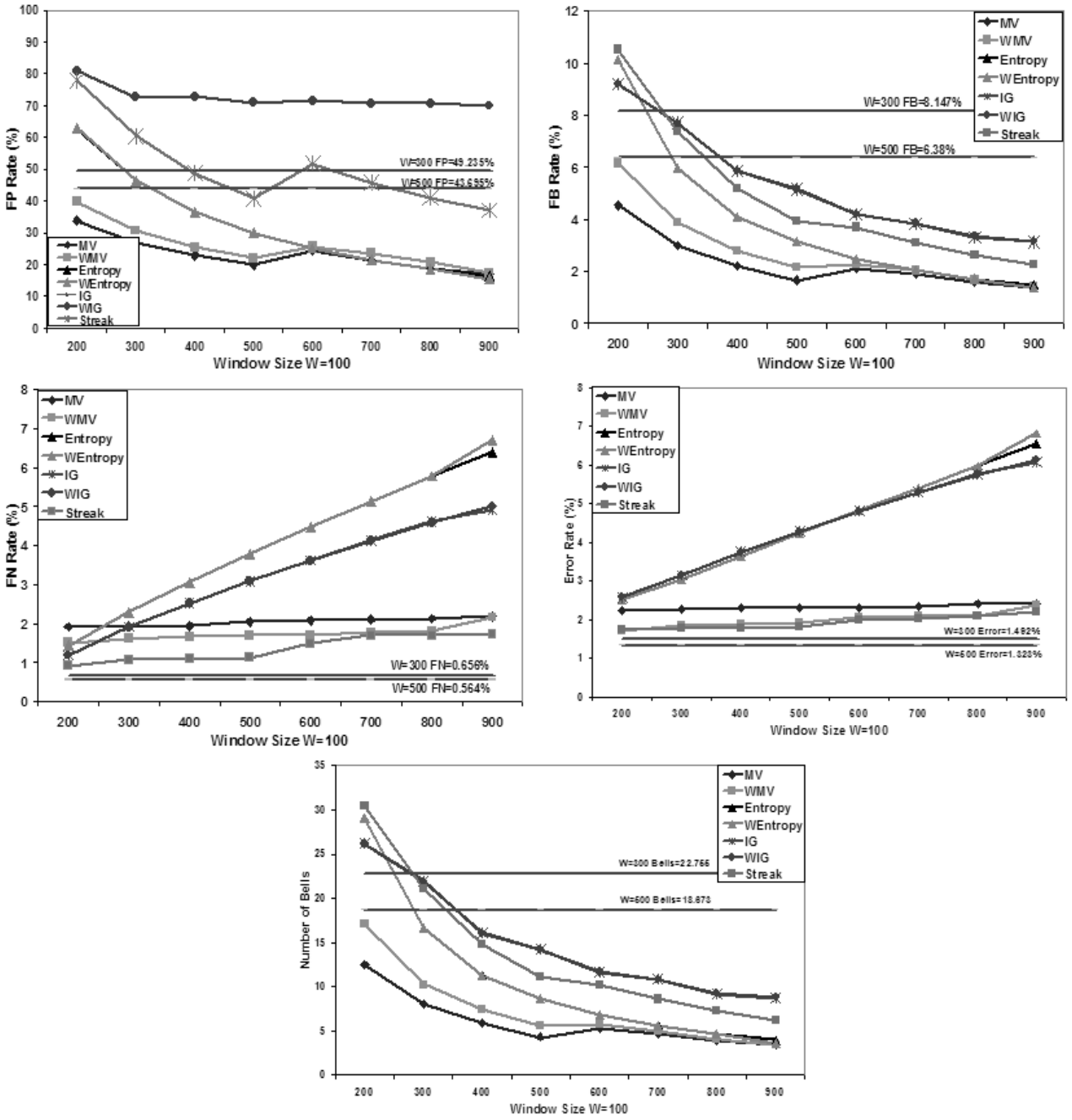


Fig. 5.1. Upper left figure shows the false positive rates; upper right shows the false bell rates; center left shows the false negative rates; center right figure shows the overall error rates, and bottom figure shows the number of false bells for $W = 100$ for all 61 users.

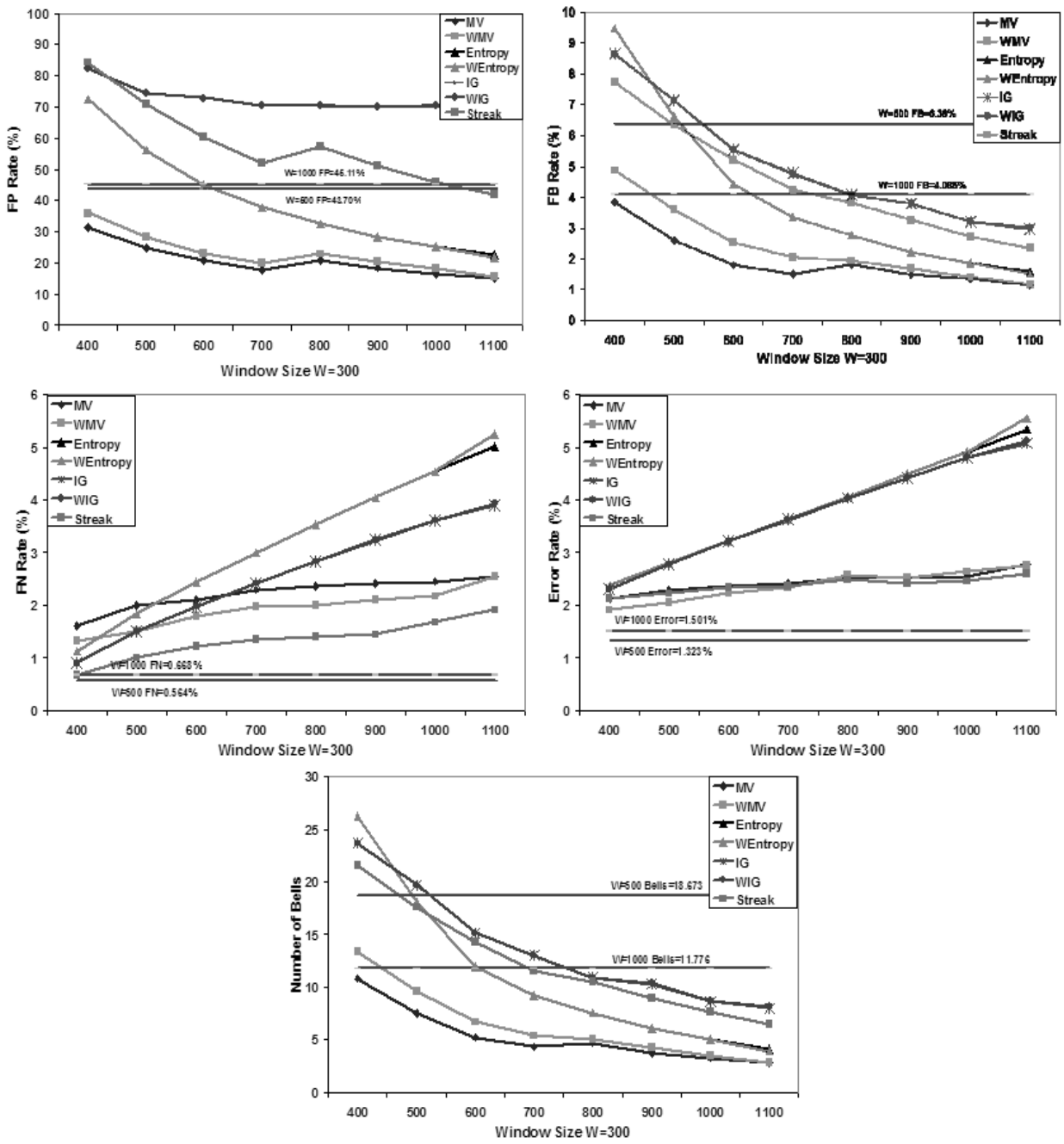


Fig. 5.2. Upper left figure shows the false positive rates; upper right shows the false bell rates; center left shows the false negative rates; center right figure shows the overall error rates, and bottom figure shows the number of false bells for $W = 300$ for all 61 users.

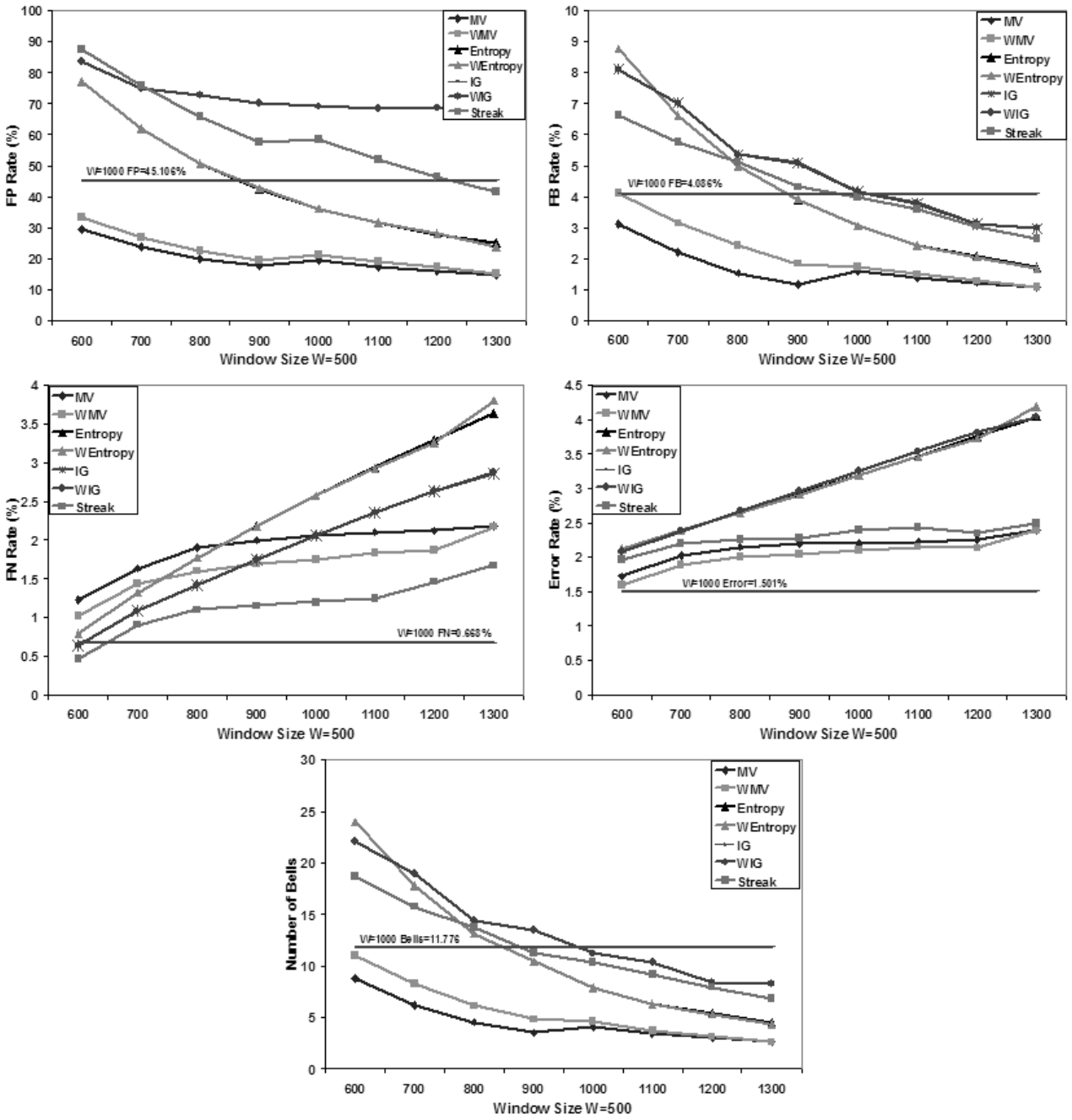


Fig. 5.3. Upper left figure shows the false positive rates; upper right shows the false bell rates; center left shows the false negative rates; center right figure shows the overall error rates, and bottom figure shows the number of false bells for $W = 500$ for all 61 users.

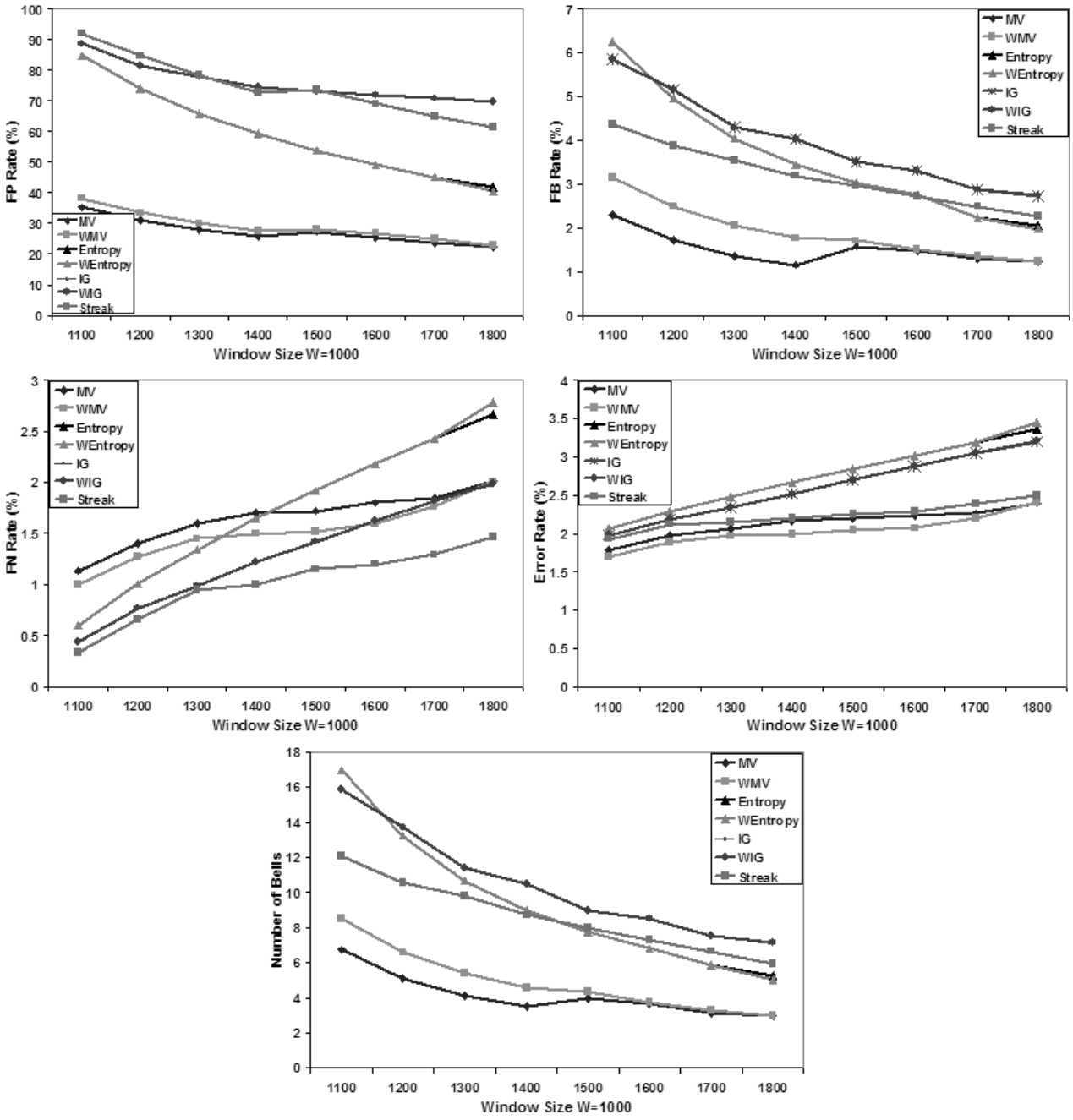


Fig. 5.4. Upper left figure shows the false positive rates; upper right shows the false bell rates; center left shows the false negative rates; center right figure shows the overall error rates, and bottom figure shows the number of false bells for $W = 1000$ for all 61 users.

5.3 Empirical Analysis

To evaluate the performance of smoothing filter functions we repeated the Anomaly Detection experiment from Section 4.3.2. We wished to determine if a *binary* classifier could build an accurate model of normal user behavior after seeing the behavior of a valid user A and the behavior of the remaining $N-1$ users in the role of intruders. We varied the window size W and computed each classification instance over a window of $W \in [100, 300, 500, 1000]$ data points. We overlapped windows so that a new window was obtained from the previous $W - 50$ and subsequent 50 points (see Figure 4.4). As a result, a classification was carried out every fifty points following the classification of the first instance. If a user utilized his/her I/O devices 50 points was approximately equal to a 5–second time interval. We applied a smoothing filter over m classification instances, where $m \in [1 : 2 : 17]$, i.e., m goes from 1 to 17 in steps of 2.

Anomaly Detection Results: The obtained results are shown in Figures 5.1 to 5.4 for the window sizes of 100, 300, 500 and 1000, respectively. Figure 5.1 shows average false positive, false bell, false negative and error rates and a bell count over all 61 users when $W = 100$ and $m \in [1 : 2 : 17]$, so the corresponding window sizes plotted on the x–axis range from 200 to 900 data points, because the windows were overlapping. We did not plot a window size of 100 because entropy, weighted entropy, information gain and weighted information gain each produced a 100% false positive and 0% false negative rate when smoothed over a single instance. This phenomenon followed from their definitions (see Section 5.2.1). We also plotted a horizontal line in each sub–figure of Figure 5.1 to show the false positive, false bell, false negative and error rate and a bell count, respectively, for $W = 300$. We did the same for $W = 500$.

Figures 5.2 to 5.4 show average false positive, false bell, false negative and error rate and a bell count over all 61 users when $W = 300$, $W = 500$ and $W = 1000$, respectively. The range of m was $[1 : 2 : 17]$, so the corresponding window sizes went

Table 5.2
The average values of FP, FB, FN and Error Rates and the Bell Count over all 61 users for the majority vote smoothing function ($W = 100$).

Window Size	**FP Rate**	**FB Rate**	**FN Rate**	**Error Rate**	**Bell Count**
100	58.37%	11.56%	0.61%	1.58%	33.29
100, $m = 1$	47.12%	8.0%	1.07%	1.84%	22.92
200, $m = 3$	33.79%	4.53%	1.92%	2.43%	12.45
300, $m = 5$	**26.23%**	**2.99%**	**2.70%**	**3.07%**	**8.04**
400, $m = 7$	21.29%	2.22%	3.43%	3.71%	5.82
500, $m = 9$	**17.75%**	**1.67%**	**4.14%**	**4.34%**	**4.26**
600, $m = 11$	25.02%	2.10%	1.42%	1.80%	5.18
700, $m = 13$	21.76%	1.92%	1.68%	2.0%	4.63
800, $m = 15$	19.0%	1.60%	1.93%	2.20%	3.84
900, $m = 17$	16.83%	1.40%	2.18%	2.41%	3.35

from 400 to 1100, 600 to 1300 and 1100 to 1800 data points for $W = 300$, $W = 500$ and $W = 1000$, respectively. Similar to Figure 5.1 we omitted plotting the results obtained for $m = 1$. In each sub–figure of Figure 5.2 we plotted two horizontal lines to show the error rates for $W = 500$ and $W = 1000$, respectively. In each sub–figure of Figure 5.3 we plotted a horizontal line to show the error rates for $W = 1000$.

The ranking by accuracy for the smoothing filter functions was 1) majority vote, 2) weighted majority vote, 3) weighted entropy, 4) entropy, 5) streaks, 6) weighted information gain and 7) information gain. The difference in performance between the weighted and non–weighted smoothing functions was negligible. Both weighted entropy and weighted information gain outperformed their non–weighted counterparts. For $W = 100$ and $m = 17$ weighted entropy produced lower false positive rate than majority vote (15.35% versus 16.83%), but the false negative rate was higher

Table 5.3
The average values of FP, FB, FN and Error Rates and the Bell Count over all 61 users for the majority vote smoothing function ($W = 300$).

Window Size	**FP Rate**	**FB Rate**	**FN Rate**	**Error Rate**	**Bell Count**
300	49.23%	8.15%	0.65%	1.49%	22.75
300, $m = 1$	41.23%	5.82%	0.99%	1.69%	16.70
400, $m = 3$	31.21%	3.83%	1.61%	2.12%	10.82
500, $m = 5$	**24.46%**	**2.60%**	**2.19%**	**2.58%**	**7.49**
600, $m = 7$	20.01%	1.81%	2.73%	3.05%	5.22
700, $m = 9$	16.83%	1.51%	3.29%	3.52%	4.32
800, $m = 11$	21.46%	1.83%	1.77%	2.11%	4.69
900, $m = 13$	18.72%	1.50%	2.03%	2.32%	3.75
1000, $m = 15$	**16.72%**	**1.34%**	**2.29%**	**2.54%**	**3.24**
1100, $m = 17$	15.07%	1.15%	2.55%	2.76%	2.80

for weighted entropy (6.69% versus 2.18%). One possible explanation for high error rates produced by the weighted and non–weighted information gain smoothing functions could be that entropy measurements were fairly consistent within m classification instances - e.g., if a user had some transient changes in his/her behavior, such behavior persisted over all m instances. Consequently, no *useful* information was obtained by computing the information gain.

Tables 5.2 to 5.5 show results for window sizes of 100, 300, 500 and 1000, respectively, obtained by the majority vote smoothing function. The second row of each table displays the results obtained by the basic classifier at a particular window size. The highlighted values show results obtained by a classifier constructed over a smaller window size W and smoothed over m classification instances, so that m smoothed windows of size W are equal to either 300, 500 or 1000 data points.

Table 5.4
The average values of FP, FB, FN and Error Rates and the Bell Count over all 61 users for the majority vote smoothing function ($W = 500$).

Window Size	**FP Rate**	**FB Rate**	**FN Rate**	**Error Rate**	**Bell Count**
500	43.69%	6.38%	0.56%	1.32%	18.67
500, $m = 1$	37.40%	4.47%	0.79%	1.44%	12.82
600, $m = 3$	29.47%	3.12%	1.22%	1.73%	8.78
700, $m = 5$	23.85%	2.21%	1.63%	2.03%	6.14
800, $m = 7$	19.87%	1.53%	2.02%	2.35%	4.53
900, $m = 9$	17.15%	1.15%	2.40%	2.67%	3.51
1000, $m = 11$	**19.79**%	**1.58**%	**1.52**%	**1.85**%	**4.04**
1100, $m = 13$	17.63%	1.37%	1.75%	2.03%	3.41
1200, $m = 15$	16.0%	1.22%	1.97%	2.21%	2.98
1300, $m = 17$	14.59%	1.08%	2.18%	2.40%	2.59

Observe that a classifier constructed over $W = 100$ data points and smoothed over $m = 5$ and $m = 9$ instances produced lower error rates than an *unsmoothed* classifier constructed over $W = 300$ and $W = 500$ data points, respectively. Also, a classifier constructed over $W = 300$ data points and smoothed over $m = 5$ and $m = 15$ instances outperformed an *unsmoothed* classifier constructed over $W = 500$ and $W = 1000$ data points, respectively. The obtained results suggested that the performance of a user re–authentication system could be boosted when the amount of data per user is limited by applying a smoothing filter function over m classification instances.

Figure 5.5 shows the results obtained with majority vote as a smoothing function for window sizes 100, 300, 500 and 1000 data points and $m \in [1 : 2 : 29]$. We increased the range of values for m to determine the error rates as m grew large.

Table 5.5
The average values of FP, FB, FN and Error Rates and the Bell Count over all 61 users for the majority vote smoothing function ($W = 300$).

Window Size	**FP Rate**	**FB Rate**	**FN Rate**	**Error Rate**	**Bell Count**
1000	45.11%	4.09%	0.67%	1.50%	11.78
1000, $m = 1$	41.08%	3.34%	0.84%	1.60%	9.63
1100, $m = 3$	35.24%	2.30%	1.13%	1.79%	6.73
1200, $m = 5$	31.09%	1.71%	1.40%	1.97%	5.12
1300, $m = 7$	27.98%	1.36%	1.66%	2.17%	4.10
1400, $m = 9$	25.50%	1.15%	1.90%	2.36%	3.53
1500, $m = 11$	27.58%	1.58%	1.50%	2.0%	3.96
1600, $m = 13$	25.55%	1.48%	1.67%	2.13%	3.65
1700, $m = 15$	23.72%	1.29%	1.84%	2.26%	3.16
1800, $m = 17$	22.28%	1.23%	2.01%	2.40%	3.0

Figure 5.5 shows that as we increase m, the overall accuracy improves. To determine the effect of smoothing on the detection time we computed the average threshold values, t, for $m = 29$ over all 61 users. We obtained the threshold values of 2.45, 2.37, 2.61 and 2.52 for the window sizes of 100, 300, 500 and 1000 data points, respectively. In Section 5.2.1 we labeled the current classification instance as "N" in the majority vote function, if $n_m^+ \geq t$, where $t \in [1, m]$, and "A" otherwise. Therefore, we needed to observe on average 26.51 intruder's instances before we raised an alarm. If the current user utilized the I/O devices 26.51 instances translated to 2.20 minutes of detection time.

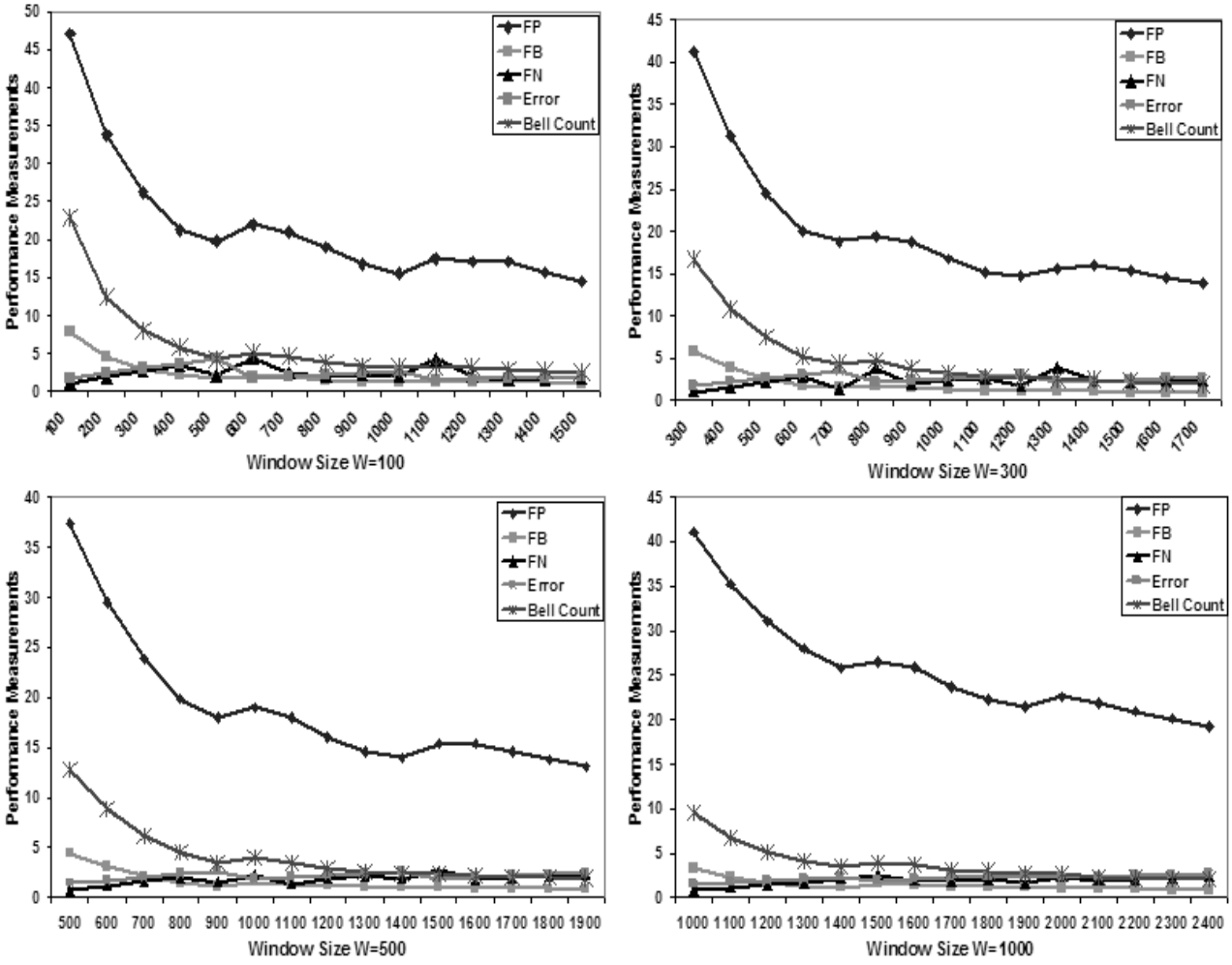


Fig. 5.5. Upper left figure shows the error rates obtained for $W = 100$; upper right shows the error rates obtained for $W = 300$; lower left shows the error rates obtained for $W = 500$, and lower right figure shows the error rates obtained for $W = 1000$ when $m \in [1 : 2 : 29]$ for all 61 users with majority vote as the smoothing function.

5.3.1 Discussion

Our empirical results showed that a *smoothed* classifier constructed over a smaller window size could produce better accuracy measurements than an *unsmoothed* classifier constructed over a larger window size. We explained this outcome with a finer granularity of user behavior that was more accurately captured over m windows of

smaller W than over a single window of $m * W$ data points. Namely, a window of $m * W$ data points averaged-out finer behavioral patterns of users to a greater extent than a window of W. As a result, inherent patterns of user behavior were better preserved when smoothing m smaller–sized windows.

Smoothing over several smaller windows was also more computationally efficient, because at each step (in our experiments each step was 50 data points) we needed to compute feature values over a window of a smaller size. One could use incremental updates [122] to improve the computational efficiency of larger window sizes, but with small window sizes such was $W = 100$ in our experiments and a window–overlap of 50 data points, computational efficiency of larger windows remained inferior.

Apparent trade–off between the overall accuracy and the detection time became non–negligible when we applied smoothing as shown in Figure 5.5. A solution to this problem should be found in an operational setting. We computed that for our user re–authentication system the detection time of a smoothed output was approximately 2.20 minutes (if user utilized I/O devices).

5.4 Summary and Conclusions

In this chapter we investigated the performance of the user re–authentication system when the amount of data was limited per user. The results obtained suggested that smoothing m classification instances each obtained over a window of W data points outperformed an unsmoothed classification of $m * W$ data points. We explained this phenomenon by a finer level of granularity of user behavior captured over smaller W and then averaged over m such instances than over a single big window of $m * W$ data points. The highlighted values in Tables 5.2 to 5.5 support this conclusion.

We also examined the performance of the system as m grew large and concluded that there was an inherent trade–off between the overall accuracy and the detection time, which could only be resolved in an operational setting.

6. USER RE-AUTHENTICATION WITH THE COMBINED DATA SOURCE

In this chapter we used the combined data source to determine:

1. The performance of the combined classifier on a previously *unseen* user dataset;
2. The ability to track a valid user across different computer and I/O configurations;
3. The degree of distinguishability among users when they were given an identical task to perform;
4. Whether the approach was scalable, i.e., did the accuracy measurement remain fairly consistent as the number of users increased; and
5. The computational efficiency of our user re–authentication system.

6.1 Detecting Previously Unseen Intruders

Empirical results obtained in Chapter 4 established the strength of the combined data source when trained on *all* N users, where $N = 61$ in our dataset. The results obtained demonstrated the ability of our system to detect an insider pretending to be another insider. In this section we wish to concentrate on a scenario where one cannot know the behavior of all users in advance (e.g., temporary workers or visitors) and therefore want to discriminate between whether it is or is not the valid user. Specifically, we investigated if we could detect an outsider pretending to be an insider. Consider the following two cases:

1. We had a 3-user dataset (Alice, Bob and Carol). We built a classifier C_1 to discriminate Alice from Bob. We wished to use C_1 to discriminate Alice from Carol.

2. We had an N–user dataset, where N is large. We built a classifier C_2 to discriminate Alice from Bob, Dave, Ed and potentially hundreds of other users. We wished to use C_2 to discriminate Alice from Carol.

We conjectured that the classifier C_2 would produce lower error rates when discriminating Alice from Carol than C_1. To validate our conjecture, we trained a biometric classifier on a large number of "seen" intruders and tested it's accuracy on an "unseen" intruder (i.e., an unlabeled sample). We investigated the performance of the classifier when all but one randomly selected user dataset (let's refer to this dataset as I) was seen in the training phase and a profile of normal user behavior for each user was built by observing the valid user's behavior and behavior of the remaining $N - 2$ intruders. We used I's dataset to compute the false negative rate during testing.

6.1.1 Experimental Methodology

For the experiments in this section we built a classifier from the combined data source from the 61–user dataset described in Section 4.2.2. We reported the results for the basic and smoothing implementation schemes in each experiment. We used the majority vote as a smoothing filter function with $m \in [1, 11]$. We computed each classification instance over a window of $W = 500$ data points and we overlapped windows so that a new window was obtained from the previous 450 and subsequent 50 points. As a result, a classification was carried out every fifty points following the classification of the first instance. To evaluate our methods we used a ten–fold cross–validation (CV). In each of the ten CV runs 90% of the user dataset was used for training and the remaining 10% was used for testing. The results reported were averaged over the inner eight CV folds.

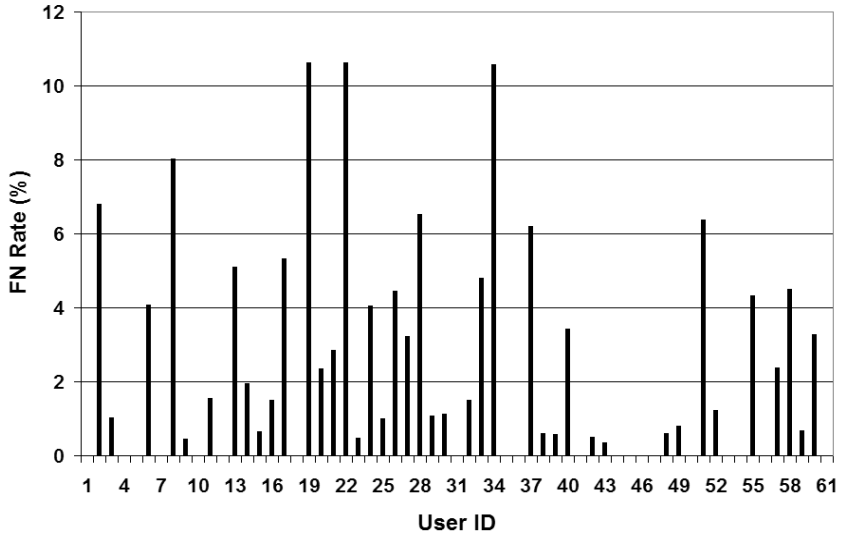


Fig. 6.1. The average FN rates on an unseen intruder for each profiled user in the 61-user dataset in the Detecting Previously Unseen Intruders experiment.

Table 6.1
The average FN rates in the Detecting Previously Unseen Intruders experiment.

Error Rate	**Basic**	**Smoothing**	**TTA**
FN Rate on an *Unseen* Intruder ($N = 61$)	0.98%	2.25%	49.3 seconds
FN Rate on an *Unseen* Intruder ($N = 3$)	51.94%	63.28%	48.5 seconds
FN Rate on $N - 1$ *Seen* Users	0.54%	1.50%	50.0 seconds

6.1.2 Empirical Analysis

The results obtained by the smoothed implementation scheme are shown in Figure 6.1. The average false negative rate and time to alarm over all 61 users was 2.25% and 49.3 seconds, respectively. Table 6.1 shows the time to alarm and the average false negative rate on 1) an unseen intruder from the 61–user dataset; 2)

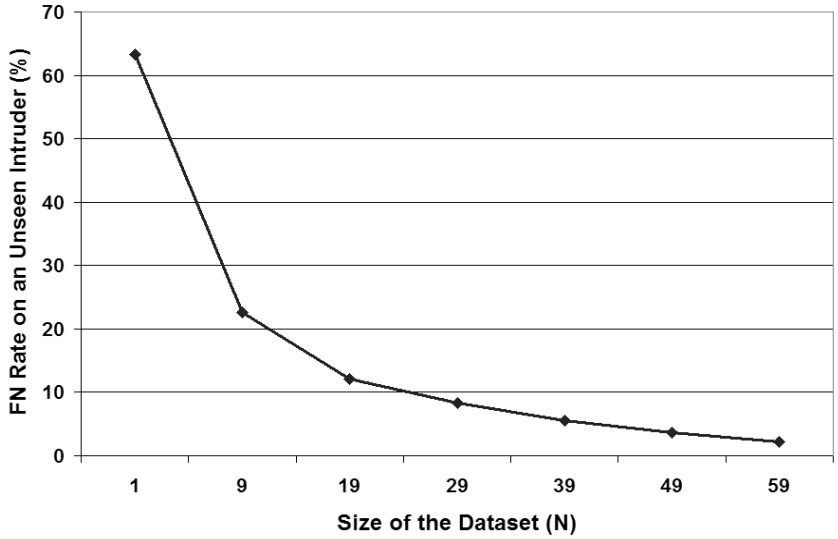


Fig. 6.2. The average FN rates as the number of seen intruders increased from 1 to 59 in the Detecting Previously Unseen Intruders experiment.

an unseen intruder from 3–user datasets;[1] and 3) a seen intruder from the Anomaly Detection experiment. The average false negative rates on an unseen intruder were higher than the false negative rate obtained in the Anomaly Detection experiment (compare the second and third with the fourth row of Table 6.1) which was to be expected considering that this intruder's dataset was not used to train the classifier in this experiment. The average false negative rate on an unseen intruder, $N = 61$, was significantly lower than the false negative rate on an unseen intruder, $N = 3$, (compare the second with the third row of Table 6.1) which validated our hypothesis that the classifier C_2 would indeed produce lower error rates when discriminating Alice from Carol than C_1.

We also determined the false negative rates on an unseen intruder as the number of seen intruders increased from 1 to 59. These results are shown in Figure 6.2. We observed that as the number of seen intruders increased the false negative rate

[1]We repeated this experiment on 61 3-user datasets to compute the average false negative rate for $N = 3$.

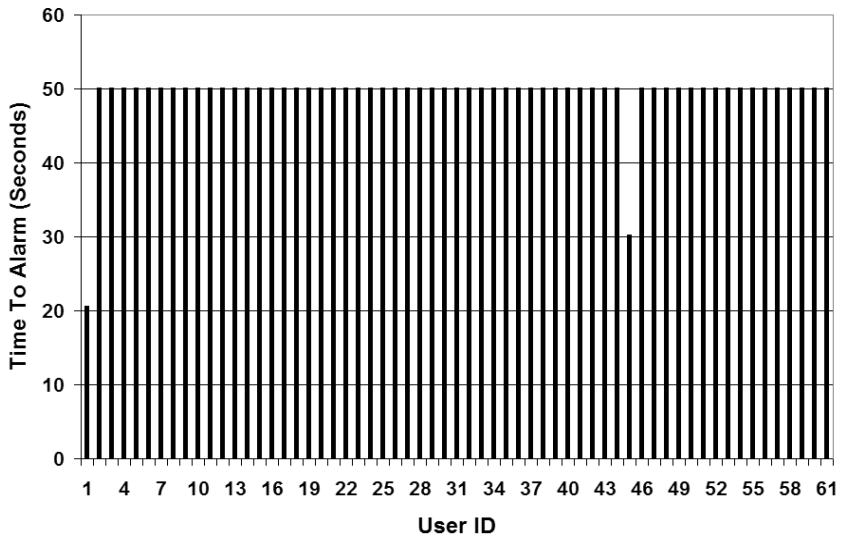


Fig. 6.3. The average TTA values for each user in the 61-user dataset in the Detecting Previously Unseen Intruders experiment.

decreased. This was not surprising considering that we measured the ability of a classifier to detect a majority class (i.e., the intruder's class) and as the majority class became more predominant the classifier produced lower false negative rates.

We next measured the amount of time it took a classifier to signal an alarm when an intrusion occurred. The TTA results obtained by the smoothed implementation scheme are shown in Figure 6.3. The average TTA values were 5.0 and 49.3 seconds for the basic and smoothing implementation schemes, respectively. We concluded that the biometric classifier could accurately detect even a previously unseen intruder with the false negative rate of 2.25% and the detection time of 49.3 seconds.

6.2 Tracking a Valid User

In Chapter 4 we introduced our 61–user dataset collected by a group of volunteers who used the Dell workstation computers. In this section, we investigated to what extent a profile of normal user behavior was dependent on the computer and I/O configuration. Specifically, we determined if we could correctly identify a valid user

as he/she was using different I/O devices and going from a desktop to a laptop computer. For each pair of computer and I/O configurations we built a binary classifier to determine if we could track a valid user across different configurations.

6.2.1 Data Set II

We asked one user to repeat the data collection process on five different computer and I/O configurations. This user was given the same reading assignment followed by a set of twenty questions. She was also given the same set of web pages to look at and answer yet another set of questions. She had no other behavioral impositions placed on her other than the specific task. The entire process lasted 7.28 hours on average; the standard deviation was 10.48 hours. The average number of raw data points for this user was 284,223 with the standard deviation of 286,604. We showed that for this user, a separate profile of normal user behavior should have been built for each configuration of computer and I/O devices. Additional users' datasets were needed to test the extent of a user profile's dependence on the computer and I/O configurations.

The following computer and I/O configurations were used to collect the data:

1. **Computer:** Laptop, 1.66MHz CPU, 1GB RAM, 80GB hard drive, Windows XP OS, Acer 2006; **Mouse:** Touchpad on a laptop computer, Acer 2006; **Keyboard:** 92–key laptop keyboard with media and web access keys, Acer 2006.

2. **Computer:** Desktop, 3.2MHz CPU, 2GB RAM, 250GB hard drive, Windows XP OS, Dell 2004; **Mouse:** Optical, ergonomic, PS–2 connected wireless mouse, Microsoft 2003; **Keyboard:** 119-key, ergonomic, USB connected wireless keyboard with media and web access keys, Microsoft 2003.

3. **Computer:** Desktop, 3.2MHz CPU, 2GB RAM, 250GB hard drive, Windows XP OS, Dell 2004; **Mouse:** Optical, USB connected mouse with media and web

access buttons, Dell 2004; **Keyboard:** 112-key, ergonomic, USB connected keyboard with web access keys, Dell 2004.

4. **Computer:** Desktop, 3.2MHz CPU, 2GB RAM, 250GB hard drive, Windows XP OS, Dell 2004; **Mouse:** Optical, USB connected mouse with media and web access buttons, Dell 2004; **Keyboard:** 105–key, USB connected keyboard, Macally 2002.

5. **Computer:** Desktop, 3.2MHz CPU, 2GB RAM, 250GB hard drive, Windows XP OS, Dell 2004; **Mouse:** Trackball, PS–2 connected mouse, Genius 1998; **Keyboard:** 105-key, PS–2 connected keyboard, Genius 1998.

6.2.2 Experimental Methodology

We reported the results for the basic and smoothing implementation schemes for a classifier constructed from the combined data source. We used the majority vote as a smoothing filter function with $m \in [1, 11]$. We computed each classification instance over a window of $W = 500$ data points and we overlapped windows every 50 data points. To evaluate our method we used a ten–fold cross–validation (CV). In each of the ten CV runs 90% of the user dataset was used for training and the remaining 10% was used for testing. The results reported were averaged over all ten CV folds.

6.2.3 Empirical Analysis

Table 6.2 shows results obtained by the smoothing implementation scheme. The eighth and ninth rows of Table 6.2 show the average FP and FN rates over all five configurations for the smoothing and basic implementation schemes, respectively. The results suggested that for a particular user, a separate profile should have been built for each configuration of computer and I/O devices. The lowest error rates (i.e., the least successful identification of a user) were produced for the laptop configu-

Table 6.2
FP and FN rates in the Tracking a Valid User experiment.

Config. ID	**Config. 1**		**Config. 2**		**Config. 3**		**Config. 4**		**Config. 5**	
Rate (%)	FP	FN	FP	FN	FP	FN	FP	FN	FP	FN
Config. 1	50.0	50.0	0.9	4.3	3.2	11.2	4.7	0.6	4.0	10.1
Config. 2	4.3	1.7	50.0	50.0	6.3	9.8	7.0	12.8	3.6	1.5
Config. 3	7.2	4.6	5.6	8.3	50.0	50.0	20.4	5.2	9.8	6.3
Config. 4	0.5	6.7	7.9	11.0	2.6	26.8	50.0	50.0	1.7	6.4
Config. 5	5.8	8.3	0.8	3.7	1.7	23.2	2.6	2.9	50.0	50.0
Average	**13.6**	**14.3**	**13.0**	**15.5**	**12.8**	**24.2**	**17.0**	**14.3**	**13.8**	**14.9**
Basic	13.3	14.8	14.4	14.2	20.0	15.3	18.5	14.1	14.5	14.3

ration; and the highest error rates were produced for configurations 3 and 4 which had the computer and the mouse device in common, but configuration 3 had an ergonomic keyboard. Configurations 2 and 5 were the two most dissimilar desktop configurations and therefore the easiest ones to differentiate.

6.3 Anomaly Detection with Behavioral Constraints

In Chapter 4 we conducted our empirical analysis on the 61-user dataset which was obtained by a group of volunteers who were given a reading assignment followed by a set of twenty questions. The volunteers were also given a set of web pages to look at and answer yet another set of questions. They were asked to behave as they would normally in any other situation i.e., they had *no* behavioral limitations placed upon them other than the specified task.

Our goal in this section was to investigate the performance of a classifier when users were given an identical task to perform. Specifically, we wished to determine

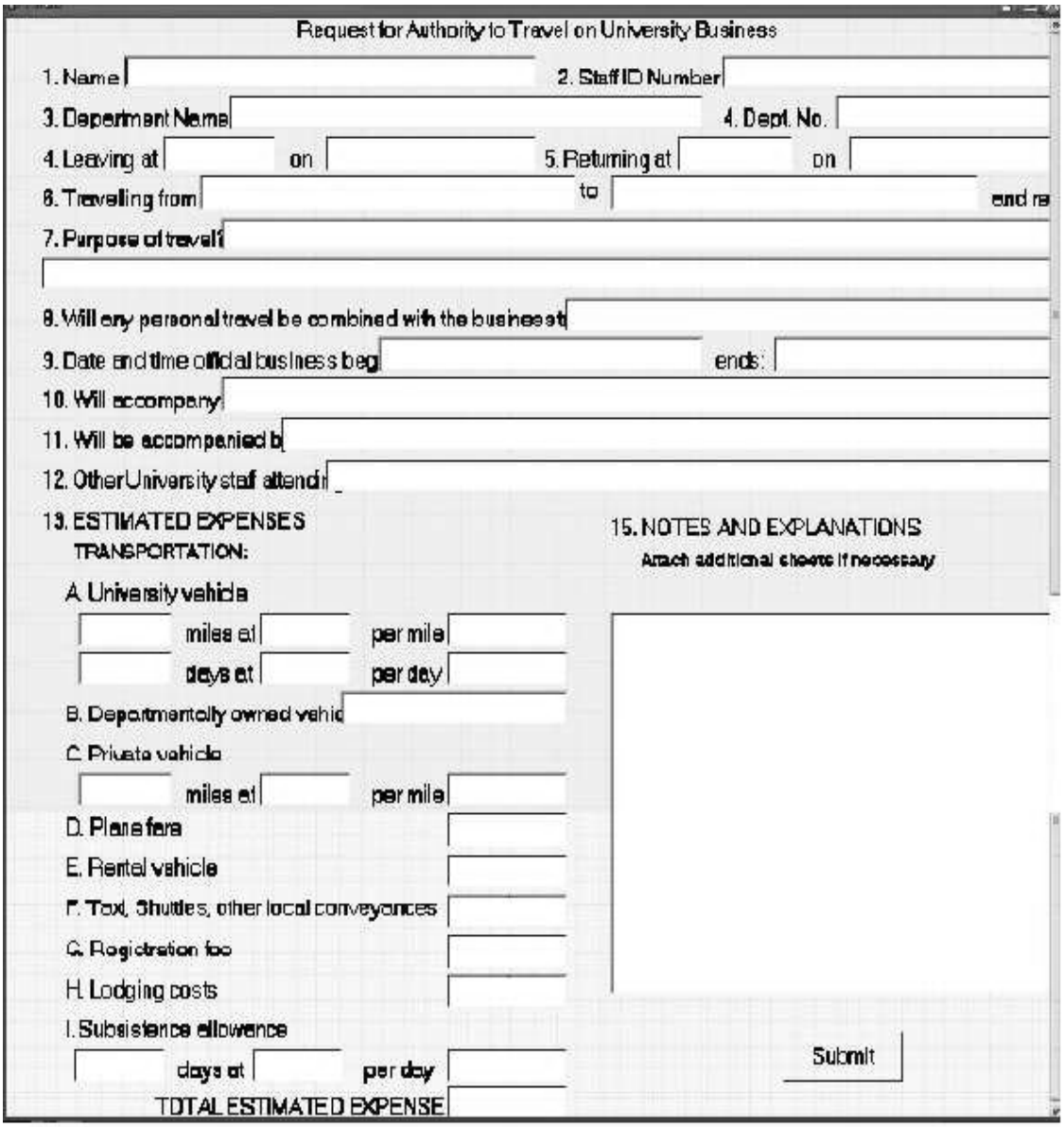


Fig. 6.4. An electronic copy of a travel expense form used to collect the 73 user dataset in the Anomaly Detection with Behavioral Constraints Experiment.

if a classifier could correctly identify a valid user and flag an intruder in a controlled environment where behavioral constraints were imposed on users. To accomplish our task, we created a computer program that collected users' keyboard and mouse input while they were filling out an electronic copy of a travel expense form shown

in Figure 6.4. They all filled in exactly the same information. We then repeated the Anomaly Detection experiment on the new dataset to determine if the binary biometric classifier could still build an accurate model of normal user behavior after seeing the behavior of a valid user A and the behavior of the remaining $N-1$ users in the role of intruders.

6.3.1 Data Set III

To test the accuracy of a classifier when users are executing the same task we collected the third dataset by a group of 73 volunteers. Our volunteers were graduate and undergraduate students and Graduate Office staff at Tufts University. Most of them were expert keyboard and mouse users. All users were asked to fill out an electronic copy of a travel expense form (one page in length) word-by–word from a template sheet that was distributed to each user in advance (see Figure 6.4).

During the data collection process typing mistakes were allowed. Users were instructed to use their mouse device when going from one field of the form to the next (instead of the "Tab" key) so we could record as many mouse movements as possible. The data collection process lasted 11.77 minutes on average; the standard deviation was 10.02 minutes. The average number of raw data points per user was 16,350 with a standard deviation of 16,428. In comparison to the 61-, 73–user dataset contained virtually no *pause* time, was much smaller in size and more dense with the I/O events.

6.3.2 Experimental Methodology

In this experiment we reported the results for the basic and smoothing implementation schemes for a classifier built from the combination of data sources (i.e., mouse, keystroke and GUI) from the 73–user dataset. We used the majority vote as a smoothing filter function with $m \in [1, 11]$. We computed each classification instance over a window of $W = 500$ data points and we overlapped windows every 50

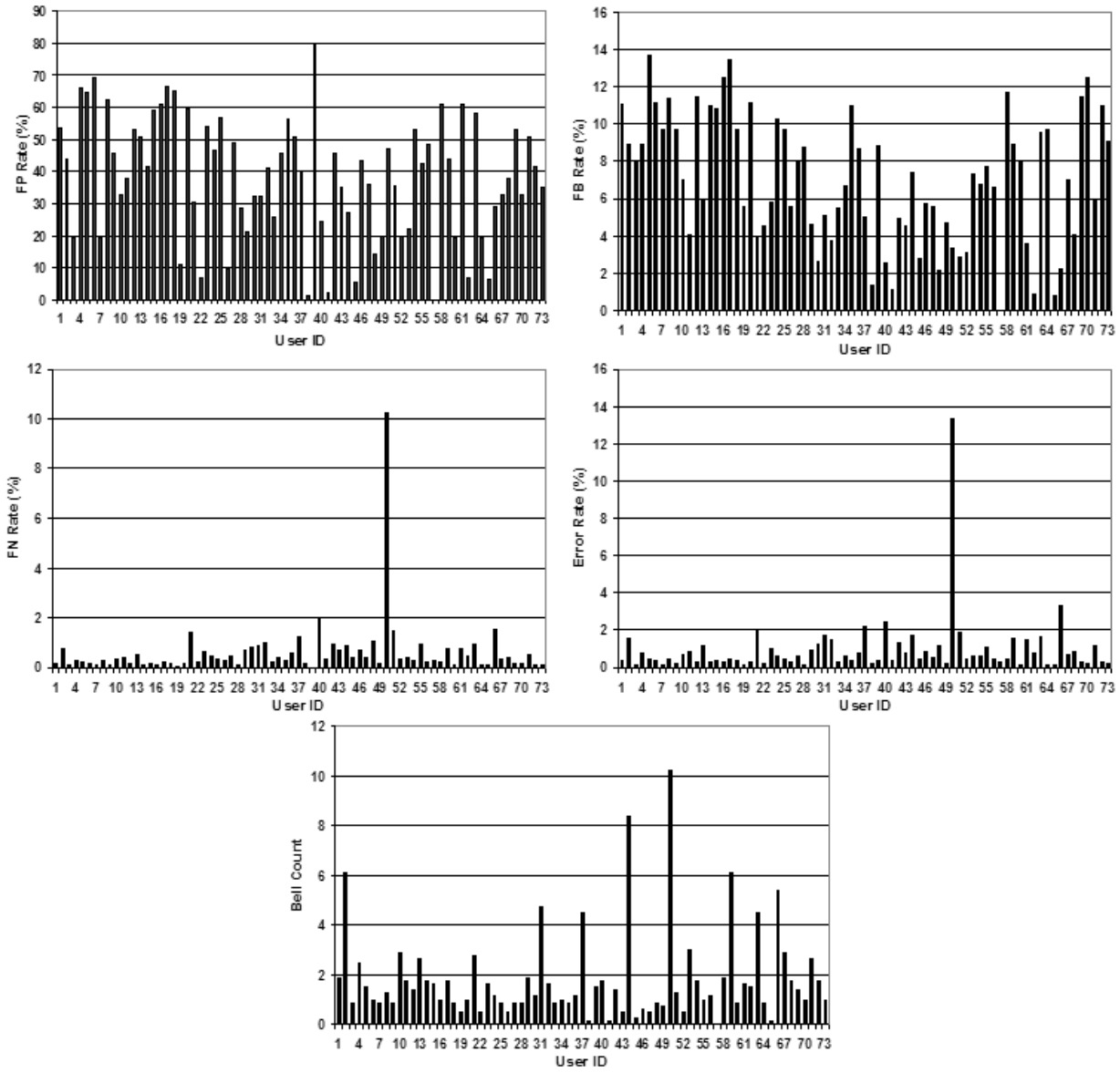


Fig. 6.5. The top left figure shows the false positive rates; the top right shows the false bell rates; center left shows the false negative rates; center right shows the error rates; and the bottom figure shows the bell count for each of the 73 users in the Anomaly Detection with Behavioral Constraints experiment.

data points. We also generated a ROC graph and reported the area under the ROC curve. To evaluate our method we used a ten–fold cross–validation (CV). In each of

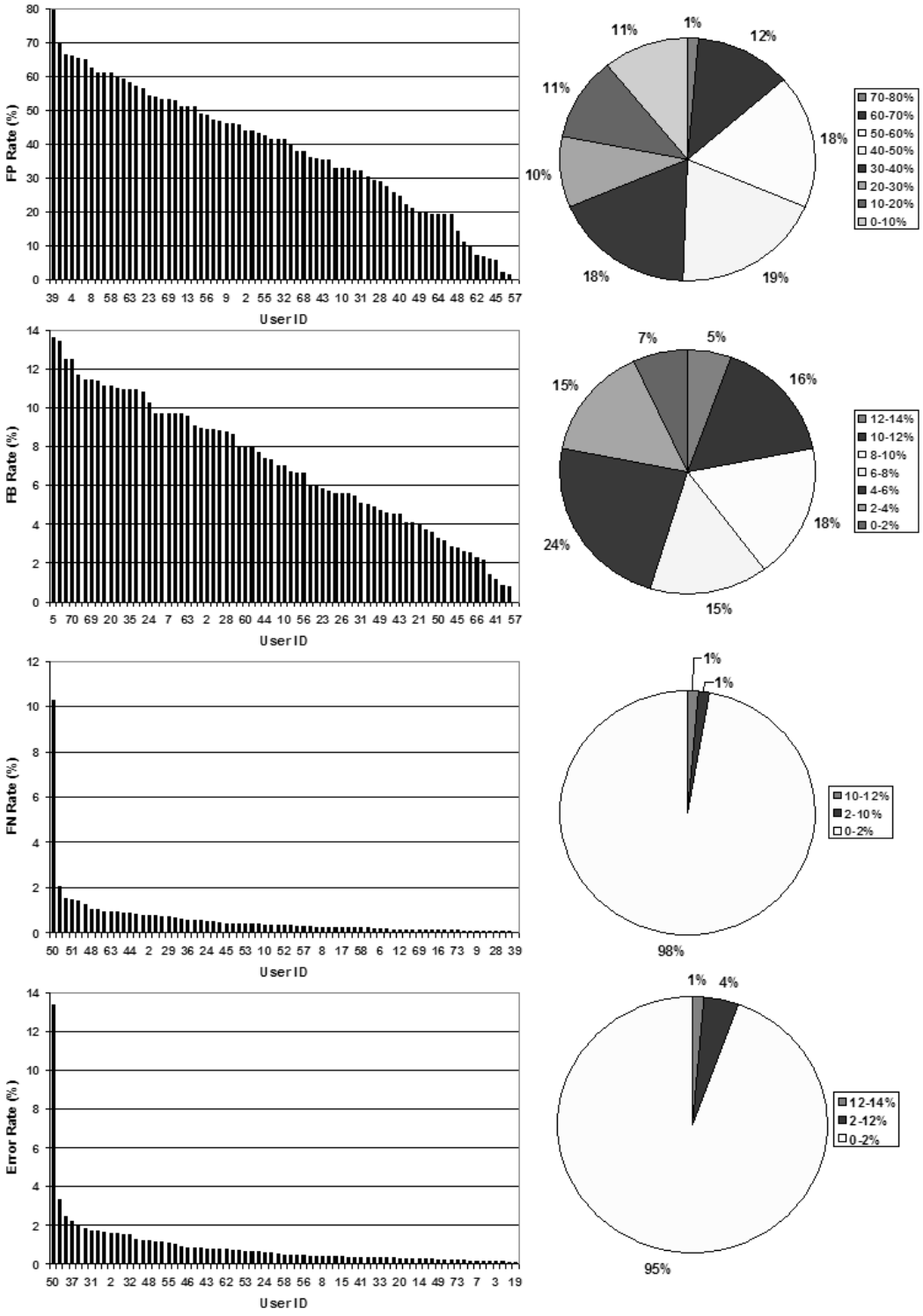


Fig. 6.6. The bar graphs show the error rates in the descending order for each of the 73 users. The pie charts show the percentage distribution of users across each error–rate range. From the top, the bar graphs and the corresponding pie charts representing FP, FB, FN and Error rates in the Anomaly Detection with Behavioral Constraints experiment are shown.

Table 6.3
The average and the standard deviation of FP, FB, FN and Error Rates and the Bell Count over all 73 users for the basic and smoothing implementation schemes in the Anomaly Detection with Behavioral Constraints experiment.

Scheme	FP Rate	FB Rate	FN Rate	Error Rate	Bell Count
Basic	48.99±17.80%	9.26±4.21	0.46±0.57	0.88±1.04	2.39±2.29
Smoothing	38.38±18.98%	6.98±3.45	0.59±1.22	0.92±1.61	1.81±1.82

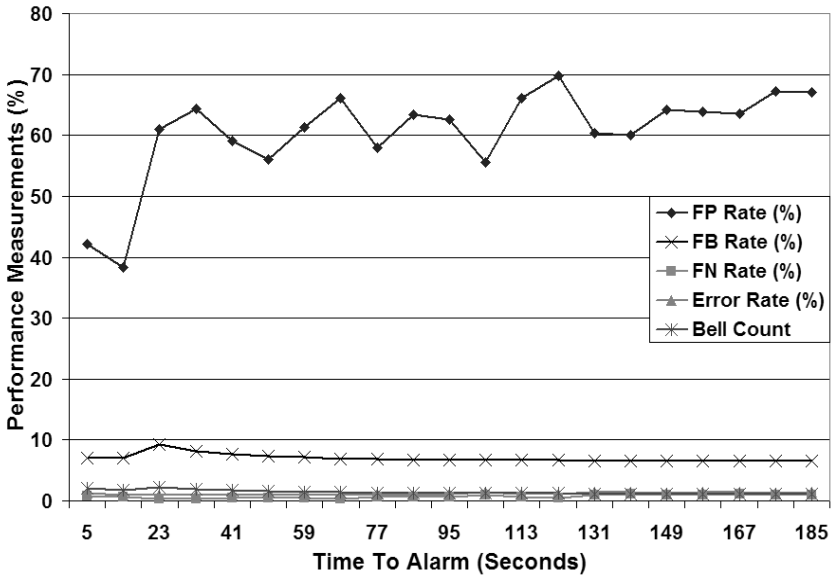


Fig. 6.7. Performance measurements versus the time to alarm in seconds for the combined data source in the Anomaly Detection with Behavioral Constraints experiment.

the ten CV runs 90% of the user dataset was used for training and the remaining 10% was used for testing. The results reported were averaged over all ten CV folds.

6.3.3 Empirical Analysis

The obtained results are shown in Figure 6.5. Figure 6.6 shows the results from Figure 6.5 in a rate–descending order. The bar graphs of Figure 6.6 show the false positive, false bell, false negative and error rates, respectively, in the descending order for each of the 73 users. The corresponding pie charts of Figure 6.6 show the percentage distribution of users across each respective error–rate range. Table 6.3 summarizes the results for the basic and smoothing implementation schemes averaged over all 73 users. The smoothed implementation scheme produced the average false positive, false bell, false negative and error rates of 38.38%, 6.98%, 0.59% and 0.92%, respectively, and a bell count of 1.81 bells. Figure 6.7 shows the error rates as the time to alarm increased from 5 to 185 seconds. The lowest error rates were obtained when the detection time was 14 seconds and the results were smoothed over $m = 3$ classification instances.[2] The ROC curve is shown in Figure 6.8. The area under the ROC curve was 0.68.

In comparison to the Anomaly Detection experiment from Section 4.3.2 this experiment yielded higher false positive and false bell rates and lower false negative and error rates and the bell count. The reason for this was a significantly shorter training time (10.6 minutes as opposed to 3.7 hours). Such a short training time did not provide the classifier with enough data to build a more accurate model of normal user behavior. More *active* data per user dataset as needed to improve the overall accuracy. This was particularly true because the number of intruders to be discriminated against increased. Considering the same amount of training time it was interesting that the false positive rate was at 38.38% and not higher. One possible explanation for this phenomenon was that the 73-user dataset was *denser* with the data points than the 61-user dataset. Namely, the density of the 73-user dataset was 1389.12 raw data points per minute which was nearly four times higher than the density of the 61–user dataset at 374.26 raw data points per minute.

[2]Higher values of m increased the average error rates, because some user datasets were small.

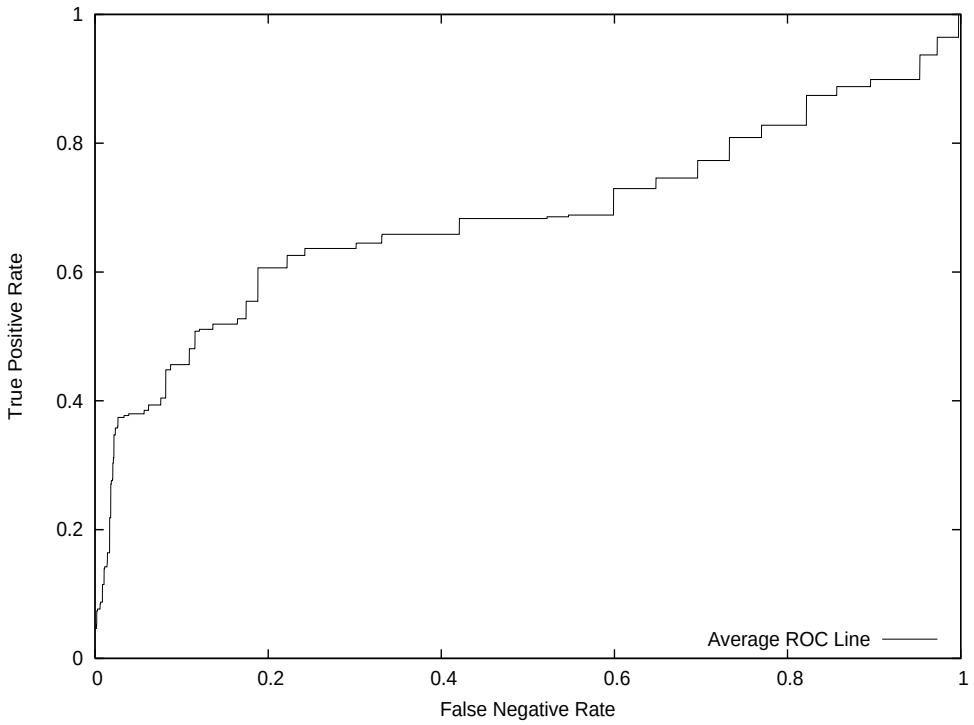


Fig. 6.8. ROC curve for the 73–user dataset in the Anomaly Detection with Behavioral Constraints experiment.

Empirical results obtained in this experiment led to the following conclusions:

1. It was possible to discriminate users even when they were executing the same task; and

2. The performance of the system improved with a) the amount of training data used to build a profile of normal user behavior and b) the density of each user's dataset in terms of the number of points collected per a particular time interval.

6.4 A Study of Scalability

In this section we investigated the scalability of our user re–authentication system. Specifically, we wished to determine the performance of the system as the

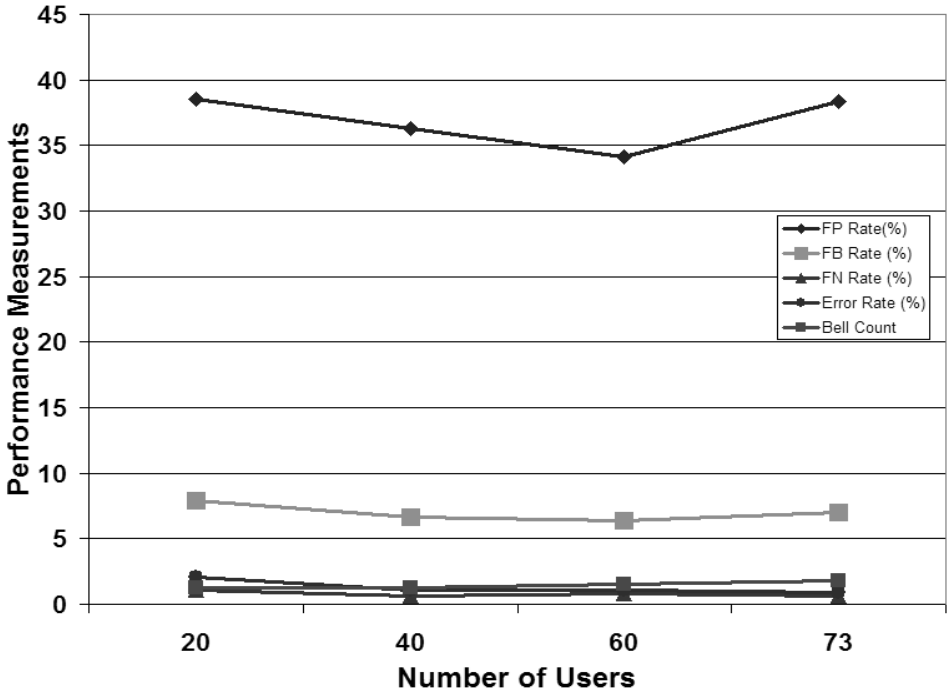


Fig. 6.9. Scalability of the system.

number of users, N, grows large. To this end, we used the larger of the two datasets (i.e., the 73–user dataset). We repeated the Anomaly Detection experiment on four subsets that we created by randomly picking 20, 40, 60 and 73 users. For each subset we computed the average false positive, false bell, false negative and error rates and a bell count over all 20, 40, 60 and 73 users, respectively. We then observed the tendencies of the error rates for each subset and drew conclusions about the system's scalability.

6.4.1 Experimental Methodology

In this experiment we reported the results for the basic and smoothing implementation schemes for a classifier constructed from the combined data source from the 73–user dataset. We used the majority vote as a smoothing filter function with $m \in [1, 11]$. We computed each classification instance over a window of $W = 500$ data

Table 6.4
The average and the standard deviation of FP, FB, FN and Error Rates and the Bell Count over 20, 40, 60 and 73–user subsets for the basic and smoothing implementation schemes in the Scalability Experiment.

Scheme	**FP Rate**	**FB Rate**	**FN Rate**	**Error Rate**	**Bell Count**
Basic	45.84±4.89%	8.98±0.76	0.70±0.28	1.38±0.55	2.04±0.51
Smoothing	36.83±2.09%	6.98±0.63	0.77±0.23	1.29±0.55	1.46±0.26

points and we overlapped windows every 50 data points. To evaluate our method we used a ten–fold cross–validation (CV). In each of the ten CV runs 90% of the user dataset was used for training and the remaining 10% was used for testing. The results reported were averaged over all ten CV folds.

6.4.2 Empirical Analysis

Table 6.4 summarizes the results. For the smoothed implementation scheme, Figure 6.9 shows that all of the error rates and the Bell Count displayed consistent and nearly constant tendencies as the number of users increased from 20 to 40 to 60 to 73. For the dataset that we had we concluded that our implementation of a user re–authentication system was scalable. However, to truly test the scalability of a system hundreds and even thousands of user datasets are needed.

6.5 Computational Efficiency

Finally, we discussed the computational efficiency of a user re-authentication system. We considered both the training time and the runtime overhead of monitoring user behavior.

The training time overhead for the decision trees was $n * m * log_2(m)$, where n was the number of classification instances and m was the number of features [108]. In practice, initial training is carried out in an off–line environment. (Note that incremental decision tree algorithms [123, 124] exist for online approaches, but we do not investigate them here.) The *runtime* overhead of monitoring consisted of collecting the data points as they were induced by the user, computing a feature vector of those features present in the decision tree model of that user, traversing the tree to determine the class of the current feature vector instance and applying the smoothing filter to reach the final decision.[3]

Let the total number of API events (i.e., system calls) that we were monitoring be p (p was known to be 161 - see Chapter 3), let the number of unique features in the user's profile be q and let the size of a decision tree be r, then the runtime of our system could be expressed as $O(c_1 * p + c_2 * q + \log_2(r) + c_3)$; $c_1, c_2, c_3 \in R^+$ where c_1, c_2 and c_3 represented the CPU time to process an API call, compute a feature and perform smoothing, respectively. We did not measure c_1, c_2 and c_3 because they were machine–specific.[4] Empirical results from the Anomaly Detection experiment in Section 4.3.2 produced $q = 150$ and $r = 372$ for the average number of features in the user's profile and the size of the decision tree (excluding leaves), respectively. Therefore, the runtime bottleneck of our system was the computation $c_2 * q$. To mediate this problem one could implement parallel processing because of the features' computational independence, but a detailed discussion of this issue is beyond the scope of this thesis.

6.6 Summary and Conclusions

In this chapter we investigated 1) the performance of a classifier on an unseen intruder dataset, 2) the ability to track a valid user across different computer and

[3]Please note that there could also be a network delay if the user's profile is stored on a secure server.

[4]Feedback from the users who participated in the data collection indicated that the entire process was unobtrusive and transparent.

I/O configurations, 3) the degree of distinguishability among different users when they are given an identical task to perform, 4) the scalability, and 5) the computational efficiency of our user re–authentication system. We concluded that a classifier constructed from the combined data source from the 61–user dataset could detect a previously unseen intruder with a false negative rate of 2.25% and a detection time of 49.3 seconds.

To examine the ability of our system to track a valid user, we asked one user to collect the data on five different hardware configurations. We showed that for this user, a separate profile of normal user behavior should have been built for each configuration of computer and I/O devices. We concluded that additional users' datasets were needed to fully test the extent of a user profile's dependence on the computer and I/O configurations.

To determine if a classifier could correctly identify a valid user and detect an intruder in a controlled environment where behavioral constraints were imposed on users we collected the second dataset from 73 volunteers who were asked to fill out an electronic copy of a travel expense sheet word-by–word from a template. In comparison to the 61-, 73–user dataset contained virtually no *pause* time, was much smaller in size and more dense with the I/O events. The experiment produced the average false positive, false bell, false negative and error rates of 38.38%, 6.98%, 0.59% and 0.92%, respectively, and a bell count of 1.81 bells. The detection time was 14 seconds. We concluded that 1) it was possible to discriminate users even when they were behaving in a nearly identical manner, and 2) the performance of the system improved with a) the amount of training data used to build a profile of normal user behavior and b) the density of each user's dataset in terms of the number of points collected per a particular time interval.

We investigated the scalability of our system on the 73–user dataset. We created four subsets by randomly picking 20, 40, 60 and 73 users and we computed the average false positive, false bell, false negative and error rates and a bell count for each subset. We then observed the tendencies of the error rates for each subset as

the number of users increased from 20 to 40 to 60 to 73. For the 73–user dataset we concluded that our implementation of a user re–authentication system was scalable.

Finally, we examined the computational efficiency of a user re–authentication system and concluded that the computation of features present in the profile of a valid user was the run-time bottleneck of the system. We suggested implementing parallel processing because of the features' computational independence as one possible solution to this problem.

7. SUMMARY AND SIGNIFICANCE

This dissertation examined a user re–authentication system via behavioral biometrics. The underlying hypothesis was that one could successfully model user behavior on the basis of the user's inputs and GUI changes as a response to those inputs. In particular, a model of normal user behavior was derived from the mechanics of the user's mouse movements, keystroke dynamics and GUI events. The proposed user re–authentication system was designed to either raise an alarm or alert a system administrator when the current behavior of user U, deviated sufficiently from learned "normal" behavior of user U.

7.1 Summary of Findings

The development and design of a state–of–the–art user re–authentication system was our primary objective. In Chapter 1 we conjectured and in the remaining chapters we presented empirical evidence from the real–world datasets that the proposed user re–authentication system was able to accomplish the following:

1. Detect insiders pretending to be other insiders (Section 4.3.2);
2. Detect outsiders pretending to be insiders (Section 6.1);
3. Discriminate users in a pair–wise sense (Section 4.3.1);
4. Determine the sensitivity of user profiles on different hardware configurations (Section 6.2);
5. Discriminate users when they were behaving in an identical manner (Section 6.3);

6. Determine the degree of system's scalability and computational efficiency (Sections 6.4 and 6.5);

7. Determine the strength of each data source (e.g., the mouse, keystrokes and GUI) individually and in combination (Sections 4.3.1 and 4.3.2);

8. Exploit granularity of the data to obtain a comprehensive feature space (Section 4.3.4);

9. Reduce the candidate feature space to a subset of most predictive features (Section 4.3.3); and

10. Improve the accuracy measure when the amount of data per user dataset was limited (Section 5.3);

The greatest challenge we faced was finding an acceptable balance among the performance criteria: 1) accuracy, 2) computational efficiency and 3) scalability. As we designed our system, we discovered both its strengths and weaknesses. It was our hope that future researchers in this area would find our results constructive and helpful. In the remainder of this section we describe our datasets and also some data–specific findings that influenced the results obtained.

We began our task with the choice of data sources to be used to model the behavior of a human being. At the start, we observed that data sources such as system and library calls and network packets were not the optimal choices for our task and we turned our attention to user–induced biometric information. We collected syntactical data from mouse and keyboard devices and we added a graphical level of interpretation to this data by collecting the GUI events. We collected three datasets: 1) a 61–user dataset obtained from a group of users who had *no* behavioral limitations placed upon them other than the specified task (Section 4.2.2); 2) a 5–user dataset obtained from a single user who was asked to repeat the original data collection task on five different computer and I/O configurations (Section 6.2.1); and 3) a 73–user dataset obtained in a controlled environment from a group of users who

were instructed to behave in an identical manner (Section 6.3.1). We next review our findings as they pertain to the feature space obtained, algorithms applied and experimental settings implemented:

Feature space: (Section 3.3) For each source of data (i.e., mouse, keystrokes and GUI events) we built a hierarchical structure to capture finer granularity of each user's mouse, keystroke and GUI behavior. We found that:

- Grouping data into a hierarchy improved the overall accuracy, but it also increased the time it took to generate a profile of valid user behavior (Section 4.3.4).
- Feature space reduction to a subset of most uncorrelated features improved the computational efficiency of the system without degrading the accuracy (Section 4.3.3).
- A subset of the ten most discriminatory features for the keystroke, mouse, GUI and combined data source was largely computed from the *leaves* of each feature hierarchy (Section 4.3.2).

Algorithm: (Section 4.2.1) We assumed a closed–setting scenario and applied a supervised learning algorithm to build a profile of normal user behavior. We found that:

- Support vector machines (SVM) algorithms produced high error rates on our data, because of the highly skewed class distributions and redundant features (Section 4.2.1).
- Decision tree algorithms produced a higher accuracy measurement than SVMs, because of the built–in feature subset selection algorithm (Section 4.2.1).

Experimental setting: (Section 4.3) To determine the accuracy measurement of the proposed user re–authentication system we conducted the following experiments:

- **Pairwise Discrimination:** A binary classifier built to discriminate between each pair of users indicated that there was indeed a signal of "normalcy" per user on the 61–user dataset. A strong quantitative imbalance[1] among keystroke, mouse and GUI data influenced the ranking of the data sources by accuracy: 1) GUI, 2) mouse and 3) keystrokes (Section 4.3.1).
- **Anomaly Detection:** A classifier built to detect an insider pretending to be another insider produced a model of normal user behavior of varying degrees of accuracy depending on the data source. The ranking of the data sources by accuracy on the 61–user dataset was: 1) GUI, 2) mouse and 3) keystrokes (Section 4.3.2). The combined data source produced the most favorable outcome (Section 4.3.2). A subset of users who utilized their keyboard and/or their mouse device produced an improved accuracy measurement. Results on the 73–user dataset indicated that it was possible to discriminate users even when they were behaving in an identical manner (Section 6.3). The tendency of error rates was measured to be consistent and nearly constant as the number of users increased from 20 to 40 to 60 to 73 (Section 6.4.2).
- **Unseen User Detection:** A binary classifier built to detect an outsider pretending to be an insider produced a higher false negative rate than the classifier from the Anomaly Detection experiment. The false negative rates decreased as the number of seen intruders the classifier was constructed from increased (Section 6.1).

We concluded that the optimal balance among accuracy, computational efficiency and scalability was deployment specific and needed to be tailored in accordance with the security policy of a particular operational setting. Faced with a trade–off between accuracy and efficiency on one end and scalability on the other we chose the former. Our implementation of a user re–authentication system achieved a false positive rate

[1]The average number of raw data instances in the 61–user dataset was 67,700.25, 11,510.54 and 1674.06 for the GUI, mouse and keystroke data, respectively.

of 23.37% and a false negative rate of 1.50% (the detection time was 50 seconds) when evaluated by a supervised learning classifier on the combined data source from the 61–user dataset. Our choice to implement a supervised learning method came at the expense of scalability, because a supervised learning classifier was not built to 1) detect unknown users and 2) discriminate users when the number of users was large. To address this problem, we conducted an experiment to determine if our supervised classifier was strong enough to detect a previously unseen intruder. The results were encouraging at the false negative rate of 2.25% and the detection time of 49.3 seconds. The tendency of the error rates tested as the number of users increased from 20 to 73 established that our implementation of a user re–authentication system was scalable on the 73–user dataset.

Faced with a trade–off between accuracy and detection time, we again chose the former. We were able to improve the overall accuracy of our system to a false positive rate of 14.47% and a false negative rate of 1.78% by applying a smoothing filter function to our original results. As a consequence, the detection time of our system increased from 50 seconds to 2.20 minutes.

7.1.1 Parameter Selection

We found that the proposed user re–authentication system had varying degrees of sensitivity to different parameters. We describe each parameter and its effect on the overall performance of the system:

Window size W: (Section 5.3) The performance of the system improved with the larger window size. The lowest error rates were produced for $W = 500$ data points.

Frequency k: (Section 3.3.7) The performance of the system was unaffected by the frequency parameter. We set $k = 8$ for the frequent events and $k = 1$ otherwise.

Smoothing parameter m: (Section 5.3) Smoothing over m smaller windows produced lower error rates than an unsmoothed larger window. However, the detection time increased with m.

Number of features q: (Section 6.5) The run–time bottleneck was the computation of features present in the model of a particular user. The run–time computational overhead increased with q.

Training time: (Section 4.3.2) The performance of the system improved with the length of the training time used to build a profile of normal user behavior. The 73–user dataset produced higher error rates than the 61–user dataset because of the smaller training dataset and a shorter training time (10.6 minutes as opposed to 3.7 hours) (Section 6.3).

Density of data: (Section 6.3) The performance of the system improved with the density of each user's dataset in terms of the number of points collected per a particular time interval. The system was able to build a more accurate model of normal user behavior when users produced more *active* data and utilized I/O devices (Section 4.3.2).

7.2 Future Directions

We consider two areas of future research: 1) implementing machine learning techniques such are boosting and cost functions to improve accuracy and 2) empirically evaluating the survivability (e.g., resilience to an attacker) of our system.

7.2.1 Boosting and Cost Functions

Machine learning provides means for achieving computer security objectives. As part of our future work we intend to investigate the performance impact of two machine learning techniques, boosting and cost functions, to user re–authentication.

The technique of boosting is a machine learning meta–algorithm for performing supervised learning [125]. Caruana and Niculescu–Mizil showed that boosted decision trees outperformed other supervised learning algorithms when applied to real–world datasets [126]. Boosting is a learning algorithm, which occurs in stages, by incrementally adding to the current learned function. At every stage a *weak* learner (i.e., one that has accuracy only slightly greater than chance) is trained with the data. The output of the weak learner is then added to the learned function, with some strength (proportional to how accurate the weak learner is). Then, the data is reweighted: examples that the current function gets wrong are "boosted" in importance, so that future weak learners will attempt to fix the errors [125].

Our datasets had highly skewed class distributions (i.e., the ratio between the minority and majority classes exceeded 1:60). In our implementation of a user re–authentication system we randomly sampled from the minority class to balance the disproportion between the classes. A more accurate method to achieve class–balance is by implementing cost functions. The cost functions assign a cost penalty to misclassifications [127]. Assignment of a high cost penalty to false positive classifications and a low cost penalty to false negative classifications would result in a lower false positive and a higher false negative rate. Our current implementation of a user re–authentication system produced a dominant false positive rate which might be re–balanced by the cost–sensitive classifier.

7.2.2 Subverting the System

A design of any computer security system is not complete without an investigation of operational conditions under which a particular system is considered "safe." Wagner and Soto investigated mimicry attacks on host based intrusion detection systems (HIDS) [96]. They showed that it was possible to generate a malicious sequence of system calls by *mimicking* normal system behavior and go undetected by the intrusion detection system. Our system was based on the mouse movements, keystroke

dynamics and GUI events and as such its space of possible outcomes surpassed that of an intrusion detection system built from the system calls data.

Nevertheless, any generative system can be reverse–engineered, which means that our user re–authentication system is also vulnerable to an attack. The question that we would like to answer in our future work is not whether the security of our system can be breached, but under what conditions can our system survive an attack. Specifically, what amount of knowledge an attacker needs to have to launch a successful and meaningful attack (e.g., the raw data files, collection and analysis algorithms and/or model of normal user behavior)?

7.3 Significance

Unlike *physical* biometrics systems which are obtrusive to the user or systems based on *tokens* and *SmartCards* that can be stolen or misplaced, the proposed system is founded on *behavioral* biometrics. The implementation of this system used mouse movements, keystroke dynamics and GUI events as behavioral biometrics to build a model of "normal" user behavior. The implemented system was characterized by its on-line use. It was designed in software and on–top of the existing hardware in such a manner that its presence did not disrupt or degrade standard operation of the computer. The high–confidence defense of the system was balanced by its scalability. Although our implementation of a user re–authentication system outperformed similar approaches, the system should be deployed in an operational setting with the defense–in–depth strategy in mind.

LIST OF REFERENCES

LIST OF REFERENCES

[1] Systems HTT. "Access control". www.htt.com, 2004.

[2] C. E. Landwehr. "Protecting unattended computers without software". In *Proceedings of the 13th Annual Computer Security Applications Conference*, pages 273–283, December 1997.

[3] Index Security. "Index security: Biometric fingerprint ID". www.index-security.com, 2004.

[4] IR Recognition. "Hand geometry technology". www.recogsys.com, 2004.

[5] Transcription Gear. "Voice recognition solutions". www.transcriptiongear.com, 2004.

[6] J. G. Daugman. "High confidence visual recognition of persons by a test of statistical independence". In *Proceedings of the IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 15, pages 1148–1161, November 1993.

[7] U.S. Biometric Consortium. "Face recognition". www.vitro.bloomington.in.us:8080/ BC, 2004.

[8] E. Spafford. "Security myths and passwords". http://www.cerias.purdue.edu/weblogs/spaf/general/post-30/, 2006.

[9] R. Gopalakrishna, E. H. Spafford, and J. Vitek. Efficient intrusion detection using automaton inlining. In *SP '05: Proceedings of the 2005 IEEE Symposium on Security and Privacy*, pages 18–31, Washington, DC, USA, 2005. IEEE Computer Society.

[10] D. Wagner and D. Dean. "Intrusion detection via static analysis". In *Proceedings of IEEE Symposium on Security and Privacy*, pages 156–169, 2001.

[11] J. T. Giffin, S. Jha, and B. P. Miller. "Detection manipulated remote call streams". In *Proceedings of the 11th USENIX Security Symposium*, pages 61–79, August 2002.

[12] R. Sekar, M. Bender, D. Dhurjati, and P. Bollineni. "A fast automation-based method for detecting anomalous program behaviors". In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 144–155, May 2001.

[13] R. Sekar and P. Uppuluri. "A fast automation-based method for detecting anomalous program behaviors". In *Proceedings of the 8th USENIX Security Symposium*, pages 63–78, August 1999.

[14] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. "A sense of self for UNIX processes". In *Proceedinges of the 1996 IEEE Symposium on Research in Security and Privacy*, pages 120–128. IEEE Computer Society Press, 1996.

[15] S. Forrest, A. S. Perelson, L. Allen, and R. Cherukuri. "Self-nonself discrimination in a computer". In *Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy*, pages 202–212, Oakland, CA, 1994. IEEE Computer Society Press.

[16] S. Forrest, S. A. Hofmeyr, and A. Somayaji. "Computer immunology". *Communications of the ACM*, 40(10):88–96, 1997.

[17] A. Wespi, M. Dacier, and H. Debar. "An intrusion-detection system based on the teiresias pattern-discovery algorithm". Technical Report RZ3103, Zurich Research Laboratory, IBM Research Division, 1999.

[18] A. Wespi, M. Dacier, and H. Debar. Intrusion detection using variable-length audit trail patterns. In *RAID '00: Proceedings of the Third International Workshop on Recent Advances in Intrusion Detection*, pages 110–129, London, UK, 2000. Springer-Verlag.

[19] C. Ko, G. Fink, and K. Levitt. "Automated detection of vulnerabilities in priviledged programs by execution monitoring". In *Proceedings of the Tenth Annual Computer Security Applications Conference*, pages 134–144, December 1994.

[20] S. N. Chari and P. C. Cheng. Bluebox: A policy-driven, host-based intrusion detection system. *ACM Trans. Inf. Syst. Secur.*, 6(2):173–200, 2003.

[21] H. H. Feng, O. M. Kolesnikov, P. Fogla, W. Lee, and W. Gong. "Anomaly detection using call stack information". In *Proceedings of IEEE Symposium on Security and Privacy*, pages 62–78, 2003.

[22] D. E. Denning and P. G. Neumann. "Requirements and model for IDES - A real-time intrusion detection system". Technical report, Computer Science Laboratory, SRI International, Menlo Park, CA, 1985.

[23] T. Lunt, J. van Horne, and L. Halme. "Automated analysis of computer system audit trails". In *Proceedings of the Ninth DOE Computer Security Group Conference*, Baltimore, MD, May 1986.

[24] T. F. Lunt. "Automated audit trail analysis and intrusion detection: A survey". In *Proceedings of the 11th National Computer Security Conference*, pages 65–73, Baltimore, MD, 1988.

[25] T. F. Lunt, R. Jagannathan, P. G. Neumann, H. S. Javitz, A. Valdes, R. Lee, S. Listgarten, and D. L. Edwards. "IDES: The enhanced prototype, A real-time intrusion detection system". Technical Report SRI-CSL-88-12, SRI Computer Science Laboratory, SRI International, 1988.

[26] T. F. Lunt, A. Tamaru, F. Gilham, R. Jagannathan, P. G. Neumann, H. S. Javitz, A. Valdes, and T. D. Garvey. "A real-time intrusion detection expert system IDES - Final report". Technical Report SRI-CSL-92-05, SRI Computer Science Laboratory, SRI International, February 1992.

[27] H. Javitz and A. Valdes. "The SRI IDES statistical anomaly detector". In *Proceedings of the 1991 IEEE Symposium on Research in Security and Privacy*, pages 316–327, Oakland, CA, 1991.

[28] J. R. Winkler and W. J. Page. "Intrusion and anomaly detection in trusted systems ". In *Proceedings of the 5th Annual Computer Security Applications Conference*, pages 39–45, 1989.

[29] H. S. Teng, K. Chen, and S. C. Lu. "Adaptive real-time anomaly detection using inductively generated sequential patterns". In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 278–284. IEEE Press, 1990.

[30] H. Debar, M. Becker, and D. Siboni. "A neural network component for an intrusion detection system". In *Proceedings of IEEE Symposium on Research in Computer Security and Privacy*, pages 240–250, 1992.

[31] A. Rapaka, A. Novokhodko, and D. Wunsch. "Intrusion detection using radial basis function network on sequences of system calls". In *Proceedings of the International Joint Conference*, volume 3, pages 1820–1825, 2003.

[32] N. Habra, B. L. Charlier, A. Mounji, and I. Mathieu. ASAX : Software architecture and rule- based language for universal audit trail analysis. In *European Symposium on Research in Computer Security (ESORICS)*, pages 435–450, 1992.

[33] P. Helman and G. Liepins. "Statistical foundations of audit trail analysis for the detection of computer misuse". *IEEE Transactions on Software Engineering*, 19(9):886–901, 1993.

[34] G. Vigna, S. T. Eckmann, and R. A. Kemmerer. "The stat tool suite". In *Proceedings DARPA Information Survivability Conference and Exposition*, volume 2, pages 46–55, 2000.

[35] N. Ye. "A Markov chain model of temporal behavior for anomaly detection". In *Proceedings of the 2000 IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop, 2000*, pages 171–174, 2000.

[36] N. Ye. "A scalable clustering technique for intrusion signature recognition". In *Proceedings of the 2000 IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop*, June 2001.

[37] W. Lee, S. J. Stolfo, and P. K. Chan. "Learning patterns from UNIX process execution traces for intrusion detection". In *Proceedings of the AAAI97 workshop on AI Approaches to Fraud Detection and Risk Management*, pages 50–56. AAAI Press, 1997.

[38] W. Lee, S. J. Stolfo, and K. W. Mok. "A data mining framework for building intrusion detection models". In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 120–132, 1999.

[39] W. Lee, S. J. Stolfo, P. K. Chan, E. Eskin, W. Fan, M. Miller, S. Hershkop, and J. Zhang. "Real time data mining-based intrusion detection". In *Proceedings of the Second DARPA Information Survivability Conference and Exposition*, pages 85–100, 2001.

[40] M. Hossain, S. M. Bridges, and R. B. Vaughn. "Adaptive intrusion detection with data mining". In *Proceedings of the IEEE International Conference on Systems Man and Cybernetics*, volume 4, pages 3097–3103, 2003.

[41] J. Shavlik, M. Shavlik, and M. Fahland. "Evaluating software sensors for actively profiling Windows 2000 users". In *Proceedings of the Fourth International Symposium on Recent Advances in Intrusion Detection*, October 2001.

[42] T. Lane and C. E. Brodley. "An application of machine learning to anomaly detection". In *Proceedings of the Twentieth NIST-NCSC National Information Systems Security Conference*, pages 366–380, 1997.

[43] T. Lane and C. E. Brodley. "Approaches to online learning and concept drift for user identification in computer security". In *Knowledge Discovery and Data Mining*, pages 259–263, 1998.

[44] T. Lane and C. E. Brodley. "Sequence matching and learning in anomaly detection for computer security". In Fawcett, Haimowitz, Provost, and Stolfo, editors, *AI Approaches to Fraud Detection and Risk Management*, pages 43–49. AAAI Press, 1997.

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

[46] T. Lane. *"Machine learning techniques for the computer security domain of anomaly detection"*. PhD thesis, CERIAS, Purdue University, West Lafayette, IN, August 2000.

[47] T. Lane and C. E. Brodley. "An empirical study of two approaches to sequence learning for anomaly detection". *Machine Learning*, 51(1):73–107, 2003.

[48] W. DuMouchel. "Computer intrusion detection based on bayes factors for comparing command transition probabilities". Technical report, National Institute of Statistical Sciences, 1991.

[49] M. Schonlau, W. DuMouchel, W.H. Ju, A. F. Karr, M. Theus, and Y. Vardi. Computer intrusion: Detecting masquerades. *Statistical Science*, 16(1):58–74, 2001.

[50] K. Sequeira and M. Zaki. "ADMIT: Anomaly-based data mining for intrusions". In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 386–395. ACM Press, 2002.

[51] T. Fawcett and F. Provost. Activity monitoring: Noticing interesting changes in behavior. In Chaudhuri and Madigan, editors, *Proceedings on the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 53–62, San Diego, CA, 1999.

[52] S. Coull, J. Branch, B. Szymanski, and E. Breimer. "Intrusion detection: A bioinformatics approach". In *Proceedings of the Nineteenth Annual Computer Security Applications Conference*, pages 24–34, Las Vegas, NE, 2003.

[53] F. Y. Leu and T. Y. Yang. "A host-based real-time intrusion detection system with data mining and forensic techniques". In *Proceedings of the 37th Internal Carnahan Conference on Security Technology*, October 2003.

[54] S. Furnell, J. Morrissey, P. Sanders, and C. Stockel. "Aplacation of keystroke analysis for improved login security and continuous user authentication". In *Proceedings of the Information and System Security Conference*, pages 283–294, 1996.

[55] P. Dowland, H. Singh, and S. Furnell. "A preliminary investigation of user authentication using continuous keystroke analysis". In *Proceedings of the Workshop Conference on Information Security Management and Small Systems Security*, 2001.

[56] P. Dowland, S. Furnell, and M. Papadaki. "Keystroke analysis as a method of advanced user authentication and response". In *Proceedings of the IFIP/SEC 17th International Conference on Information Security*, 2002.

[57] F. Bergadano, D. Gunetti, and C. Picardi. User authentication through keystroke dynamics. *ACM Trans. Inf. Syst. Secur.*, 5(4):367–397, 2002.

[58] F. Bergadano, D. Gunetti, and C. Picardi. Identity verification through dynamic keystroke analysis. *Intelligent Data Analysis*, 7(5):469–496, 2003.

[59] A. A. E. Ahmed and I. Traore. Detecting computer intrusions using behavioral biometrics. In *Proceedings of the 3rd Annual Conference on Privacy, Security and Trust*. ACM Press, October 2005.

[60] J. Goecks and J. Shavlik. "Automatically labeling web pages based on normal user actions". In *Procedings of the IJCAI Workshop on Machine Learning for Information Filtering*, July 1999.

[61] Z. Ghahramani. Unsupervised learning. *Advanced Lectures in Machine Learning: Lecture Notes in Computer Science*, 3176:72–112, 2004.

[62] L. A. Gordon, M. P. Loeb, W. Lucyshyn, and R. Richardson. "Computer crime and security survey". http://i.cmpnet.com/gocsi/db_area/pdfs/fbi/FBI2006.pdf, 2006.

[63] United States Secret Service. "National threat assessment center - insider threat study". www.asisonline.org/newsroom/crisisResponse/insiderthreat.pdf, August 2004.

[64] R. Joyce and G. Gupta. "User authorization based on keystroke latencies". *Journal of the ACM*, 33(2):168–176, 1990.

[65] Wikipedia. "Dictionary attack". en.wikipedia.org/wiki/, 2005.

[66] F. Monrose, M. K. Reiter, and S. Wetzel. Password hardening based on keystroke dynamics. In *CCS '99: Proceedings of the 6th ACM Conference on Computer and Communications Security*, pages 73–82, New York, NY, USA, 1999. ACM Press.

[67] B. Miller. "Vital signs of identity". *IEEE Spectrum*, 31(2):22–30, 1994.

[68] N. M. Herbst and C. N. Liu. "Automatic signature verification based on accelerometry". Technical report, IBM, 1977.

[69] K. Zimmermann and J. Werner. "SIRSYS-A program facility for handwriting signature analysis ". In *Procedings of the Carnahan Conference*, pages 153–155, May 1978.

[70] G. Bills and K. Zimmerman. Spectral analysis of right-handed versus left handed on-line script. In *Proceedings of the IEEE Conference on Frontiers of Engineering and Computing in Health Care*, September 1985.

[71] M. Eden. "Handwriting and pattern recognition". In *Procedings of the IEEE Transactions in Information Theory Conference*, volume 8, pages 160–166, February 1962.

[72] J. S. Lew. "Optimal accelerometer layouts for data recovery in signature verification ". *IBM J. Res. Develop.*, 24(4):496–511, 1980.

[73] C. Liu, N. Herbst, and J. Anthony. "Automatic signature verification: system description and field test results". *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):35–38, 1979.

[74] J. Verdenbregt and W. Koster. "Analysis and synthesis of handwriting ". *Philips Technical Review*, 32(3):73–78, 1971.

[75] R. Gaines, W. Lisowski, S. Press, and N. Shapiro. "Authentication by keystroke timing: Some preliminary results". Technical report, Rand Corporation, Santa Monica, CA, 1980.

[76] J. Leggett and G. Williams. "Verifying identity via keyboard charactersistics". *Journal of Man and Machine Studies*, 23(1):67–76, 1988.

[77] J. Garcia. "Personal identification aparatus". US Patent and Trademark Office, 1986.

[78] J. R. Young and R. W. Hammon. "Method and apparatus for verifying an individual's identity". US Patent and Trademark Office, 1989.

[79] S. Bleha, C. Slivinsky, and B. Hussein. "Computer-access security systems using keystroke dynamics". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12:1217–1222, 1990.

[80] M. Brown and S. J. Rogers. "User identification via keystroke characteristics of typed names using neural networks". *International Journal of Man and Machine Studies*, 39:999–1014, 1993.

[81] M. S. Obaidat and B. Sadoun. "A simulation evaluation study of neural network techniques to computer user identification". *Information Science*, 102:239–258, 1997.

[82] M. S. Obaidat and B. Sadoun. "Verification of computer users using keystroke dynamics". *IEEE Transactions on Systems, Man and Cybernetics*, 27(2):261–269, 1997.

[83] F. Monrose and A. Rubin. "Authentication via keystroke dynamics". In *Proceedings of the Fourth ACM Conference on Computer and Communications Security*, pages 48–56, April 1997.

[84] F. Monrose and A. Rubin. "Keystroke dynamics as a biometric for authentication". http://citeseer.nj.nec.com/monrose99keystroke.html, 1999.

[85] J. T. Tou and R. C. Gonzalez. *Pattern Recognition Principles*. Addison-Wesley, 1981.

[86] Y. Sang, H. Shen, and P. Fan. Novel impostors detection in keystroke dynamics by support vector machine. In K. M. Liew, H. Shen, S. See, W. Cai, P. Fan, and S. Horiguchi, editors, *PDCAT*, volume 3320 of *Lecture Notes in Computer Science*, pages 666–669. Springer, 2004.

[87] Y. Sang, H. Shen, and P. Fan. Keystroke characteristics identity authentication based on levenberg-marquardt algorithm. *Computer Applications*, 24(7):1–3, 2004.

[88] R. A. J. Everitt and P. W. McOwan. "Java-based internet biometric authentication systems". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(9):1166–1172, 2003.

[89] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison–Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[90] J. P. Anderson. "Computer security threat monitoring and surveillance". Technical report, James P. Anderson Co., Fort Washington, PA, 1980.

[91] R. Wright. "2003 CSI/FBI computer security survey". http://www.security.fsu.edu/docs/FBI2003.pdf, 2003.

[92] C. Warrender, S. Forrest, and B. A. Pearlmutter. "Detecting intrusions using system calls: Alternative data models". In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 133–145, 1999.

[93] I. Rigoutsos and A. Floratos. "Combinatorial pattern discovery in biological sequences". *Journal of Bioinformatics*, 14:55–67, 1998.

[94] R. A. Wagner and M. J. Fisher. "Tstring-to-string correction problem". *Journal of the ACM*, 21:168–173, 1974.

[95] M. Pusara and C. E. Brodley. User re-authentication via mouse movements. In *VizSEC/DMSEC '04: Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, pages 1–8, New York, NY, USA, 2004. ACM Press.

[96] D. Wagner and P. Soto. "Mimicry attacks on host based intrusion detection systems". In *Proceedings Ninth ACM Conference on Computer and Communications Security*, pages 255–264. ACM Press, 2002.

[97] R. M. Gray and L. D. Davisson. *An introduction to statistical signal processing*. Cambridge University Press, Cambridge, UK, January 2005.

[98] E. Gabrilovich and S. Markovitch. Text categorization with many redundant features: Using aggressive feature selection to make svms competitive with c4.5. In *Proceedings of The Twenty-First International Conference on Machine Learning*, pages 321–328, Banff, Alberta, Canada, 2004. Morgan Kaufmann.

[99] L. Tang and H. Liu. Bias analysis in text classification for highly skewed data. In *ICDM '05: Proceedings of the Fifth IEEE International Conference on Data Mining*, pages 781–784, Washington, DC, USA, 2005. IEEE Computer Society.

[100] H. Zhao and S. Ram. Combining schema and instance information for integrating heterogeneous data sources. *Data Knowl. Eng.*, 61(2):281–303, 2007.

[101] S. Aksoy, K. Koperski, C. Tusk, and G. Marchisio. Interactive training of advanced classifiers for mining remote sensing image archives. In *Proceedings of the 2004 ACM International Conference on Knowledge Discovery and Data Mining*, pages 773–782. ACM Press, August 2004.

[102] A. Kapoor, H. Ahn, and R. W. Picard. Mixture of gaussian processes for combining multiple modalities. In *Multiple Classifier Systems Workshop*, pages 86–96, 2005.

[103] E. Kim and J. Ko. Dynamic classifier integration method. In *Multiple Classifier Systems Workshop*, pages 97–107, 2005.

[104] R. Patenall, D. Windridge, and J. Kittler. Multiple classifier fusion performance in networked stochastic vector quantisers. In *Multiple Classifier Systems Workshop*, pages 128–135, 2005.

[105] C. Hsu and C. Lin. "A comparison of methods for multi-class support vector machines". *IEEE Transactions on Neural Networks*, 13:415–425, 2002.

[106] S. Axelsson. The base-rate fallacy and its implications for the difficulty of intrusion detection. In *Proceedings of the Sixth ACM Conference on Communications and Computer Security*, Kent Ridge Digital Labs, Singapore, November 1999.

[107] J. G. Dy and C. E. Brodley. Feature subset selection and order identification for unsupervised learning. In *ICML '00: Proceedings of the Seventeenth International Conference on Machine Learning*, pages 247–254, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[108] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, US, 2005.

[109] T. Joachims. *Learning to classify text using support vector machines*. Kluwer, 2002.

[110] C. C. Chang and C. J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[111] T. Joachims. Optimizing search engines using clickthrough data. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142, New York, NY, USA, 2002. ACM Press.

[112] G. Forman. "An extensive empirical study of feature selection metrics for text classification". *Journal of Machine Learning*, 3:1289–1305, 2003.

[113] T. Mitchell. *"Machine learning"*. McGraw-Hill Companies, Inc., United States of America, 1997.

[114] Y. Freund. Boosting a Weak Learning Algorithm by Majority. *Information and Computation*, 121(2):256–285, 1995.

[115] R. E. Schapire. A brief introduction to boosting. In *Proceedings of the 1999 International Joint Conference on Artificial Intelligence*, pages 1401–1406, 1999.

[116] Y. Yuan and M. J. Shaw. "Induction of fuzzy decision trees". In *Fuzzy Sets and Systems*, volume 69, pages 125–139, 1995.

[117] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.

[118] R. A. Fisher. *Statistical methods and scientific inference*. Oliver and Boyd, Edinburgh, UK, 1956.

[119] J. Catlett. Overpruning large decision trees. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 764–769, 1991.

[120] B. D. Davison and H. Hirsh. Predicting sequences of user actions. In *Predicting the Future: AI Approaches to Time-Series Problems*, pages 5–12, Madison, WI, July 1998. AAAI Press. Proceedings of AAAI–98/ICML–98 Workshop, published as Technical Report WS98–07.

[121] J. Avery. *Information theory and evolution*. World Scientific, 2003.

[122] B. P. Welford. "Note on a method for calculating corrected sums of squares and products". *Technometrics*, 4(3):419–420, 1962.

[123] P. E. Utgoff, N. C. Berkman, and J. A. Clouse. Decision tree induction based on efficient tree restructuring. *Machine Learning*, 29(1):5–44, 1997.

[124] A. Fern, R. Givan, B. Falsafi, and T. Vijaykumar. Dynamic feature selection for hardware prediction. Technical report, School of Electrical and Computer Engineering, Purdue University, 2000.

[125] R. E. Schapire and Y. Singer. Improved boosting using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999.

[126] R. Caruana and A. Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, pages 161–168, New York, NY, USA, 2006. ACM Press.

[127] P. Domingos. Metacost: a general method for making classifiers cost-sensitive. In *KDD '99: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 155–164, New York, NY, USA, 1999. ACM Press.

APPENDICES

APPENDIX A
DATA COLLECTION EXECUTABLE ALGORITHM

```
unit MapCallRetUnit1;
{
 This program initiates interception of all messages and then it saves
 msg information into a data file located in the current directory and
 named "data_log.txt".
}
interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms
  Dialogs, StdCtrls, ExtCtrls;

{ define the library file and the groups of messages of interest
  NOTE: The constant definitions for msgs must coincide with the
  definitions in the DLL library file. }
const
   MyDLL     = 'MapWndRetDll.dll';
   CM_WND_MSG = WM_USER + $1000;       //system msgs
   CM_KEY_MSG = WM_USER + $1001;       //keyboard msgs
   CM_MOUSE_MSG = WM_USER + $1002;     //mouse msgs

type
   TMapWndRetHook=procedure; stdcall;

type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Label1: TLabel;
    Timer1: TTimer;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
    MapHandle, StrHandle      : THandle;
    PReceptor       : ^Integer;
    SReceptor       : PChar;
    HandleDLL       : THandle;
    HookOn,
    HookOff         : TMapWndRetHook;

    procedure MemoHook1(var message: TMessage); message  CM_WND_MSG;
```

Fig. A.1. Declaration and Initialization steps in the EXE file.

```
var
  Form1: TForm1;
  msgID, x_pt, y_pt: array[1..10010] of integer;
  sys_time: array[1..10010] of TDateTime;
  app_name: array[1..10010] of string;
  counter, x_old, y_old, x_new, y_new: integer;
  Curfile: textfile;
  msg_file: string;
  mouse: TMouse;
  Present: TDateTime;
  Year, Month, Day: Word;
```

Fig. A.2. Global variables in the EXE file.

```
implementation

{$R *.dfm}

{
 On "Stop" button click library and the timer are deactivated,
 the collection stops and all of the remaining data in arrays is saved in
 the text file.
}
procedure TForm1.Button1Click(Sender: TObject);
var
i: integer;
begin
  timer1.Interval:=0;
  {Uninstall the Hook}
  if Assigned(HookOff) then HookOff;

  {Free the DLL}
  if HandleDLL<>0 then
   FreeLibrary(HandleDLL);

 {Close the memfile and the View}
  if MapHandle<>0 then
  begin
    UnmapViewOfFile(PReceptor);
    CloseHandle(MapHandle);
  end;

  Present:= Now;
  DecodeDate(Present, Year, Month, Day);
  msg_file:=
ExtractFilePath(Application.Exename)+inttostr(month)+'_'+inttostr(day)+'.tx
t';
  assignfile(CurFile,msg_file);
  if fileexists(msg_file)=false then rewrite(CurFile) else append(CurFile);
       for i:=1 to (counter-1) do
            writeln(CurFile,msgID[i],' ',x_pt[i],' ',
                      y_pt[i],' ',sys_time[i],' ',app_name[i]);
  flush(CurFile); closefile(CurFile);
  showmessage('The program will now close. Please e-mail the data file to
maja@purdue.edu.');
  close;
end;
```

Fig. A.3. Initial setup.

```
{
 On formcreate event "data_log.txt" file is defined to be in the
 current directory. All pointers are initialized and the shared memory
 is defined.
}
procedure TForm1.FormCreate(Sender: TObject);
begin
   timer1.Interval:=0;
   counter:=1;
   x_old:=mouse.CursorPos.X;        //get the current cursor coordinates
   y_old:=mouse.CursorPos.Y;

   //load library
   HandleDLL:=LoadLibrary( PChar(ExtractFilePath(Application.Exename)+
                                 MyDLL ) );
   if HandleDLL = 0 then raise Exception.Create('Cannot load a DLL');

   @HookOn :=GetProcAddress(HandleDLL, 'HookOn');
   @HookOff:=GetProcAddress(HandleDLL, 'HookOff');

   IF not assigned(HookOn) or
      not assigned(HookOff)  then
      raise Exception.Create('Cannot find the required DLL functions');

   //create shared memory

MapHandle:=CreateFileMapping( $FFFFFFFF,nil,PAGE_READWRITE,0,SizeOf(Integer
),
                              'Receptor');
   StrHandle:=CreateFileMapping( $FFFFFFFF,nil,PAGE_READWRITE,0,2000,
                              'StrReceptor');

    if MapHandle=0 then
      raise Exception.Create( 'Error while creating a mapping file');
    if StrHandle=0 then
      raise Exception.Create( 'Error while creating a string mapping file')

    PReceptor:=MapViewOfFile(MapHandle,FILE_MAP_WRITE,0,0,0);
    PReceptor^:=application.Handle;
    SReceptor:=MapViewOfFile(StrHandle,FILE_MAP_WRITE,0,0,0);

end;
```

Fig. A.4. Executable and Library shared mapping.

```
{
 Read selected system msgs from the shared memory, populate the appropriate
 arrays with the msg info values and if the array buffer is full save
 the msg info into the text file.
}
procedure TForm1.MemoHook1(var message: TMessage);
 var
    i, msg: Integer;
 begin
    msg:=message.WParam;
    // the following is the list of msgs that are NOT recorded due to their
sheer volume
    if (msg<WM_USER)then
    begin
      case msg of

0..5,7..11,16..19,22..24,36,125,127..131,134,272..274,529..534,561..563,43.
.48,50,

51,57,768..783,61,65,72,73,75,80..85,536,537,785,794,55,290..297,38..40,123
,276..288,
        135,306,308..311,258..263,786:
        begin
          // if buffer is full record msg info in the arrays into the txt
file
          if (counter=10000) then
          begin
            Present:= Now;
            DecodeDate(Present, Year, Month, Day);
            msg_file:=
ExtractFilePath(Application.Exename)+inttostr(month)+'_'+inttostr(day)+'.tx
t';
            assignfile(CurFile,msg_file);
            if fileexists(msg_file)=false then rewrite(CurFile) else
append(CurFile);
               for i:=1 to 10000 do
                  writeln(CurFile,msgID[i],' ',x_pt[i],' ',
                      y_pt[i],' ',sys_time[i],' ',app_name[i]);
            flush(CurFile); closefile(CurFile);
            counter:=1;
          end;
          // populate arrays with the current msg info
          x_pt[counter]:=mouse.CursorPos.X;
          y_pt[counter]:=mouse.CursorPos.Y;
          sys_time[counter]:=gettime;
```

Fig. A.5. Records GUI data.

```
{
 Read keyboard msgs from the shared memory, populate the appropriate
 arrays with the msg info values and if the array buffer is full save
 the msg info into the text file.
}
procedure TForm1.MemoHook2(var message: TMessage);
 var
    i, msg: Integer;
 begin
   msg:=message.WParam;
   if (counter=10000) then
        begin
            Present:= Now;
            DecodeDate(Present, Year, Month, Day);
            msg_file:=
ExtractFilePath(Application.Exename)+inttostr(month)+'_'+inttostr(day)+'.tx
t';
            assignfile(CurFile,msg_file);
            if fileexists(msg_file)=false then rewrite(CurFile) else
append(CurFile);
               for i:=1 to 10000 do
                  writeln(CurFile,msgID[i],' ',x_pt[i],' ',
                      y_pt[i],' ',sys_time[i],' ',app_name[i]);
            flush(CurFile); closefile(CurFile);
            counter:=1;
        end;
        x_pt[counter]:=mouse.CursorPos.X;
        y_pt[counter]:=mouse.CursorPos.Y;
        sys_time[counter]:=gettime;
        app_name[counter]:='noname';  // no appl. name for key msgs
        //I add 1100 to msg in order to distinguish key msgs from sys msgs
        msgID[counter]:=msg+1100; counter:=counter+1;
end;
```

Fig. A.6. Records keystroke data.

```
{
 Read selected mouse msgs from the shared memory, populate the appropriate
 arrays with the msg info values and if the array buffer is full save
 the msg info into the text file.
}
procedure TForm1.MemoHook3(var message: TMessage);
 var
    i, msg: Integer;
 begin
   msg:=message.WParam;
    // the following msgs are NOT recorded due to volume
    if((msg<WM_USER)and(msg<>WM_MOUSEMOVE)) then
   begin
   if (counter=10000) then
        begin
            Present:= Now;
            DecodeDate(Present, Year, Month, Day);
            msg_file:=
ExtractFilePath(Application.Exename)+inttostr(month)+'_'+inttostr(day)+'.tx
t';
            assignfile(CurFile,msg_file);
            if fileexists(msg_file)=false then rewrite(CurFile) else
append(CurFile);
               for i:=1 to 10000 do
                  writeln(CurFile,msgID[i],' ',x_pt[i],' ',
                      y_pt[i],' ',sys_time[i],' ',app_name[i]);
            flush(CurFile); closefile(CurFile);
            counter:=1;
        end;
        x_pt[counter]:=mouse.CursorPos.X;
        y_pt[counter]:=mouse.CursorPos.Y;
        sys_time[counter]:=gettime;
        app_name[counter]:=SReceptor; // appl. name is recorded
        msgID[counter]:=msg; counter:=counter+1;
    end;
end;
```

Fig. A.7. Records mouse data.

```
{
 On "Start" button click library is initiated, the collection begins
 and the timer for the time series data (10 times per second) is activated.
}
procedure TForm1.Button2Click(Sender: TObject);
begin
     counter:=1;
     timer1.Interval:=100;
     HookOn;
end;
```

Fig. A.8. Program activation code.

```
procedure TForm1.Timer1Timer(Sender: TObject);
var
i: integer;
begin
    x_new:=mouse.CursorPos.X;
    y_new:=mouse.CursorPos.Y;
    // if cursosr has moved within 100msec record new points, otherwise do
nothing
    if(x_new<>x_old)or(y_new<>y_old) then
    begin
        if (counter=10000) then
        begin
            Present:= Now;
            DecodeDate(Present, Year, Month, Day);
            msg_file:=
ExtractFilePath(Application.Exename)+inttostr(month)+'_'+inttostr(day)+'.tx
t';
            assignfile(CurFile,msg_file);
            if fileexists(msg_file)=false then rewrite(CurFile) else
append(CurFile);
               for i:=1 to 10000 do
                  writeln(CurFile,msgID[i],' ',x_pt[i],' ',
                      y_pt[i],' ',sys_time[i],' ',app_name[i]);
            flush(CurFile); closefile(CurFile);
            counter:=1;
        end;
        x_pt[counter]:=x_new;
        y_pt[counter]:=y_new;
        sys_time[counter]:=gettime;
        app_name[counter]:='noname';
        // I recognize time series data by identifier of 2000 as a msg ID
        msgID[counter]:=2000; counter:=counter+1;
        x_old:=x_new; y_old:=y_new;
    end;
end;
```

Fig. A.9. Rate limited client–area mouse movement setup via a timer.

```
{
 On form destroy/close library and the timer are deactivated,
 the collection stops and all of the remaining data in arrays is saved in
 the text file.
}
procedure TForm1.FormDestroy(Sender: TObject);
var
i: integer;
begin
    timer1.Interval:=0;

    if Assigned(HookOff) then HookOff;
    if HandleDLL<>0 then FreeLibrary(HandleDLL);

    {Close the memfile and the View}
    if MapHandle<>0 then
    begin
      UnmapViewOfFile(PReceptor);
      CloseHandle(MapHandle);
    end;

    Present:= Now;
    DecodeDate(Present, Year, Month, Day);
    msg_file:=
ExtractFilePath(Application.Exename)+inttostr(month)+'_'+inttostr(day)+'.tx
t';
    assignfile(CurFile,msg_file);
    if fileexists(msg_file)=false then rewrite(CurFile) else
append(CurFile);
       for i:=1 to (counter-1) do
            writeln(CurFile,msgID[i],' ',x_pt[i],' ',
                     y_pt[i],' ',sys_time[i],' ',app_name[i]);
    flush(CurFile); closefile(CurFile);
    showmessage('The program will now close. Please e-mail the data file to
maja@purdue.edu.');

end;

end.
```

Fig. A.10. Wrap-up code.

APPENDIX B
DATA COLLECTION LIBRARY ALGORITHM

```
library MapWndRetDll;
{
 This is a library DLL file. This file intercepts all messages after the
messages
 have been processed by application.
}
uses
  Windows, SysUtils,
  Messages;

//define groups of messages of interest
const
  CM_WND_MSG = WM_USER + $1000;       //system msgs
  CM_KEY_MSG = WM_USER + $1001;       //keyboard msgs
  CM_MOUSE_MSG = WM_USER + $1002;     //mouse msgs

 var
  WndRetHook, KeyHook, MouseHook    : HHook;
  MapHandle     : THandle;
  PReceptor    : ^Integer;
  StrHandle    :  THandle;
  SReceptor    : PChar;
```

Fig. B.1. Declaration and Initialization steps in the DLL file.

```
//This function intercepts and maps to the shared memory all GUI msgs
 function CallWndRetHook( Code: Integer;wParam: WPARAM;lParam: LPARAM
                          )       : LRESULT; stdcall;
 var
  msg_ptr: ^CWPRETSTRUCT;    //Windows defined data structure that carries
msg details
 begin

  if code=HC_ACTION then
  begin
   {if the mapfile exists}
   MapHandle:=OpenFileMapping(FILE_MAP_WRITE,False,'Receptor');
   {If dont, send nothing to receiver application}
   if MapHandle<>0 then
   begin
     msg_ptr:=Ptr(lparam);
     PReceptor:=MapViewOfFile(MapHandle,FILE_MAP_WRITE,0,0,0);
     PostMessage(PReceptor^,CM_WND_MSG,msg_ptr^.message,msg_ptr^.hwnd);
     UnmapViewOfFile(PReceptor);
     CloseHandle(MapHandle);
   end;
  end;
 {call to next hook of the chain}
  Result := CallNextHookEx(WndRetHook, Code, wParam, lParam)
 end;
```

Fig. B.2. Interception of GUI events in the DLL file.

```
//This function intercepts and maps to the shared memory all keyboard msgs
function KeyboardHook( Code: Integer;wParam: WPARAM;lParam  : LPARAM
                          ) : LRESULT; stdcall;
begin
 {if a key was pressed/released}
 if code=HC_ACTION then
 begin
  {if the mapfile exists}
  MapHandle:=OpenFileMapping(FILE_MAP_WRITE,False,'Receptor');
  {If dont, send nothing to receiver application}
  if MapHandle<>0 then
  begin
    PReceptor:=MapViewOfFile(MapHandle,FILE_MAP_WRITE,0,0,0);
    PostMessage(PReceptor^,CM_KEY_MSG,wparam,lparam);
    UnmapViewOfFile(PReceptor);
    CloseHandle(MapHandle);
  end;
 end;
{call to next hook of the chain}
 Result := CallNextHookEx(KeyHook, Code, wParam, lParam)
end;
```

Fig. B.3. Interception of keystroke events in the DLL file.

```
//This function intercepts and maps to the shared memory all mouse msgs
 function MouseProcHook( Code: Integer; wParam: WPARAM; lParam: LPARAM
                          )       : LRESULT; stdcall;
 var
    msg_ptr: ^MOUSEHOOKSTRUCT;
    class_name: string;
    iLength: cardinal;
 begin
  if code=HC_ACTION then
  begin
   {if the mapfile exists}
   MapHandle:=OpenFileMapping(FILE_MAP_WRITE,False,'Receptor');
   StrHandle:=OpenFileMapping(FILE_MAP_WRITE,False,'StrReceptor');

   {If dont, send nothing to receiver application}
   if ((MapHandle<>0)and(StrHandle<>0)) then
   begin
     msg_ptr:=Ptr(lParam);
     iLength := 255;
     setLength(class_name, iLength);
     iLength :=
GetWindowmodulefilename(msg_ptr^.hwnd,PChar(class_name),iLength);
     setLength(class_name, iLength);

     SReceptor:=MapViewOfFile(StrHandle,FILE_MAP_WRITE,0,0,0);
     StrPLCopy(SReceptor,class_name,iLength);
     UnmapViewOfFile(SReceptor);
     CloseHandle(StrHandle);

     PReceptor:=MapViewOfFile(MapHandle,FILE_MAP_WRITE,0,0,0);
     PostMessage(PReceptor^,CM_MOUSE_MSG,wparam,lparam);
     UnmapViewOfFile(PReceptor);
     CloseHandle(MapHandle);
   end;
  end;
 {call to next hook of the chain}
  Result := CallNextHookEx(MouseHook, Code, wParam, lParam)
 end;
```

Fig. B.4. Interception of mouse events in the DLL file.

```
{ Three hooks are installed. For more information on each one of WH_?
commands
 see Windows manual }
 procedure HookOn; stdcall;
 {procedure for install the hook}

 begin
   WndRetHook:=SetWindowsHookEx(WH_CALLWNDPROCRET, @CallWndRetHook,
HInstance , 0);
   KeyHook:=SetWindowsHookEx(WH_KEYBOARD, @KeyboardHook, HInstance , 0);
   MouseHook:=SetWindowsHookEx(WH_MOUSE, @MouseProcHook, HInstance , 0);
 end;

 procedure HookOff;  stdcall;
 begin
   {procedure to uninstall the hook}
   UnhookWindowsHookEx(WndRetHook);
   UnhookWindowsHookEx(KeyHook);
   UnhookWindowsHookEx(MouseHook);
 end;

 exports
 {Export the procedures}
  HookOn,
  HookOff;

begin
end.
```

Fig. B.5. Exported functions in the DLL file.

APPENDIX C
GUI EVENTS

Table C.1: List of GUI Events

ID_{16}	**Event Name**	ID_{16}	**Event Name**
0	WM_NULL	1	WM_CREATE
2	WM_DESTROY	3	WM_MOVE
5	WM_SIZE	7	WM_SETFOCUS
8	WM_KILLFOCUS	A	WM_ENABLE
B	WM_SETREDRAW	10	WM_CLOSE
11	WM_QUERYENDSESSION	12	WM_QUIT
13	WM_QUERYOPEN	16	WM_ENDSESSION
17	WM_SYSTEMERROR	1B	WM_DEVMODECHANGE
2B	WM_DRAWITEM	2C	WM_MEASUREITEM
2D	WM_DELETEITEM	2E	WM_VKEYTOITEM
2F	WM_CHARTOITEM	30	WM_SETFONT
32	WM_SETHOTKEY	33	WM_GETHOTKEY
37	WM_QUERYDRAGICON	39	WM_COMPAREITEM
3D	WM_GETOBJECT	41	WM_COMPACTING
47	WM_WINDOWPOSCHANGED	48	WM_POWER
4B	WM_CANCELJOURNAL	50	WM_INPUTLANGCHANGEREQUEST
51	WM_INPUTLANGCHANGE	52	WM_TCARD
53	WM_HELP	54	WM_USERCHANGED
55	WM_NOTIFYFORMAT	7B	WM_CONTEXTMENU

Continued on next page

ID_{16}	**Event Name**	ID_{16}	**Event Name**
7D	WM_STYLECHANGED	7F	WM_GETICON
80	WM_SETICON	81	WM_NCCREATE
82	WM_NCDESTROY	83	WM_NCCALCSIZE
86	WM_NCACTIVATE	87	WM_GETDLGCODE
FF	WM_INPUT	102	WM_CHAR
103	WM_DEADCHAR	104	WM_SYSKEYDOWN
105	WM_SYSKEYUP	106	WM_SYSCHAR
107	WM_SYSDEADCHAR	110	WM_INITDIALOG
111	WM_COMMAND	112	WM_SYSCOMMAND
114	WM_HSCROLL	115	WM_VSCROLL
116	WM_INITMENU	117	WM_INITMENUPOPUP
11F	WM_MENUSELECT	120	WM_MENUCHAR
122	WM_MENURBUTTONUP	123	WM_MENUDRAG
124	WM_MENUGETOBJECT	125	WM_UNINITMENUPOPUP
126	WM_MENUCOMMAND	127	WM_CHANGEUISTATE
128	WM_UPDATEUISTATE	129	WM_QUERYUISTATE
132	WM_CTLCOLORMSGBOX	134	WM_CTLCOLORLISTBOX
135	WM_CTLCOLORBTN	136	WM_CTLCOLORDLG
137	WM_CTLCOLORSCROLLBAR	211	WM_ENTERMENULOOP
212	WM_EXITMENULOOP	213	WM_NEXTMENU
214	WM_SIZING	215	WM_CAPTURECHANGED
216	WM_MOVING	217	WM_POWERBROADCAST
218	WM_DEVICECHANGE	10D	WM_IME_STARTCOMPOSITION
10E	WM_IME_ENDCOMPOSITION	10F	WM_IME_COMPOSITION
281	WM_IME_SETCONTEXT	282	WM_IME_NOTIFY
283	WM_IME_CONTROL	284	WM_IME_COMPOSITIONFULL
285	WM_IME_SELECT	286	WM_IME_CHAR

Continued on next page

ID_{16}	**Event Name**	ID_{16}	**Event Name**
288	WM_IME_REQUEST	290	WM_IME_KEYDOWN
291	WM_IME_KEYUP	220	WM_MDICREATE
221	WM_MDIDESTROY	222	WM_MDIACTIVATE
223	WM_MDIRESTORE	224	WM_MDINEXT
225	WM_MDIMAXIMIZE	226	WM_MDITILE
227	WM_MDICASCADE	228	WM_MDIICONARRANGE
229	WM_MDIGETACTIVE	230	WM_MDISETMENU
231	WM_ENTERSIZEMOVE	232	WM_EXITSIZEMOVE
233	WM_DROPFILES	234	WM_MDIREFRESHMENU
300	WM_CUT	301	WM_COPY
302	WM_PASTE	303	WM_CLEAR
304	WM_UNDO	305	WM_RENDERFORMAT
306	WM_RENDERALLFORMATS	307	WM_DESTROYCLIPBOARD
308	WM_DRAWCLIPBOARD	309	WM_PAINTCLIPBOARD
30A	WM_VSCROLLCLIPBOARD	30B	WM_SIZECLIPBOARD
30C	WM_ASKCBFORMATNAME	30D	WM_CHANGECBCHAIN
30E	WM_HSCROLLCLIPBOARD	30F	WM_QUERYNEWPALETTE
311	WM_PALETTECHANGED	312	WM_HOTKEY
31A	WM_THEMECHANGED	3E0	WM_DDE_INITIATE
3E1	WM_DDE_TERMINATE	3E2	WM_DDE_ADVISE
3E3	WM_DDE_UNADVISE	3E4	WM_DDE_ACK
3E5	WM_DDE_DATA	3E6	WM_DDE_REQUEST
3E7	WM_DDE_POKE	3E8	WM_DDE_EXECUTE

APPENDIX D

LIST OF FEATURES

Table D.1: A complete list of mouse features.

ID	Feature Description
1	Number of *all* Mouse points
2	Number of Mouse events
3	Number of NC Moves
4	Number of Mouse Moves
5	Number of Clicks
6	Numbers of Wheel Moves
7	Number of Single clicks
8	Number of Double Clicks
9	Mean of X coordinates of mouse events
10	Standard deviation of X coordinates of mouse events
11	Skewness of X coordinates of mouse events
12	Mean of Y coordinates of mouse events
13	Standard deviation of Y coordinates of mouse events
14	Skewness of Y coordinates of mouse events
15	Mean of X coordinates of NC moves
16	Standard deviation of X coordinates of NC moves
17	Skewness of X coordinates of NC moves
18	Mean of Y coordinates of NC moves
19	Standard deviation of Y coordinates of NC moves

Continued on next page

ID	Feature Description
20	Skewness of Y coordinates of NC moves
21	Mean of X coordinates of mouse moves
22	Standard deviation of X coordinates of mouse moves
23	Skewness of X coordinates of mouse moves
24	Mean of Y coordinates of mouse moves
25	Standard deviation of Y coordinates of mouse moves
26	Skewness of Y coordinates of mouse moves
27	Mean of X coordinates of wheel moves
28	Standard deviation of X coordinates of wheel moves
29	Skewness of X coordinates of wheel moves
30	Mean of Y coordinates of wheel moves
31	Standard deviation of Y coordinates of wheel moves
32	Skewness of Y coordinates of wheel moves
33	Mean of X coordinates of clicks
34	Standard deviation of X coordinates of clicks
35	Skewness of X coordinates of clicks
36	Mean of Y coordinates of clicks
37	Standard deviation of Y coordinates of clicks
38	Skewness of Y coordinates of clicks
39	Mean of X coordinates of single clicks
40	Standard deviation of X coordinates of single clicks
41	Skewness of X coordinates of single clicks
42	Mean of Y coordinates of single clicks
43	Standard deviation of Y coordinates of single clicks
44	Skewness of Y coordinates of single clicks
45	Mean of X coordinates of double clicks
46	Standard deviation of X coordinates of double clicks

Continued on next page

ID	Feature Description
47	Skewness of X coordinates of double clicks
48	Mean of Y coordinates of double clicks
49	Standard deviation of Y coordinates of double clicks
50	Skewness of Y coordinates of double clicks
51	Mean of 1–graph of mouse events
52	Standard deviation of 1–graph of mouse events
53	Skewness of 1–graph of mouse events
54	Mean of 2–graph of mouse events
55	Standard deviation of 2–graph of mouse events
56	Skewness of 2–graph of mouse events
57	Mean of 3–graph of mouse events
58	Standard deviation of 3–graph of mouse events
59	Skewness of 3–graph of mouse events
60	Mean of 4–graph of mouse events
61	Standard deviation of 4–graph of mouse events
62	Skewness of 4–graph of mouse events
63	Mean of 5–graph of mouse events
64	Standard deviation of 5–graph of mouse events
65	Skewness of 5–graph of mouse events
66	Mean of 6–graph of mouse events
67	Standard deviation of 6–graph of mouse events
68	Skewness of 6–graph of mouse events
69	Mean of 7–graph of mouse events
70	Standard deviation of 7–graph of mouse events
71	Skewness of 7–graph of mouse events
72	Mean of 8–graph of mouse events
73	Standard deviation of 8–graph of mouse events

Continued on next page

ID	Feature Description
74	Skewness of 8–graph of mouse events
75	Mean of 1–graph of NC moves
76	Standard deviation of 1–graph of NC moves
77	Skewness of 1–graph of NC moves
78	Mean of 2–graph of NC moves
79	Standard deviation of 2–graph of NC moves
80	Skewness of 2–graph of NC moves
81	Mean of 3–graph of NC moves
82	Standard deviation of 3–graph of NC moves
83	Skewness of 3–graph of NC moves
84	Mean of 4–graph of NC moves
85	Standard deviation of 4–graph of NC moves
86	Skewness of 4–graph of NC moves
87	Mean of 5–graph of NC moves
88	Standard deviation of 5–graph of NC moves
89	Skewness of 5–graph of NC moves
90	Mean of 6–graph of NC moves
91	Standard deviation of 6–graph of NC moves
92	Skewness of 6–graph of NC moves
93	Mean of 7–graph of NC moves
94	Standard deviation of 7–graph of NC moves
95	Skewness of 7–graph of NC moves
96	Mean of 8–graph of NC moves
97	Standard deviation of 8–graph of NC moves
98	Skewness of 8–graph of NC moves
99	Mean of 1–graph of mouse moves
100	Standard deviation of 1–graph of mouse moves

Continued on next page

ID	Feature Description
101	Skewness of 1–graph of mouse moves
102	Mean of 2–graph of mouse moves
103	Standard deviation of 2–graph of mouse moves
104	Skewness of 2–graph of mouse moves
105	Mean of 3–graph of mouse moves
106	Standard deviation of 3–graph of mouse moves
107	Skewness of 3–graph of mouse moves
108	Mean of 4–graph of mouse moves
109	Standard deviation of 4–graph of mouse moves
110	Skewness of 4–graph of mouse moves
111	Mean of 5–graph of mouse moves
112	Standard deviation of 5–graph of mouse moves
113	Skewness of 5–graph of mouse moves
114	Mean of 6–graph of mouse moves
115	Standard deviation of 6–graph of mouse moves
116	Skewness of 6–graph of mouse moves
117	Mean of 7–graph of mouse moves
118	Standard deviation of 7–graph of mouse moves
119	Skewness of 7–graph of mouse moves
120	Mean of 8–graph of mouse moves
121	Standard deviation of 8–graph of mouse moves
122	Skewness of 8–graph of mouse moves
123	Mean of 1–graph of wheel moves
124	Standard deviation of 1–graph of wheel moves
125	Skewness of 1–graph of wheel moves
126	Mean of 2–graph of wheel moves
127	Standard deviation of 2–graph of wheel moves

Continued on next page

ID	Feature Description
128	Skewness of 2–graph of wheel moves
129	Mean of 3–graph of wheel moves
130	Standard deviation of 3–graph of wheel moves
131	Skewness of 3–graph of wheel moves
132	Mean of 4–graph of wheel moves
133	Standard deviation of 4–graph of wheel moves
134	Skewness of 4–graph of wheel moves
135	Mean of 5–graph of wheel moves
136	Standard deviation of 5–graph of wheel moves
137	Skewness of 5–graph of wheel moves
138	Mean of 6–graph of wheel moves
139	Standard deviation of 6–graph of wheel moves
140	Skewness of 6–graph of wheel moves
141	Mean of 7–graph of wheel moves
142	Standard deviation of 7–graph of wheel moves
143	Skewness of 7–graph of wheel moves
144	Mean of 8–graph of wheel moves
145	Standard deviation of 8–graph of wheel moves
146	Skewness of 8–graph of wheel moves
147	Mean of 1–graph of clicks
148	Standard deviation of 1–graph of clicks
149	Skewness of 1–graph of clicks
150	Mean of 2–graph of clicks
151	Standard deviation of 2–graph of clicks
152	Skewness of 2–graph of clicks
153	Mean of 3–graph of clicks
154	Standard deviation of 3–graph of clicks

Continued on next page

ID	Feature Description
155	Skewness of 3–graph of clicks
156	Mean of 4–graph of clicks
157	Standard deviation of 4–graph of clicks
158	Skewness of 4–graph of clicks
159	Mean of 5–graph of clicks
160	Standard deviation of 5–graph of clicks
161	Skewness of 5–graph of clicks
162	Mean of 6–graph of clicks
163	Standard deviation of 6–graph of clicks
164	Skewness of 6–graph of clicks
165	Mean of 7–graph of clicks
166	Standard deviation of 7–graph of clicks
167	Skewness of 7–graph of clicks
168	Mean of 8–graph of clicks
169	Standard deviation of 8–graph of clicks
170	Skewness of 8–graph of clicks
171	Mean of 1–graph of single clicks
172	Standard deviation of 1–graph of single clicks
173	Skewness of 1–graph of single clicks
174	Mean of 2–graph of single clicks
175	Standard deviation of 2–graph of single clicks
176	Skewness of 2–graph of single clicks
177	Mean of 3–graph of single clicks
178	Standard deviation of 3–graph of single clicks
179	Skewness of 3–graph of single clicks
180	Mean of 4–graph of single clicks
181	Standard deviation of 4–graph of single clicks

Continued on next page

ID	Feature Description
182	Skewness of 4–graph of single clicks
183	Mean of 5–graph of single clicks
184	Standard deviation of 5–graph of single clicks
185	Skewness of 5–graph of single clicks
186	Mean of 6–graph of single clicks
187	Standard deviation of 6–graph of single clicks
188	Skewness of 6–graph of single clicks
189	Mean of 7–graph of single clicks
190	Standard deviation of 7–graph of single clicks
191	Skewness of 7–graph of single clicks
192	Mean of 8–graph of single clicks
193	Standard deviation of 8–graph of single clicks
194	Skewness of 8–graph of single clicks
195	Mean of 1–graph of double clicks
196	Standard deviation of 1–graph of double clicks
197	Skewness of 1–graph of double clicks
198	Mean of 2–graph of double clicks
199	Standard deviation of 2–graph of double clicks
200	Skewness of 2–graph of double clicks
201	Mean of 3–graph of double clicks
202	Standard deviation of 3–graph of double clicks
203	Skewness of 3–graph of double clicks
204	Mean of 4–graph of double clicks
205	Standard deviation of 4–graph of double clicks
206	Skewness of 4–graph of double clicks
207	Mean of 5–graph of double clicks
208	Standard deviation of 5–graph of double clicks

Continued on next page

ID	Feature Description
209	Skewness of 5–graph of double clicks
210	Mean of 6–graph of double clicks
211	Standard deviation of 6–graph of double clicks
212	Skewness of 6–graph of double clicks
213	Mean of 7–graph of double clicks
214	Standard deviation of 7–graph of double clicks
215	Skewness of 7–graph of double clicks
216	Mean of 8–graph of double clicks
217	Standard deviation of 8–graph of double clicks
218	Skewness of 8–graph of double clicks
219	Mean of distance of mouse events
220	Standard deviation of distance of mouse events
221	Skewness of distance of mouse events
222	Mean of angle of mouse events
223	Standard deviation of angle of mouse events
224	Skewness of angle of mouse events
225	Mean of speed of mouse events
226	Standard deviation of speed of mouse events
227	Skewness of speed of mouse events
228	Mean of distance of NC moves
229	Standard deviation of distance of NC moves
230	Skewness of distance of NC moves
231	Mean of angle of NC moves
232	Standard deviation of angle of NC moves
233	Skewness of angle of NC moves
234	Mean of speed of NC moves
235	Standard deviation of speed of NC moves

Continued on next page

ID	Feature Description
236	Skewness of speed of NC moves
237	Mean of distance of mouse moves
238	Standard deviation of distance of mouse moves
239	Skewness of distance of mouse moves
240	Mean of angle of mouse moves
241	Standard deviation of angle of mouse moves
242	Skewness of angle of mouse moves
243	Mean of speed of mouse moves
244	Standard deviation of speed of mouse moves
245	Skewness of speed of mouse moves
246	Mean of distance of wheel moves
247	Standard deviation of distance of wheel moves
248	Skewness of distance of wheel moves
249	Mean of angle of single wheel moves
250	Standard deviation of angle of wheel moves
251	Skewness of angle of wheel moves
252	Mean of speed of wheel moves
253	Standard deviation of speed of wheel moves
254	Skewness of speed of wheel moves
255	Mean of distance of clicks
256	Standard deviation of distance of clicks
257	Skewness of distance of clicks
258	Mean of angle of clicks
259	Standard deviation of angle of clicks
260	Skewness of angle of clicks
261	Mean of speed of clicks
262	Standard deviation of speed of clicks

Continued on next page

ID	Feature Description
263	Skewness of speed of clicks
264	Mean of distance of single clicks
265	Standard deviation of distance of single clicks
266	Skewness of distance of single clicks
267	Mean of angle of single clicks
268	Standard deviation of angle of single clicks
269	Skewness of angle of single clicks
270	Mean of speed of single clicks
271	Standard deviation of speed of single clicks
272	Skewness of speed of single clicks
273	Mean of distance of double clicks
274	Standard deviation of distance of double clicks
275	Skewness of distance of double clicks
276	Mean of angle of double clicks
277	Standard deviation of angle of double clicks
278	Skewness of angle of double clicks
279	Mean of speed of double clicks
280	Standard deviation of speed of double clicks

Table D.2: A complete list of Keystroke features.

ID	Feature Description
1	Number of *all* Keystroke points
2	Number of Function keys
3	Number of Control keys
4	Number of Regular keys
5	Number of Mouse–keys

Continued on next page

ID	Feature Description
6	Numbers of Other keys
7	Number of Letters
8	Number of Numbers
9	Number of letter A
10	Number of letter B
11	Number of letter C
12	Number of letter D
13	Number of letter E
14	Number of letter F
15	Number of letter G
16	Number of letter H
17	Number of letter I
18	Number of letter J
19	Number of letter K
20	Number of letter L
21	Number of letter M
22	Number of letter N
23	Number of letter O
24	Number of letter P
25	Number of letter Q
26	Number of letter R
27	Number of letter S
28	Number of letter T
29	Number of letter U
30	Number of letter V
31	Number of letter W
32	Number of letter X

Continued on next page

ID	Feature Description
33	Number of letter Y
34	Number of letter Z
35	Number of numerals 0
36	Number of numerals 1
37	Number of numerals 2
38	Number of numerals 3
39	Number of numerals 4
40	Number of numerals 5
41	Number of numerals 6
42	Number of numerals 7
43	Number of numerals 8
44	Number of numerals 9
45	Mean of 1–graph of *all* keystrokes
46	Standard deviation of 1–graph of *all* keystrokes
47	Skewness of 1–graph of *all* keystrokes
48	Mean of 2–graph of *all* keystrokes
49	Standard deviation of 2–graph of *all* keystrokes
50	Skewness of 2–graph of *all* keystrokes
51	Mean of 3–graph of *all* keystrokes
52	Standard deviation of 3–graph of *all* keystrokes
53	Skewness of 3–graph of *all* keystrokes
54	Mean of 4–graph of *all* keystrokes
55	Standard deviation of 4–graph of *all* keystrokes
56	Skewness of 4–graph of *all* keystrokes
57	Mean of 5–graph of *all* keystrokes
58	Standard deviation of 5–graph of *all* keystrokes
59	Skewness of 5–graph of *all* keystrokes

Continued on next page

ID	Feature Description
60	Mean of 6–graph of *all* keystrokes
61	Standard deviation of 6–graph of *all* keystrokes
62	Skewness of 6–graph of *all* keystrokes
63	Mean of 7–graph of *all* keystrokes
64	Standard deviation of 7–graph of *all* keystrokes
65	Skewness of 7–graph of *all* keystrokes
66	Mean of 8–graph of *all* keystrokes
67	Standard deviation of 8–graph of *all* keystrokes
68	Skewness of 8–graph of *all* keystrokes
69	Mean of 1–graph of Function keys
70	Standard deviation of 1–graph of Function keys
71	Skewness of 1–graph of Function keys
72	Mean of 2–graph of Function keys
73	Standard deviation of 2–graph of Function keys
74	Skewness of 2–graph of Function keys
75	Mean of 3–graph of Function keys
76	Standard deviation of 3–graph of Function keys
77	Skewness of 3–graph of Function keys
78	Mean of 4–graph of Function keys
79	Standard deviation of 4–graph of Function keys
80	Skewness of 4–graph of Function keys
81	Mean of 5–graph of Function keys
82	Standard deviation of 5–graph of Function keys
83	Skewness of 5–graph of Function keys
84	Mean of 6–graph of Function keys
85	Standard deviation of 6–graph of Function keys
86	Skewness of 6–graph of Function keys

Continued on next page

ID	Feature Description
87	Mean of 7–graph of Function keys
88	Standard deviation of 7–graph of Function keys
89	Skewness of 7–graph of Function keys
90	Mean of 8–graph of Function keys
91	Standard deviation of 8–graph of Function keys
92	Skewness of 8–graph of Function keys
93	Mean of 1–graph of Control keys
94	Standard deviation of 1–graph of Control keys
95	Skewness of 1–graph of Control keys
96	Mean of 2–graph of Control keys
97	Standard deviation of 2–graph of Control keys
98	Skewness of 2–graph of Control keys
99	Mean of 3–graph of Control keys
100	Standard deviation of 3–graph of Control keys
101	Skewness of 3–graph of Control keys
102	Mean of 4–graph of Control keys
103	Standard deviation of 4–graph of Control keys
104	Skewness of 4–graph of Control keys
105	Mean of 5–graph of Control keys
106	Standard deviation of 5–graph of Control keys
107	Skewness of 5–graph of Control keys
108	Mean of 6–graph of Control keys
109	Standard deviation of 6–graph of Control keys
110	Skewness of 6–graph of Control keys
111	Mean of 7–graph of Control keys
112	Standard deviation of 7–graph of Control keys
113	Skewness of 7–graph of Control keys

Continued on next page

ID	Feature Description
114	Mean of 8–graph of Control keys
115	Standard deviation of 8–graph of Control keys
116	Skewness of 8–graph of Control keys
117	Mean of 1–graph of Regular keys
118	Standard deviation of 1–graph of Regular keys
119	Skewness of 1–graph of Regular keys
120	Mean of 2–graph of Regular keys
121	Standard deviation of 2–graph of Regular keys
122	Skewness of 2–graph of Regular keys
123	Mean of 3–graph of Regular keys
124	Standard deviation of 3–graph of Regular keys
125	Skewness of 3–graph of Regular keys
126	Mean of 4–graph of Regular keys
127	Standard deviation of 4–graph of Regular keys
128	Skewness of 4–graph of Regular keys
129	Mean of 5–graph of Regular keys
130	Standard deviation of 5–graph of Regular keys
131	Skewness of 5–graph of Regular keys
132	Mean of 6–graph of Regular keys
133	Standard deviation of 6–graph of Regular keys
134	Skewness of 6–graph of Regular keys
135	Mean of 7–graph of Regular keys
136	Standard deviation of 7–graph of Regular keys
137	Skewness of 7–graph of Regular keys
138	Mean of 8–graph of Regular keys
139	Standard deviation of 8–graph of Regular keys
140	Skewness of 8–graph of Regular keys

Continued on next page

ID	Feature Description
141	Mean of 1–graph of Mouse–keys
142	Standard deviation of 1–graph of Mouse–keys
143	Skewness of 1–graph of Mouse–keys
144	Mean of 2–graph of Mouse–keys
145	Standard deviation of 2–graph of Mouse–keys
146	Skewness of 2–graph of Mouse–keys
147	Mean of 3–graph of Mouse–keys
148	Standard deviation of 3–graph of Mouse–keys
149	Skewness of 3–graph of Mouse–keys
150	Mean of 4–graph of Mouse–keys
151	Standard deviation of 4–graph of Mouse–keys
152	Skewness of 4–graph of Mouse–keys
153	Mean of 5–graph of Mouse–keys
154	Standard deviation of 5–graph of Mouse–keys
155	Skewness of 5–graph of Mouse–keys
156	Mean of 6–graph of Mouse–keys
157	Standard deviation of 6–graph of Mouse–keys
158	Skewness of 6–graph of Mouse–keys
159	Mean of 7–graph of Mouse–keys
160	Standard deviation of 7–graph of Mouse–keys
161	Skewness of 7–graph of Mouse–keys
162	Mean of 8–graph of Mouse–keys
163	Standard deviation of 8–graph of Mouse–keys
164	Skewness of 8–graph of Mouse–keys
165	Mean of 1–graph of Other keys
166	Standard deviation of 1–graph of Other keys
167	Skewness of 1–graph of Other keys

Continued on next page

ID	Feature Description
168	Mean of 2–graph of Other keys s
169	Standard deviation of 2–graph of Other keys
170	Skewness of 2–graph of Other keys
171	Mean of 3–graph of Other keys
172	Standard deviation of 3–graph of Other keys
173	Skewness of 3–graph of Other keys
174	Mean of 4–graph of Other keys
175	Standard deviation of 4–graph of Other keys
176	Skewness of 4–graph of Other keys
177	Mean of 5–graph of Other keys
178	Standard deviation of 5–graph of Other keys
179	Skewness of 5–graph of Other keys
180	Mean of 6–graph of Other keys
181	Standard deviation of 6–graph of Other keys
182	Skewness of 6–graph of Other keys
183	Mean of 7–graph of Other keys
184	Standard deviation of 7–graph of Other keys
185	Skewness of 7–graph of Other keys
186	Mean of 8–graph of Other keys
187	Standard deviation of 8–graph of Other keys
188	Skewness of 8–graph of Other keys
189	Mean of 1–graph of Letters
190	Standard deviation of 1–graph of Letters
191	Skewness of 1–graph of Letters
192	Mean of 2–graph of Letters
193	Standard deviation of 2–graph of Letters
194	Skewness of 2–graph of Letters

Continued on next page

ID	Feature Description
195	Mean of 3–graph of Letters
196	Standard deviation of 3–graph of Letters
197	Skewness of 3–graph of Letters
198	Mean of 4–graph of Letters
199	Standard deviation of 4–graph of Letters
200	Skewness of 4–graph of Letters
201	Mean of 5–graph of Letters
202	Standard deviation of 5–graph of Letters
203	Skewness of 5–graph of Letters
204	Mean of 6–graph of Letters
205	Standard deviation of 6–graph of Letters
206	Skewness of 6–graph of Letters
207	Mean of 7–graph of Letters
208	Standard deviation of 7–graph of Letters
209	Skewness of 7–graph of Letters
210	Mean of 8–graph of Letters
211	Standard deviation of 8–graph of Letters
212	Skewness of 8–graph of Letters
213	Mean of 1–graph of Numbers
214	Standard deviation of 1–graph of Numbers
215	Skewness of 1–graph of Numbers
216	Mean of 2–graph of Numbers
217	Standard deviation of 2–graph of Numbers
218	Skewness of 2–graph of Numbers
219	Mean of 3–graph of Numbers
220	Standard deviation of 3–graph of Numbers
221	Skewness of 3–graph of Numbers

Continued on next page

ID	Feature Description
222	Mean of 4–graph of Numbers
223	Standard deviation of 4–graph of Numbers
224	Skewness of 4–graph of Numbers
225	Mean of 5–graph of Numbers
226	Standard deviation of 5–graph of Numbers
227	Skewness of 5–graph of Numbers
228	Mean of 6–graph of Numbers
229	Standard deviation of 6–graph of Numbers
230	Skewness of 6–graph of Numbers
231	Mean of 7–graph of Numbers
232	Standard deviation of 7–graph of Numbers
233	Skewness of 7–graph of Numbers
234	Mean of 8–graph of Numbers
235	Standard deviation of 8–graph of Numbers
236	Skewness of 8–graph of Numbers

Table D.3: A complete list of GUI features.

ID	Feature Description
1	Number of *all* GUI points
2	Number of Spatial+Temporal events
3	Number of Temporal events
4	Number of Window events
5	Number of Control events
6	Numbers of Menu events
7	Number of Item events
8	Number of Icon events

Continued on next page

ID	Feature Description
9	Number of Dialog events
10	Numbers of Query events
11	Number of Combo box events
12	Number of Miscellaneous events
13	Mean of X coordinates of *all* GUI events
14	Standard deviation of X coordinates of *all* GUI events
15	Skewness of X coordinates of *all* GUI events
16	Mean of Y coordinates of *all* GUI events
17	Standard deviation of Y coordinates of *all* GUI events
18	Skewness of Y coordinates of *all* GUI events
19	Mean of X coordinates of Spatial+Temporal events
20	Standard deviation of X coordinates of Spatial+Temporal events
21	Skewness of X coordinates of Spatial+Temporal events
22	Mean of Y coordinates of Spatial+Temporal events
23	Standard deviation of Y coordinates of Spatial+Temporal events
24	Skewness of Y coordinates of Spatial+Temporal events
25	Mean of X coordinates of Window events
26	Standard deviation of X coordinates of Window events
27	Skewness of X coordinates of Window events
28	Mean of Y coordinates of Window events
29	Standard deviation of Y coordinates of Window events
30	Skewness of Y coordinates of Window events
31	Mean of X coordinates of Control events
32	Standard deviation of X coordinates of Control events
33	Skewness of X coordinates of Control events
34	Mean of Y coordinates of Control events
35	Standard deviation of Y coordinates of Control events

Continued on next page

ID	Feature Description
36	Skewness of Y coordinates of Control events
37	Mean of X coordinates of Menu events
38	Standard deviation of X coordinates of Menu events
39	Skewness of X coordinates of Menu events
40	Mean of Y coordinates of Menu events
41	Standard deviation of Y coordinates of Menu events
42	Skewness of Y coordinates of Menu events
43	Mean of X coordinates of Item events
44	Standard deviation of X coordinates of Item events
45	Skewness of X coordinates of Item events
46	Mean of Y coordinates of Item events
47	Standard deviation of Y coordinates of Item events
48	Skewness of Y coordinates of Item events
49	Mean of 1–graph of *all* GUI events
50	Standard deviation of 1–graph of *all* GUI events
51	Skewness of 1–graph of *all* GUI events
52	Mean of 2–graph of *all* GUI events
53	Standard deviation of 2–graph of *all* GUI events
54	Skewness of 2–graph of *all* GUI events
55	Mean of 3–graph of *all* GUI events
56	Standard deviation of 3–graph of *all* GUI events
57	Skewness of 3–graph of *all* GUI events
58	Mean of 4–graph of *all* GUI events
59	Standard deviation of 4–graph of *all* GUI events
60	Skewness of 4–graph of *all* GUI events
61	Mean of 5–graph of *all* GUI events
62	Standard deviation of 5–graph of *all* GUI events

Continued on next page

ID	Feature Description
63	Skewness of 5–graph of *all* GUI events
64	Mean of 6–graph of *all* GUI events
65	Standard deviation of 6–graph of *all* GUI events
66	Skewness of 6–graph of *all* GUI events
67	Mean of 7–graph of *all* GUI events
68	Standard deviation of 7–graph of *all* GUI events
69	Skewness of 7–graph of *all* GUI events
70	Mean of 8–graph of *all* GUI events
71	Standard deviation of 8–graph of *all* GUI events
72	Skewness of 8–graph of *all* GUI events
73	Mean of 1–graph of Spatial+Temporal events
74	Standard deviation of 1–graph of Spatial+Temporal events
75	Skewness of 1–graph of Spatial+Temporal events
76	Mean of 2–graph of Spatial+Temporal events
77	Standard deviation of 2–graph of Spatial+Temporal events
78	Skewness of 2–graph of Spatial+Temporal events
79	Mean of 3–graph of Spatial+Temporal events
80	Standard deviation of 3–graph of Spatial+Temporal events
81	Skewness of 3–graph of Spatial+Temporal events
82	Mean of 4–graph of Spatial+Temporal events
83	Standard deviation of 4–graph of Spatial+Temporal events
84	Skewness of 4–graph of Spatial+Temporal events
85	Mean of 5–graph of Spatial+Temporal events
86	Standard deviation of 5–graph of Spatial+Temporal events
87	Skewness of 5–graph of Spatial+Temporal events
88	Mean of 6–graph of Spatial+Temporal events
89	Standard deviation of 6–graph of Spatial+Temporal events

Continued on next page

ID	Feature Description
90	Skewness of 6–graph of Spatial+Temporal events
91	Mean of 7–graph of Spatial+Temporal events
92	Standard deviation of 7–graph of Spatial+Temporal events
93	Skewness of 7–graph of Spatial+Temporal events
94	Mean of 8–graph of Spatial+Temporal events
95	Standard deviation of 8–graph of Spatial+Temporal events
96	Skewness of 8–graph of Spatial+Temporal events
97	Mean of 1–graph of Temporal events
98	Standard deviation of 1–graph of Temporal events
99	Skewness of 1–graph of Temporal events
100	Mean of 2–graph of Temporal events
101	Standard deviation of 2–graph of Temporal events
102	Skewness of 2–graph of Temporal events
103	Mean of 3–graph of Temporal events
104	Standard deviation of 3–graph of Temporal events
105	Skewness of 3–graph of Temporal events
106	Mean of 4–graph of Temporal events
107	Standard deviation of 4–graph of Temporal events
108	Skewness of 4–graph of Temporal events
109	Mean of 5–graph of Temporal events
110	Standard deviation of 5–graph of Temporal events
111	Skewness of 5–graph of Temporal events
112	Mean of 6–graph of Temporal events
113	Standard deviation of 6–graph of Temporal events
114	Skewness of 6–graph of Temporal events
115	Mean of 7–graph of Temporal events
116	Standard deviation of 7–graph of Temporal events

Continued on next page

ID	Feature Description
117	Skewness of 7–graph of Temporal events
118	Mean of 8–graph of Temporal events
119	Standard deviation of 8–graph of Temporal events
120	Skewness of 8–graph of Temporal events
121	Mean of 1–graph of Window events
122	Standard deviation of 1–graph of Window events
123	Skewness of 1–graph of Window events
124	Mean of 2–graph of Window events
125	Standard deviation of 2–graph of Window events
126	Skewness of 2–graph of Window events
127	Mean of 3–graph of Window events
128	Standard deviation of 3–graph of Window events
129	Skewness of 3–graph of Window events
130	Mean of 4–graph of Window events
131	Standard deviation of 4–graph of Window events
132	Skewness of 4–graph of Window events
133	Mean of 5–graph of Window events
134	Standard deviation of 5–graph of Window events
135	Skewness of 5–graph of Window events
136	Mean of 6–graph of Window events
137	Standard deviation of 6–graph of Window events
138	Skewness of 6–graph of Window events
139	Mean of 7–graph of Window events
140	Standard deviation of 7–graph of Window events
141	Skewness of 7–graph of Window events
142	Mean of 8–graph of Window events
143	Standard deviation of 8–graph of Window events

Continued on next page

ID	Feature Description
144	Skewness of 8–graph of Window events
145	Mean of 1–graph of Control events
146	Standard deviation of 1–graph of Control events
147	Skewness of 1–graph of Control events
148	Mean of 2–graph of Control events
149	Standard deviation of 2–graph of Control events
150	Skewness of 2–graph of Control events
151	Mean of 3–graph of Control events
152	Standard deviation of 3–graph of Control events
153	Skewness of 3–graph of Control events
154	Mean of 4–graph of Control events
155	Standard deviation of 4–graph of Control events
156	Skewness of 4–graph of Control events
157	Mean of 5–graph of Control events
158	Standard deviation of 5–graph of Control events
159	Skewness of 5–graph of Control events
160	Mean of 6–graph of Control events
161	Standard deviation of 6–graph of Control events
162	Skewness of 6–graph of Control events
163	Mean of 7–graph of Control events
164	Standard deviation of 7–graph of Control events
165	Skewness of 7–graph of Control events
166	Mean of 8–graph of Control events
167	Standard deviation of 8–graph of Control events
168	Skewness of 8–graph of Control events
169	Mean of 1–graph of Menu events
170	Standard deviation of 1–graph of Menu events

Continued on next page

ID	Feature Description
171	Skewness of 1–graph of Menu events
172	Mean of 2–graph of Menu events
173	Standard deviation of 2–graph of Menu events
174	Skewness of 2–graph of Menu events
175	Mean of 3–graph of Menu events
176	Standard deviation of 3–graph of Menu events
177	Skewness of 3–graph of Menu events
178	Mean of 4–graph of Menu events
179	Standard deviation of 4–graph of Menu events
180	Skewness of 4–graph of Menu events
181	Mean of 5–graph of Menu events
182	Standard deviation of 5–graph of Menu events
183	Skewness of 5–graph of Menu events
184	Mean of 6–graph of Menu events
185	Standard deviation of 6–graph of Menu events
186	Skewness of 6–graph of Menu events
187	Mean of 7–graph of Menu events
188	Standard deviation of 7–graph of Menu events
189	Skewness of 7–graph of Menu events
190	Mean of 8–graph of Menu events
191	Standard deviation of 8–graph of Menu events
192	Skewness of 8–graph of Menu events
193	Mean of 1–graph of Item events
194	Standard deviation of 1–graph of Item events
195	Skewness of 1–graph of Item events
196	Mean of 2–graph of Item events
197	Standard deviation of 2–graph of Item events

Continued on next page

ID	Feature Description
198	Skewness of 2–graph of Item events
199	Mean of 3–graph of Item events
200	Standard deviation of 3–graph of Item events
201	Skewness of 3–graph of Item events
202	Mean of 4–graph of Item events
203	Standard deviation of 4–graph of Item events
204	Skewness of 4–graph of Item events
205	Mean of 5–graph of Item events
206	Standard deviation of 5–graph of Item events
207	Skewness of 5–graph of Item events
208	Mean of 6–graph of Item events
209	Standard deviation of 6–graph of Item events
210	Skewness of 6–graph of Item events
211	Mean of 7–graph of Item events
212	Standard deviation of 7–graph of Item events
213	Skewness of 7–graph of Item events
214	Mean of 8–graph of Item events
215	Standard deviation of 8–graph of Item events
216	Skewness of 8–graph of Item events
217	Mean of 1–graph of Icon events
218	Standard deviation of 1–graph of Icon events
219	Skewness of 1–graph of Icon events
220	Mean of 2–graph of Icon events
221	Standard deviation of 2–graph of Icon events
222	Skewness of 2–graph of Icon events
223	Mean of 3–graph of Icon events
224	Standard deviation of 3–graph of Icon events

Continued on next page

ID	Feature Description
225	Skewness of 3–graph of Icon events
226	Mean of 4–graph of Icon events
227	Standard deviation of 4–graph of Icon events
228	Skewness of 4–graph of Icon events
229	Mean of 5–graph of Icon events
230	Standard deviation of 5–graph of Icon events
231	Skewness of 5–graph of Icon events
232	Mean of 6–graph of Icon events
233	Standard deviation of 6–graph of Icon events
234	Skewness of 6–graph of Icon events
235	Mean of 7–graph of Icon events
236	Standard deviation of 7–graph of Icon events
237	Skewness of 7–graph of Icon events
238	Mean of 8–graph of Icon events
239	Standard deviation of 8–graph of Icon events
240	Skewness of 8–graph of Icon events
241	Mean of 1–graph of Dialog events
242	Standard deviation of 1–graph of Dialog events
243	Skewness of 1–graph of Dialog events
244	Mean of 2–graph of Dialog events
245	Standard deviation of 2–graph of Dialog events
246	Skewness of 2–graph of Dialog events
247	Mean of 3–graph of Dialog events
248	Standard deviation of 3–graph of Dialog events
249	Skewness of 3–graph of Dialog events
250	Mean of 4–graph of Dialog events
251	Standard deviation of 4–graph of Dialog events

Continued on next page

ID	Feature Description
252	Skewness of 4–graph of Dialog events
253	Mean of 5–graph of Dialog events
254	Standard deviation of 5–graph of Dialog events
255	Skewness of 5–graph of Dialog events
256	Mean of 6–graph of Dialog events
257	Standard deviation of 6–graph of Dialog events
258	Skewness of 6–graph of Dialog events
259	Mean of 7–graph of Dialog events
260	Standard deviation of 7–graph of Dialog events
261	Skewness of 7–graph of Dialog events
262	Mean of 8–graph of Dialog events
263	Standard deviation of 8–graph of Dialog events
264	Skewness of 8–graph of Dialog events
265	Mean of 1–graph of Query events
266	Standard deviation of 1–graph of Query events
267	Skewness of 1–graph of Query events
268	Mean of 2–graph of Query events
269	Standard deviation of 2–graph of Query events
270	Skewness of 2–graph of Query events
271	Mean of 3–graph of Query events
272	Standard deviation of 3–graph of Query events
273	Skewness of 3–graph of Query events
274	Mean of 4–graph of Query events
275	Standard deviation of 4–graph of Query events
276	Skewness of 4–graph of Query events
277	Mean of 5–graph of Query events
278	Standard deviation of 5–graph of Query events

Continued on next page

ID	Feature Description
279	Skewness of 5–graph of Query events
280	Mean of 6–graph of Query events
281	Standard deviation of 6–graph of Query events
282	Skewness of 6–graph of Query events
283	Mean of 7–graph of Query events
284	Standard deviation of 7–graph of Query events
285	Skewness of 7–graph of Query events
286	Mean of 8–graph of Query events
287	Standard deviation of 8–graph of Query events
288	Skewness of 8–graph of Query events
289	Mean of 1–graph of Combo box events
290	Standard deviation of 1–graph of Combo box events
291	Skewness of 1–graph of Combo box events
292	Mean of 2–graph of Combo box events
293	Standard deviation of 2–graph of Combo box events
294	Skewness of 2–graph of Combo box events
295	Mean of 3–graph of Combo box events
296	Standard deviation of 3–graph of Combo box events
297	Skewness of 3–graph of Combo box events
298	Mean of 4–graph of Combo box events
299	Standard deviation of 4–graph of Combo box events
300	Skewness of 4–graph of Combo box events
301	Mean of 5–graph of Combo box events
302	Standard deviation of 5–graph of Combo box events
303	Skewness of 5–graph of Combo box events
304	Mean of 6–graph of Combo box events
305	Standard deviation of 6–graph of Combo box events

Continued on next page

ID	Feature Description
306	Skewness of 6–graph of Combo box events
307	Mean of 7–graph of Combo box events
308	Standard deviation of 7–graph of Combo box events
309	Skewness of 7–graph of Combo box events
310	Mean of 8–graph of Combo box events
311	Standard deviation of 8–graph of Combo box events
312	Skewness of 8–graph of Combo box events
313	Mean of 1–graph of Miscellaneous events
314	Standard deviation of 1–graph of Miscellaneous events
315	Skewness of 1–graph of Miscellaneous events
316	Mean of 2–graph of Miscellaneous events
317	Standard deviation of 2–graph of Miscellaneous events
318	Skewness of 2–graph of Miscellaneous events
319	Mean of 3–graph of Miscellaneous events
320	Standard deviation of 3–graph of Miscellaneous events
321	Skewness of 3–graph of Miscellaneous events
322	Mean of 4–graph of Miscellaneous events
323	Standard deviation of 4–graph of Miscellaneous events
324	Skewness of 4–graph of Miscellaneous events
325	Mean of 5–graph of Miscellaneous events
326	Standard deviation of 5–graph of Miscellaneous events
327	Skewness of 5–graph of Miscellaneous events
328	Mean of 6–graph of Miscellaneous events
329	Standard deviation of 6–graph of Miscellaneous events
330	Skewness of 6–graph of Miscellaneous events
331	Mean of 7–graph of Miscellaneous events
332	Standard deviation of 7–graph of Miscellaneous events

Continued on next page

ID	Feature Description
333	Skewness of 7–graph of Miscellaneous events
334	Mean of 8–graph of Miscellaneous events
335	Standard deviation of 8–graph of Miscellaneous events
336	Skewness of 8–graph of Miscellaneous events
337	Mean of distance of *all* GUI points
338	Standard deviation of distance of *all* GUI points
339	Skewness of distance of *all* GUI points
340	Mean of angle of *all* GUI points
341	Standard deviation of angle of *all* GUI points
342	Skewness of angle of *all* GUI points
343	Mean of speed of *all* GUI points
344	Standard deviation of speed of *all* GUI points
345	Skewness of speed of *all* GUI points
346	Mean of distance of Spatial+Temporal events
347	Standard deviation of distance of Spatial+Temporal events
348	Skewness of distance of Spatial+Temporal events
349	Mean of angle of Spatial+Temporal events
350	Standard deviation of angle of Spatial+Temporal events
351	Skewness of angle of Spatial+Temporal events
352	Mean of speed of Spatial+Temporal events
353	Standard deviation of speed of Spatial+Temporal events
354	Skewness of speed of Spatial+Temporal events
355	Mean of distance of Window events
356	Standard deviation of distance of Window events
357	Skewness of distance of Window events
358	Mean of angle of Window events
359	Standard deviation of angle of Window events

Continued on next page

ID	Feature Description
360	Skewness of angle of Window events
361	Mean of speed of Window events
362	Standard deviation of speed of Window events
363	Skewness of speed of Window events
364	Mean of distance of Control events
365	Standard deviation of distance of Control events
366	Skewness of distance of Control events
367	Mean of angle of single Control events
368	Standard deviation of angle of Control events
369	Skewness of angle of Control events
370	Mean of speed of Control events
371	Standard deviation of speed of Control events
372	Skewness of speed of Control events
373	Mean of distance of Menu events
374	Standard deviation of distance of Menu events
375	Skewness of distance of Menu events
376	Mean of angle of Menu events
377	Standard deviation of angle of Menu events
378	Skewness of angle of Menu events
379	Mean of speed of Menu events
380	Standard deviation of speed of Menu events
381	Skewness of speed of Menu events
382	Mean of distance of Item events
383	Standard deviation of distance of Item events
384	Skewness of distance of Item events
385	Mean of angle of Item events
386	Standard deviation of angle of Item events

Continued on next page

ID	Feature Description
387	Skewness of angle of Item events
388	Mean of speed of Item events
389	Standard deviation of speed of Item events
390	Skewness of speed of Item events

APPENDIX E
DATA COLLECTION ASSIGNMENT

The students were instructed to read "Java–based Internet Biometric Authentication System" by R. A. J. Everitt and P.W. McOwan [88] and answer the following questions:

1. How can we avoid pitfalls encountered with the traditional authentication systems?
2. What is the difference between the physiological and behavioral data?
3. Describe in your own words the characteristics and operation of a "hybrid" system.
4. In the context of the assigned reading material describe what keyboard and mouse inputs were required from each user.
5. How many stages did the authors' proposed solution have?
6. What is "registration"?
7. How did the authors handle noisy data?
8. Explain the "normalization" process.
9. What are the *hold* and *latency* times?
10. What is the difference between the *within* and *between* class variance?
11. Describe the difference between the *ranking* and *genetic* approach.
12. Explain the user's profile space.
13. What is the "boundary" space?
14. Why do we need a validation set?

15. Describe why the generalization is important.
16. What is FAR and FRR?
17. Explain how FAR and FRR were computed in the paper?
18. What results did the authors obtain?
19. What part of the paper did you like the best?
20. What part of the paper would you improve?

VITA

VITA

Maja Pusara

Research Interests Computer Security, Machine Learning and Data Mining.

Education

- **Ph.D.**, School of Electrical and Computer Engineering, Purdue University, August 2007.
- **M.S.**, School of Electrical and Computer Engineering, Purdue University, December 2003.
- **B.S.**, School of Electrical Engineering, Lake Superior State University, May 2001.
- **B.S.**, School of Computer Engineering, Lake Superior State University, August 2001.
- **B.S.**, School of Computer Science, Lake Superior State University, May 2001.
- **A.S.**, School of Engineering, Dodge City Community College, August 1998.
- **A.A.**, School of Engineering, Dodge City Community College, August 1998.

Honors

- **Valedictorian**, Graduating Class, Lake Superior State University, 2001
- **Recipient**, Outstanding Electrical and Computer Engineering Graduate, Lake Superior State University, 2001
- **Recipient**, Outstanding Computer Science and Mathematics Graduate, Lake Superior State University, 2001

- **Meritorious Winner**, Mathematical Modeling Contest (COMAP), 2001
- **Recipient**, Judge Fenlon Award for the Best Female Student, Lake Superior State University, 2000
- **Recipient**, American Society of Military Engineers Scholarship, Lake Superior State University, 2000
- **Recipient**, Gerald M. Samson Mathematical Scholarship, Lake Superior State University, 2000
- **Recipient**, Presidential Scholarship, Dodge City Community College, 1997
- **Recipient**, Endowment Scholarship, Dodge City Community College, 1997

Activities

- **Program Committee Member**, MIT Organization of Serbian Students (MOST), MIT, 2004–present
- **Volunteer**, Charitable Concert to Benefit War Orphans in the Balkans, MIT, 2005
- **Member**, Eta Kappa Nu Electrical Engineering Honorary Society, 2004–present
- **Treasurer**, Friends of Europe Society, Purdue University, 2002
- **Member**, Friends of Europe Society, Purdue University, 2001–2004
- **Student Representative**, Board to Select and Outstanding Faculty Member of the Year, 2001
- **Member**, Alpha Chi Engineering Honorary Society, 2000
- **President**, Society of Women Engineers (SWE), 1998 and 2000
- **Member**, Society of Women Engineers (SWE), 1998-present
- **Director**, Financial Committee, Lake Superior State University, 2000
- **Counselor**, Campus Connections, Lake Superior State University, 2000

- **Advisor**, Business and Management Program, Dodge City Community College, 1998

Publications

- Pusara, M. and Brodley, C. E., 2007. "Boosting performance when the amount of data is limited". Under Review for the *Proceedings of the 2007 IEEE International Conference on Data Mining (ICDM).*
- Pusara, M. and Brodley, C. E., 2007. "Dynamics of biometric sources for user re–authentication". Under Review for the *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS).*
- Pusara, M. and Brodley, C. E., 2007. "Analysis of mouse dynamics for user re–authentication". Under Review for the *ACM Transactions on Information and System Security (TISSEC).*
- Pusara, M. and Brodley, C. E., 2004. "User reauthentication via mouse movements". In Proceedings of the *2004 ACM CCS workshop on Visualization and Data Mining for Computer Security.*

Invited Presentations and Poster Sessions

- *Dynamics of Biometric Sources for User Re-authentication*, Poster Session for the Graduate Student Fair, Computer Science Department, Tufts University, October, 2005.
- *User Re-authentication via Mouse Movements*, CERIAS Information Security Symposium, Purdue University, March, 2004.
- *User Re-authentication via Mouse Movements*, Workshop on Statistical and Machine Learning Techniques in Computer Intrusion Detection, George Mason University, September 24-26, 2003.

Professional Activities

- Society of Women Engineers (SWE)
- Reviewer for the *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.*
- Reviewer for the *SIAM International Conference on Data Mining.*
- Reviewer for the *Journal of Distributed Systems Computing.*

Work Experience

- **Lecturer** September 2006 - May 2007
  Department of Mathematics, Tufts University Medford, MA
- **Research Assistant** August 2003 - Present
  Department of ECE, Purdue University W. Lafayette, IN
- **Teaching Assistant** August 2002 - August 2003
  Department of Mathematics, Purdue University W. Lafayette, IN
- **Research and Development Engineer** September 2000 - May 2001
  DaimlerChrysler Chelsea, MI
- **Resident Assistant** September 1999 - May 2001
  Lake Superior State University Sault Ste. Marie, MI
- **Process Control Intern** June 2000 - September 2000
  Mead Paper Escanaba, MI
- **Lab Instructor, Women in Robotics Program** June 1999 - July 1999
  Lake Superior State University Sault Ste. Marie, MI