Chapter 4

Relevance of Machine Learning

V. Rao Vemuri
University of California, Davis
rvemri@ucdavis.edu

1. Introduction

This book is about enhancing computer security through smart technology. The previous three chapters examined various facets of the security problem. Chapters 5-9 will discuss various methods of introducing the "smarts" into the security domain. This chapter provides some rationale for the need of smart technology and, in a brisk manner, covers some of the relevant ideas from artificial intelligence and machine learning.

Today's computer systems comprise a broad range of processors, communication networks, and information depositories. These systems are increasingly ubiquitous, and consequently they are increasingly subject to attack, misuse, and abuse. The complexity of these systems makes it exceedingly difficult to reason about their behavior. It is difficult to design security policies that are simple to understand and flexible to tolerate. Power and bandwidth limitations constrain security features in lightweight wireless devices. Cost considerations limit the usage of high assurance implementation methods. Software bundling policies make the software unwieldy; many vulnerable functions present in these bundles are rarely used by many users. System engineering tradeoffs are rarely based on technology issues alone; social, organizational, economic, regulatory and legal factors play a major role. Unfortunately, at today's state of the art, we do not have adequate understanding to develop an integrated solution to these challenging problems. The best we can in the rest of the book is to present those technology issues we do understand and hope to lay the foundation so as to develop an ability to address the broader issues at a future time.

An examination of security issues reveals that security threats and vulnerabilities have been evolving very rapidly. Indeed, over the past 25 years, the knowledge and expertise required to make a successful attack have been decreasing and the quality of tools available to attackers have been increasing. This changing environment obsoletes many security measures. For example, the current methods of malware (short for *malicious software*, i.e., software designed specifically to damage or disrupt a system, such as a virus or a Trojan horse) detection do not work with polymorphic malware (i.e., malware that uses encryption as a defense mechanism to change itself to avoid detection, typically by encrypting the malware itself with an encryption routine, and then providing a different decryption key for each mutation). Even if detection method is successful in some cases, it often gives precious little time to react and respond. Serendipitous seeding of malware makes the attacks hard to detect until the damage is done. The key to defend against this type of threat is to recognize that the attack mechanisms (viruses, worms, DDOS, etc. are evolving and mutating. New connectivity options are opening doors for new types of attacks. The old

model of “perimeter protection” is not keeping pace with these threats and vulnerabilities. This realization opens the doors for intelligent and adaptive methods for developing protection strategies.

According to current conventional wisdom, a promising way of protecting a computer system is based on an approach called *defense-in-depth*, which advocates multiple layers of protection to guard against failure of a single security component: hardware devices can fail, software can have flaws and bugs, and system administrators can make configuration errors. To overcome these potential vulnerabilities, defense-in-depth advocates a layered approach. The first step of this approach is the separation of systems into several "network sections" in one of the defense layers. Placement of a firewall to control the flow of traffic between section boundaries is another step. Another defense mechanism would be a "border router," placed between the Internet Service Provider (ISP) and the firewall, to filter traffic entering and leaving a network. Another layer of defense could be the placement of switches (a combination of a hub and a bridge) on individual sections of a network in order to make sniffing less effective. Yet another layer of defense is to use encryption. In spite of this battery of defensive weapons, intrusions still occur. The final layer of defense includes network based intrusion detection systems (NIDS) and host based intrusion detection systems (HIDS). Much of the AI and machine learning work, to date, has been targeted at this layer.

The defense-in-depth strategy is a general concept. Instead of looking at the system as a collection of physical layers, one can look at logical or functional layers for defense. For example, in the so-called enclave/policy tuned approach, the “normal” mode of operation has security policies that allow free flow of information between clients and servers. An event, such as Internet propagation of a new worm, could trigger a “degraded-mode” of operation, tightening security policies to allow communication between essential users and servers, but excluding or at least severely limiting access by non-essential entities. Non-essential clients can be quarantined either through brute-force network methods (turn off router port), or through implementation of autonomics/circuit breakers on hosts themselves. Essential IT services can include directory services, DNS, messaging, and e-mail. “Priority mode” policy changes can include rate-limiting connections, throttling system connectivity to the network to minimal levels and elevated intrusion detection and prevention.

2. Place of Intrusion Detection in the Security Landscape

Information security is best achieved in three stages: prevention, detection, and response. The earlier one intervenes, the more cost effective the solution will be. Although prevention gives the biggest bang for the buck, to rely solely on prevention would be a bad tactic; one may end up paying a heavy price if an attack eventually manages to get through. Preventing subversion by building completely secure systems from the requirements stage upwards is considered by many to be a very difficult task. The vast installed base of systems world wide is a virtual guarantee that any transition to a fully secure design would take a long time. Secure systems, designed and built from the requirements stage up, are nevertheless vulnerable to insider attacks. Enforcing levels of access-control mechanisms

is believed to lower efficiency and user-friendliness. Thus there is a need for intrusion detection systems.

Detecting intrusions in order to take remedial actions is a more reachable goal than preventing them altogether. The term "intrusion detection" refers to the broad range of techniques used to protect computer systems from malicious attacks. An intruder can be someone from within an organization (insider) or an outsider. An efficient detection system and a well articulated incident response procedure should be an integral part of any defensive strategy.

Network-based intrusion detection systems are necessary because most of the attacks come from the Internet. Although most of these attacks can be stopped by a properly configured firewall, one needs to be concerned about attacks that succeed in penetrating this outside perimeter. The final defensive layer is the host-based intrusion detection system. In this case, the defensive software is installed in every protected host computer. This requires an expenditure of system resources as well as administrative resources to monitor each and every system. Furthermore, this may demand a customized detection policy to reduce false alarm rate. In spite of these drawbacks, HIDS do have a place because NIDS do have limitations imposed by high-speed, switched and encrypted networks.

Many intrusion detection systems (IDS) are based on detecting signatures of previously seen attacks. Schemes that identify attacks based on anomalies (i. e., behaviors deviating from "normal" activity) exhibit unacceptably high false alarm rates and relatively poor coverage of the attack space. Whichever method is used, these methods often use data gathered from sensors that were originally designed for audit purposes – not for detecting attacks. These audit trails are records of activities that are logged to a file in chronological order and are of the order of 100 MB/day. These records can be inspected and attributes that are believed to shed some light on the intrusive behavior of a user are extracted: machine-oriented attributes (host attributes) such as user ID, host ID, time of log-in, duration of a session as well as items such as instructions used, speed of keyboard entry, CPU and memory resources consumed, number of processes created, system calls generated, and so on. Instead of analyzing a host machine's audit trails, one can also look at network-related attributes such as TCP/IP packet data and TCP/IP connection data and so on. Instead of building one centralized IDS passively defending a system, one can conceivably have a team of cooperating autonomous agents *actively* defending and maintaining the integrity and trustworthiness of a system.

Although much emphasis, in the subsequent chapters, is placed on the detection problem, the issue of responding to an intrusion is no less important. A majority of intrusion response systems (IRS) react to intrusions by generating reports and alarms. Research indicates that the greater the time gap between detection and response the greater the probability of success of the attack. For rapid response, manual methods are not quite adequate. Most of the automated response methods depend on using stateless methods (say, a decision table) where a particular response is associated with a particular attack; the same response is used for the same type of attack – always. More work is necessary in automating the response mechanism. Perhaps intelligent software agents and cooperating

agents can play a role here. Some of the ideas discussed in Section 6 of this chapter will come in handy while developing a single policy decision based on inputs from multiple agents, each having its own world view and each monitoring the system and making recommendations to a "head agent" [Vemuri, '00].

Although there is no shortage of ideas in detection *per se*, there is, however, very little understanding – at a theoretical level – on how attacks manifest. Indeed, there is inadequate understanding on how to characterize an attack. On the flip side, there is no adequate definition of what constitutes normal behavior either. In the absence of this knowledge, it is not easy to detect wide excursions from the norm. One possibility is to design systems that are capable of learning, over time, their own normal behavior. With this capability, a system would be better capable of analyzing itself, tune its operation on the fly and exhibit better robustness in performance.

Unlike in natural sciences - where one proposes a hypothesis, tests it using well defined metrics and validates it in order to build a theoretical foundation – much of the current research in intrusion detection is *ad hoc*. As the rest of the book is devoted to an examination of the issues related to intrusion detection from the point of view of machine learning, this chapter is devoted to a brief introduction to a brief overview of machine learning formulations and identify considerations that might lead to a better insight into the intrusion detection problem.

3. Machine Learning Beyond Intrusion Detection

Clearly there is a role for machine learning in computer security research that transcends intrusion detection. A typical university course in computer security covers such topics as authentication and identification, policies and models, architectures, malicious software, cryptographic algorithms and protocols, access control, network security, database security, social engineering and awareness, intrusion detection and response, and cyber forensics. Machine learning has a potential role to play in many of these areas.

For example, one can consider the design of self-adaptive systems that can survive an attack. There is certainly a need to understand multi-stage attack processes in which cascading failures can occur because the compromise of one resource may lead to the compromise of a more valuable resource. Modeling these processes can lead to a better design of self-adaptive systems that can survive attacks.

Learning for the diagnosis of a system state as a classification problem is another way machine learning can be harnessed for computer security. In this case, the current system state can be assigned to "normal" or abnormal state

Phishing is a mechanism used by spoofers for identity theft, that is, the theft of a name, Social Security Number, credit card number or some other piece of personal information to fraudulently apply for a credit card, make an unauthorized purchases, gain access to a person's bank account, and so on.

4. Machine Learning and Computational Learning Theory

Unlike in natural sciences - where one proposes a hypothesis, tests it using well defined metrics and validates it in order to build a theoretical foundation – much of the current research in intrusion detection is *ad hoc*. The use of empirical observations to augment knowledge derived from first principles to develop a scientific model (or hypothesis) is called *learning*. In many learning scenarios, it has become popular to use the computer to learn the correct model based on examples of observed behavior. When computers are used to implement these learning algorithms, the discipline is Machine Learning.

Besides artistic creativity, ethical behavior and social responsibility, the most difficult intellectual skill to computerize is learning. A possible reason for this difficulty is that learning is the result of the confluence of several intellectual capabilities. At the current state of the art, four different types of activities appear to fall under the machine learning rubric: symbol-based, connectionist-based, behavior-based and immune system-based learning.

Symbol-based learning has its roots in classical AI. It draws its strength from the Symbol System Hypothesis which states that all knowledge can be represented in symbols and the ability to manipulate these symbols to produce new symbols - and therefore new knowledge – is the essence of intelligence. Classical AI methods such as search and decision tree induction belong to this category.

Connectionist-based learning, inspired by the biology of the brains, de-emphasizes the explicit representation of knowledge using symbols. Neural network based methods are exemplars of connectionist systems. Symbol-based systems are implemented as computer programs and draw their conclusions from logical inference procedures. Connectionist systems are also implemented, quite often, as computer programs, but they are "trained" and draw their conclusions by recognizing patterns. Supervised classification methods such as Perceptrons, Support Vector Machines, kernel machines and a whole host of unsupervised classification (called clustering) methods belong to this category.

Behavior-based learning is inspired by Darwinian evolution. Here one assumes that a population of candidate solutions is always available and the challenge is to search this pool to find one that fits the problem at hand. These methods can be characterized by the phrase, "solutions in search of problems." Genetic and evolutionary algorithms fall in this category.

Immune-system-based learning draws its strength from the observation that the human body (or for that matter any biological system) is very adept at recognizing "foreign" objects entering the body. This ability to discriminate "self" from "non-self" can be exploited to develop powerful pattern recognition and classification algorithms.

Which of these four approaches is considered superior? The spate of experimental evidence from the 1990's suggests that none of these is markedly superior to the others.

However, there is reason to believe that factors such as feature selection and the encoding methods used for their representation may have some beneficial impact.

In contrast to the representational view, the learning task can also be viewed by considering the utilitarian objective of the learned system: learning for classification, learning for planning, learning for acting, learning for understanding, and so on.

The theoretical underpinning of machine learning is called computational learning theory (COLT). Machine learning, then, is the science of building predictors from data randomly sampled from an assumed probability distribution while accounting for the computational complexity of the learning algorithm and the predictor's performance on future data. Much of the work in machine learning is empirical. In such research, the performance of learning algorithms heavily depends upon the type of training experience from which the learning machine will learn. The learning algorithms themselves are typically judged by their performance on sample sets of data. Stated differently, in machine learning, training data is used to search (or build or learn) for a model (or a hypothesis) in a space of possible models (hypotheses). Given some training data, machine learning is tantamount to searching the hypothesis space (or model space) for the best possible hypothesis that describes the observed training data. Evidently, a well-defined learning problem requires a well-posed problem, a performance metric and a source of training experience.

Although *ad hoc* approaches do provide some insight, it is difficult to compare two learning algorithms carefully and rigorously, or to understand situations where a given learning algorithm performs well. Therefore the following issues become central to machine learning.

- Given sufficient training data, what algorithms exist so as to guarantee convergence to a hypothesis?

- How much training data is sufficient? Is there a particular sequence in which training data is to be presented for optimum learning experience? Does more training data give more confidence in what the machine learned?

- How does one go about getting the training data? If sufficient amount of real data is not available or hard to get, can one create training data sets via simulation experiments conducted in a laboratory set up? In such a case how much confidence one can place on a machine's predictions while it is operating in a real operational condition?

- Can prior knowledge about potential hypotheses help guide the learning process? If learning is not a memorization of what the machine saw during training but an ability to generalize from examples, how can prior knowledge help even if it is only approximately correct?

Computational learning theory provides a framework under which a rigorous analysis of both the predictive power and computational efficiency of learning algorithms can be carried out. COLT can shed light on some important questions: What kinds of guarantees

can one provide about learning algorithms? What are good algorithms for achieving certain types of goals? Can one devise models that are both amenable to mathematical analysis and make sense empirically? What can be said about the inherent ease or difficulty of learning problems? Addressing these questions will require pulling in notions and ideas from artificial intelligence, probability and statistics, computational complexity theory, cognitive psychology, game theory, and empirical machine learning research.

5. Some Popular Machine Learning Methods

Among the more popular methods implementing the classification step are neural nets, clustering methods, decision trees, Bayesian nets and a whole host of hybrid methods. Within the broad category of neural nets fall Perceptrons, support vector machines, multilayer nets with gradient type training, radial basis function networks and self-organizing nets.

Multi-layer networks with Back Propagation. This is a multi-layer (typically an input, hidden and output layers) feed forward neural network that relies on the classical gradient descent method for error minimization. During the forward pass, a feature vector is presented at the input layer which propagates toward the output layer and produces an output. This output is compared with the expected output, assumed to be known, and the resulting error is propagated backwards through a system of interconnected weights between the neural layers. These weights are adjusted until an error measure (typically the sum of the squares of errors) is minimized. This process is called *training*. The algorithm is a well established procedure and is described in many text books. The challenge really is at the pre-processing stage: selecting the features in the feature vector, the feature vectors to use during training and testing, deciding when to stop training, and so on. This method has been applied by many for intrusion detection [Dao and Vemuri, '02].

Support Vector Machines. These are really sophisticated versions of Perceptrons. Whereas a Perceptron allows many, theoretically infinite, possible separating hyperplanes between classes, a SVM produces a unique, optimal hyperplane. If a pattern is not linearly separable in the original feature space, a SVM permits linear separability in a higher dimensional space. SVM's have been successfully applied to the intrusion detection problem [Hu, et al, '03]).

Probabilistic models. It appears that many probabilistic models such as Markov Chains, Hidden Markov Models as well as non-Markovian models like Gaussian classifiers, naïve Bayes, [Valdes and Skinner, '00], Fuzzy neural systems (ARTMAPs, neuro-fuzzy ART [Liao, et al, '04; Hoffmann, et al. '03]), statistical models (decision trees, Markov models, etc.) can be effectively used for intrusion detection with comparable results. For example, the generalized Markov chain may improve the accuracy of detecting statistical anomalies. However, these are complex and time consuming to construct.

Clustering. The k-nearest neighbor method [Liao and Vemuri, '02; Rawat, et al, '04] and one-class classification for masquerade detection [Pasos, '03] also gave promising results. Some research is also going on in behavior-based security via user profiling [Dao and

Vemuri, '00; Stolfo, et al, '03]. Profiling users, characterizing user intent and capturing profile drift are problems that are waiting for a satisfactory solution.

Decision trees. Although decision trees are easy to learn and implement, they do not seem to enjoy as much popularity as, say, neural nets in the context of intrusion detection. A possible reason for this is that the problem of finding the smallest decision tree that is consistent with a set of training examples is known to be NP-hard.

Bayesian Networks. Bayes networks are powerful tools for decision and reasoning under uncertainty. A very simple form of Bayes networks is called naive Bayes, which is particularly efficient for inference tasks. However, naive Bayes is based on a very strong independence assumption. Surprisingly, the naïve Bayes still gives good results even if the independence assumption is violated, triggering further research in this area.

The exemplar for using any of the methods to develop intrusion detection systems is based on implementing the following four canonical steps: (a) data collection, (b) feature extraction, (c) creation of training and test data sets, (d) pattern recognition and classification. This recipe has been followed by a number of investigators. Implementation methods may differ, but the general recipe has been the same. Insofar as intrusion detection applications are concerned, much of the published work seems to follow the above sequence of steps: selecting a feature set to characterize a user, selecting a metric to measure similarity (or distance) between users, using a training data set, the most widely used being the DARPA data set, [Lipmann, et al '00] to train and test their models, and representing the results in terms of detection rates, false alarm rates, ROC curves, and so on.

6. Making Machine Learning More Useful

In a canonical *supervised* learning problem, a learner (or a learning program) is given a set of *training examples* in the form $\{(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \ldots (\mathbf{x_m}, y_m)\}$ and the learner is asked to learn some unknown function $y = f(\mathbf{x})$ from this data. Here the values $\mathbf{x_i}$ are typically vectors of the form $< x_{i1}, x_{i2}, \ldots, x_{in} >$ where the components x_{ij} are typically discrete or real valued such as height, weight, color, age and are termed *features* (or attributes) of the vector $\mathbf{x_i}$ – the feature vector. The notation x_{ij} refers to the *j th* feature of the feature vector $\mathbf{x_i}$. The subscript *j* is usually dropped if the context makes the meaning obvious. The *y* values are drawn from a discrete set of classes {1, 2, ..., K} if the problem is a classification problem, and the real line if it is a regression problem. It is generally assumed that the training examples might be corrupted with noise. It is generally convenient to assume that the training examples are indeed a sample drawn from a known probability distribution (say, the normal distribution) whose parameters may or may not be known.

Given a sample set S of training examples, a learning algorithm outputs a *classifier*. The classifier is a *hypothesis* of the unknown function *f*. Just like a number of regression lines can be drawn for a given data set, one can develop a family of classifiers (or hypotheses)

from a given set of training examples. Given a new **x,** each of these hypotheses $h_1, h_2, \ldots h_L$ can be used to make a prediction.

The classification step can be implemented by a variety of supervised methods such as neural nets (e. g. back propagation, radial basis functions and support vector machines), probabilistic methods (e. g. decision trees, Gaussian, Bayesian and Markov models) as well as by unsupervised methods such as clustering (e. g. k-nearest neighbor, k-means). Some of these methods will be discussed later in this chapter as well as in the next two chapters.

Most of the methods listed in the preceding paragraph share some common drawbacks. For example, no rationale seems to exist in selecting the features (number of features and the features themselves) defining a feature vector. A similar comment can be made about the selection of distance metrics (see Chapter 6). Finally, little progress appears to have been made in defining such fundamental concepts like "normal behavior" and "intrusive behavior." Is there an abstract way to characterize an intrusion? Can one build a model that can generate intrusive sequences with specified characteristics? Is there a way to explain the behavior of the current crop of intrusion detection models? Are there any limits to the learning methods in terms of their performance (say, in terms of false positives)?

There are several ways of overcoming these drawbacks. The bulk material in the rest of this section is summarized from a 1997 survey paper by Dietterich.

6.1 Ensemble of Classifiers

One method of improving the classification accuracy is to combine the outputs of different classifiers (hypotheses) in some suitable fashion, by some sort of voting. If all the classifiers agree, the question of voting becomes mute. If the classifiers disagree and if the errors committed by the classifiers are uncorrelated, then a majority vote would produce a good answer [Dietterich, ‘97]. However if the error rates of individual classifiers exceeds 0.5, then the error rate of the voted ensemble increases as a result of voting. The key to the success of this method is to make the individual classifiers as good as possible to start with.

6.2 Constructing an Ensemble by Manipulating Training Data

A fairly general method of constructing an ensemble of classifiers is by manipulating the training examples. Instead of using up all of the training data for one swoop, it is divided into subsets and different classifiers are trained with each subset. This strategy works well when the hypothesis generated is fairly sensitive to small changes in training data. This is indeed the case with neural nets and decision trees.

Cross-Validation: A popular method of manipulating training data is to subdivide the training data into *m* disjoint subsets and to re-construct training sets by leaving out some of the subsets, in turn, from the training process. For example, if the training data is divided randomly into 10 disjoint subsets, 10 overlapping training sets can be constructed by

dropping out *one* of the subsets, in turn, to get the "leave-one-out method of cross validation." This process can be generalized to get "leave k-out cross validation."

Bagging: A second method of manipulating the training data is to pick a sub set of *m* training examples as a random sample "with replacement." Such a sample is called *bootstrap replicate* of the original training data. Each of these bootstrap data sets is used to train a different *component classifier*. It is customary to select the same type of learning machine for all component classifiers – say, all neural nets, all hidden Markov models, all decision trees and so on. The final classification decision is based on a voting procedure. For this reason the method is also called *bootstrap aggregation*, from which the name *bagging* is derived [Breiman, '96]. It can be shown that each bootstrap aggregate contains, on the average, 63.2% of the original training set. In general bagging improves the classification accuracy of those classifiers whose classification accuracy is sensitive to small changes in training data. Decision trees and neural nets, which are deemed sensitive, are therefore called *unstable*.

Boosting. A third method of manipulating training data - similar to bagging - is boosting, which is a general method of converting a rough rule of thumb (or, a weak hypothesis) into a highly accurate prediction rule.

Adaboost: Adaboost, a variation of boosting, picks - at iteration l - a sub set of m training examples "with replacement," according to a probability distribution $p_l(\mathbf{x})$. The hypothesis generated with this training data is labeled h_l. The error rate ε_l of this classifier is computed and used to adjust $p_l(\mathbf{x})$. This procedure puts more weight on instances that were misclassified. A final classifier is then constructed by a weighted average of the individual classifiers [Freund and Schapire, '96]

6.3 Constructing an Ensemble by Manipulating Input Features

Another general method of constructing an ensemble of classifiers is by manipulating the input features. Selecting what and how many features to be included in the feature vector is an early design decision. Quite often this decision is made for expediency by looking at features that are already there in the available training data. For example, in intrusion detection, NIDES uses up to 30 attributes such as CPU and I/O usage, commands used, local network activity, and so on, to define a feature vector. Values of these attributes are typically logged routinely. Is it really necessary to use all these? Would a subset suffice? If so, what subset? How does one select that subset? As the size of the feature vector influences the size of the classifier, and therefore the effort in building it, there is some merit in trying to work with feature vectors of small dimension.

How does one select the subsets from the training data? There are no standard rules. Typically they are selected manually by using some ad hoc criterion, not necessarily justified by any problem-driven considerations. For example, 100 contiguous samples of a signal (in the time domain or in the frequency domain) can be grouped into several contiguous non-overlapping segments, or into several overlapping segments by sliding a window, or every 10th sample is placed in a bin to create ten bins, and so on. Or, in a

preprocessing stage the input data is first clustered and into groups and training data sets created by using some systematic selection from the clusters. In any event, this process of manipulating input features seems to work if the features are highly redundant.

6.4 Constructing an Ensemble by Injecting Randomness

This is probably the most commonly used method in neural networks. In backpropagation training, for example, a recommended method is to start with a randomly selected initial weight set, train the network and then re-start with a different random set of initial weights and repeat the experiment several times. Each such trial results in a different classifier. This approach is fairly widespread and has been successfully applied to decision trees and rule based expert systems.

6.5 Constructing an Ensemble Using Different Learning Algorithms

Finally, it is possible to build an ensemble of classifiers each using a different learning algorithm. Learning algorithms using radically different principles probably will produce very different classifiers, but there is no guarantee that they do produce the necessary diversity.

Indeed, there are many more ways of creating ensembles of classifiers.

6.6 Combining the Results form an Ensemble of Classifiers

Once results from a family of classifiers are available, there are many ways of combining the results. Indeed there are as many ways as there are voting procedures. Three prominent methods are: un-weighted voting, weighted voting and gating.

Majority Vote. The simplest of these is to take a simple majority vote. This idea can be extended if each classifier can produce not just a classification decision but also a class-probability estimate. The class probability estimate for data point **x** is the probability that the true class is k, $k = 1, 2, ..., K$, given the hypothesis h_l. Then the class probability of the ensemble is given by

$$P(f(x) = k) = \frac{1}{L}\sum_{l=1}^{L} P(f(x) = k \mid h_l)$$

The class with the highest probability (analogue to majority vote) becomes the predicted class.

Gating. Here the idea is to learn a gating function that takes **x** as input and produces as output the weights w_l to be associated with classifier h_l. That is, one seeks to simultaneously learn the gating function while learning the classifier.

Stacking. This starts with the assumption that there are L different training algorithms A_1, A_2, ..., A_L. Each of these algorithms take the training data as input and produce L different hypotheses $h_1, h_2, ... h_L$. In stacking the goal is to find a classifier h^* such that the final

classification will be computed by h^* ($h_1(x)$, $h_2(x)$, ... $h_L(x)$). In other words one has to learn h^* in some fashion.

6.7 Why the Ensemble Idea Works?

The ensemble idea works because uncorrelated errors made by individual classifiers can be removed by voting. Do individual classifiers commit uncorrelated errors? Is it possible to find a single classifier that works as well as a voting ensemble?

Answers to these questions can be found by looking at some of the theoretical foundations of machine learning and computational learning theory. Learning in Machine Learning is tantamount to a search in the hypothesis space, *H*. The search is carried out until one finds a hypothesis *h* that best approximates the unknown function *f*. The success of this search depends upon two important factors: size of the hypothesis space and the fact that such a search indeed ends in finding a good hypothesis that meets the criterion for "best."

If the hypothesis space is large, then a large training set is required to progressively constrain the search until a good approximation is found. Indeed, each training example can be used to rule out all the hypotheses that misclassify it. In a two-class classification problem – the most common scenario in intrusion detection – it is theoretically possible to use each training example to rule out one of the hypotheses in *H*. This suggests that O (log |H|) training examples would suffice to select a classifier.

Finding the weights for the smallest possible neural net consistent with the training data is NP-hard. Therefore people use local search methods, such as the gradient method, to find locally optimal weights. Due to these reasons it is quite possible that one may never succeed in finding the best hypothesis even after factoring in prior knowledge to assist the search process.

In spite of these positive attributes of the ensemble methods, very few attempts seem to have been made to apply this method to intrusion detection.

7. Summary

The complexity of many learning problems of today goes well beyond the capabilities of current machine learning methods. Specifically, in the computer security area, the machine learning technology that has been used until now is not adequate to allow the calculation of acceptable solutions while simultaneously assessing their accuracy. This inadequacy is responsible, in part, for the poor performance (high false positive rates, for instance) of the current crop of machine learning methods. Although computational learning theory, which has revolutionized the solution of classification, prediction and regression problems, has established itself as the framework of choice in machine learning, its full potential has not yet been brought to bear in addressing computer security problems.

In the intrusion detection problem domain, the available first principle knowledge is generally not adequate to characterize an intruder. The available empirical knowledge (in the form of labeled examples for training and testing) is inadequate because of small sample sizes. Much of the available data has been collected under artificial conditions and even that data does not contain adequate number of intrusion scenarios to learn from [McHugh, '00].

In order to apply COLT to "real-life" problems, such as computer security in general and intrusion detection in particular, the following methodological steps are suggested:

1. Precisely define the problem, preserving key features, while maintaining simplicity.
2. Select the appropriate formal learning model
3. Design a learning algorithm
4. Analyze the performance of the algorithm using the formal model.

While selecting the formal learning model, in Step 2 above, there are a number of issues to consider:

- What is being learned?
- How does the learner interact with the environment? (Is there a helpful teacher, a critic or an adversary?)
- What is the prior knowledge of the learner?
- How is the learner's hypothesis represented?
- What are the criteria for successful learning?
- How efficient is the learner in time, space and data?

In the computer security area, it seems reasonable to restrict the type of problems to the so-called concept learning problems, where there are a set of *instances* (training data) and a single *target concept* that classifies each instance. The goal of the learner is to device a hypothesis (say, a neural net) that correctly classifies each instance as a positive (intruder) or negative instance (non-intruder). The hypotheses in the hypothesis space must make a binary prediction every time an instance of data is presented; it is not acceptable for the learning machine to return a "I do not know" for some instances.

References

Dao, Vu and V. Vemuri, “Profiling Users in the UNIX OS Environment," *International ICSC Conference on Intelligent Systems and Applications*, University of Wollongong, Australia, Dec. 11-15, 2000.

Dao, Vu and V. Rao Vemuri, “Computer Network Intrusion Detection: A Comparison of Neural Networks Methods," *Differential Equations and Dynamical Systems*, 10:1/2, pp 201-214, 2002.

Dietterich, T. D., “Machine Learning Research: Four Current Directions,” AI Magazine, *AAAI*, Winter 1997

Dietterich, T. D. and P. Langley, “Machine Learning for Cognitive Networks: Technology Assessment and Research Challenges,” http://web.engr.oregonstate.edu/~tgd/kp/dl-report.pdf, May 11, 2003.

Freund, Y. and Schapire, R. E., Experiments with a New Boosting Algorithm, Proc. 13th Intl. Conf. Machine Learning, Ed. L. Sattina, pp 148-156, Morgan-Kaufmann, San Francisco, 1996.

Hoffmann, A. and B. Sick, "Evolutionary Optimization of Radial Basis Function Networks for Intrusion Detection," *Joint International Conference ICANN/ICONIP 2003*, Istanbul, Turkey, June 2003

Hoffman, A, C. Schmitz , and B. Sick, "Intrusion Detection in Computer Networks with Neural and Fuzzy Classifiers," *Lecture Notes in Computer Science*, Springer-Verlag Heidelberg, Volume 2714 / 2003, pp. 316 – 324, August 2003.

Hu, Wenjie, Yihua Liao, and V. Rao Vemuri, "Robust Support Vector Machines for Anomaly Detection in Computer Security," *International Conference on Machine Learning and Applications*, Los Angeles, CA, July 2003.

Liao, Yihua and V. Rao Vemuri, “Using Text Categorization Techniques for Intrusion Detection," *Proc. Usenix San Francisco,* August, 2002.

Liao, Yihua and V. Rao Vemuri and A. Pasos, "A General Framework for Adaptive Anomaly Detection with Evolving Connectionist Systems, *SIAM Inter. Conference on Data Mining*, Lake Buena Vista, FL, April 22-24, 2004.

Lippmann, R. P., R. K. Cunningham, D. J. Fried, S. L. Garfinkel, A. S. Gorton, I. Graf, K. R. Kendall, D. J. McClung, D. J. Weber, S. E. Webster, D. Wyschogrod and M. A. Zissman, "MIT Lincoln Laboratory offline component of DARPA 1998 Intrusion detection Evaluation," 2000 http://www.ll.mit.edu/IST/ideval/docs/docs_index.html

McHugh, John, "Testing Intrusion Detection Systems: A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations as Performed by Lincoln Laboratories," *ACM Trans. on Information and Systems Security*, 3:4, pp 262-294, November 2000.

Pasos, Alejandro, *Machine Learning Techniques in Masquerader Detection,* M. S. Thesis, University of California, Davis, 2003

Rawat, Sanjay, A. K. Pujari, V. P. Gulati, V. Rao Vemuri, "Intrusion Detection using Text Processing Techniques with a Binary-Weighted Cosine Metric," *International Journal of Information Security,* Springer-Verlag, Submitted 2004.
http://www.cs.ucdavis.edu/~vemuri/publications

Stolfo, S. J., S. Hershkop, K. Wang, O. Nimerkern and C-W Hu, "A Behavior-based Approach to Securing Email Systems," *Proc. Mathematical Methods, Models and Architectures for Computer Networks Security,* Springer Verlag, Sept. 2003.

Valdes, A. and K. Skinner, "Adaptive, Model-based Monitoring for Cyber Attack Detection," *Lecture Notes in Computer Science*, Number 1907, Springer-Verlag, Toulouse, France. October, 2000. In (Ed. Edited by H. Debar and L. Me and F. Wu), *Recent Advances in Intrusion Detection (RAID 2000)*.

Vemuri, Rao V. “Agent Assist for Computer Games," *International ICSC Conference on Intelligent Systems and Applications*, University of Wollongong, Australia, Dec. 11-15, 2000.