Spring 5-21-2005

Computer Intrusion Detection Through Statistical Analysis and Prediction Modeling

Paul Evangelista
United States Military Academy, paul.evangelista@westpoint.edu

COMPUTER INTRUSION DETECTION THROUGH STATISTICAL ANALYSIS AND PREDICTION MODELING

By

Paul F. Evangelista

A Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF OPERATIONS RESEARCH AND STATISTICS

Approved:

_____

Mark J. Embrechts, Co-Thesis Adviser

_____

Boleslaw K. Szymanski, Co-Thesis Adviser

Rensselaer Polytechnic Institute
Troy, New York

April 2005
(For Graduation May 2005)

ii

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENT

I would like to briefly mention those who have helped me in completing this thesis and making it through the last two years of graduate study. First to my advisors, who have demonstrated unmatched patience and generously shared both their time and wisdom. Their insightful thoughts and guidance were invaluable. I would also like to acknowledge the United States Army and the Department of Systems Engineering at West Point. These organizations provided me with this opportunity to pursue graduate work, and I thank them for entrusting me to accomplish these goals. I would especially like to mention those who have been there every day, for better or for worse - my family. My two children, Alexander and Jacqueline, have the ability to make any person smile and quickly remember what is truly important. They are the meaning of joy and happiness. Most importantly, I would like to thank my wife Heather. She is the one who performs the thankless tasks every day, asking for nothing in return. She is an amazing mother, superb wife, and a brilliant woman. None of this would be possible without her selfless support and guidance.

ABSTRACT

Information security is very important in today's society. Computer intrusion is one type of security infraction that poses a threat to all of us. Almost every person in modern parts of the world depend upon automated information. Information systems deliver paychecks on time, manage taxes, transfer funds, deliver important information that enables decisions, and maintain situational awareness in many different ways. Interrupting, corrupting, or destroying this information is a real threat. Computer attackers, often posing as intruders masquerading as authentic users, are the nucleus of this threat. Preventive computer security measures often do not provide enough; digital firms need methods to detect attackers who have breached firewalls or other barriers. This thesis explores techniques to detect computer intruders based upon UNIX command usage of authentic users compared against command usage of attackers. The hypothesis is that computing behavior of authentic users differs from the computing behavior of attackers. In order to explore this hypothesis, seven different variables that measure computing commands are created and utilized to perform predictive modeling to determine the presence or absence of a attacker. This is a classification problem that involves two known groups: intruders and non intruders. Techniques explored include a proven algorithm published by Matthius Schonlau in [17] and several predictive model variations utilizing the aforementioned seven variables; predictive models include linear discrimination analysis, clustering, kernel partial least squares learning machines.

CHAPTER 1
INTRODUCTION

1.1 Statement of the Problem

Security poses a problem for today's society. Numerous threats from our environment aim to breach security, and since 9-11, the importance of vigilant security measures has escalated. There is a very serious threat that exists today that is silent, virtually transparent, and seemingly anonymous; this threat is computer intrusion. Society today depends on an uninterrupted flow of vital information. Malicious attackers intend to stop, interrupt, and alter this flow of information. As society grows increasingly reliant on computers, the importance of computer security grows.

There are numerous techniques used to enhance the security of information and computers. Many of these techniques attempt to block or deter attackers, preventing them from intruding in the first place. These techniques include many of our everyday encounters such as user-accounts and passwords, firewalls, and VPN encryption. All of these techniques aim to keep the wrong people out and the right people in. However, attackers continually strive to break through these barriers and invade protected information. These barriers are preventive measures, and digital firms must augment these preventive measures by implementing intrusion detection systems (IDS) that monitor the network and detect fraudulent or unusual activity.

This thesis explores a computer security problem known as host based intrusion detection, and in particular, masquerade detection. This is a binary classification problem that involves authentic users and low instances of masqueraders posing as authentic users. Utilizing captured command usage logs of UNIX users, statistics that measure these commands serve as a platform for classification. Some of the statistics created included recent published methods by other researchers, and several of the statistics are novel. The goal of this thesis involves:

1. Replicating work performed by other researchers with this data to capture proven successful techniques and statistics for the masquerade detection problem.

2. Merge these successful proven statistics with independently developed

statistics and explore interaction with multivariate statistics.

3. Applying proven cutting edge supervised classification models to achieve superior classification rates.

The purpose of a host based IDS is to detect malicious use of user privileges, often detected as novelties or anomalies. Certain characteristics of computer IDS define their effectiveness. The typical measures of an IDS involve accuracy of detection, often expressed as a true positive rating and false positive rating. The overall effectiveness of an IDS can be shown on a receiver operating characteristic (ROC) curve, which will be explained later in detail. A typical problem with intrusion detection involves minimizing false positive alarms while maintaining an acceptable rate of true positives. For example, if an IDS identifies 50% of users as intruders, but less than 10% of those identified are actual intruders, the digital firm will investigate the activities of one half of all users and consider suspending user accounts until the investigation is complete. Examining 50% of all user activities and creating frustration for 50% of all users is not acceptable for most digital firms, realizing that only a fraction of the 50% are attackers. Minimizing false positive ratings is a key criterion when evaluating an IDS, even while realizing that a lower true positive rating will occur.

Intrusion detection is inherently a statistical problem [5]. The problem involves collecting a sample of data, describing the data through statistical attributes, and then classifying the data based upon these attributes. Oftentimes intrusion detection involves identifying intruders without any training data and without any indication of their intrusion technique. The brand of intrusion detection explored in this project is driven by developing profiles of authentic users and developing methods to detect anomalies based upon a stream of new data that may or may not be from the authentic user. If the attributes of the new stream of data significantly strays from a particular user's typical patterns of behavior, then the models explored should indicate an intruder.

1.2   Recent Work

Schonlau et. al. [5, 6, 7, 17, 16] conducted the original work with the data examined in this thesis. Their contributions included a thorough analysis of several statistical techniques for identifying masqueraders. Schonlau et. al. explored approaches that include: Bayes one-step Markov model, hybrid multistep Markov model, text compression, Incremental Probabilistic Action Modeling (IPAM), sequence matching, and a uniqueness algorithm[5]. Schonlau stressed the importance of minimizing false positives, setting a goal of 1% or less for all of his classification techniques. Schonlau's uniqueness algorithm, explained in [17], achieved a 40% true positive rating before crossing the 1% false positive boundary. Wang [19] used one-class training based on data representative of only one user and demonstrated that it worked as well as multi-class training. Coull [4] applied bioinformatics matching algorithm for a semi-global alignment to this problem. Lee [12] built a data mining framework for constructing features and model for intrusion detection.

Roy Maxion contributed insightful work with this data that challenged both the design of the data set and previous techniques used on this data [14, 13]. Maxion uses a 1v49 approach in [14], where he trains a Naive Bayes Classifier one user at a time using the training data from one user as true negative examples versus data from the forty-nine other users (hence 1v49) as true positive (masquerader) examples. Maxion claimed the best performance to date in [14], achieving a true positive rating of 60% while maintaining a false positive rating of 1% or less. Maxion also examines masquerade detection with a similar data set that contain command arguments in [13].

1.2.1   The Necessity to Investigate Supervised Learning Approaches for Intrusion Detection

The research documented in this thesis differs from the abovementioned authors for several reasons. The prediction model or learning machine used for our experiments is Rosipal's Kernel Partial Least Squares (KPLS) [15]. As the title indicates, it is based upon the partial least squares techniques popularized in chemometrics. KPLS is an extremely efficient learning machine, especially for high dimensional

data, but it only applies to the supervised learning domain. The existing literature that explores the Schonlau et. al. (SEA) dataset exclusively considers unsupervised learning. Intrusion detection, and masquerade detection, should be considered as both a supervised and unsupervised learning problem. Supervised and unsupervised learning represent two domains of learning theory that should contribute and provide synergy to each other. The criticism for using supervised learning for intrusion detection, specifically in the masquerade detection scenario, is that we never know what the masquerader actually does, so it is impossible to train with true positive examples. The SEA dataset, along with other masquerade detection datasets that were constructed in a similar fashion, is a complete anomaly to this criticism. The SEA dataset utilizes surrogate masqueraders. The masquerading incidents within the SEA test data are nothing more than commands taken from another user's stream of authentic, non-malicious commands and probabilistically inserted (at a very sparse rate) in place of another user's authentic commands. Therefore, this dataset does not contain authentic masquerading data. The SEA dataset shortcomings also include a lack of context over objects as indicated by Maxion in [13]. However, the extensive research based upon this dataset proves that the SEA data poses a non-trivial problem. The underlying assumption surrounding the examination of these surrogate masqueraders is that techniques that work well with these surrogates should extend into the real domain.

The argument behind using a supervised learning approach with this dataset ties directly to this assumption. An assumption is that simulating true positive examples, or using existing true positive examples from this data set for training is valid. Therefore, I propose that a supervised learning machine can and should be applied with this data.

1.2.2 Schonlau's Analysis

Matthius Schonlau is the researcher who created the dataset analyzed in this thesis. Schonlau developed, analyzed, and compared numerous detection algorithms which have already been mentioned.

Particular attention is given to the uniqueness algorithm (which is based on

unpopular commands)[17], replicating this technique with several programs written in perl. Following the work of Schonlau accomplished two critical goals: by replicating his work and achieving the same results through new text mining programs, a validity check for the programs is accomplished. Secondly, his work with this dataset is comprehensive, and understanding and replicating his work provides a solid foundation of knowledge that enables further exploration of this dataset.

Shortly after finishing the initial programming and validity check with Schonlau's algorithm based upon unpopular commands[17], new techniques were explored. It is possible to measure this data in numerous ways and provide quantities that a learning machine can use to predict outcomes. After creating a new set of variables that measure user behavior, these variables serve as a basis for experimentation in predictive modeling with learning machines. The learning machine utilized in this article is Rosipal's K-PLS[15]. With numerous combinations of variables presented to the learning machine with different preprocessing techniques, exploration of improved classification begins.

CHAPTER 2
ANALYSIS OF THE DATA

2.1   Understanding the Dataset

Understanding the structure of the data is necessary before any programming or initial analysis can be conducted. Schonlau collected this data from an AT&T lab in New Jersey, observing the computing behavior of 50 users. The data set consists of 50 users with each user contributing a stream of 15,000 truncated UNIX commands. The user's data stream is further divided into blocks of 100 commands, thus creating 150 blocks of commands for each user. The first 50 blocks of data is training data only (contains no masquerading data), and the remaining 100 blocks of data contains masquerading data that appears based on a probability. Given that there are 50 users, this implies that in total there are 5,000 tuples of data in the non-intruded initial set and 10,000 tuples of data that potentially contain intrusion data. Since the true outcome of each of these subsequent 10,000 tuples is known, understand that there are only 256 intruded tuples. Only 2.56% of the data contains intruders, which makes this a very difficult problem. It is a very unbalanced classification problem. The objective is to determine if a block of data contains masquerading data or not.

The initial programming effort involved determining the data dictionary (list of every unique command). In addition to generating this list, the programs also report the frequency of the commands and the popularity of the commands (number of users who use distinct command). In an effort to validate the data dictionary, frequency, and popularity of the commands, I relied upon a graph that plots distinct commands vs. unique commands[17] and collaboration with Yongqiang Zhang based upon his work in [18].
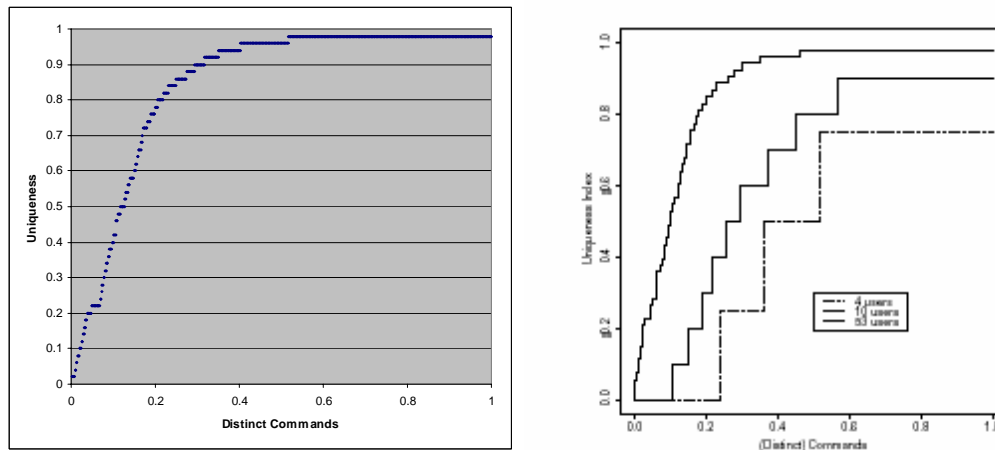
Figure 2.1: I plotted the above left graph based upon results from Perl programs that develop the data dictionary, frequency, and popularity of the commands (based upon all 50 users). The above right graph comes directly from Schonlau's analysis of the data[17]. As you can see, these two graphs (for approximately 50 users) are virtually identical.

The above graphs illustrate the uniqueness (1 - (total users of that command / total users)) of the distinct commands. As you can see, almost 50% of the distinct commands have a uniqueness of .98, meaning that almost 50% of the distinct commands are used by only one user. In addition to gleaning this useful analysis describing the popularity of the commands, replicating Schonlau's graph reinforces the validity of the Perl programs. In total, there are 856 distinct commands within the entire data set, and within the training data set (first 5000 commands for each user) there are 635 distinct commands.

2.2   Schonlau's Algorithm

Now that the structure of this data is understood, this section will explain how Schonlau utilized some of the variables and properties mentioned above to predict intrusion or non intrusion for each test data tuple[17]. He utilizes a creative algorithm for this. The algorithm truly formed the basis for this project, providing an example of a successful technique that predicts the presence or absence of an intruder.

$$x_u = \frac{1}{n_u} \sum_{k=1}^{K} W_{uk}(1 - \frac{U_k}{U})n_{uk} \quad , \tag{2.1}$$

where the weights W_{uk} are

$$W_{uk} = \begin{cases} \frac{v_{uk}}{v_k} & \text{if user } u\text{'s training data contains command } k \\ -1 & \text{otherwise} \end{cases}$$

where $v_{uk} = \frac{N_{uk}}{N_u}$ and $v_k = \sum_u v_{uk}$

Table 2.1: Description of Schonlau's variables

Name of Variable	Description
N_u	Number of commands in training data (5000)
n_u	Number of commands in test data (100)
N_{uk}	Number of times user uses command k in training data
n_{uk}	Number of times user uses command k in test data
U	Number of users (50)
U_k	Number of users who use command k
K	Number of distinct commands in training data (635)

A toy problem illustrating this algorithm is in Appendix A. The algorithm computation follows:

1. Preprocess the training data (initial 5000 commands from each user) to determine number of distinct commands, uniqueness of each command (number of users who use command k), and command frequency for population and each user.

2. For each tuple of test data, calculate x_u value (intruded data should score close to -1, authentic data close to 1).

3. Determine acceptable threshold (between -1 and 1) and calculate confusion matrix.

This technique is fundamentally different from typical machine learning methods; it is simply a fancy statistic. This technique calculates a test statistic x_u, which is based upon command history and uniqueness properties of the commands in the dataset. Schonlau utilizes both the command usage history of the particular user and the entire population to predict on the test data. The first 5000 tuples of

data are training data, and there is no intrusion present. This is the data utilized to develop the data dictionary, uniqueness, and frequencies. When the x_u value is calculated for a tuple of test data, the algorithm relies upon the uniqueness and frequency of commands seen from both the user and the population to determine each summed value. Unique commands never used by the user tend to drive the summation towards -1, where commands that are typical of the user drive the summation towards +1.

The algorithm achieves exceptional results considering the unsupervised nature of the learning. In order to better understand the algorithm and the dataset, I wrote several perl programs that created the necessary data dictionary and executed the above algorithm.

These programs will generate the x_u value, a number between -1 and 1 for each stream of 100 commands. The total output is an array of 5000 numbers between -1 and 1 (50 users, each user has 100 data-blocks (100 commands each) of test data, thus an array of length 5000). Based upon the algorithm, a user whose data-block contains commands similar to those used in the training data will tend to score high, whereas a data block containing commands previously unused by the user (especially if unpopular) will tend to score negative values. Once the array of 5000 test statistics is built, a single classification threshold can be determined in order to develop a confusion matrix. If the test statistic is less than the threshold, the program predicts an intruder; if the statistic is above the threshold, the program predicts no intruder. Figure 2 illustrates three different confusion matrices produced by the program that analyzes the 5000 test statistics.

The confusion matrix shows the performance of the IDS at a particular operating point, however an ROC curve is necessary to illustrate the overall performance of the IDS. In order to build the data for an ROC curve, the technique utilized generated results for 1600 incremental values of the classification threshold (increments of .001 from -.89 to .7). This essentially generated 1600 confusion matrices, and when listed in an array format one can easily calculate points along the ROC curve for the system. The below ROC curves represent the results of two algorithms that I analyzed (the second algorithm is a very slight modification of the original).

Figure 2.2: **The confusion matrices shown above were generated by a Perl program that analyzes the 5000 test statistics. From top to bottom, the corresponding classification thresholds are -.3, -.25, and -.28.**

ROC curves measure the false positive rating vs. the true positive rating (see Appendix B). In order to compare two ROC curves, the area under the curve (AUC) represents the comparative measure. Since an ROC curve represents a classification system, the AUC is a comparative measure for several classification systems. The AUC is an overall performance measure of a classifier, however it is important to understand that minimal false positives is paramount with IDS. I will illustrate with the best ROC curve attained that the AUC is equivalent to Schonlau's uniqueness algorithm AUC, however a false positive rate of 0 is maintained for much higher true positives. This indicates a better classifier. The next section describes the development of new variables that describe user behavior and explores the statistical nature and predictive power of these variables.

2.3 Programming in Perl

Several programs written in perl enabled the analysis of this dataset. perl is designed to manipulate and mine textual data, which was ideal for this dataset. There are six programs that formed the basis for the analysis of this dataset. Each user's commands existed in a single file, and the Perl programs opened these files and extracted the statistics necessary to describe user behavior and perform predictive modeling. The first program, *dictionarytrain*, analyzes the initial 5000 commands

AREA UNDER THE CURVE : 0.9490
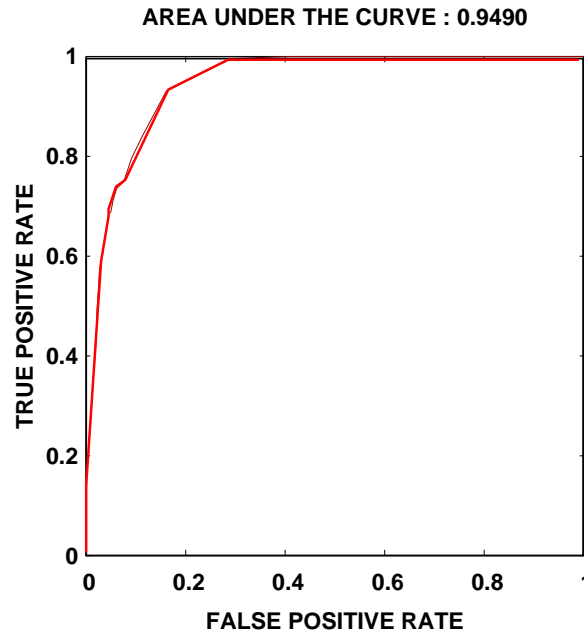
Figure 2.3: The above ROC curves represent results from the Perl programs written to execute and analyze the performance of Schonlau's uniqueness algorithm which is based on unpopular commands.

of each user and creates a data dictionary of distinct commands. The program also determines the overall frequency an popularity of commands. The program called *user_popularity* determines the number of times each user utilized each distinct command. There is also a program that calculates Schonlau's algorithm, and this program titled *algorithm_debug* creates a file that contains the x_u value for the 5000 tuples of test data. The next step involves analyzing the x_u values, the the program *results_analysis_debug* calculates the confusion matrices and the true positive / false positive ratings necessary to build the ROC curve. Chapter 3 discusses several new variables introduced to provide further insight and explore the possibility of building a more powerful prediction model. There are two programs that support the analysis of these new variables, titled *features* and *601features*. The below figure illustrates the flow of data and through the Perl programs, and Appendix C contains the source code of these programs.

RAW DATA
50 different files, each containing 15,0000 truncated Unix commands from a single user, provide the raw data necessary to begin the analysis. The first 5,000 commands of each file are known to be authentic, however the last 10,000 commands could contain the presence of an intruder.

PREPROCESSING
The *dictionarytrain* program reads every file and creates an output file that contains a dictionary of distinct commands and the frequency and popularity of each distinct command. The *userpopularity* program determines the number of times each user utilized each distinct command.

CALCULATING SCHONLAU'S ALGORITHM
The *algorithm_debug* program calculates Schonlau's algorithm, producing an *xu* value for each of the 5000 tuples of test data. The *results_analysis_debug* program calculated the confusion matrices and true positive / false positive ratings to build an ROC curve.

PREDICTION MODELING WITH NEW VARIABLES AND A K-PLS LEARNING MACHINE
The programs titled *features* and *601features* calculate variable for newly created variables, built largely from the statistics produced by *dictionarytrain* and *userpopularity,* and creates output files in *MetaNeural* format prepared to feed into the *Analyze* software for prediction modeling using a K-PLS learning machine.

Figure 2.4: Flow chart of Perl programs

CHAPTER 3
INTRODUCTION AND ANALYSIS OF NEW VARIABLES

3.1 The New Variables

After validating the Perl programs and developing a solid understanding of the dataset, new variables were necessary to create different techniques that accurately predict the presence or absence of an intruder. Several of these variables are also utilized in Schonlau's algorithm, but generally in a different manner. These variables were created with the goal of developing a technique to measure the behavior of computer users in a manner that captures the differences between intruders and authentic users. There are a number of techniques available to capture computer user behavior and/or network behavior. This technique is largely a function of the dataset utilized. Remember that the dataset contains truncated UNIX commands collected from fifty different users. Out of the 15,000 commands contributed from a user, the first 5,000 commands are untouched and the latter 10,000 contain intrusion data based upon a low probability. The new variables and the prediction models involving these new variables work primarily with this latter 10,000 commands from each user. Therefore, with each user contributing 100 tuples of data for this section, there are a total of 5,000 tuples that we will examine. Typically, these 5,000 tuples will be scrambled, divided into a training set and test set, and fed into a learning machine. These new variables we are discussing measure the overall characteristics of the 100 commands contained within the tuple, and these variables can be seen in Table 1.

An important aspect that remained true throughout the search for effective classification involved the importance of Mahalanobis scaling the data - essentially normalizing every entry in the data matrix - subtracting the mean and dividing by the standard deviation of the variable. This was due to the extreme ranges of different variables; with similar reasoning, the covariance matrix provides little useful information, however the correlation matrix provides a true measure of the

Table 3.1: Description of new variables

	Name of Variable	Description
1	% of Unix commands	Percent of commands that are Unix
2	% Top 20 most popular commands	Fraction of commands that are within the top 20 most popular commands in the data set
3	% of internet commands	Fraction of commands that are internet/email commands (sendmail and netscape)
4	Average Uniqueness	Averages Uniqueness (number between .02 and .98) for entire data set
5	Average Frequency	Averages the frequency of each command in the dataset
6	% of foreign commands	Percent of commands never seen before - this will be zero for all training data
7	x_u value	x_u value from Schonlau's algorithm[17], essentially providing a signature index that is based upon usage pattern of that particular user

interaction between variables.

3.2   Correlation Matrices

The correlation matrix is a scaled representation of the covariance; correlation is the scaled linear association between two variables eliminating the impact of units of measure. The correlation matrix shows that there is a definite relationship between the "% foreign commands" and the "% of UNIX commands" since the value of this coefficient is very close to -1. One other pair of variables, "% Top 20" and "Average Frequency", were highly correlated at a value of .7. In order to create a physical representation of the correlation matrix, an option in *Analyze*(computational intelligence software package written by Professor Mark Embrechts, RPI) produced the below colored representation of the correlation matrix. The highly correlated variables - coefficients that take on values close to 1 and -1 - were represented as darker colors, while the uncorrelated variables - coefficients that take on values close

to zero - were represented as light colors such as white. The software created a color coded matrix that was mostly light colored with a couple spots of darker colors, confirming the numerical observations of the correlation coefficients.

Correlation Matrix

	1	2	3	4	5	6	7
1	1	0.214	0.047	0.243	0.196	-0.998	0.018
2	0.214	1	0.298	0.391	0.686	-0.202	0.178
3	0.047	0.298	1	0.154	0.341	-0.049	-0.117
4	0.243	0.391	0.154	1	0.47	-0.245	-0.196
5	0.196	0.686	0.341	0.47	1	-0.194	0.056
6	-0.998	-0.202	-0.049	-0.245	-0.194	1	-0.011
7	0.018	0.178	-0.117	-0.196	0.056	-0.011	1
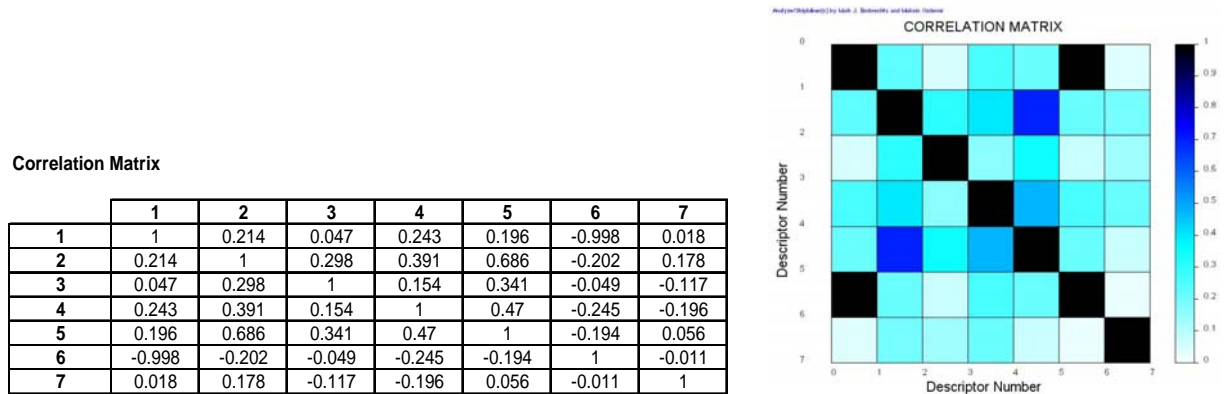
Figure 3.1: Numerical and Colored Correlation Matrices

3.3   Color Description of Variables

It is sometimes possible to predict a response directly from the values of the variables, without any learning machine or algorithm.
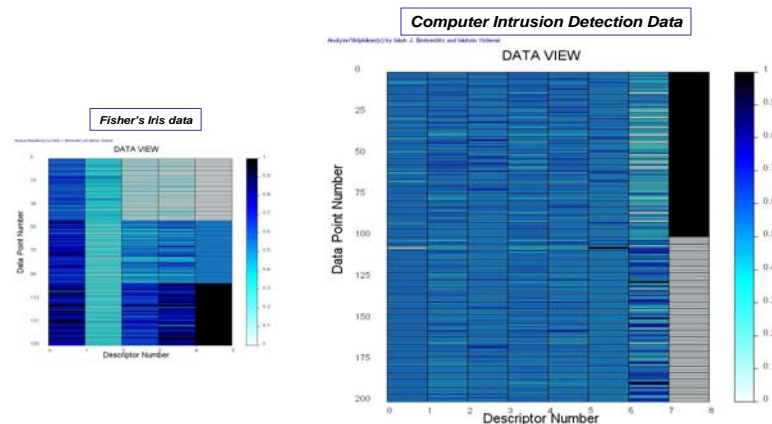
Figure 3.2: Colored Description of Variables

If a variable takes on a certain range of values for one response, and a separate range for another response, it is sometimes possible to infer with a good degree of certainty the response based on a single variable or multiple variables that have this

property. "Fisher's Iris data" is a great illustration of this point. Four predictor variables - petal length, petal width, sepal length, and sepal width are used to determine the type of iris. The *Analyze* software package is used to color code observations from the dataset.

In the Iris dataset it is apparent that the sepal length and width (first two variables) cannot be used to determine the type of iris since the colors do not have a distinct similarity with those in the response column. On the other hand, the colors generated for the petal length and width are distinctly similar to the response column. So we can say that petal length or petal width can be used to fairly accurately predict the type of iris. The computer intrusion dataset was sorted by the "Intrusion/ Non-Intrusion" column and color coded in the same way as the Iris data set. These new variables introduced for the computer intrusion detection problem do not exhibit this type of behavior.

3.4   Principal Component Analysis

The primary purpose of principal component analysis is to determine the utility in reducing the variables into linear combinations that explain the majority of variance from the dataset; this is essentially a technique in reducing dimensionality. Additionally, one can observe that principle components often naturally segregate or cluster groups within the data. This natural clustering or grouping was the purpose behind the principal component analysis conducted on this data set. It is possible that the score plot of the principal component loadings from a data set would naturally discriminate. Unfortunately, this phenomenon did not occur. As seen in the below graphs, the intrusion points (red) are centered near the centroid of all points. These plots do not provide good discrimination between intrusion and non intrusion.
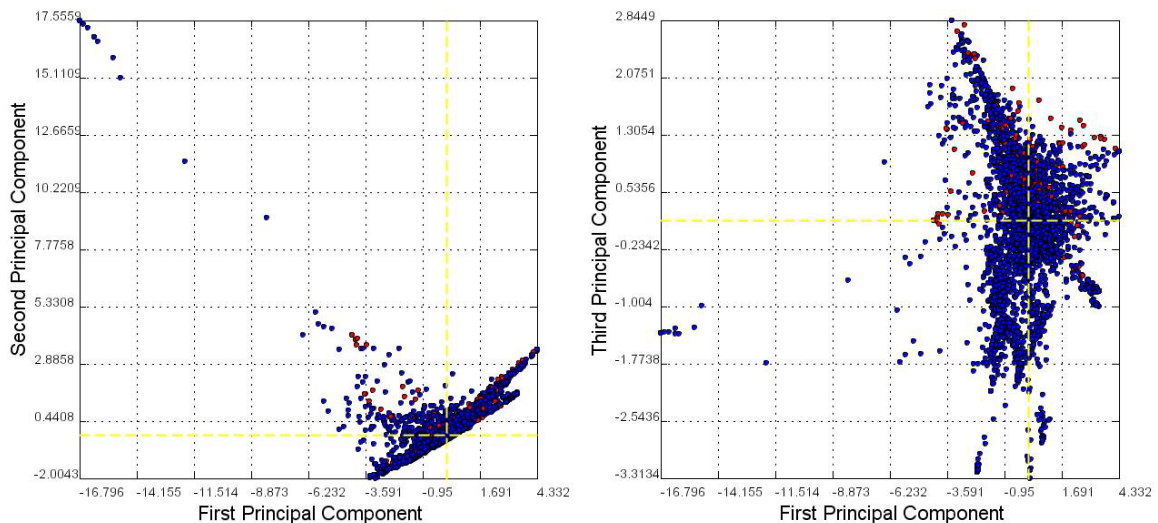
Figure 3.3: Principal component score plots for the 1st PC vs 2nd PC and 1st PC vs 3rd PC. Red dots indicate intruders, blue dots are authentic users.

3.5   Linear Discrimination Analysis

Prediction modeling was the next effort to follow the initial exploration of the data. An obvious technique to explore for the classification of two groups from a multivariate data set is discrimination analysis. We explore discrimination analysis

as a first step in prediction modeling of this data. If a simple linear discrimination model can sufficiently predict the groups of this data set, there may be no need to explore further. Secondly, linear discrimination analysis could expose characteristics of this data not seen before. Linear discrimination analysis proved more effective than quadratic discrimination analysis, with linear discrimination analysis consistently providing superior true positive and false positive rates. This is likely due to the fact that although the correlation matrices of these two groups are not identical, they are similar enough to gain significant predictive power through linear discrimination analysis. The linear discrimination rule can be succinctly stated as follows[11]:

Let $\bar{x}_1$ represent the mean vector of intruded training data, and let $\bar{x}_2$ represent the mean vector of non intruded training data. Allocate test data tuple x_0 to non intrusion if:

$$(\bar{x}_1 - \bar{x}_2)^T (S_{pooled}^{-1}) x_0 - \frac{1}{2}(\bar{x}_1 - \bar{x}_2)^T (S_{pooled}^{-1})(\bar{x}_1 - \bar{x}_2) \geq \ln w \qquad (3.1)$$

where
$$w = \frac{\text{cost}(1|2)}{\text{cost}(2|1)} \frac{\text{probability(no intrusion)}}{\text{probability(intrusion)}}$$

The general procedure for applying this discrimination rule involves utilizing a training set to develop the mean vectors and the pooled correlation matrix. The classification step proceeded with introducing a tuple from the test data and applying the above rule. The results from testing 1000 tuples of test data produces a confusion matrix, essentially itemizing the number of correct classifications vs. incorrect classifications in accordance with the groups. The below figure illustrates and example of a confusion matrix. (note: we use a binary value to denote an intruder or non intruder; a binary value of 0 represents no intrusion, where a value of 1 represents an intruder).

This particular confusion matrix represents the performance of the linear discrimination analysis when the probabilities for each type of group are equal (.5 for each group). We'll explain the significance of examining multiple ranges of proba-

	pred 0	pred 1
act 0	799	165
act 1	21	15

TP%   0.416667
FP%   0.171162

Figure 3.4: Confusion matrix for Linear Discrimination Analysis

bilities below. The above confusion matrix captures an important result from the classification procedure: it enables the calculation of true positive and false positive which forms the basis for comparing classification systems. We define true positive as correctly classifying an intruder (predict 1 when actually 1), and a false positive is defined is incorrectly classifying a non-intruder (predict 1 when actually 0).

The above discrimination rule enables classification for a certain operating point, where the costs and probabilities are fixed. However, comparing classification systems at one operating point does not provide sufficient comparison. The performance of classification systems must be considered across a sufficient range of operating characteristics, providing a thorough representation of the systems performance. Therefore, the range of consideration could be explored by varying the probability of intrusion and probability of no intrusion. Changing these probabilities by subtracting .1 from one and adding .1 to the other produced changes in the confusion matrix; this essentially changes the tolerance of the test, allowing analysis across a range of operating points. The result of this range of operating points is a Receiver Operating Characteristic Curve, or ROC curve, that plots false positive on the x axis and true positive on the y axis. Plotting the operating characteristics of several confusion matrices produces an ROC curve as shown below. A common method of evaluating a classification system is to measure the area under the curve (AUC).

The area under the curve for the Linear Discrimination Analysis classification system is .702, which illustrates an effective classification system. Throughout this report the area under the curve will serve as a measure to compare alternative classification systems.
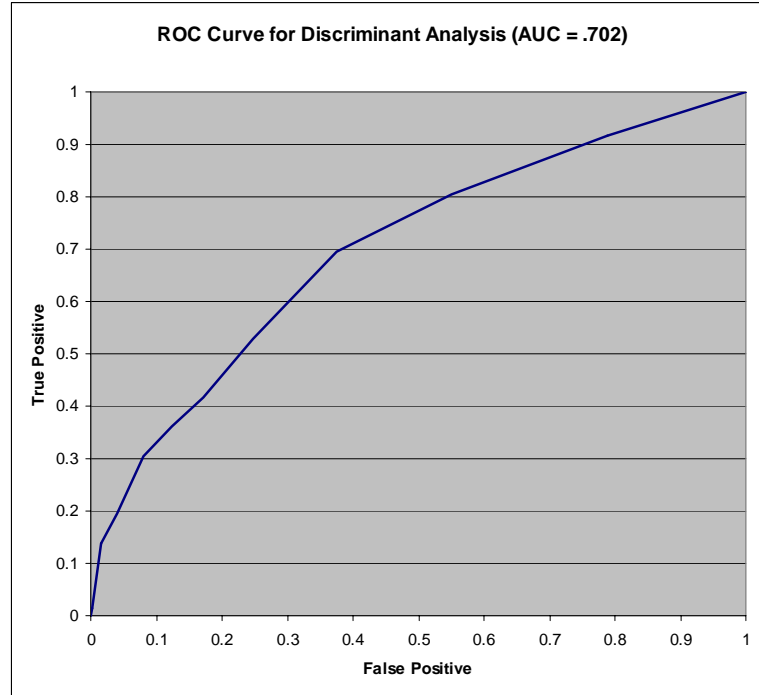
Figure 3.5: ROC Curve for Linear Discrimination Analysis

3.6   Clustering

Clustering is a technique commonly used for more than two groups or when the number of groups is unknown. However, due to the nature of clustering (determining a measure or method to separate groups), clustering can serve as an effective tool for discrimination. An initial clustering attempt applied K-means clustering to this data set, however poor results occurred. The next technique explored involved distance comparisons, using distance to the centroid of each group as a similarity measure. An initial attempt comparing Euclidean distances to the centroid of each group produced an interesting result that is commonly known as the curse of dimensionality. Every point in this seven dimensional hyperspace was almost equidistant from another point! The next obvious step involved reducing dimensionality by choosing only two variables that produced a good clustering of the groups. Through trial and error, we determined that the variable measuring the percent of internet commands and the x_u value from Schonlau's algorithm produced good separation between the groups. Euclidean distance measurements provided adequate classifi-

cation, however Minkoski's equation for distance provided the best classification, utilizing a value of 5 for m. For each tuple of data in the test data, we measure Minkoski's distance to the centroid observed in the training data, and classify into the group that measures the closest[11].

$$d(\mathbf{x}, \bar{\mathbf{x}}_{\mathbf{no\_int}}) = [\sum_{i=1}^{p} |x_i - \bar{x}_{no\_int}|^m]^{1/m} \tag{3.2}$$

$$d(\mathbf{x}, \bar{\mathbf{x}}_{\mathbf{int}}) = [\sum_{i=1}^{p} |x_i - \bar{x}_{int}|^m]^{1/m} \tag{3.3}$$

The above equations represent Minkoski's distance equation, where $\bar{x}_{int}$ represents the centroid of intrusion training data, and $\bar{x}_{no\_int}$ represents the centroid of non intrusion training data. p is the number of variables measured ($p = 2$ provided the best results).

Once again, it was important to collect a range of values for this classification tool in order to evaluate overall performance of the system. In order to obtain a range of confusion matrices, we introduced an offset value. Initially, if $d(\mathbf{x}, \bar{\mathbf{x}}_{\mathbf{int}})$ represents the distance between a test data point and the centroid of intrusion data, if $d(\mathbf{x}, \bar{\mathbf{x}}_{\mathbf{int}})$ was greater than $d(\mathbf{x}, \bar{\mathbf{x}}_{\mathbf{no\_int}})$ then the decision rule was to classify the point as a non intruder. However, by introducing an offset variable w, the new rule assigned a point as a non intruder if $d(\mathbf{x}, \bar{\mathbf{x}}_{\mathbf{no\_int}}) < d(\mathbf{x}, \bar{\mathbf{x}}_{\mathbf{no\_int}}) - w$, where w ranged from .2 to 1.2. This range of values for w produced an array of confusion matrices, again providing a method of plotting an ROC curve. The ROC curve for our clustering analysis with Minkoski's distance metric is shown below.

The area under the curve is .722, which shows a slight improvement from our discrimination analysis. Of particular interest in this ROC curve is the very steep slope for small values of the false positive. An important aspect of computer intrusion detection involves minimizing false positive. A false positive indicates that some authentic user has been targeted as a attacker. The implications of this could involve temporary suspension of user privileges until the situation is resolved or the employment of manual inspection to determine the accuracy of this classification (most intrusion detection systems pass all reports of an intruder on to an employee
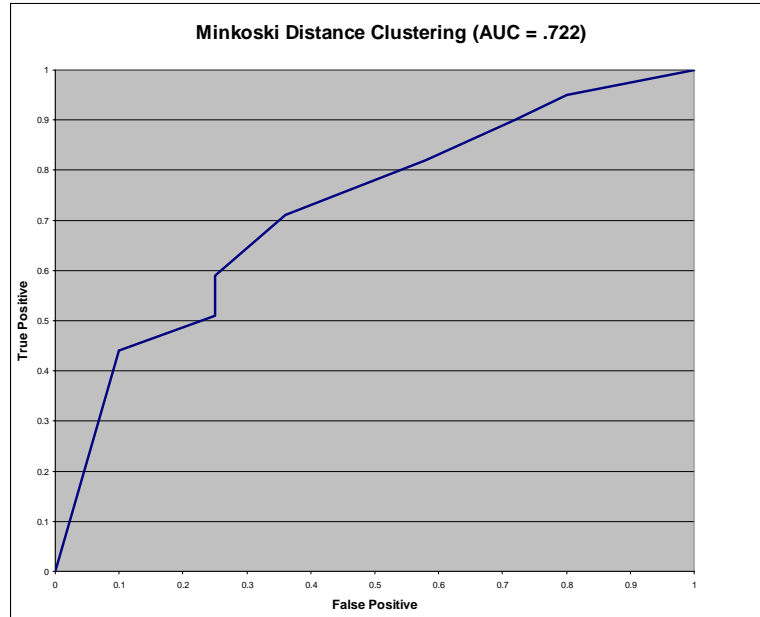
Figure 3.6: ROC Curve for Clustering with Minkoski's Distance Metric

who manually inspects the case; a high number of false positives is costly for this reason).

CHAPTER 4
APPLICATION OF KERNEL PARTIAL LEAST SQUARES

4.1   Kernel Partial Least Squares

Partial Least Squares Regression (PLS) was conceived by the Swedish statistician Herman Wold for econometrics modeling of multi-variate time series[20]. The first PLS publication was a sociology application in 1975[21]. His son, Svante Wold, applied PLS to chemometrics in the early eighties [23, 22] and currently PLS has become one of the most popular and powerful tools in chemometrics, mainly because of the quality of building models with many variables. PLS is not easy to explain and the mathematics involved is far from transparent. Partially for that reason PLS has a low emphasis in mainstream statistics and machine learning.

KPLS is a technique that has grown from partial least squares analysis. The study of partial least squares(PLS) is similar to principal components analysis (PCA).

PLS analysis considers the response vector (or matrix for multiple responses), typically denoted as $\mathbf{Y}$. PLS regression is a technique that maximizes latent variable correlation with the response vector. Therefore, the first latent variable (which is again a linear combination of the input variables), possesses maximum correlation with the response variable while remaining orthogonal to the remaining latent variables. Since the first few partial least squares components or latent variables capture the majority of correlation with the response variable, powerful prediction models result with desirable dimension reduction properties.

Rosipal explains in [15] how to extract these PLS components. Utilizing the NIPALs approach, this is the algorithm that Rosipal discusses:

1. randomly initialize $\mathbf{u}$

2. $\mathbf{w} = \mathbf{X}^T\mathbf{u}$

3. $\mathbf{t} = \mathbf{Xw}$, $\mathbf{t} \leftarrow \frac{\mathbf{t}}{\|\mathbf{t}\|}$

4. $\mathbf{c} = \mathbf{Y}^T\mathbf{t}$

5. $\mathbf{u} = \mathbf{Yc}, \mathbf{u} \leftarrow \frac{\mathbf{u}}{\|\mathbf{u}\|}$

6. continue to repeat steps 2-5 until $\mathbf{t}$ and $\mathbf{u}$ converge within a specified tolerance.

7. deflate $\mathbf{X,Y}$:

$$\mathbf{X} \leftarrow \mathbf{X} - \mathbf{tt}^T\mathbf{X}$$
$$\mathbf{Y} \leftarrow \mathbf{Y} - \mathbf{tt}^T\mathbf{Y}$$

After step 7, the first PLS component is found and the next PLS component can be extracted from the deflated $\mathbf{X}$ and $\mathbf{Y}$ matrices using the same algorithm.

At each full iteration (completion of step 7), store the $\mathbf{t,u,w}$ and $\mathbf{c}$ vectors. These vectors will create matrices $\mathbf{T,U,W}$ and $\mathbf{C}$ which will be used to complete the PLS regression model. Writing the typical regression model as

$$\mathbf{Y} = \mathbf{XB} + \mathbf{F}$$

where $\mathbf{B}$ is our regression matrix and $\mathbf{F}$ is the residual matrix, Rosipal shows in [15] the following:

$$\mathbf{B} = \mathbf{X}^T\mathbf{U}(\mathbf{T}^T\mathbf{XX}^T\mathbf{U})^{-1}\mathbf{T}^T\mathbf{Y}$$

The key to Kernel PLS (KPLS) is realizing the kernel matrix formed in the algorithm shown above between steps 2 and 3. The algorithm for KPLS is no different than what is shown for the PLS algorithm, except steps 2 and 3 combine to create:

$$\mathbf{t} = \phi\phi^T\mathbf{u}, \mathbf{t} \leftarrow \frac{\mathbf{t}}{\|\mathbf{t}\|}$$

ϕ represents the nonlinear function, or kernel function, that transforms the input variables into feature space. $\phi\phi^T$ is the well known kernel matrix. KPLS provides the attractive aspect of feature reduction while also combining the powerful similarity technique that exists within nonlinear kernels. Our typical choice for the

kernel function is the gaussian kernel, defined as:

$$K(x_i, x_j) = e^{\frac{\|x_i - x_j\|}{2\sigma^2}}$$

The free parameter σ is necessary with this kernel, requiring minimal tuning.

4.2   Seven Feature Model

The analysis of the new variables in chapter 3 provides insight into the relationship between the variables and the descriptive power of these variables. Exploring fundamental properties such as correlation, principal components, and colored descriptive plots often reveals dynamics amongst the variables that is simple but not intuitive or obvious. If a response could be predicted simply from one variable or a linear combination that reduces dimensionality, there is no need to use learning machines - it is a simple problem. Unfortunately, this is not a simple problem. Further analysis is required.

The $Analyze^{TM}$ software package is the primary vehicle for analysis from this point forward [8]. Previous analysis and preprocessing utilized Perl programs and some simple spreadsheet calculations, however now that all of the variables have been produced and represented in the MetaNeural format required by $Analyze^{TM}$ Analyze, predictive modeling can begin.

The most obvious method for prediction modeling given the set of new variables described is to simply represent each tuple of data with a combination of these seven variables. Let us call this the seven feature model. This is the initial approach. The file to be processed by the learning machine contained 5000 tuples of data, each containing 7 feature variables, the outcome (0 for non intrusion, 1 for intrusion), and an identification number. The identification number is a six digit number that uniquely identifies each tuple. Here is an example of a tuple of preprocessed data:

% Top 20 most popular commands	% internet commands	Average uniqueness	Average frequency	% of foreign commands	XU	Intrusion/Not Intrusion	ID#
0.61	0	0.8549	6309.74	0	-0.2784	0	143101

Figure 4.1: A example tuple of preprocessed data with the new variables

One final crucial step in data preprocessing consisted of the scaling of the data. During the previous discussion of principal components and the correlation matrix, I indicated that the relationship between variables is much more meaningful with scaled data. Learning machines also require scaled data to extract meaning and accurately learn and predict. Each variable, to include the response, was Mahalonobis scaled. After scaling the variables, the final step involved splitting the data into a training and testing set (4000 tuples were used for training, 1000 tuples for testing), and then the learning machines processed the information. Once the raw data is in MetaNeural format, Analyze operators perform and provide the scaling, splitting, and predictive modeling, and all necessary analytical results.

The ROC curve for the seven feature model is illustrated below. The AUC is .907.
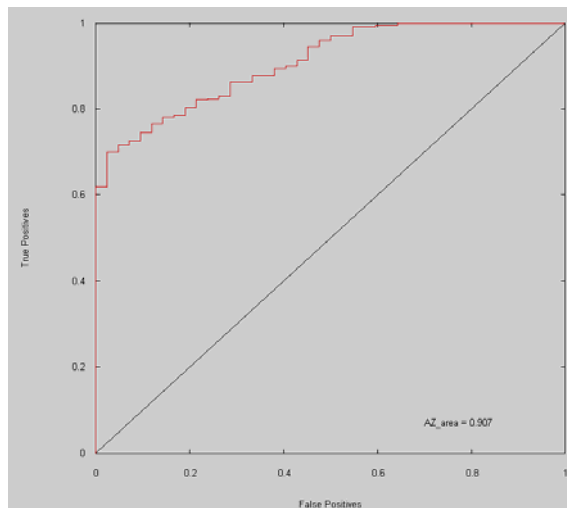
Figure 4.2: The ROC curve for the seven feature model. The AUC is .907.

4.3   601 feature model

In addition to the seven feature model, I expanded the detail of the features in an attempt to improve the prediction. Each variable in the seven feature model is an average or percentage for the entire tuple of 100 commands considered. The only exception to this is the xu value, which is a measure of the entire tuple. Therefore,

each command can be represented by six different values, each value representing variables one through six. The result of this approach is a tuple of 601 features (the extra feature is Schonlau's xu value). The results of this 601 feature model were surprisingly not better than the seven feature model. The 601 feature model was cumbersome because of its size, required a much longer processing time, and did not predict as well. I have two educated guesses why this model did not predict as well:

1. Consider each variable as some type of random variable that could be modeled with a probability density function. The original seven feature model contained averages and percentages - a function of a summation of each variable for each command of the tuple of 100 commands. This average or percentage represents a statistic, and it is likely that these statistics are functions of sufficient statistics, which implies that regardless of how the observations are manipulated, further information regarding the true parameters of the underlying probability density functions that form these statistics cannot be obtained. Therefore, improvement is not an option if the variables calculated are already functions of sufficient statistics. The theory of sufficient statistics is addressed in [1]

2. This file is too large and cumbersome for the KPLS learning machine. There is a curse of dimensionality. When the dimensionality grows too large, classification becomes very difficult because every tuple becomes equally dissimilar. Reduced and meaningful dimensions provide much more predictive power.

The discussion of the performance of Rosipal's KPLS with these two models introduces again the topic of the σ tuning parameter. This parameter greatly impacts the performance. This model utilizes Gaussian kernels. The Gaussian kernels represent a dissimilarity measure as seen in the below equation.

$$k_{ij} \equiv e^{-\frac{\|\vec{x_j} - \vec{x_l}\|^2}{2\sigma^2}} \tag{4.1}$$

The σ value in this equation is a tuning parameter - it needs to be adjusted to

create optimal performance. The tuning of this σ value proved critical for achieving optimal performance with Rosipal's KPLS learning machine. Consider the 601 feature model. Rosipal's KPLS achieved a number of different values as I modified the sigma value. The below table illustrates the impact of tuning sigma with all other conditions remaining constant. The table shows a range of sigma values and the observed AUC of the ROC curve.

Table 4.1: Illustration of the impact of the sigma value against the performance of the learning machine, measured here through the AUC

σ **value**	**Area under the curve (AUC)**
1	.6505
2	.7036
3	.7084
4	.6954
5	.7443
6	.7738
8	.6956
9	.7383
10	.7658
15	.7653
30	.7545
100	.7745
500	.7379

Although this table illustrates only integer values for sigma, sigma can actually take any real number value. The purpose of the table is to illustrate the impact of changes in sigma. The table also illustrates how much worse the 601 feature model performed vs. the seven feature model. As you can see, the best AUC for the 601 feature model was .7745.

4.4   IMPROVING THE MODEL

The previous analysis indicates that KPLS is a robust learning machine that captures all of the relevant information presented by the variables, regardless of the form of the variables presented. This is similar to the logical argument regard-

ing sufficient statistics previously mentioned. In order to improve the performance of the model, the data needed to improve with the addition of relevant variables that uncover information that is not contained in the previous variables. Two new variables are presented.

A simple transition probability matrix model can be constructed to capture the likelihood of a user transitioning from one command to the next based upon previous behavior. For every user, given that there are 635 distinct commands, a transition matrix, $\mathbf{T} \in \mathbb{R}^{635 \times 635}$, is created. Allowing t_{ij} to represent the ith row and j^{th} column, this value is equivalent to the probability of transitioning from the ith command to the jth command. Given the kth test tuple of $l = 1...100$ commands, the probability can be calculated as the following:

$$P_k = \prod_{l=1}^{100} t_l \text{ where } t_l = t_{ij} \text{ when } i = l - 1, j = l \tag{4.2}$$

For each test tuple we calculate this variable that we will refer to as the *transition* variable. This variable is different from previous variables created in that it measures patterns; the order of the commands presented matters.

The other variable created is a very simple measurement inspired by the x_u variable. Referred to as the *new* variable, it is a count of how many new commands are in the test tuple that have never been used by the user. Ranging from 1 to 100, this variable has a surprising predictive power.

This variable by itself performs better as a single scalar predictor than any of the previously mentioned techniques with the exclusion of the x_u variable. This certainly reinforces the cliche of "garbage in - garbage out". However, it is interesting to see what occurs when this variable is combined with the others. The AUC jumps to .95, which is equivalent to the x_u variable, however notice the value of the true positive axis when the false positive rate increases from 0. The true positive rate is slightly greater than .7 at this point. This illustrates the caution that is necessary when considering the AUC as the overall performance measure. The AUC is largely considered the best scalar performance measure for binary classifiers [2, 9]. However, caution must be taken before selecting this as the one and only
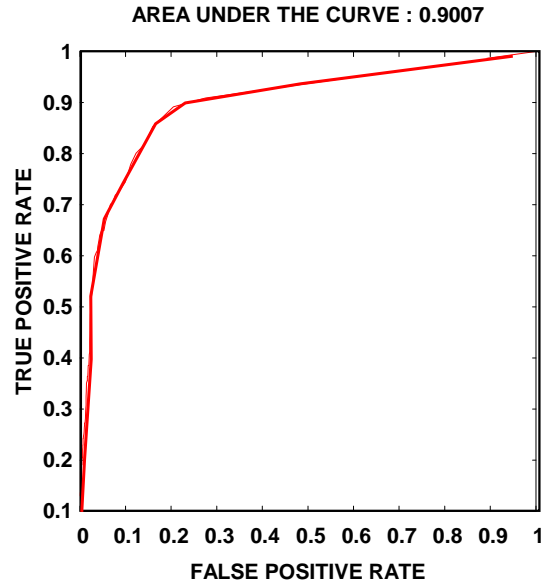
Figure 4.3: ROC curve illustrating the predictive power of solely the new variable

measure of performance. If minimizing false positives is important, a false positive boundary can be chosen, such as FP = 1% as indicated by Schonlau in [5]; the IDS is measured by the true positive rate that it achieves before crossing the false positive threshold of 1%.
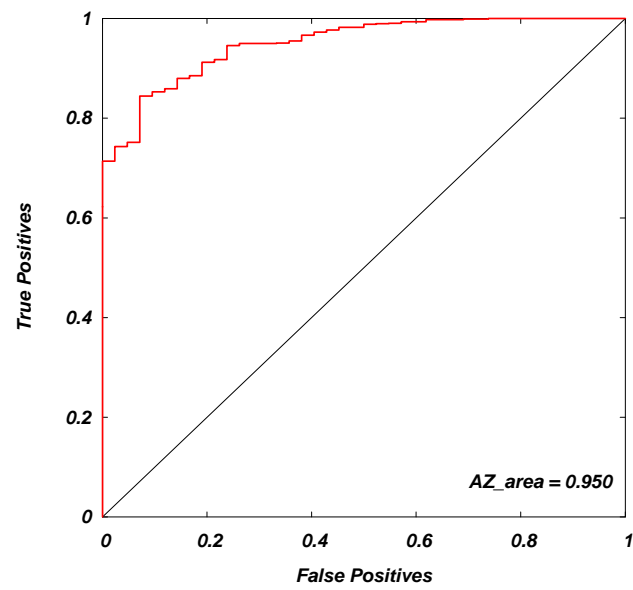
Figure 4.4: ROC curve of the performance achieved using every available variable

CHAPTER 5
CONCLUSION AND FUTURE WORK

5.1 Conclusion

This thesis addresses a unique dataset that provides opportunity for the application of data mining, statistics, and state of the art learning methods. Fundamental theoretical concepts of multivariate statistics have been discussed, and advanced learning methods such as Kernel Partial Least Squares demonstrated remarkable performance. Regardless of the models utilized and manipulation of statistics, it was also reinforced that independent statistics with strong correlation with the dependent variable consistently improve the model. The ROC curve is an invaluable tool for the measurement of an IDS. It is an elegant, concise description of performance. The area under the ROC curve (AUC) is an excellent scalar measurement of the performance of a binary classifier, however the AUC has limitations. Two curve of equivalent AUC may not represent equivalent classifiers in the eyes of a decision maker. If maximizing performance in the low false positive range is a priority, or perhaps maximizing true positives is a priority as it is in the medical field, other metrics must be considered.

The significance of research in this field expands well beyond the digital realm and computer intrusion detection. Intrusion detection is a security problem. This security could involve an airport, a national asset, vital infrastructure, or even the physical borders of our country. Every day Americans spend millions of dollars to secure their way of life, and this research presents innovative techniques to provide enhanced security. Although the vehicle of this research involves computer intrusion and attackers, these same techniques could be generalized to enhance the security of any important resource.

5.2 Future Work

An approach yet to be explored extensively involves combining several ROC curves in an attempt to achieve a synergistic effect that results in optimal perfor-

mance beyond the capabilities of any single classification system.

An overall system of IDS that generates a synergistic curve from component systems.

TRUE POSITIVE

FALSE POSITIVE

1

1
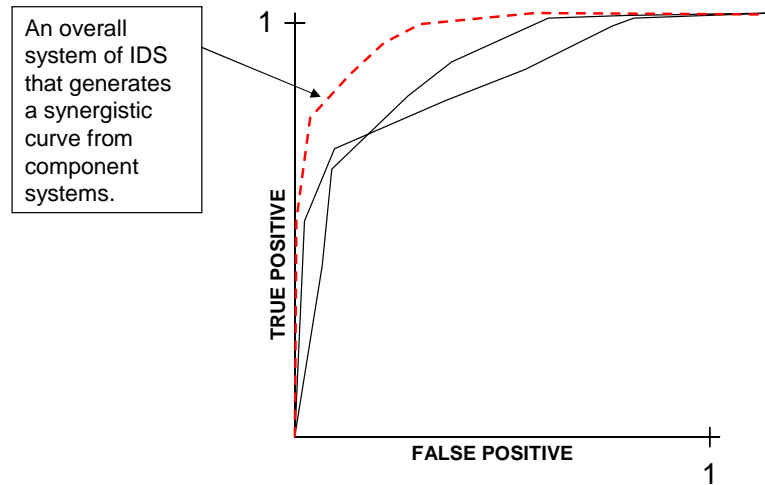
Figure 5.1: The above graph represents three ROC curves. The two black curves are component algorithms or component systems, and the dashed ROC curve represents a system that combines the component systems and generates a synergistic effect with improved performance.

The above illustrated approach differs from several alternatives that have been explored by others. This chart, showing multiple ROC curves and an optimal synergistic ROC curve, represents a proposal to search for an optimal classification system by combining several systems. The vehicle for this research is intrusion detection systems, however the potential application is much broader.

Utilizing multiple IDS to devise a superior hybrid system is not a new idea. Tom Fawcett and Foster Provost explored the concept of an ROC convex hull. This approach essentially maps several ROC curves onto one graph, and depending upon the tolerance accepted by the digital firm, the optimal operating position is derived from the convex hull of the ROC curves [10]. Huseyin Cavusoglu, Birendra Mishra, and Srinivasan Raghunathan also analyze several IDS as an overall system, and they describe optimal operating conditions based upon variables devised from a game theory approach[3]. This game theory approach views the digital firm and the attacker partaking in a game where they each stand to gain or lose, and based upon the cost benefit ratio of both the digital firm and the attacker, there is an optimal operating condition for the overall system. Both of these approaches mentioned

above combine IDS algorithms in a parallel fashion, which differs from the in series approach that I will take. Several IDS algorithms placed in series could also be viewed as several filters placed in series. Each filter will classify data, but subsequent filters will only classify data passed from previous filters. Any filter has the authority to classify data as containing negative intrusion and authentic, and if this is the case that user is classified as authentic and will not be analyzed any further. However, if a filter classifies data as containing an intruder then the next filter will also analyze this data.

REFERENCES

[1] Lee J. Bain and Max Engelhardt. *Introduction to Probability and Mathematical Statistics*. Duxbury, second edition, 1991.

[2] Andrew P. Bradley. The Use of the Area Under the ROC Curve in the Evaluation of Machine Learning Algorithms. *Pattern Recognition*, Volume 30 (7):1145–1159, 1997.

[3] Huseyin Cavusoglu, Birendra Mishra, and Srinivasan Raghunathan. Configuration of Intrusion Detection Systems. Seattle, December 2003. International Conference on Information Systems (ICIS).

[4] Scott Coull, Joel Branch, and Boleslaw K. Szymanski. Intrusion Detection: A Bioinformatics Approach. Las Vegas, Nevada, December 2001. Proceedings of the 19th Annual Computer Security Applications Conference.

[5] William DuMouchel, Wen Hua Ju, Alan F. Karr, Matthius Schonlau, Martin Theus, and Yehuda Vardi. Computer Intrusion: Detecting Masquerades. *Statistical Science*, 16(1):1–17, 2001.

[6] William DuMouchel and Matthius Schonlau. A Fast Computer Intrusion Detection Algorithm Based on Hypothesis Testing of Command Transition Probabilities. pages 189–193. The Fourth International Conference of Knowledge Discovery and Data Mining, August 1998.

[7] William DuMouchel and Matthius Schonlau. A Comparison of Test Statistics for Computer Intrusion Detection Based on Principal Components Regression of Transition Probabilities. pages 404–413. Proceedings of the 30th Symposium on the Interface: Computing Science and Statistics, 1999.

[8] Mark J. Embrechts. *Analyze^{TM}*, Version 6.86, Rensselear Polytechnic Institute. http://www.drugmining.com, Accessed 4 June, 2004.

[9] Tom Fawcett. ROC Graphs: Notes and Practical Considerations for Data Mining Researchers. Palo Alto, CA, 2003. Technical Report HPL-2003-4, Hewlett Packard.

[10] Tom Fawcett and Foster Provost. Robust Classification for Imprecise Environments. *Machine Learning Journal*, 42(3):203–231, 2001.

[11] Richard A. Johnson and Dean W. Wichern. *Applied Multivariate Statistical Analysis*. Prentice Hall, fifth edition, 2002.

[12] Wenke Lee and Salvatore J. Stolfo. A Framework for Constructing Features and Models for Intrusion Detection Systems. *ACM Transactions on Information and System Security (TISSEC)*, 3(4):227–261, 2000.

[13] Roy A. Maxion. Masquerade Detection Using Enriched Command Lines. San Francisco, CA, June 2003. International Conference on Dependable Systems and Networks.

[14] Roy A. Maxion and Tahlia N. Townsend. Masquerade Detection Using Truncated Command Lines. Washington, D.C., June 2002. International Conference on Dependable Systems and Networks.

[15] Roman Rosipal and Leonard J. Trejo. Kernel Partial Least Squares Regression in Reproducing Kernel Hilbert Space. *Journal of Machine Learning Research*, 2:97–123, 2001.

[16] Matthius Schonlau and Martin Theus. Intrusion Detection Based on Structural Zeroes. *Statistical Computing and Graphics Newsletter*, 9(1):12–17, 1998.

[17] Matthius Schonlau and Martin Theus. Detecting Masquerades in Intrusion Detection Based on Unpopular Commands. *Information Processing Letters*, 76(1-2):33–38, 2000.

[18] Boleslaw K. Szymanski and Yongqiang Zhang. Recursive Data Mining for Masquerade Detection and Author Identification. West Point, NY, 9-11 June 2004. 3rd Annual IEEE Information Assurance Workshop.

[19] Geoffrey I. Webb and Zijan Zheng. Multi-Strategy Ensemble Learning: Reducing Error by Combining Ensemble Learning Techniques. *IEEE Transactions on Knowledge and Data Engineering*, 16(8):980–991, 2004.

[20] Herman Wold. Estimation of Principal Components and Related Models by Iterative Least Squares. In P. R. Krishnaiah, editor, *Multivariate Analysis*, pages 391–420. Academic Press, NY, 1966.

[21] Herman Wold. Path Models with Latent Variables. In H.M. Ballock, editor, *Quantitative Sociology: International Perspectives on Mathematical and Statistical Model Building*, pages 307–357. Academic Press, NY, 1975.

[22] Svante Wold. Personal Memories of the Early PLS Development. *Chemometrics and Intelligent Laboratory Systems*, 58:83–84, 2001.

[23] Svante Wold, Michael Sjlstrm, and Lennart Erikson. PLS Regression: A Basic Tool of Chemometrics. *Chemometrics and Intelligent Laboratory Systems*, 58: 109–130, 2001.

APPENDIX A

Tutorial in Calculating the x_u Value

A.1   A Toy Problem with Five Commands

The calculation of Schonlau's x_u is not entirely intuitive. It is also helpful to discuss why this statistic is so powerful for anomaly detection and exactly where this predictive power is contained. In chapter 2, we presented the x_u value as follows:

$$x_u = \frac{1}{n_u} \sum_{k=1}^{K} W_{uk}(1 - \frac{U_k}{U})n_{uk} \quad , \tag{A.1}$$

where the weights W_{uk} are

$$W_{uk} = \begin{cases} \frac{v_{uk}}{v_k} & \text{if user } u\text{'s training data contains command } k \\ -1 & \text{otherwise} \end{cases}$$

where $v_{uk} = \frac{N_{uk}}{N_u}$ and $v_k = \sum_u v_{uk}$

Table A.1: Description of Schonlau's variables for Toy Problem

Name of Variable	Description
N_u	Number of commands in training data (5000)
n_u	Number of commands in test data (5)
N_{uk}	Number of times user uses command k in training data
n_{uk}	Number of times user uses command k in test data
U	Number of users (50)
U_k	Number of users who use command k
K	Number of distinct commands in training data (635)

Now let us consider an example that contains only five commands in the test tuple, as reflected above. These commands are *cpp, sh, xrdb, cpp, sh*, in that order. There are three distinct commands within these five, and we can begin to calculate the necessary variables as shown in figure A.1.

In calculating W_{uk}, notice that the behavior of the other users influences the

calculation. This is shown in figure A.2. This is a consistent theme with Schonlau's uniqueness approach. The power of the approach comes from measuring one users behavior not only against her own previous behavior but also against the behavior of the population. A strong penalty occurs if a command that has never been used by a user appears in the test data. We will see in step three how this penalty is magnified or dampened depending on the popularity of the new command seen.

Figure A.3 illustrates how the popularity of a command influences the calculation. Popular commands receive minimal weight, assuming that it is more likely for a user to suddenly start using a popular command rather than an unpopular command. Notice from the calculation how a string of unpopular commands never used by a user can drive the x_u value in the negative direction.

The final step involves a simple summation. This short toy problem has a positive value, indicating an authentic user, which it is.

a. **Tuple is: cpp, sh, xrdb, cpp, sh**
   3 distinct commands; user 1's first 5
   commands.

	cpp	sh	xrdb
N_{uk}	38	394	38
n_{uk}	2	2	1
U_k	45	49	43

Figure A.1: Step 1 of the toy problem

b. **Calculate W_{uk} for each command:**
   Consider cpp:
   $W_{uk} = (N_{uk} / N_u) / \sum_u v_{uk}$
   $W_{uk} = (38 / 5000) / (2225 / 5000) = .01708$

   Notice from algorithm that if a distinct
   command is not used, null effect…if
   user never used command in training,
   but it shows up in test data, $W_{uk} = -1$

	cpp	sh	xrdb
N_{uk}	38	394	38
n_{uk}	2	2	1
U_k	45	49	43
W_{uk}	.01708	.01809	.06518

Figure A.2: Step 2 of the toy problem

c. Let $y = W_{uk} (1 - (U_k/U)) n_{uk}$ (for notation)
Calculate y for each distinct command:
Consider cpp:
$y = .01708 (1 - (45 / 50)) 2 = .00342$

Notice from algorithm that if many users use command, impact minimal…

	cpp	sh	xrdb
N_{uk}	38	394	38
n_{uk}	2	2	1
U_k	45	49	43
W_{uk}	.01708	.01809	.06518
y	.00342	.00072	.00913

Notice that although xrdb is only used once, the impact is the most significant…this is because xrdb is more unique than other commands

Figure A.3: Step 3 of the toy problem

	cpp	sh	xrdb
N_{uk}	38	394	38
n_{uk}	2	2	1
U_k	45	49	43
W_{uk}	.01708	.01809	.06518
y	.00342	.00072	.00913

d.
$x_u = 1/5 (\sum y) = .002654$

Figure A.4: Step 4 of the toy problem

APPENDIX B
RECEIVER OPERATING CHARACTERISTIC (ROC) CURVES

B.1 The Confusion Matrix and Hypothesis Testing Definitions

Confusion matrices, ROC curves, and hypothesis testing have very much in common. Many of the terms used with these tools are synonymous, and often these terms are misunderstood. The below matrix which illustrates the four regions of a typical hypothesis testing problem defines the applicable terms.

DECISION: STATE OF NATURE:	Reject H_0 (Predict No Intrusion)	Do Not Reject H_0 (Predict Intrusion)
H_0 is False **(No Intrusion)**	Good Probability = $1 - \beta$ Frequently called **Power** *(The medical community refers to this as **specificity**.)*	Bad – Type II Error Probability = β *(This will be referred to as a **False Positive** – medical community often plots this on the x axis as 1-specificity.)*
H_0 is True **(Intrusion)**	Bad – Type I Error Probability = α	Good Probability = $1 - \alpha$ Frequently called **Confidence** *(This will be referred to as **True Positive** when discussing ROC curves; an alternative term is **sensitivity**, typically used by the medical community.)*

Figure B.1: Hypothesis testing regions and terms

As seen in the above figure, there are numerous terms used to describe the performance of a classification system. The medical community uses only specificity and sensitivity. For our purposes, we will use false positive and true positive.

The confusion matrix stems directly from the above discussion. A confusion matrix represents the outcome of an attempt to classify known groups. When dealing with an IDS, there are only two groups: intruders or non intruders. Therefore,

a confusion matrix for an IDS would contain four regions, representing the four potential types of classification. The below confusion matrix illustrates this point.

PREDICTED: ACTUAL:	0 (Negative)	1 (Positive)
0 (Negative)	a	b
1 (Positive)	c	d

Figure B.2: Hypothesis testing regions and terms

From a confusion matrix, one can calculate both the false positive rating and the true positive rating. These calculations follow:

$$TRUE\ POSITIVE: \quad \frac{d}{c+d}$$

$$FALSE\ POSITIVE: \quad \frac{b}{a+b}$$

B.2   The ROC Curve

A Receiver Operating Characteristics Curve is a very complete, simple, and elegant way to display the performance of a classification system. An ROC curve is a graphic representation of the relationship between the probability of a true positive outcome (sensitivity, 1-α error) and the probability of a false positive outcome (Type II error(β) or 1-specificity). The earliest use of ROC curves can be traced back to World War II during initial implementation of radar systems. It was very important for radar systems to accurately detect aircraft. Radar systems could be set to a very high and sensitive level, where every aircraft would definitely be detected, however this would result in a tremendous number of false positives. Imagine the number of false air alarms that could have been sounded over Great Britain if the Allied Forces did not carefully manage the sensitivity setting of their radar systems! ROC curves represent reality, where perfect detection or classification is usually not a possibility. Users must determine classification settings, depending on a cost-benefit analysis of what they are willing to accept as a true positive rating and a false positive rating.

An ROC curve is a representation of the entire range of operating points that a classification system has the capability to attain, and typically the user has the flexibility to decide where to operate on the curve. The overall curve reflects the quality of the classification system. The Area under the curve (AUC) is typically used as method of comparing alternate ROC curves; the better ROC curve typically has more AUC.
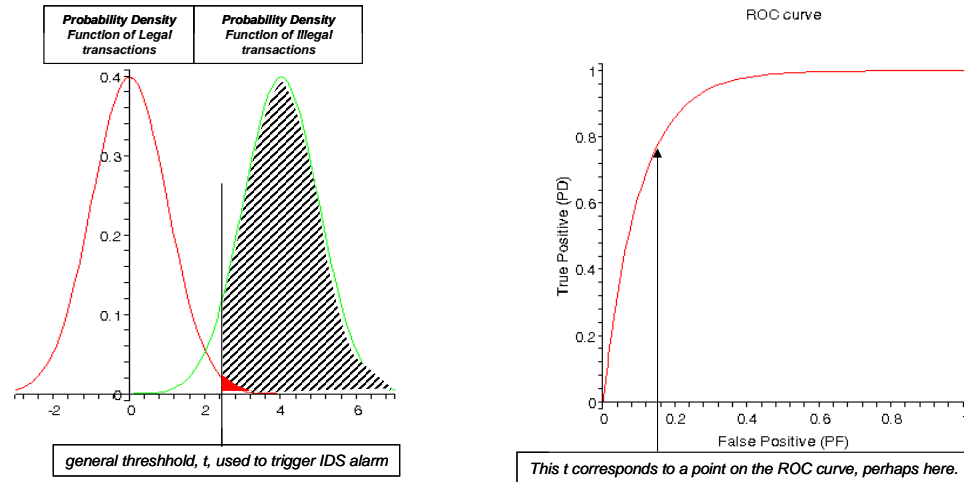
Figure B.3: The plot on the left shows the PDFs of legal and illegal transactions (non intruders and intruders), respectively. The ROC curve on the right shows the possible operating point represented by the tolerance threshold shown on the PDF plots.

The plot of the PDFs illustrates the idea of false positives and true positives. The hashed area represents the probability of a true positive, and the small red area represents a false positive. Imagine shifting the threshold, t, to the left. The true positive rating would definitely increase, but so would the false positive. The user must identify what point on the curve is acceptable, and this is usually accomplished through a cost-benefit analysis.

APPENDIX C
Perl Programs

C.1   Introduction to the Programs

This appendix contains the source code from the six primary Perl programs that the calculations necessary to analyze both Schonlau's algorithm and prediction models using the new variables. The following table briefly describes each program.

Table C.1: Description of Perl Programs

Name of Program	Description
dictionarytrain	This program analyzes each users initial 5000 commands and creates a data dictionary of distinct commands. This program also determines the overall frequency and popularity of commands.
user_popularity	This program determines the number of time each user utilized each distinct command.
algorithm_debug	This program calculates Schonlau's algorithm. The majority of variables from his algorithm should look similar to the Perl variables. This program creates a file that contains the xu value for the 5000 tuples of test data.
results_analysis_debug	This program calculates the confusion matrices and data necessary to build the ROC curve from Schonlau's algorithm.
features	This program creates seven features for each tuple of test data (5000 tuples).
601features	This program creates 601 features for each tuple of test data (5000 tuples).

C.2 Source Code

C.2.1 The *dictionarytrain* program

```
open (traintotal,">traintotal"); open (trainvar,">trainvar"); $j=1;

@total=@_;

for($j=1;$j<51;$j++)   #loops one time for each user (1-50)
{
    $fh="user$j";
    open $fh,"$fh";
    @all_lines=<$fh>;
    $i=0;
    for($i=0;$i<5000;$i++)
    {
        $a=(5000*($j-1));
        @total[$i+$a]=@all_lines[$i]
    }

print traintotal @total;        #saves every command as an array
length 250K

}

$count=0; $a=1; $b=2; for ($i=0;$i<250000;$i++) {
    if ($count==0)      #initiates cmd dictionary with first command
    {
        $count=count+1;
        $k=$count-1;
        $dictionary[$k][$a]=@total[$i];
        $dictionary[$k][$b]=1;
    }

    $j=0;
    $z=0;
    for ($j=0;$j<$count;$j++)   #determines if cmd is already in
dictionary
    {   if (@total[$i] eq $dictionary[$j][$a])
        {
            $dictionary[$j][$b]=($dictionary[$j][$b])+1;
            $z=$z+1;
        }
    }
    if ($z==0)              #if new cmd, adds to dictionary
```

```
    {
        $count=$count+1;
        $k=$count-1;
        $dictionary[$k][$a]=@total[$i];
        $dictionary[$k][$b]=1;
    }
}

$i=0; for ($i=0;$i<$count;$i++) {
    chomp $dictionary[$i][$a];
    print "$dictionary[$i][$a] $dictionary[$i][$b] \n";
}

$i=0; for ($i=0;$i<$count;$i++) {
    print trainvar "$dictionary[$i][$a],$dictionary[$i][$b]\n";
}

print "\n"; print "$count distinct commands."; print "\n";
$total=@total; print "$total total commands.\n"; print "\n";

open (traincmddict,">traincmddict"); open
(traincmdcount,">traincmdcount");

$i=1; for ($i=0;$i<$count;$i++) {
    print traincmddict "$dictionary[$i][$a]\n";
    print traincmdcount "$dictionary[$i][$a]\n";
}

##Below lines determine the popularity of the commands.

open (traincmddict,"traincmddict"); open (popular, ">popular"); open
(popvar,">popvar");

$j=1; @total=@_; @popularity=@_;

$k=0; for ($k=0;$k<635;$k++) {
    @popularity[$k]=0;
}

for($j=1;$j<51;$j++)  #loops one time for each user (1-50) {

    $fh="user$j";
    open $fh,"$fh";
    @all_lines=<$fh>;
```

```
    $i=0;
    for($i=0;$i<5000;$i++)
    {
        $k=0;
        $count=0;
        for ($k=0;$j<635;$j++)
        {
            if (@all_lines[$i] eq @traincmddict[$j])
                {
                    @popularity[$j]=@popularity[$j]+1;
                    $j=635;
                }


        }
    }
}



$i=0; for ($i=0;$i<635;$i++) {
    print popular @popular[$i];
    print popvar "@popularity[$i],\n";
}
```

C.2.2   The *user_popularity* program

```
open (traincmddict,"traincmddict"); open (popular, ">popular"); open
(popvar,">popvar");

$j=1; @total=@_; @popularity=@_;

@traincmddict=<traincmddict>;

$k=0; for ($k=0;$k<635;$k++) {
    @popularity[$k]=0;
}

for ($i=0;$i<50;$i++) {
    $k=0;
    for ($k=0;$k<635;$k++)
    {
        $usertraindict[$k][$i]=0;
    }
}
```

```
for($j=1;$j<51;$j++)  #loops one time for each user (1-50) {
    @counter=@_;
    $fh="user$j";
    open $fh,"$fh";
    @all_lines=<$fh>;
    $i=0;
    for($i=0;$i<5000;$i++)
    {
        $k=0;
        for ($k=0;$k<635;$k++)
        {
            if (@all_lines[$i] eq @traincmddict[$k])
                {   if (@counter[$k]!=1)
                    {
                        @popularity[$k]=@popularity[$k]+1;
                        @counter[$k]=1;
                    }
                    $usertraindict[$k][$j-1]=
$usertraindict[$k][$j-1]+1;
                }
        }
    }
}

$i=0; for ($i=0;$i<635;$i++) {
    print popular "@popularity[$i]";
    print popvar "@popularity[$i]\n";
}

open (usertraindict,">usertraindict");

for ($i=0;$i<=50;$i++) {
    $k=0;
    for ($k=0;$k<635;$k++)
    {
        print usertraindict "$usertraindict[$k][$i]\n";
    }
}
```

C.2.3 The *algorithm_debug* program

```
$start=time(); print "Start time:  $start\n"; open
(usertraindict,"usertraindict"); @utraintotal=<usertraindict>;
```

```perl
$i=0; for ($i=0;$i<50;$i++) {
    $j=0;
    for ($j=0;$j<635;$j++)
    {
        $cmd=(635*$i)+$j;
        $usertraindict[$j][$i]=@utraintotal[$cmd];
    }
}

open (userpop,">userpop"); $i=0; for ($i=0;$i<635;$i++) {
    $j=0;
    for ($j=0;$j<49;$j++)
    {
        chomp $usertraindict[$i][$j];
        print userpop "$usertraindict[$i][$j],";
    }
    print userpop "$usertraindict[$i][49]";
}

$i=1; for ($i=1;$i<51;$i++) {
    $fh="user$i";
    open $fh, "$fh";
    @all_lines=<$fh>;
    $j=0;
    for($j=0;$j<100;$j++)
    {
        $k=0;
        for($k=0;$k<100;$k++)
        {
            $line=5000+(($j*100)+$k);
            $testblock[$i-1][$j][$k]=@all_lines[$line];
            #this nested loop creates 100 arrays of test data
for every user
        }
    }
} print "User test blocks complete.\n"; $time=(time()-$start)/60;
$time=sprintf("%.2f",$time);
print "Total elapsed time:  $time min\n"; print "Now comparing test
blocks with user training dictionary...\n\n"; print @all_lines[0];

open (traincmddict,"traincmddict"); @traincmddict=<traincmddict>;
$i=1; for ($i=0;$i<50;$i++) {
    @vk[$i]=0;
    $j=0;
```

```perl
    for ($j=0;$j<100;$j++)
    {
        $l=0;
        for($l=0;$l<635;$l++)
        {
            $nuk[$i][$l][$j]=0;
            $wuk[$i][$l][$j]=-1;
            $vuk[$l][$i]=0;
            $modwuk[$i][$l][$j]=1;
        }
        $l=0;
        for($l=0;$l<635;$l++)
        {
            $k=0;
            for($k=0;$k<100;$k++)
            {
                if($testblock[$i][$j][$k] eq $traincmddict[$l])
                {
                    $nuk[$i][$l][$j]=($nuk[$i][$l][$j])+1;
                    $vuk[$l][$i]=($usertraindict[$l][$i])/5000;
                }
            }
        }
    }
}

print "Test blocks analyzed against user training dictionary.\n";
$time=(time()-$start)/60;
$time=sprintf("%.2f",$time);
print "Total elapsed time:  $time min\n"; print "Now computing
algorithm...\n";

$i=0; for ($i=0;$i<635;$i++) {
    $j=0;
    for ($j=0;$j<50;$j++)
    {
        @vk[$i]=@vk[$i]+$vuk[$i][$j];
    }
}

open (popvar,"popvar"); @userpopularity=<popvar>;

$i=0; for ($i=0;$i<50;$i++) {
    $j=0;
```

```
    for ($j=0;$j<100;$j++)
    {
        $l=0;
        for($l=0;$l<635;$l++)
        {
            if(($vuk[$l][$i]==0) and ($nuk[$i][$l][$j]>0))
            {
                $modwuk[$i][$l][$j]=-1;
            }
            if($vuk[$l][$i]==0)
            {
                $wuk[$i][$l][$j]=-1;
            }
            if($vuk[$l][$i] > 0)
            {
                $wuk[$i][$l][$j]=$vuk[$l][$i]/@vk[$l];
            }
            $calcxu=.01*(($wuk[$i][$l][$j])*($nuk[$i][$l][$j])*
            (1-(@userpopularity[$l]/50)));
            $xu[$j][$i]=$xu[$j][$i]+$calcxu;
            $modcalcxu=.01*(($modwuk[$i][$l][$j])*($nuk[$i][$l][$j])*
            (1-(@userpopularity[$l]/50)));
            $modxu[$j][$i]=$modxu[$j][$i]+$modcalcxu;

        }
    }
}

open (xu,">xu"); $i=0; for ($i=0;$i<50;$i++) {
    $k=0;
    for ($k=0;$k<100;$k++)
    {
        print xu "$xu[$k][$i]\n";
    }
}

open (modxu,">modxu"); $i=0; for ($i=0;$i<50;$i++) {
    $k=0;
    for ($k=0;$k<100;$k++)
    {
        print modxu "$modxu[$k][$i]\n";
    }
}
```

```
print "Program complete.\n"; $time=(time()-$start)/60;
$time=sprintf("%.2f",$time);
print "Total elapsed time:  $time min\n";
```

C.2.4 The *results_analysis_debug* program

```
$start=time(); print "Start time:  $start\n";

open (xu,"xu"); @xu=<xu>; open
(masquerade_summary,"masquerade_summary.txt"); open
(masq_summary,">masq_summary"); print masq_summary
<masquerade_summary>; open (masq_summary,"masq_summary");

@trueresults=<masq_summary>;

open(trueresults,">trueresults");

print trueresults @trueresults;

$test=substr(@trueresults[99],1,1); print $test; #print
@trueresults[99];

@true=@_; $i=0; for($i=0;$i<50;$i++) {
    $j=0;
    for($j=0;$j<100;$j++)
    {
        $cmd=(($i*100)+$j);
        $length=$i*2;
        @true[$cmd]=substr(@trueresults[$j],$length,1);
    }
}

open(true,">true");

$i=0; for($i=0;$i<5000;$i++) {
    print true "@true[$i]\n";
}

$i=0; $tolerance=-.25; for($i=0;$i<5000;$i++) {
    if (@xu[$i]<$tolerance)
    {
        @testresult[$i]=1;
    }
    if (@xu[$i]>=$tolerance)
```

```perl
        {
            @testresult[$i]=0;
        }
    }
}

$truepositive=0; $falsepositive=0; $falsenegative=0;
$truenegative=0; for ($i=0;$i<5000;$i++) {
    if (@testresult[$i]==@true[$i])
    {
        @finalresult[$i]=1;
    }
    if (@testresult[$i]!=@true[$i])
    {
        @finalresult[$i]=0;
    }
    if (@testresult[$i]==1 && @true[$i]==1)
    {
        $truepositive=$truepositive+1;
    }
    if (@testresult[$i]==0 && @true[$i]==0)
    {
        $truenegative=$truenegative+1;
    }
    if (@testresult[$i]==1 && @true[$i]==0)
    {
        $falsepositive=$falsepositive+1;
    }
    if (@testresult[$i]==0 && @true[$i]==1)
    {
        $falsenegative=$falsenegative+1;
    }
}

print "        predicted positive  predicted negative"; print "\n";
print "actual positive      $truepositive $falsenegative\n"; print
"actual negative      $falsepositive $truenegative\n";

open (ROC,">ROC"); $j=0; for ($j=0;$j<1600;$j++) {
    $i=.3;
    $tolerance=-.8963+($j*.001);
    for($i=0;$i<5000;$i++)
    {
        if (@xu[$i]<$tolerance)
        {
```

```
            @testresult[$i]=1;
        }
        if (@xu[$i]>=$tolerance)
        {
            @testresult[$i]=0;
        }
    }

    $truepositive=0;
    $falsepositive=0;
    $falsenegative=0;
    $truenegative=0;
    for ($i=0;$i<5000;$i++)
    {
        if (@testresult[$i]==@true[$i])
        {
            @finalresult[$i]=1;
        }
        if (@testresult[$i]!=@true[$i])
        {
            @finalresult[$i]=0;
        }
        if (@testresult[$i]==1 && @true[$i]==1)
        {
            $truepositive=$truepositive+1;
        }
        if (@testresult[$i]==0 && @true[$i]==0)
        {
            $truenegative=$truenegative+1;
        }
        if (@testresult[$i]==1 && @true[$i]==0)
        {
            $falsepositive=$falsepositive+1;
        }
        if (@testresult[$i]==0 && @true[$i]==1)
        {
            $falsenegative=$falsenegative+1;
        }
    }
    @truepositive[$j]=
($truepositive/($truepositive+$falsenegative));
    @falsepositive[$j]=
($falsepositive/($truenegative+$falsepositive));
    print ROC "@falsepositive[$j],@truepositive[$j]\n";
```

```
}

print "Program complete.\n"; $time=(time()-$start)/60;
$time=sprintf("%.2f",$time);
print "Total elapsed time:  $time min\n";
```

C.2.5 The *features* program

```
$start=time(); $now=time()-$start; print "Start time:  ($now)\n";

open (unix_commands,"unix_commands.txt"); open (traincmddict,
"traincmddict"); @unix_commands=<unix_commands>;
@traincmddict=<traincmddict>; $i=0; @traindict_unix=@_; for
($i=0;$i<635;$i++) {
    @dict_unix[$i]=0;
    $j=0;
    for ($j=0;$j<2006;$j++)
    {
        if (@traincmddict[$i] == @unix_commands[$j])
        {
            @traindict_unix[$i]=1;
            $j=2007;
        }
    }
}

open (traindict_unix, ">traindict_unix"); $i=0; for
($i=0;$i<635;$i++) {
    print traindict_unix "@traindict_unix[$i]\n";
}

open (toptwenty,">toptwenty"); @toptwenty=@_; $i=0; for
($i=0;$i<635;$i++) {
    @toptwenty[$i]=0;
}

@toptwenty[1]=1; @toptwenty[12]=1; @toptwenty[47]=1;
@toptwenty[35]=1; @toptwenty[26]=1; @toptwenty[59]=1;
@toptwenty[48]=1; @toptwenty[31]=1; @toptwenty[22]=1;
@toptwenty[25]=1; @toptwenty[139]=1; @toptwenty[36]=1;
@toptwenty[18]=1; @toptwenty[68]=1; @toptwenty[155]=1;
@toptwenty[175]=1; @toptwenty[15]=1; @toptwenty[32]=1;
@toptwenty[21]=1; @toptwenty[40]=1;
```

```
open (internet_cmd,">internet_cmd"); @internet_cmd=@_; $i=0; for
($i=0;$i<635;$i++) {
    @internet_cmd[$i]=0;
}

@internet_cmd[47]=1; @internet_cmd[59]=1;

$i=0; for ($i=0;$i<635;$i++) {
    print internet_cmd "@internet_cmd[$i]\n";
}

$i=1; for ($i=1;$i<51;$i++) {
    $fh="user$i";
    open $fh, "$fh";
    @all_lines=<$fh>;
    $j=0;
    for($j=0;$j<150;$j++)
    {
        $utraindict_unix[$i][$j]=0;
        $utoptwenty[$i][$j]=0;
        $uinternet_cmd[$i][$j]=0;
        $uuniqueness[$i][$j]=0;
        $utrainfreq[$i][$j]=0;
        $uforeign[$i][$j]=0;
        $k=0;
        for($k=0;$k<100;$k++)
        {
            $line=(($j*100)+$k);
            $datablock[$i-1][$j][$k]=@all_lines[$line];
            #this nested loop creates 150 arrays of total data
for every user
        }
    }
}

open (popvar,"popvar"); @popvar=<popvar>; open
(trainfreq,"trainfreq.txt"); @trainfreq=<trainfreq>;

$i=0; $j=0; for ($i=0;$i<50;$i++) {
    $j=0;
    for ($j=0;$j<150;$j++)
    {
        $k=0;
        for ($k=0;$k<100;$k++)
```

```perl
        {
            $z=0;
            $l=0;
            for ($l=0;$l<635;$l++)
            {
                if ($datablock[$i][$j][$k] eq @traincmddict[$l])
                {
                    $utraindict_unix[$i][$j]=@traindict_unix[$l]+
                    $utraindict_unix[$i][$j];
                    $utoptwenty[$i][$j]=@toptwenty[$l]+
                    $utoptwenty[$i][$j];
                    $uinternet_cmd[$i][$j]=@internet_cmd[$l]+
                    $uinternet_cmd[$i][$j];
                    $uuniqueness[$i][$j]=$uuniqueness[$i][$j]+
                    @popvar[$l];
                    $utrainfreq[$i][$j]=$utrainfreq[$i][$j]+
                    $trainfreq[$l];
                    $l=700;
                    $z=1;
                }
                if (($l==634) && ($z==0))
                {
                    $uforeign[$i][$j]=$uforeign[$i][$j]+1;
                }
            }
        }
    }
}

$now=time()-$start; print "features created for all training
commands;\n"; print "now measuring against data; time elapsed:
$now sec\n";

open (true,"true"); @true=<true>; open
(featurematrix,">featurematrix"); open
(testfeaturematrix,">testfeaturematrix"); open (xu,"xu"); @xu=<xu>;
$i=0; for ($i=0;$i<50;$i++) {
    $j=0;
    for ($j=0;$j<150;$j++)
    {
        $utraindict_unix[$i][$j]=($utraindict_unix[$i][$j])/100;
        $utoptwenty[$i][$j]=($utoptwenty[$i][$j])/100;
        $uinternet_cmd[$i][$j]=($uinternet_cmd[$i][$j])/100;
        $uuniqueness[$i][$j]=(
```

```
1-((50-($uuniqueness[$i][$j])/100))/100);
        $utrainfreq[$i][$j]=($utrainfreq[$i][$j])/100;
        $uforeign[$i][$j]=($uforeign[$i][$j])/100;
        $id=((101+$i)*1000)+($j+1);
        if ($j<50)
        {
        print featurematrix "$utraindict_unix[$i][$j]
$utoptwenty[$i][$j]
        $uinternet_cmd[$i][$j]  $uuniqueness[$i][$j]
$utrainfreq[$i][$j]
        $uforeign[$i][$j]   0   $id\n";
        }
        if ($j>=50)
        {
        $result=($i*100)+($j-50);
        chomp @true[$result];
        chomp @xu[$result];
        print featurematrix "$utraindict_unix[$i][$j]
$utoptwenty[$i][$j]
        $uinternet_cmd[$i][$j]  $uuniqueness[$i][$j]
$utrainfreq[$i][$j]
        $uforeign[$i][$j]   @true[$result]  $id\n";
        print testfeaturematrix "$utraindict_unix[$i][$j]
$utoptwenty[$i][$j]
        $uinternet_cmd[$i][$j]  $uuniqueness[$i][$j]
$utrainfreq[$i][$j]
        $uforeign[$i][$j]   @xu[$result]
@true[$result]  $id\n";
        }
    }
}

$now=time()-$start; print "program complete; time elapsed:  $now
sec\n";
```

C.2.6   The *601features* program

```
$start=time(); $now=time()-$start; print "Start time:  ($now)\n";

open (unix_commands,"unix_commands.txt"); open (traincmddict,
"traincmddict"); @unix_commands=<unix_commands>;
@traincmddict=<traincmddict>; $i=0; @traindict_unix=@_; for
($i=0;$i<635;$i++) {
    $dict_unix[$i]=0;
```

```perl
    $j=0;
    for ($j=0;$j<2006;$j++)
    {
        if ($traincmddict[$i] == $unix_commands[$j])
        {
            $traindict_unix[$i]=1;
            $j=2007;
        }
    }
}

open (traindict_unix, ">traindict_unix"); $i=0; for
($i=0;$i<635;$i++) {
    print traindict_unix "$traindict_unix[$i]\n";
}

open (toptwenty,">toptwenty"); @toptwenty=@_; $i=0; for
($i=0;$i<635;$i++) {
    $toptwenty[$i]=0;
}

$toptwenty[1]=1; $toptwenty[12]=1; $toptwenty[47]=1;
$toptwenty[35]=1; $toptwenty[26]=1; $toptwenty[59]=1;
$toptwenty[48]=1; $toptwenty[31]=1; $toptwenty[22]=1;
$toptwenty[25]=1; $toptwenty[139]=1; $toptwenty[36]=1;
$toptwenty[18]=1; $toptwenty[68]=1; $toptwenty[155]=1;
$toptwenty[175]=1; $toptwenty[15]=1; $toptwenty[32]=1;
$toptwenty[21]=1; $toptwenty[40]=1;

open (internet_cmd,">internet_cmd"); @internet_cmd=@_; $i=0; for
($i=0;$i<635;$i++) {
    $internet_cmd[$i]=0;
}

$internet_cmd[47]=1; $internet_cmd[59]=1;

$i=0; for ($i=0;$i<635;$i++) {
    print internet_cmd "$internet_cmd[$i]\n";
}

$i=1; for ($i=1;$i<51;$i++) {
    $fh="user$i";
    open $fh, "$fh";
    @all_lines=<$fh>;
```

```
    $j=0;
    for($j=0;$j<150;$j++)
    {
        $k=0;
        for($k=0;$k<100;$k++)
        {
            $line=(($j*100)+$k);
            $datablock[$i-1][$j][$k]=@all_lines[$line];
            #this nested loop creates 150 arrays of total data
for every user
            $utraindict_unix[$i][$j][$k]=0;
            $utoptwenty[$i][$j][$k]=0;
            $uinternet_cmd[$i][$j][$k]=0;
            $uuniqueness[$i][$j][$k]=0;
            $utrainfreq[$i][$j][$k]=0;
            $uforeign[$i][$j][$k]=0;
        }
    }
}

open (popvar,"popvar"); @popvar=<popvar>; open
(trainfreq,"trainfreq.txt"); @trainfreq=<trainfreq>;

$i=0; $j=0; for ($i=0;$i<50;$i++) {
    $j=0;
    for ($j=50;$j<150;$j++)
    {
        $k=0;
        for ($k=0;$k<100;$k++)
        {
            $z=0;
            $l=0;
            for ($l=0;$l<635;$l++)
            {
                if ($datablock[$i][$j][$k] eq $traincmddict[$l])
                {
                    $utraindict_unix[$i][$j][$k]=$traindict_unix[$l];
                    $utoptwenty[$i][$j][$k]=$toptwenty[$l];
                    $uinternet_cmd[$i][$j][$k]=$internet_cmd[$l];
                    $uuniqueness[$i][$j][$k]=$popvar[$l];
                    $utrainfreq[$i][$j][$k]=$trainfreq[$l];
                    $l=700;
                    $z=1;
                }
```

```perl
                if (($l==634) && ($z==0))
                {
                     $uforeign[$i][$j][$k]=1;
                }
            }
        }
    }
}

$now=time()-$start; print "features created for all training
commands;\n"; print "now measuring against data; time elapsed:   $now
sec\n";

open (true,"true"); @true=<true>; open (sevenC,">sevenC"); open
(xu,"xu"); @xu=<xu>; $i=0; for ($i=0;$i<50;$i++) {
    $j=50;
    for ($j=50;$j<150;$j++)
    {
        $id=((101+$i)*1000)+($j+1);
        $id=((101+$i)*1000)+($j+1);
        if ($j>=50)
        {
            $result=($i*100)+($j-50);
            chomp @true[$result];
            chomp @xu[$result];
            $x=@xu[$result];
            $x=$x*1;
            $k=0;
            @a=@_;
            $b=$_;
            for ($k=0;$k<100;$k++)
            {
                $uuniqueness[$i][$j][$k]=
1*$uuniqueness[$i][$j][$k];
                $utrainfreq[$i][$j][$k]=
1*$utrainfreq[$i][$j][$k];
                $c=1*$utrainfreq[$i][$j][$k];
                $utraindict_unix[$i][$j][$k]=
1*$utraindict_unix[$i][$j][$k];
                $utoptwenty[$i][$j][$k]=
1*$utoptwenty[$i][$j][$k];
                $uinternet_cmd[$i][$j][$k]=
1*$uinternet_cmd[$i][$j][$k];
                $uforeign[$i][$j][$k]=
```

```
1*$uforeign[$i][$j][$k];
                $a[$k]=join ("  ",$utraindict_unix[$i][$j][$k],
                $utoptwenty[$i][$j][$k],
$uinternet_cmd[$i][$j][$k],
                $uuniqueness[$i][$j][$k],$c,
$uforeign[$i][$j][$k]);
                #print "@a[$k]\n";
            }
            $b=join ("  ",@a);
            print sevenC "$b     $xu[$result]
$true[$result]  $id\n";
        }
    }
}

$now=time()-$start; print "program complete; time elapsed:  $now
sec\n";
```