Statistical Learning Methods for Information Security: Fundamentals and Case Studies [*]

Hsing-Kuo Pao   Yuh-Jye Lee
Dept. of Computer Science & Information Engineering
National Taiwan University of Science & Technology
pao@mail.ntust.edu.tw   yuh-jye@mail.ntust.edu.tw

Chun-Ying Huang
Dept. of Computer Science & Engineering
National Taiwan Ocean University
chuang@ntou.edu.tw

Abstract

One of the most traditional methods for information security can be as easy as sequence matching, such as the signature-based methods for virus detection. However, it is now well-accepted that the signature-based methods are no longer satisfactory solutions for many security problems. The signature is usually too rigid, resulting in detection that is hard to adjust and easy to bypass. Statistical learning approaches can complete the puzzle to form an integrated defense system. Numerous statistical learning methods have been proposed in the last couple of decades for various applications. To solve information security problems statistically, we need to carefully choose

[*] This work received grants from National Science Council under the grant numbers NSC 99-2218-E-011-019, NSC 100-2218-E-011-010, NSC 101-2218-E-011-009, and NSC 102-2219-E-019-001; also from National Science Council, National Taiwan University and Intel Corporation under grant numbers NSC 102-2911-I-002-001 and NTU 103R7501. It was also supported in part by Taiwan Information Security Center (TWISC). The authors acknowledge anonymous referees for their constructive criticisms.

appropriate statistical learning methods and evaluation procedures so that what seems to be a meaningful and effective method in terms of the statistical analysis can also be beneficial when the method is deployed to the real world. This paper aims to give an introductory and as self-contained as possible overview for how to correctly and effectively apply statistical methods to information security problems. We also demonstrate a couple of applications of the statistical learning methods on the problems of botnet detection and account security.

Keywords — Anomaly detection, Information security, Intrusion detection, Signature-based methods. Statistical learning.

1 Introduction

Information security has been one of the cornerstones in modern cyber technology. People dream of automatically detecting intrusions to ensure a secure system all the time. Information security researchers did not use *statistical learning* or *machine learning* algorithms substantially in their applications until recent decades. The traditional approach of information security research relies heavily on the understanding of various domain knowledge such as computer network, database, operating system, web technology, etc. That is to say, however difficult it may be, mastering the domain knowledge is a necessary condition to build a robust defense system to prevent a domain or server from intrusions.

To detect an intrusion, we may solve a symbol matching problem using either an exact string matching [36] or an approximate Longest Common Subsequence (LCS) matching [15] algorithm. More generally, we can check signatures to detect intrusive activities, so-called the *signature-based approach.* On the other hand, we can adopt rule-based detection algorithms to separate intrusions from normal behavior. Nevertheless, for both approaches, the blacklist for signature-based methods or the rules for rule-based methods may get longer and longer as time goes by, and the performance eventually degrades as both its time and storage requirements increase. In recent years, zero-day exploits [18] that take advantage of unsynchronized signature databases often cause disasters for infrequent patch updating. Moreover, increases in the automation of intrusive procedures complicate the detection rule designation and speed up the intrusion propagation. In summary, for the traditional approach to work properly, we need experienced domain experts, a complete (which could be infinitely long) signature database, an efficient and effective detection method, and frequent and prompt signature updating to make the defense system work.

An alternative technique to deal with the above issues is the *probabilistic* approach. In the probabilistic method, we can combine similar *patterns* that are generated along with some malicious activities into one group to compress the blacklist. A high-level guideline is to "extract" *knowledge* from the activity patterns and classify the intention behind the patterns into either a benign type or a malicious type. A theoretical treatment includes using a Markov chain or hidden Markov model [58, 72] to model sequential data, which is a type of *generative* model in the sense that we can really generate more sequential data (such as an attack sequence in a network environment) given the model. A *Probabilistic Graphical Model* [38, 31] is a general model that can describe a dataset which consists of various dependencies in the set, including spatial and temporal dependencies.

To detect intrusions, anomaly detection techniques can complement the weakness of signature-based methods. While signature-based methods which belong to a *specific* hypothesis in concept learning [51] may suffer from high false-negative problems[1], most anomaly-based detection methods which belong to a *general* hypothesis could give relatively high false-positives. Generally speaking, integrating both the signature-based approach and anomaly-based detection approach should give a better result than using only one of them in intrusion detection [30]. However, to apply this integration effectively in real intrusion detection systems is still challenging. Especially, it is not trivial to make a final decision when two approaches give contradicting results.

Probabilistic learning and anomaly-based detection techniques, where both belong to the *data driven* approach, are two examples of statistical learning methods that can be helpful to solve information security problems. Statistical machine learning has been receiving attention in recent decades due to several successful stories in which learning algorithms have been applied to problems such as speech recognition [58], bioinformatics [4], face detection [70], autonomous driving [55], etc. One of the reasons for such success is due to the development of *discriminative* models, such as the support vector machines (SVM) [11], where the goal is to separate data that belong to different patterns. For instance, given some malicious and benign training

[1] We consider an attack or an intrusion positive data in this case and throughout the article.

data, we can train a model and use it to classify new incoming data into one of the two categories, called *supervised learning*. Generative modeling and discriminative modeling are different in their algorithms, the goals they try to achieve, and the applications that they perform well at [52].

Statistical learning methods have already solved numerous information security problems, especially *system security problems*. This is because a system security problem can be transformed into a clustering or a classification task by nature; for instance, to distinguish between benign and malicious activities. With well-developed learning methods, researchers are able to concentrate more on analyzing the security problem itself and identifying the key point to solve the problem. To our knowledge, Forrest et al. [21] is the first research group that suggested detecting intrusions by using learning methods. Numerous researches based on generative models [71, 74, 34] are then followed to explore the effectiveness of learning methods on solving security problems. Researchers have also used discriminative models to deal with security issues. They utilized RIPPER [39, 8], decision tree [42, 62], or SVM [8, 27, 28] to solve different security problems.

Researchers have shown that learning methods can solve security problems; however, it is still challenging when, for example, *the training dataset is unbalanced*, *the labels of data are not* 100% *accurate*, or *the training data is not representative*, etc. Even we have so many learning methods available in the public domain these days, it is still not a straightforward task to successfully apply the learning methods to real-world information security problems because of the gap between academic research and industrial deployment [61]. Information security consists of many sub-fields such as network security, software security, database security, identification/authentication, spam filtering, privacy preserving, and many others. Applying learning algorithms to deal with those different problems may lead to different difficulties, and we illustrate how to deal with various problems given a wide range of algorithms.

In this article, we show how and why machine learning methods can help solve information security problems. We introduce the basic statistical learning concept, and also demonstrate how the learning algorithms can solve real problems. Statistical learning methods, like any other methods, also have their limitations. The key idea is to utilize the methods in appropriate domains, perhaps in synergy with other methods such as the traditional approach, for a robust and efficient problem solving package. The remainder of this paper is organized as follows. Section 2 contains a discussion of a few well-known information security problems and issues if solving the problems by statistical learning methods. In Section 3, we introduce the basic statistical leaning concept, followed by a series of guidelines in choosing learning and statistical methods for real security problems. Following that, we offer some case studies in Section 4; then, in Section 5, we summarize our conclusion.

2 Solving Information Security Problems Statistically

In this section, we discuss some of the more recent problems in information security and the issues we face when solving them via statistical learning methods. First, we focus on the debate between signature-based methods and anomaly-based methods. Second, a few recent information security problems are reviewed; following that, we give suggestions on how to deal with those problems. The statistical learning methods belong to the category of data-oriented approach, so in the last, we discuss the issues when the data-oriented approach is applied to information security problems.

2.1 Signature-based Methods versus Anomaly-based Methods

The majority of traditional methods are signature-based (also called misuse-based methods in intrusion detection). Signature-based methods have the advantage of using an attack database to detect previously seen attacks. On the other hand, finding the unusual behaviors in daily security data belongs to the category of anomaly-based detection methods [12, 23, 35]. While the signature-based methods are usually efficient and well-defined, anomaly-based methods generally need no pre-defined categories of benign or different types of malicious behaviors. That is why the anomaly-based methods can detect novel attacks. As we mentioned in the introduction section, signature-based methods usually produce more false negatives, and anomaly-based methods are likely to have more false positives.

The input for the signature-based method is a pre-defined attack database. The attack database has a set of attack rules that is usually defined by experts. On the other hand, the input for the anomaly-based method is a dataset that consists of mostly benign data. Statistical learning methods can profile normal behaviors based on the data. Compared to the signature-based method which is an expert/model-oriented approach, the anomaly-based method is a data-oriented approach. When we build a model from data, the model's generalization power is expected to make correct labeling for some unseen cases. Given the anomaly detection approach, we expect to detect novel intrusions that domain experts may not think of. That makes the data-driven or anomaly-detection-based approach more robust than the methods that heavily rely on experts' knowledge. Nevertheless, when deploying a model to a new environment, a data-driven approach has the advantage of being able to adjust the model of previous moments to reflect the current status. Before we discuss more issues regarding data-driven approaches, we review a set of various information security problems and illustrate how statistical learning methods can help us solve the problems.

2.2 A Set of Current Information Security Problems

Most attacks evolve rapidly in a dynamic environment. With different times come different types of intrusions and threats topping security lists. We discuss a few of the most notorious attacks, malicious items, or important security issues in recent years and use them to illustrate how statistical learning methods can help us to detect attacks.

Understanding Botnets Botnets are one of the most talked-about attacks nowadays, thanks to the rise of Web 2.0 and computerized social networks. A botnet can hijack a network or a host to execute some pre-assigned tasks. A large-scale botnet attack can compromise hosts/networks and cause a severe loss. By nature, a botnet has a spatial connection in the social or non-social network; therefore, we should take advantage of those spatial relationships for an effective detection. Moreover, considering the dynamic behavior of a botnet, we can understand more about how the botnet spreads out through time and provide a better defense for the attacks.

We discuss several issues in the botnet detection. First, as we mentioned previously, to detect a botnet effectively, one has to consider the spatial and temporal aspects of a botnet. Second, botnet activities are usually hidden in the environment; therefore, to acquire a benchmark dataset for a fair evaluation is generally difficult. Collecting data in a restricted domain can only claim a limited rather than a general statement. Also, label information is not always trustful in this case. All imply that the evaluation of botnet detection methods must be carefully handled. A detailed discussion of botnet detection can be found in Section 4.1.

Malicious URL Detection A malicious URL references a website that traps web surfers with some unauthorized actions when they visit the site. Some typical examples include an installation of Trojans, worms, or unwanted scripts in the local client when the client's owner visits a malicious URL. Malicious URLs are a serious problem on the Internet because not all web surfers have a basic understanding of information security. In particular, many cannot differentiate malicious websites from normal ones. Therefore, malicious URL detection is an urgent task to ensure web security. Moreover, it is ideal to perform the detection in real time or close to real time so that the damage to the victims of malicious websites can be minimized.

Understanding the content of a malicious site can help us classify it as one that is designed with bad intentions and make appropriate defensive actions afterwards. However, content analysis usually takes a lot of computational effort, not to mention the bandwidth load to retrieve the content before the analysis. Another approach is to only use the URL string to pre-screen a significant amount of malicious sites so that the need to do further analysis based on the website content can be avoided [45, 44, 53]. To build a model that can judge whether a website is malicious or not given the website URL, clearly, having a rule-based model is not appropriate; instead, a *black-box* method may work better which can combine several features to have an integrated judgment. We will discuss the choice between a transparent model and a black-box model in Section 3.4.

Account Security Most Internet users have more than a couple of web accounts for various cyber activities. In recent years, the methods used to secure users accounts have become a serious issue. To login to an account, the user normally provides credentials such as a username and password, but biometric methods like fingerprint matching, facial recognition, or iris scan may also be used for personal identification. Sometimes, a web connection is built on an untrusted network and unauthorized persons may steal a user's personal information and even his or her identity. We need a procedure to confirm or identify the user's true identity.

To detect account hijackers, most traditional approaches use a rule-based method. For example, most account login systems prevent users from account access after a few password trials; remote web servers usually shut down a network connection when the connection reaches a maximum transmission limit. Clearly, strict rules like those stated above could be annoying if we set easy-to-reach thresholds for the rules to avoid high false negatives. We can apply statistical learning methods to learn the true account owner's behavior so that when a user who has a different behavior pattern tries to access the account, the method can give warning to the system. In Section 4.2, we demonstrate an approach that can learn different behavior patterns to decide whether or not a given user is the true account owner when he or she intends to access an account. To build a profile of the true account owner's behavior, we need to profile the so-called normal behavior, further discussed in the next subsection.

2.3 Issues for Data-oriented Approach

To apply statistical learning methods to information security problems, we are concerned with some issues which may not be found for the traditional signature-based approach due to the need of acquiring a dataset for training. Let us further discuss the issues below:

i. *Acquiring Benchmark and Data Labeling:*

   In general, it is not easy to acquire a benchmark dataset when we want to evaluate statistical methods that are proposed to solve security problems. To adopt a supervised learning method for intrusion detection in a network environment, we need label information for a naturally collected dataset. However, it takes significant effort from humans to label massive network data to the correct class: either an attack or a benign one. What should be emphasized is that many security problems need professional experts to build such labeled datasets, which is a quite different from e.g., face detection problems where almost everybody can take the labeling job, and crowd sourcing can be applied when it is necessary. On the other hand, people do use synthesized datasets, such as the well-known DARPA dataset [37] which was built in an isolated network environment. Researchers believe that synthesized datasets such as the DARPA dataset may not be appropriate to simulate the real-world network environment [49, 50].

ii. *Profiling Normal Behavior:* An alternative choice to build a benchmark dataset is to find a set with all or most of the data in one label. Given the dataset, we can train a one-class model to describe the "normal" behavior in the set. In practice, we also allow the situation in which some or many of the label information is not perfectly reliable, but the majority of the whole set is still believed to have the same label. When detecting network intrusions, it is likely that most network data in the network environment is "clean", and we can focus on how to model the majority of the data, which is normal behavior profiling. Understanding normal behavior can help us understand the "anomalies" (also known as "outliers") in a task of anomaly detection. We need no effort to label the data in this case; however, keep in mind that to judge an environment that contains no attacks at all is usually risky.

iii. *Unbalanced Distributions:* In various security datasets, attacks are usually less common than non-attacks. In this case, the *rare events*, i.e., the attacks, are easily classified as non-attacks because it may not increase the error rate by much. To give a fair evaluation criterion of model selection, we can consider F-measure which can balance the counting between the false positives and false negatives. We should also take into account different priors, or to assign different costs for data of different labels according to the problem types [3].

iv. *Hybrid Data Types:* The data that we usually discuss for information security problems may consist of hybrid data types, the *categorical* data (such as alert types, used protocol, etc.), and the *numerical* data (such as the bandwidth, transmission rate, and duration, etc.). In general, we need to apply different methods for different types of data. In the case we must transform the data from one type to the other (categorical to numerical or vice versa) to fit the model's need, we need to make sure no artifacts are introduced in such a transformation.

We need to consider the problems mentioned above when we solve real world security problems using statistical methods. Even so, we believe that statistical methods generally open a possible "complementary" viewpoint to common approaches that use only rigid detection rules. In this article, we discuss some security problems in which a statistical approach may better help to solve the problems.

3 Statistical Learning Methods Applied in Information Security

Traditionally, machine learning methods are categorized as three disciplines: *supervised*, *unsupervised*, and *reinforcement* learning methods [1], depending on *whether or not* and *how* we acquire the label information of data. For instance, we can consider attacks as positive-labeled data and non-attacks as negative-labeled data in an intrusion detection task so that we can apply binary classification techniques, which are among the most discussed learning methods, to solve the problem. In this case, labeled data are given for training, and the goal is to predict or give a label for fresh data that have no label. Alternatively, as a more conservative approach[2], we can use an unsupervised method to cluster data into groups of the same types when we either do not have the label information, or the given label information cannot be trusted. Usually we can detect novel type attacks by unsupervised learning instead of supervised learning, which makes the unsupervised learning method more applicable in a dynamic environment.

Some other candidate methods for novel attack detection include *one-class methods* [60], where we have data of one label, and we are interested in modeling data that belong to this class; *semi-supervised learning*, where the given data for training are only partially labeled; and many asymmetric classification methods [48, 47] that give different penalties (costs) for false positives and false negatives. We can also apply different strategies for data that belong to the known controlled label(s) and the data that do not belong to any known labels. More issues and related methods are discussed further below. Before that, we introduce the notations that we plan to use in this work.

Notations We have a dataset $\mathcal{D} = \{\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_m\} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_m, y_m)\}$ that consists of m data and each data[3] $(\mathbf{x}_i, y_i) = (x_{i1}, x_{i2}, \ldots, x_{in}; y_i)$ can be summarized by its n attributes and the label/class information y_i. Knowing the relationship between $\mathbf{x}_i$ and y_i is the basis of our modeling for prediction. Sometime we choose a different index system, $s, t, \ldots$ rather than $i, j, \ldots$ for data index if we believe there is a known or unknown de-

[2] Unsupervised learning is more conservative than supervised learning in the sense that we do not (fully) trust the label information in the given dataset.

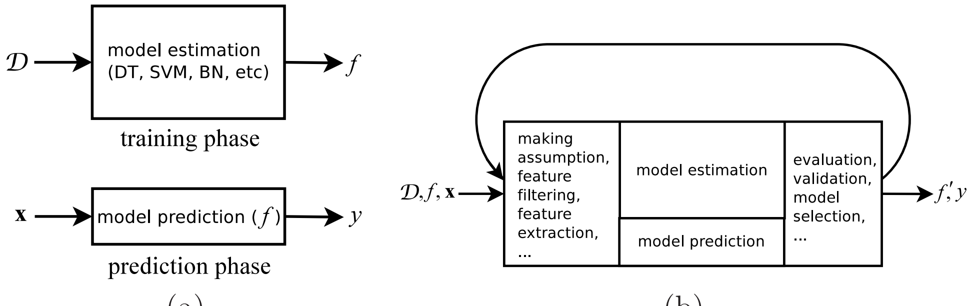
[3] A bold face letter indicates a vector in this article.

pendency between data. In this case, we may interpret $s, t, \ldots$ as time for a series of data with a clear order or data with dependencies in between. The attributes can be *categorical* (or called *discrete*) or *numerical* (or called *continuous*)[4]. Below we discuss the basic learning concepts that can be helpful for information security problems.

3.1 The Complete Learning Procedure

The typical learning procedure consists of the training phase which produces a function f to describe the relationship between $\mathbf{x}$ and y given the training data $\mathcal{D}$; and the prediction phase, which takes the trained function f and fresh unseen $\mathbf{x}'$ to predict y', as shown in Figure 1(a). It says that one of the key components in machine learning is model estimation, where we look for the most appropriate model f to predict y given $\mathbf{x}$. A list of well-known models for the purpose includes decision trees [57, 56], Support Vector Machine (SVM) [68], artificial neural networks, and boosting (e.g., Adaboost [22], to name a few. When we need to deal with data that have dependencies between them, we may consider *Hidden Markov Model* (HMM) [58] for sequential data or *Probabilistic Graphical Models* (PGM) [38, 31] for modeling more complicated dependencies. A related classification problem aims to predict labels $(y_{i1}, y_{i2}, \ldots, y_{ik})$ given attributes $(x_{i1}, x_{i2}, \ldots, x_{in})$, called *multi-label classification* [66] may also need to model dependencies between different labels (and maybe attributes too). Given various advanced learning methods proposed in the last decades, one should not forget the baseline methods which can be as simple as k-nearest neighbor (kNN) or naïve Bayes. In some cases, by choosing an appropriate Mahalanobis distance, kNN can perform as well as many other methods [24]. In fact, kNN can be considered a special case of SVM, when we choose a small enough neighborhood range. On the other hand, we often see naïve Bayes outputs competitive result even the conditional independence assumptions violate [75]. Thanks to the open source package WEKA [25], researchers can easily try out many learning methods to get a basic understanding of how difficult a problem could be. Before applying various learning models to problems, visualization of data on low-dimensional space is also helpful when we have no clue on choosing from many learning models.

[4] Using the terms "discrete" and "continuous" in machine learning should not be confused with the mathematical meanings of the same keywords.



(a) (b)

Figure 1: The complete learning procedure.

Finding a good model to describe the data is not the only focus when we apply learning algorithms to security problems. The ultimate goal of learning is to be able to predict the correct label y' for a given fresh attribute set $\mathbf{x}'$. We can take both the training data $\mathcal{D}$ and the attribute set $\mathbf{x}'$ together to predict y'. In this case, the training and prediction procedures are integrated as one procedure. kNN is an example, which belongs to a category of *lazy* learning methods when we do not explicitly build a model for prediction until there is a need to do so (at the time for prediction), or we simply skip the training part and combine training and prediction together as one. In another case, we also take both $\mathcal{D}$ and $\mathbf{x}'$ together as the input, but we build a "side product" model f in the time for prediction. Sometimes, to deal with dynamic learning and prediction tasks, we can operate the training/prediction procedure continually and periodically. In this case, at time t, we take $\mathcal{D}_t = \{d_1, \ldots, d_t\}$, the attribute set $\mathbf{x}_{t+1}$ and the model that was learned from the previous step f_t as the input to produce the model for the next step f_{t+1}, as well as the predicted label y_{t+1}. The procedure is especially useful for *continuous monitoring* in information security.

For most security problems, we are worried about novel types of intrusions. In these cases, a dynamic intrusion detection model is preferred where we continually tune the model based on current network or server status, rather than a *batch-mode* trained model which is built upon a fixed dataset $\mathcal{D}$. We need a one-time training model (such as an intrusion detection model) that is efficient particularly in its prediction/detection phase; on the other hand, the dynamic model (such as an intrusion prevention model) needs to be efficient or close to real-time in both its training and prediction phases.

We also need to emphasize that a dynamic model is inevitably necessary when we build a model in an *adversarial* environment. In this case, a model needs to evolve as attackers continue to revise their attack scenarios [6].

To successfully build a good model is not only to correctly estimate model parameters or to select a good model from a set of candidates. It is often the case that we have to make appropriate assumptions (*inductive bias*) about the security problems, and to extract a good feature set from available attributes to effectively find intrusions. After the model estimation, finding good criteria for model evaluation or validation is also important. Simply speaking, when a learning model fails to find intrusions or malicious patterns, not only should we criticize the model choice, we must also re-visit the whole learning procedure, which includes 1) finding an informative feature representation for the problem; 2) selecting a model that fits the problem; 3) knowing how to fairly evaluate a set of models; and 4) checking how trustful and noisy the data is. Finding a model that does not *overfit* or *underfit* the dataset is also a key for successful learning. The complete learning procedure is shown in Figure 1(b).

Researchers in the information security community have criticized careless use of learning algorithms in solving security problems [61]. When we use statistical learning methods to solve problems, we often need the help of some expert knowledge for labeling, model selection, and model improvement, etc. To label data for training, for instance, the difficulty exists in information security rather than in other applications. Considering the learning problems such as face detection, autonomous driving, and speech recognition, most of us can recognize faces, drive cars, and understand spoken languages without much difficulty at all. To label intrusions or attacks in security problems, however, the needed expert knowledge is not as trivial. Moreover, labeling large-scale data based on expert knowledge can be a burden whether it is for inexperienced people or even for domain experts. Acquiring expert knowledge is expensive in information security. Using the expert knowledge should also be with care. To solve security problems, what should be done is to introduce as little bias as possible to make the modeling robust. For instance, working on an expert-chosen feature set may not be as good as working on a rich feature set and letting the training decide what the effective features are. Overall, a careful choice of learning models as well as a list of experienced decisions from the learning result is necessary to effectively detect intrusions. We discuss more issues about how to apply learning algorithms from different viewpoints in the following subsections.

3.2 Supervised Learning versus Unsupervised Learning

In most cases, to choose between supervised learning and unsupervised learning methods relies on the availability of data label information. However, we need to consider several issues when dealing with security problems. First, acquiring data label information for security problems is often expensive because a carefully executed stealthy or masquerade attack may not be identified easily even for experts. Therefore, it is not expected to have such label information for large-scale data. To work with the security problems that are full of large-scale data, we have to choose between semi-supervised learning and unsupervised learning methods. That is also the case when we cannot fully trust the label information.

Second, we should consider the problem of *labeling errors* and *measurement errors* in training sets. When a set contains more labeling errors rather than measurement errors, especially when data are more or less formed as clusters and data are not too sparse, we should choose unsupervised methods. For rare types of intrusions, we seldom collect enough (or maybe even collect none) data to form separate groups. In this case, we should apply one-class methods or anomaly-based detection methods [12] to detect intrusions. In this case, "intrusions" are considered anomalies or outliers from the normal benign patterns and we can learn the model with or without the label information and with or without the novel attack patterns.

We need to emphasize that unsupervised learning rather than supervised learning methods are more distribution dependent; therefore, some *parametric* assumptions for distribution estimation could be inevitable. When the label dependent distribution (likelihood) is hard to model, using label information in training is helpful if the labels are reliable. That is, supervised learning rather than unsupervised learning is still recommended in the normal cases when we have reliable labeled data for training. In the last few decades or so, supervised learning has dominated the mainstream research of statistical learning community. One of the reasons is due to its well-accepted evaluation procedure. This situation has been changing in the last decade [69].

3.3 To Be or Not to Be, Stochastic Models, Bayesian Models and Stochastic Algorithms

To detect intrusions, both the detection result and the detection method itself can be formed probabilistically. In the typical sense of supervised learning, given training data $\mathcal{D}$, we can build a detection model f that can predict a coming activity to be either an intrusion or a benign one by evaluating $f(\mathbf{x})$ for a given observed attribute set $\mathbf{x}$. Alternatively, as a probabilistic approach, we instead estimate the *probability distribution*[5] $P(f \mid \mathcal{D})$, and choose the most *probable* model by computing

$$\arg\max_{f} P(f \mid \mathcal{D}) \,, \tag{1}$$

and then we find the correct label by evaluating $f(\mathbf{x})$ afterwards. We can further improve the detection result based on a group of classifiers and their weighted prediction. First, we compute

$$P(y \mid \mathcal{D}, \mathbf{x}) = \sum_{k} P(y \mid f_k, \mathbf{x}) P(f_k \mid \mathcal{D}) \,, \tag{2}$$

where f_k is a function or a hypothesis that is feasible to the given training data $\mathcal{D}$, and $P(y \mid f_k, \mathbf{x})$ is the probability of having a label y when the function/hypothesis is f_k given the attribute set $\mathbf{x}$. When we have the probability of $P(y \mid \mathcal{D}, \mathbf{x})$ for all kinds of labels y as in Eq. 2, we can predict the label to be the most probable label by

$$\arg\max_{y} P(y \mid \mathcal{D}, \mathbf{x}) \,. \tag{3}$$

This is called the *Bayes optimal classifier* [51]. We compute the integral instead of summation in Eq. 2 when we have an infinite hypothesis space.

We can generally use the probabilistic approach to obtain richer information for intrusion or malicious behavior detection. For instance, a 90% chance that a site is under attack is very much different from a 60% chance. Moreover, the solution from Bayes optimal classifier can be proved to be optimal [51]. However, building a successful prediction model based on the probabilistic approach is difficult when we do not have enough prior information when we need to apply Bayes rule, or do not have enough computation

[5] It can be a probability mass function or a probability density function.

resources to enumerate all items in Eq. 2 to estimate Bayes optimal classification result.

A systematic approach for probabilistic modeling is *Probabilistic Graphical Models*, which work under assumptions of independence or conditional independence. Popular choices include naïve Bayes, which assumes conditional independence between attributes given the label information, and hidden Markov models [58] for the detection problem given sequential data. PGM can encode complex interactions, including spatial and temporal relationships between attributes (or between data) in a graphical and modular form [38, 31]. One has to know that structural learning in PGM is generally difficult [76] unless we have small parent sets [14].

We can use a graphical model, a general purpose probabilistic model, to detect intrusions or other types of malicious behaviors. For instance, we can use HMM to predict whether the site is under attack at any moment given a series of observations. In this case, we aim at modeling the temporal relationship between the observed data. On the other hand, a group of hosts may share a similar safety status if they are in the same network domain or from the same social network. In this case, we can use directed or undirected graphical models to construct the spatial relationship between different hosts' status. In a graphical model, we can represent a host's safety status by a random variable, and we can model how a host interacts with another host with a link in between. Finally, *probabilistic inference* in the model can help us to check whether or not a set of hosts act similarly as a group. In practice, graphical models can deal with categorical attributes directly. In a situation where we have numerical attributes, we often must deal with attribute values that are either not drawn from well-defined distributions or are generated by unknown distributions that are hard to describe. In this case, we can choose either a *nonparametric* approach or transform numerical attributes to categorical ones before we apply the categorical type of graphical models.

At times, even the model itself is not a stochastic model; however, we may use a stochastic algorithm to find the model. Many learning algorithms, such as the online perceptron algorithm, Passive and Aggressive algorithm [17], and Confidence Weight algorithm [20] use an iterative approach to find the optimal solution. In this case, if we use a gradient descent algorithm to find the solution, we may use an incremental and randomized approach in which we feed the data into the training process one at a time, and randomly select the data for each moment. This approach is called *stochastic gradient descent* [10, 9]. The approach works better than other types of gradient descent algorithms because the stochastic manner has a chance to jump out of *local optimum*.

To use a gradient descent algorithm, we need to understand how fast the algorithm converges to the optimum. If we need a model that does not allow long training time, iterative algorithms may not be appropriate. However, we can use an approximate version to find the answer, such as using only a few data to compute the gradient in each run. That is another reason why stochastic gradient descent is favored over the typical gradient algorithm.

3.4 Transparent Models versus Black-box Models

When we look for a detection model, we aim to choose either a model that has a high detection accuracy, or a model that is *transparent* enough that domain experts and model users can understand the model and further improve/tune the model based on their domain expertise. A transparent model is a model that clearly shows what the useful features are to detect intrusions; on the other hand, a *black-box* model can provide high detection accuracy, but it may not be easily understood how the model achieves the high detection accuracy.

An example of a transparent model is a decision tree. Given a decision tree, we can predict the label of incoming data by sequentially checking the decisions and their outcomes from the tree root to one of the leaves with label information; on the other hand, we can convert the tree to a series of detection rules. Given either approach, the detection is transparent so that experts can manipulate the tree or the rules to improve the detection result. That is not quite the case for *artificial neural networks* (ANN) [7], a typical black-box model where we only know the weights between consecutive layers in networks. However, it is not comprehensive to us how the detection is done or how to further improve the detection result. A similar case is seen in support vector machines [68]. In an SVM model, we often work on the feature space rather than the input space when a nonlinear method is used to project original attributes to a high-dimensional space, called the *kernel trick* [11]. In a black-box model, detection features are usually abstract from any physical meanings or concrete features that can be measured or collected in the real world.

The distinction between the transparent model and the black-box model is not always clear. For the decision tree model, once we adopt the *boosting* technique to build an ensemble of trees to detect intrusions, the set of trees may not be as transparent as before, and we are no longer clear on what the most “influential factors” are to achieve high performance. On the other hand, for a single tree, we cannot conclude that all the attributes in the tree are important if the tree is very large. Similarly, interpreting the attribute’s importance based on its level in the tree is not always accurate because the interactions between not-always-independent attributes could be sophisticated. The attribute located in the tree root is the most important one in the set; however, the importance of any other attributes further down in the tree can only be interpreted *conditionally*. We can only find the second, third, etc. important attributes in the tree when we have independence between all the attributes in the set.

With the black-box models, ANNs, SVMs and others, we can indeed extract explainable features from the model once we acquire a detection model. For different black-box models, one of the most focused topics is to extract rules from them, such as the approaches proposed for SVMs [46, 5], and the ones proposed for ANNs [2]. For SVMs, an alternative approach for feature selection, called LASSO [65] is to apply L_1-norm in the optimization so that coefficients to unimportant attributes will be pushed to zero.

Whether to choose a transparent model or black-box model depends on how and what we need for a detection model. If an organization has enough man power as well as the expertise to deal with the detection model and the detection result, a transparent model is preferred. On the other hand, if a high accuracy model is the ultimate goal and there are no technicians with enough training to maintain the detection model, a black-box model or similar could be the solution. We also need to know that in an *adversarial* environment, choosing a black-box may be more favorable than a transparent model in the sense that the rules to decide a transparent model may be easily worked around and targeted by adversarial opponents. In recent years, people have consistently asked for intrusion detection models that can remain effective even under the adversarial environment, which is called *adversarial learning* [6, 43, 67].

3.5 Feature Selection and Feature Extraction

Researchers in machine learning, information security, and many other fields know how difficult it is to build an attribute set that is complete enough to design a robust model to detect intrusions. To build an attribute set for learning a detection model, we often face the trade-off between choosing a complete attribute set that may be high-dimensional, and a truncated attribute set that may not be complete but small enough to be handled for modeling. Clearly, to separate data of different patterns in a truncated space may not be possible. On the other hand, developing a high-dimensional model may suffer from the *curse of dimensionality* phenomenon [7] because the data are relatively sparse in high-dimensional space when we need to consider the distance between pairwise data points, for instance, for kNN classification.

When we have an attribute set that includes almost all necessary information for the detection, a good *representation* of the attributes is still helpful, at least in the following senses.

i. In some cases, learning or detection on the representation space can be as easy as a kNN classification/detection where we can use the simple Euclidean distance to separate data of different patterns [24].

ii. A good representation may imply a natural parameterization where we can extract "insights" from the representation. One of the well-known examples is that computer vision researchers can use Isomap [64], Locally Linear Embedding (LLE) [59] or Hessian eigenmaps [19] to find a good parameterization for a set of images where we have extremely high dimensional space if we use all pixels as the attribute dimensions.

Overall, we can do our best to find an attribute set that is as complete as possible, followed by a good representation, and to detect intrusions in the representation space.

A representation (or parameterization) system is a mapping from the original input space $(x_1, x_2, \ldots, x_n)$ that consists of n attributes to a new space $(\sigma_1(\mathbf{x}), \sigma_2(\mathbf{x}), \ldots, \sigma_d(\mathbf{x}))$ where $\mathbf{x} = (x_1, \ldots, x_n)$. The new space is called a representation space and each attribute in the new space is called a *feature*. Usually we are interested in a case where $d \ll n$, and the technique is called *dimensionality reduction*. When $\sigma_k = x_\ell$ for some ℓ for all k, such dimensionality reduction technique is called *feature selection*. Otherwise, it is called *feature extraction*.

Apart from the techniques that were mentioned above, other useful dimensionality reduction techniques include Principal Component Analysis (PCA) [32] and Multidimensional Scaling (MDS) [16] for the unsupervised cases, and Sliced Inverse Regression (SIR) [41] and its (nonlinear) kernelized version [73] for the supervised cases. PCA and MDS are linear techniques.

When we expect that data are located on a nonlinear *manifold*, we can choose Isomap, LLE or Hessian eigenmaps [19]. MDS is applied to the case when we are given pairwise distances rather than coordinates as the input. Note that MDS will produce a result identical to the one from PCA if we start from a distance matrix D that is computed from the Euclidean distances of known pairwise coordinates [1].

The dataset for intrusion detection may not be in a high-dimensional space, such as the well-known KDD dataset [33, 63] that was extracted from DARPA 1999 [37]. To detect malware, however, we may need to check the source code for static analysis and the system logs for dynamic analysis. We expect relatively high-dimensional data in this case. Regardless of whether data were originally in a low- or high-dimensional space, when we need to consider temporal or spatial dependencies in a problem, we may work on high-dimensional data where each datum is formed by combining a few data that are from different time or "space". For example, in a *time series* prediction problem, we can consider a dataset[6] $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t, \mathbf{x}_{t+1}, \ldots\}$ where data are given with time stamps. In this case, to consider the temporal relationship in the dataset, we can build a "giant" datum $\mathcal{X}_t = (\mathbf{x}_t, \mathbf{x}_{t+1}, \ldots, \mathbf{x}_{t+k})$, and find patterns within the set of giant data. The n-gram method that is often used for *text mining* belongs to this category. Similar cases can be applied when we have spatial dependencies between data, or have both spatial and temporal dependencies between data at the same time. Feature selection or extraction is also useful even when we do not have high-dimensional data, but a visualization of the data in low-dimensional space is necessary.

4 Case Study

We discuss two case studies in this article. The first security problem is to detect botnets given the network statistics, and in the second case, we study an account security problem in which we attempt to distinguish between account intruders and the genuine account owner. Before we discuss the two cases, let us compare the two cases in terms of the "checklists" that we reviewed in the last section. As we can see in Table 1, we should select different algorithms for different types of applications. There may not be a "perfect" algorithm for any given application; however, if we choose an inappropriate

[6] We may or may not have label information in this case.

Table 1: Comparison of two case studies following the guidelines in the previous section when choosing a learning algorithm.

	Botnet detection	Account security
Method	C4.5	Markov Chain, dissimilarity measure, kNN, SSVM, Isomap
Learning Procedure	Batch	Batch/Continuous
Supervised/Unsupervised	Supervised	Supervised
Deterministic/Stochastic	Deterministic	Stochastic
Transparent/Black-box	Transparent	Black-box
Feature Selection and Extraction	With vector inputs, without feature extraction	With distance inputs, using Isomap for data representation
Evaluation	Precision, recall, F-measure under 10-fold cross-validation, & ROC	10-fold cross-validation, visualization

algorithm, we cannot see how well statistical learning algorithms can benefit from solving a security problem.

4.1 Botnet Detection

Problem and Motivation Due to the distributed design and implementation of a botnet, a number of features can be observed and retrieved as the key to detect bot activities. In this case, we discuss one solution [29] that detects bot activities in a monitored network. It is assumed that a bot often has a differentiable failure pattern due to its distributed behavior. Since a victim or a command and control (C&C) server can be temporarily unavailable, network requests sent from a bot to those unavailable hosts would generate failures different from regular Internet applications. By monitoring failures caused by a single host for a short period, it is possible to determine whether the host is running bot software. In this case, we detection bot activities using the C4.5 decision tree algorithm.

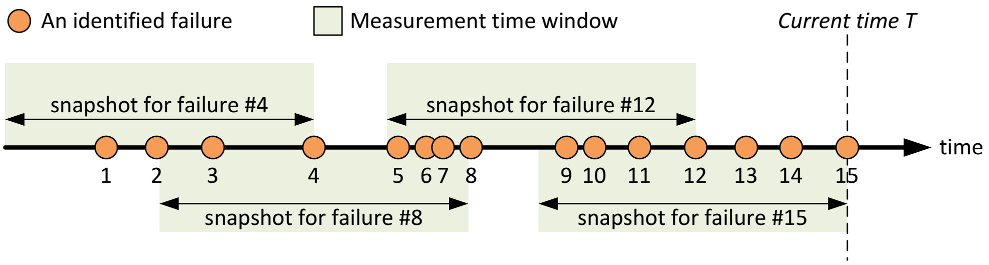


Figure 2: Examples of snapshots for a single host. Snapshots are used to calculate feature vectors.

Feature Collection One fundamental task in this case study is to identify proper features used to classify bots and non-bots. In the discussed solution, network failures are collected from TCP, UDP, ICMP, and DNS communications and the features are then obtained from a series of failures. To shorten the detection time, failures used to generate features are collected in a short period. The group of failures collected within a period is called a *snapshot of failures* (or simply *snapshot*) and the period is called a *measurement time window* (Δt). A snapshot is taken when a failure occurs, as in the example provided in Figure 2. Snapshots are taken on a per-host basis. Each snapshot is then transformed into a *feature vector*, which consists of numerous features. For the complete list of features please refer to [29]. The solution assumes that feature vectors for normal, peer-to-peer, and bot hosts are differentiable. Hence, by collecting numerous feature vectors and assigning proper labels to feature vectors, these data can be used to train and build a classification model to differentiate normal, peer-to-peer, and bot hosts.

Feature Selection and the Classification Model We collected approximately 30GB of normal traces, 2GB of peer-to-peer traces, and 2GB of botnet traces. Feature vectors generated from normal, peer-to-peer, and bot traces are labeled `NORM`, `P2P`, and `BOT`, respectively. It is not clear whether a selected feature is effective or not. A quick solution to evaluate the selected features is to look at their scatter plots. Figure 3 shows two scatter plots (out of 561 total plots) for various pairwise selected features. We find that a few features are not good for classification. For example, the left one in Figure 3 shows that `F9` is not a good feature because it is almost impossible to classify the three traces. Most of the rest of the figures, which look similar to the right one in Figure 3, should be useful for the purpose of classification.

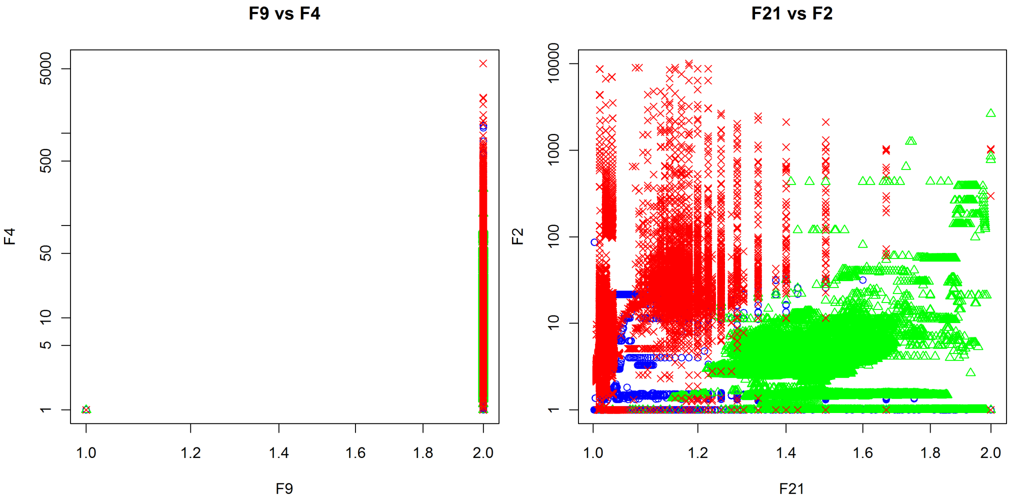
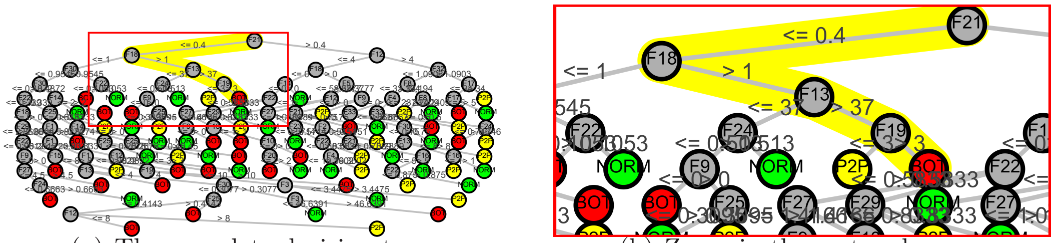


Figure 3: Scatter plots for selected pairwise features.



(a) The complete decision tree.

(b) Zoom in the rectangle area.

```
F13:  Total number of TCP RESETs.
F18:  Total number of UDP timeout failures.
F19:  Total number of DNS failures.
F21:  Ratio of distinct destination ports to all destination ports.
```

(c) Part of features used to detect bot activities in the zoom-in area.

Figure 4: Decision tree generated from the training traces. The measurement time window (Δt) is 120 seconds.

The classification model is built from the collected traces using the C4.5 algorithm [56]. Figure 4(a) shows the decision tree generated from feature samples collected in a 120-second time window. There are 137 nodes in the tree. The maximum tree depth is 14 and the maximum tree width is 18. The detection algorithm has to walk an average of seven to eight steps in the decision tree before it is able to make a decision. We notice that those less effective features are almost excluded from the resulting decision tree. We found only four, one, zero, and two nodes made decisions based on features F9, F10, F11, and F33 respectively. In addition to detecting bot activities, we can also infer detection rules given the decision tree. Based on the marked path {F21 → F18 → F13 → F19 → BOT} in Figure 4(b) and the features listed in Figure 4(c), we are able to know how a bot is detected and understand activities of one type of bot hosts:

(1) F21 is a small value ($\leq$0.4), indicating that this type of bot would use similar port numbers for communications.

(2) F18 is greater than one, indicating that this type of bot would have a few UDP failures.

(3) F13 is a larger value ($>$37), indicating that this type of bot often generates many TCP RESETs.

(4) F19 is greater than three, indicating that this type of bot would have several DNS failures.

Note that given such a transparent model, we can tune the model such as choosing between a less accurate but smaller tree, or a more accurate but larger tree; between a tree with fewer false negatives or fewer false positives, etc. The tree generated by the C4.5 algorithm is not guarantee to be optimal anyway, so an improvement from experts could be possible.

Performance The performance of the classifier was evaluated using the collected training traces as well. We used 10-fold cross-validation to verify the accuracy of the discussed solution. Figure 5 shows the precision, recall, F-measure, and false positive rates of all 10-fold cross-validation results. Different scales of measurement time windows ranging from tens of seconds to half a day were used to evaluate the performance. The results show that a larger measurement time window would obtain enhanced detection performance in accuracy and false positive rates. Although it is possible to use

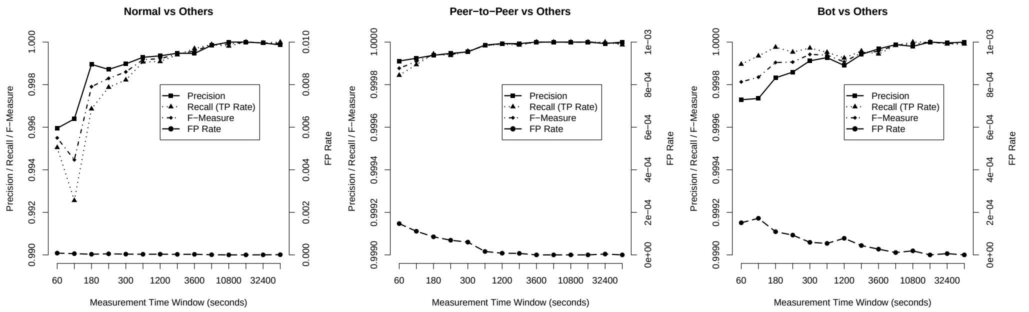


Figure 5: Summary of detection performance: Precision, recall, F-measure, and false positive rates.

a large measurement time window such as half a day (43200 seconds), the response time to detect an unwanted event is also delayed. Based on the detection performance, to have a balance between the required detection time and the detection performance, a measurement time window between 180 and 600 seconds would be ideal.

Using ROC curve plots is common to judge whether a classifier is better than another without giving any thresholds. Figure 6 shows ROC curve plots for measurement time windows of 60, 120, and 180 seconds. A perfect classifier would have an area under curve (AUC) of 1.0 [26]. The figures show that the AUCs for the three small measurement time windows are all greater than 0.999. We can conclude that the C4.5 decision tree algorithm performs well on classifying normal, peer-to-peer, and bot hosts with well-chosen features.

4.2 Account Security Based on Behavior Profiling

In the second case study, we discuss the account security problem. We use user behavior to decide the users' identity and verify whether the user is the genuine account owner.

Problem and Motivation To logon to a web service such as a bank, social network, or online game, a user normally needs to provide a password among other credentials for user authentication/verification. In this study, we discuss a statistical learning-based method that can analyze user behavior to verify whether or not a user is the genuine account owner. In reality,

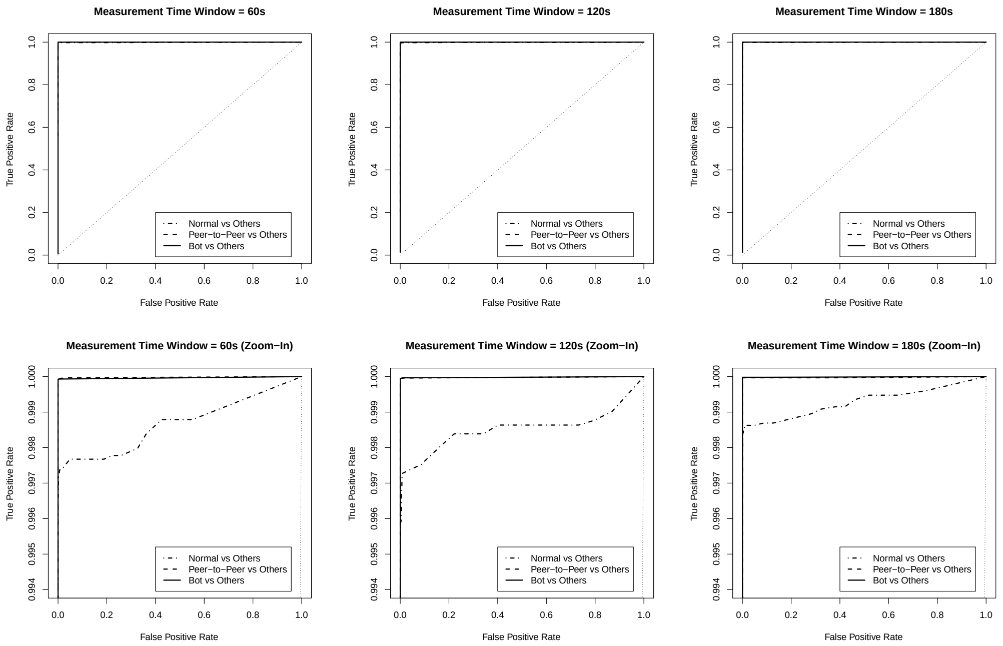
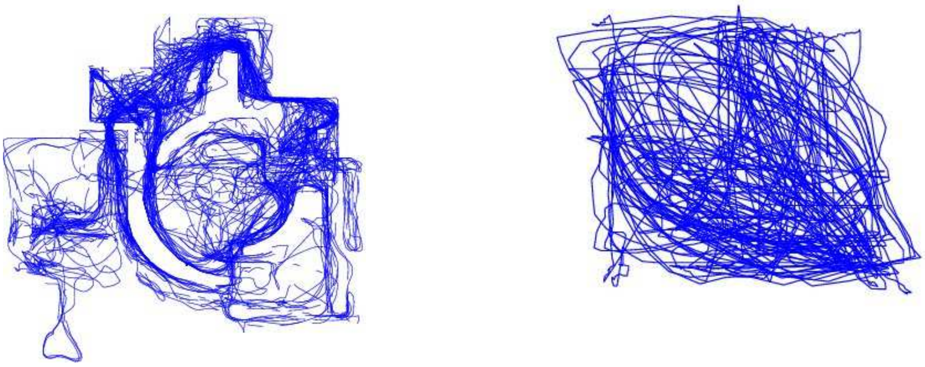


Figure 6: ROC curves for small measurement time windows.

we aim to detect both intruders, the ones not the genuine user and bots, automatic programs. In this case study, we are interested in using *user trajectories*, which are sequential coordinates of user inputs to represent user behavior to decide a user's identity.

Some input trajectory examples are shown in Figure 7, which include an online game trace made by a human user, and a mouse trace from a right-handed user. We shall test the same method for those various types of trajectories to demonstrate how effective our learning method is for a wide range of inputs. It is not surprising that a method built upon a "rigid" signature without statistical consideration may not detect unlawful access too well, given so many different kinds of inputs.

Classification Model We describe the proposed method and explain how we detect unlawful account users by exploiting the trajectory input. Given the trajectory input of varied length, a typical approach is to extract a set of features that form a *fixed*-length vector; then, we utilize a classifier to detect intruders based on the feature vector. In practice, building an informative



(a) Human playing online game (b) Mouse trace from a right-handed user

Figure 7: Different types of user trajectory: (a) is the trace of a human who played an online game called Quake 2; and (b) is the mouse trace that belongs to a right-handed user.

and complete feature set which needs domain expertise is usually difficult. Alternatively, in this case study, we compute dissimilarity/distance between each pair of trajectories. Once we have the dissimilarity measure, classifying a user to be the genuine account owner or intruders/bots becomes trivial given the user's trajectory. More about the implementation details can be found in [54].

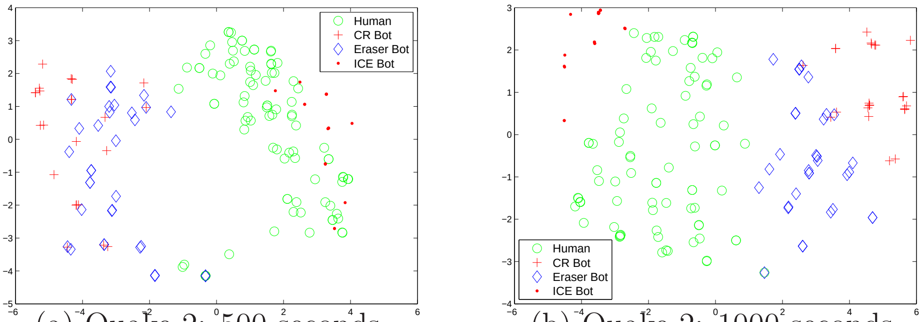
Formally, for each trajectory $\mathbf{s}$, we would like to use a model $\mathcal{M}$ to describe the trajectory. For instance, we can use a Markov chain or HMM to model the trajectory. Given a trajectory $\mathbf{s}$ and a description model $\mathcal{M}$, we compute the code length of the trajectory $\mathbf{s}$ with respect to the model $\mathcal{M}$ as a negative logarithm of the likelihood, as follows:

$$c(\mathbf{s} \mid \mathcal{M}) = -\ell(\mathbf{s}; \mathcal{M}) = -\log L(\mathbf{s}; \mathcal{M}) \,, \tag{4}$$

where $L(\mathbf{s}; \mathcal{M}) = P(\mathbf{s} \mid \mathcal{M})$ denotes the likelihood of $\mathbf{s}$ given the model $\mathcal{M}$. Based on the code length, we define the dissimilarity[7] between two trajectories $\mathbf{s}_1$ and $\mathbf{s}_2$ as follows:

$$d(\mathbf{s}_1, \mathbf{s}_2) = \frac{c(\mathbf{s}_1 \,|\, \mathcal{M}_2) + c(\mathbf{s}_2 \,|\, \mathcal{M}_1)}{c(\mathbf{s}_{12} \,|\, \mathcal{M}_{12})} - 1 \,, \tag{5}$$

[7] The smaller the value, the closer will be the relationship between $\mathbf{s}_1$ and $\mathbf{s}_2$.



(a) Quake 2: 500 seconds

(b) Quake 2: 1000 seconds

Figure 8: (a) (b) given inputs of different length, the representations of the Quake 2 traces after projection by Isomap into a 2-D space, where a point represents a human trace (green circles) or a bot trace (other symbols), after projection by Isomap into a 2-D space. The x- and y-axes are the first and second principal coordinates from Isomap. Classification is usually performed in a higher dimensional space, called the space of *intrinsic dimensionality*.

where $\mathcal{M}_i$ is the associated model for $\mathbf{s}_i$; also, $\mathbf{s}_{12}$ and $\mathcal{M}_{12}$ denote the trajectory that concatenates $\mathbf{s}_1$ and $\mathbf{s}_2$ one after another, and its associated model of $\mathbf{s}_{12}$ respectively. We have $d(\mathbf{s}_1, \mathbf{s}_2) \geq 0$ and $d(\mathbf{s}_1, \mathbf{s}_2) = d(\mathbf{s}_2, \mathbf{s}_1)$.

Given the pairwise dissimilarities of trajectories derived by Eq. 5, we either: (1) utilize a classification model that allows distance inputs, such as the kNN classifier, to determine if a trajectory is similar to the trajectories of the real account owner or if it belongs to an intruder; (2) adopt a map, such as (linear) MDS [16] or (non-linear) Isomap [64] to output data coordinates given pairwise data distances and apply a classification task afterwards. In this study, we choose a *manifold learning* approach called Isomap where we seek an embedded feature space to represent a set of trajectories. Figure 8 shows some examples of embedding in a 2-D space after applying Isomap. Note that the "optimal" dimensionality (also called the *intrinsic dimensionality*), where we can (more or less) separate different kinds of trajectories effectively, is not necessarily two dimensions. Ideally, we should be able to use any classifier in the feature space to determine whether a trajectory belongs to the true account owner or an intruder. In this study, we use SSVM [40] to evaluate the performance of the proposed method.

Table 2: Summary of the verification results for various inputs. The table shows the average error rates (in percentages) of the SSVM classification after performing ten-fold cross-validation three times.

Data Set	Training Error	Test Error
Handwriting	1.18	3.91
Mouse	6.67	8.10
Game	10.80	15.62

Performance In this study, we test the proposed method on three types of trajectories: online game traces, handwriting traces, and mouse traces. The game trace dataset is comprised of avatar movements obtained from Quake 2, a popular FPS game. The dataset comprises of human traces from many players and traces from well-known game bots such as CR Bot, Eraser Bot, and ICE Bot. The handwriting dataset from SVC 2004 handwritten signature verification competition is a benchmark for user verification. We also created the mouse movement dataset by ourselves based on 14 users' daily mouse controlling traces for a total of 217 instances. We aim to detect unlawful intruders and bots in the online game dataset and the intruders in the handwriting and mouse trace datasets. More details can be found in [13, 54].

We design the experiment as follows. First, given all trajectories, based on the dissimilarity measure in Eq. 5, we utilize Isomap to find the representation space for all the trajectories. Second, given two identities (a true account owner and an intruder), we select all trajectories belonging to the two identities in the representation space, and then we operate a binary classification (under ten-fold cross-validation, for three repeats) to label the trajectories as either belonging to the true account owner or not. Table 2 shows the performance of the proposed method on different kinds of trajectories. As expected, verification of the handwritten trajectory dataset achieves the best error rate (3.91%), followed by verification of the mouse trace (8.10%), and verification of the game trace (15.62%). Handwriting traces give us the best discriminative power because they are based on finer motions. In contrast, game traces are usually collected in a restricted environment, so they lack some degree of freedom in their movement to show the true identity of the trace owners.

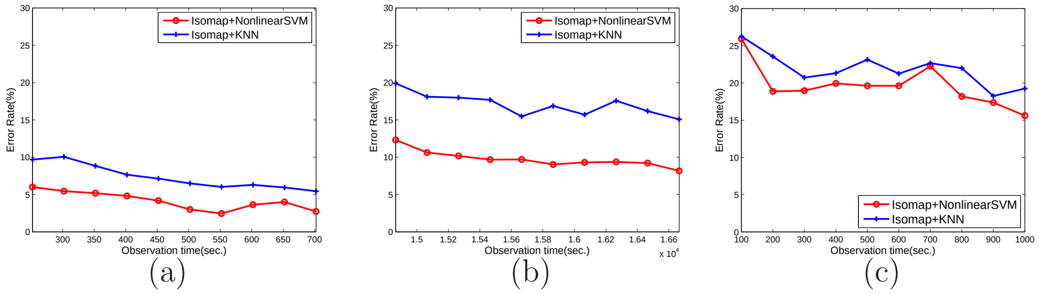


Figure 9: The test error rates obtained by applying two detection schemes, SSVM and kNN, to: (a) handwriting traces, (b) mouse traces and (c) online game traces, of different length. All verification errors decrease over time; also, the SSVM-based method outperforms the kNN-based method.

Since we want to verify the user's identity as early as possible given a trajectory, we are interested in analyzing the performance when only a shorter input trajectory is given, rather than waiting for a longer one. In Figure 9, given inputs such as handwriting traces, mouse traces, or online game traces of different length, we show the error rates of the verification task using SSVM and kNN. We use both SSVM and kNN as the classifiers after finding the representation by Isomap. The verification results (i.e., whether the trace belongs to the true account owner or not) can be summarized as follows. First, the verification errors on all types of inputs decrease when we use longer traces. Second, the SVM-based method outperforms the kNN-based method. Third, similar to the result in Table 2, handwriting trajectory dataset again achieves the best verification performance, followed by verification of the mouse traces, and verification of the game traces. In particular, we observe that for handwriting traces, after several hundred seconds (such as after observing 550 sampling points), the (nonlinear) SVM classifier yields an error rate as low as 2-4%.

5 Conclusion

Information security is an endless effort for computer scientists. The traditional method on signature matching can help detect known intrusions or threats efficiently. Based on statistical machine learning methods, we can find attacks with novel behavior. Moreover, statistical learning methods can adapt to the novel attacks constantly as the attacks evolve over time. The statistical learning methods for information security should not be considered as a substitute for traditional methods for intrusion or threat detection. In fact, having both approaches, the traditional signature-based and statistical learning methods work together could aid in fighting cyber-crime more effectively and efficiently than ever before. To correctly and thoughtfully use statistical learning methods, we should not be restricted to the most common methods. In fact, we should take into account different data statistics and properties to decide the best method to build our detection framework. Combining different disciplines that include domain knowledge and appropriate methodology, we can solve the problems that could not be solved before.

References

[1] Ethem Alpaydin. *Introduction to Machine Learning.* The MIT Press, Cambridge, MA, USA, 2004.

[2] Robert Andrews, Joachim Diederich, and Alan B. Tickle. Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-Based Systems*, 8(6):373–389, December 1995.

[3] Stefan Axelsson. The base-rate fallacy and the difficulty of intrusion detection. *ACM Trans. Inf. Syst. Secur.*, 3:186–205, August 2000.

[4] Pierre Baldi and Søren Brunak. *Bioinformatics - the Machine Learning Approach (2nd Edition).* MIT Press, 2001.

[5] Nahla H. Barakat and Andrew P. Bradley. Rule extraction from support vector machines: A review. *Neurocomputing*, 74(1-3):178–190, 2010.

[6] Marco Barreno, Peter L. Bartlett, Fuching Jack Chi, Anthony D. Joseph, Blaine Nelson, Benjamin I. P. Rubinstein, Udam Saini, and J. Doug Tygar. Open problems in the security of learning. In *Proceedings of the 1st ACM workshop on Workshop on AISec*, AISec '08, pages 19–26, New York, NY, USA, 2008. ACM.

[7] Christopher M. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006.

[8] Damiano Bolzoni, Sandro Etalle, and Pieter Hartel. Panacea: Automating attack classification for anomaly-based network intrusion detection systems. In *Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection*, 2009.

[9] Léon Bottou. Stochastic gradient learning in neural networks. In *Proceedings of Neuro-Nîmes 91*, Nimes, France, 1991. EC2.

[10] Léon Bottou. Online algorithms and stochastic approximations. In David Saad, editor, *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998. revised, oct 2012.

[11] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[12] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):15:1–15:58, July 2009.

[13] Kuan-Ta Chen, Andrew Liao, Hsing-Kuo Pao, and Hao-Hua Chu. Game bot detection based on avatar trajectory. In *Proceedings of IFIP ICEC 2008*, 2008.

[14] C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467, May 1968.

[15] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 3 edition, 2009.

[16] Trevor F. Cox and Michael A. A. Cox. *Multidimensional Scaling (2nd Edition)*. Chapman & Hall/CRC, 2000.

[17] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-aggressive algorithms. *J. Mach. Learn. Res.*, 7:551–585, December 2006.

[18] Jedidiah R Crandall, Zhendong Su, S Felix Wu, and Frederic T Chong. On deriving unknown vulnerabilities from zero-day polymorphic and metamorphic worm exploits. In *Proceedings of the 12th ACM conference on Computer and communications security*, pages 235–248. ACM, 2005.

[19] David L Donoho and Carrie Grimes. Hessian eigenmaps: Locally linear embedding techniques for high-dimensional data. *Proceedings of the National Academy of Sciences*, 100(10):5591–5596, 2003.

[20] Mark Dredze and Koby Crammer. Confidence-weighted linear classification. In *In ICML 08: Proceedings of the 25th international conference on Machine learning*, pages 264–271. ACM, 2008.

[21] Stephanie Forrest, Steven A. Hofmeyr, and Thomas A. Longstaff. A sense of self for unix processes. In *Proceedings of IEEE Symposium on Security and Privacy*, 1996.

[22] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1):119–139, 1997.

[23] N. Goernitz, Marius Kloft, Konrad Rieck, and Ulf Brefeld. Toward supervised anomaly detection. *J. Artif. Intell. Res. (JAIR)*, 46:235–262, 2013.

[24] Jacob Goldberger, Sam Roweis, Geoff Hinton, and Ruslan Salakhutdinov. Neighbourhood components analysis. In *Advances in Neural Information Processing Systems 17*, pages 513–520. MIT Press, 2004.

[25] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: an update. *SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[26] James A. Hanley and Barbera J. McNeil. A method of comparing the areas under receiver operating characteristic curves derived from the same cases. *Radiology*, 148(3):839–843, September 1983.

[27] Ching-Hsiang Hsu, Chun-Ying Huang, and Kuan-Ta Chen. Fast-flux bot detection in real time. In *Proceedings of the 13th International Symposium on Recent Advances in Intrusion Detection*, 2010.

[28] Xin Hu, Matthew Knysz, and Kang G. Shin. Measurement and analysis of global ip-usage patterns of fast-flux botnets. In *Proceedings of IEEE INFOCOM*, 2011.

[29] Chun-Ying Huang. Effective bot host detection based on network failure models. *Computer Networks*, 57(2):514–525, February 2013.

[30] Kai Hwang, Min Cai, Ying Chen, and Min Qin. Hybrid intrusion detection with weighted signature generation over anomalous internet episodes. *Dependable and Secure Computing, IEEE Transactions on*, 4(1):41–55, Jan 2007.

[31] Finn V. Jensen. *Bayesian Networks and Decision Graphs*. Springer, New York, 2001.

[32] Ian T. Jolliffe. *Principal Component Analysis*. Springer Verlag, 2 edition, 2002.

[33] KDD Cup Data. `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`.

[34] Rahul Khanna and Huaping Liu. System approach to intrusion detection using hidden markov model. In *Proceedings of the International Conference on Wireless Communication and Mobile Computing*, 2006.

[35] Marius Kloft and Pavel Laskov. Security analysis of online centroid anomaly detection. *J. Mach. Learn. Res.*, 13(1):3681–3724, December 2012.

[36] Donald E. Knuth, James H. Morris Jr., and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6(2):323–350, 1977.

[37] MIT Lincoln Lab. 1999 DARPA intrusion detection evaluation data set. `http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/data/1999data.html`, 1999.

[38] Steffen L. Lauritzen. *Graphical Models*. Clarendon Press, Oxford, 1996.

[39] Wenke Lee, Salvatore J. Stolfo, and Kui W. Mok. A data mining framework for building intrusion detection models. In *Proceedings of IEEE Symposium on Security and Privacy*, 1999.

[40] Yuh-Jye Lee and Olvi L. Mangasarian. SSVM: A smooth support vector machine for classification. *Comput. Optim. Appl.*, 20(1):5–22, 2001.

[41] Ker-Chau Li. Sliced inverse regression for dimension reduction. *J. Am. Statistical Assoc.*, pages 316–342, 1991.

[42] Carl Livadas, Robert Walsh, David Lapsley, and W. Timothy Strayer. Using machine learning technliques to identify botnet traffic. In *Proceedings of the 31st IEEE Conference on Local Computer Networks*, pages 967–974. IEEE, 2006.

[43] Daniel Lowd and Christopher Meek. Adversarial learning. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, KDD '05, pages 641–647, New York, NY, USA, 2005. ACM.

[44] Justin Ma, Lawrence K. Saul, Stefan Savage, and Geoffrey M. Voelker. Beyond blacklists: learning to detect malicious web sites from suspicious URLs. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1245–1254, New York, NY, USA, 2009. ACM.

[45] Justin Ma, Lawrence K. Saul, Stefan Savage, and Geoffrey M. Voelker. Identifying suspicious URLs: an application of large-scale online learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 681–688, New York, NY, USA, 2009. ACM.

[46] David Martens, Bart Baesens, and Tony Van Gestel. Decompositional rule extraction from support vector machines by active learning. *IEEE Transactions on Knowledge and Data Engineering*, 21(2):178–191, 2009.

[47] Hamed Masnadi-Shirazi and Nuno Vasconcelos. Risk minimization, probability elicitation, and cost-sensitive SVMs. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 759–766, 2010.

[48] Hamed Masnadi-Shirazi and Nuno Vasconcelos. Cost-sensitive boosting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(2):294–309, 2011.

[49] John McHugh. The lincoln laboratory intrusion detection evaluation: A critique. In *Proceedings of the 2000 DARPA Information Survivability Conference and Exposition*, 2000.

[50] John Mchugh. Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. *ACM Transactions on Information and System Security*, 3:262–294, 2000.

[51] Tom Mitchell. *Machine Learning*. McGraw Hill, 1997.

[52] Andrew Y. Ng and Michael I. Jordan. On discriminative vs. generative classifiers: a comparison of logistic regression and naive Bayes. In *Advances in Neural Information Processing Systems (NIPS)*, volume 14, 2001.

[53] Hsing-Kuo Pao, Yan-Lin Chou, and Yuh-Jye Lee. Malicious URL detection based on Kolmogorov complexity estimation. In *Proceedings of the The 2012 IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technology - Volume 01*, WI-IAT '12, pages 380–387, Washington, DC, USA, 2012. IEEE Computer Society.

[54] Hsing-Kuo Pao, Junaidillah Fadlil, Hong-Yi Lin, and Kuan-Ta Chen. Trajectory analysis for user verifcation and recognition. *Knowledge-Based Systems*, 34:81–90, 2012.

[55] Dean Pomerleau. ALVINN: An autonomous land vehicle in a neural network. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 1*. Morgan Kaufmann, 1989.

[56] J. Ross Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., 1993.

[57] J. Ross Quinlan. Bagging, boosting, and C4.5. In *AAAI/IAAI, Vol. 1*, pages 725–730, 1996.

[58] Lawrence R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[59] Sam T. Roweis and Lawrence K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

[60] Bernhard Schölkopf, Robert C. Williamson, Alex J. Smola, John Shawe-Taylor, and John C. Platt. Support vector method for novelty detection. In *NIPS*, pages 582–588, 1999.

[61] Robin Sommer and Vern Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, SP '10, pages 305–316, Washington, DC, USA, 2010. IEEE Computer Society.

[62] W. Timothy Strayer, David Lapsely, Robert Walsh, and Carl Livadas. Botnet detection based on network behavior. *Advances in Information Security*, 36:1–24, 2008.

[63] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A. Ghorbani. A detailed analysis of the KDD CUP 99 data set. In *Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on*, pages 1–6, 2009.

[64] Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, December 2000.

[65] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, 58:267–288, 1994.

[66] Grigorios Tsoumakas and Ioannis Katakis. Multi-label classification: An overview. *International Journal of Data Warehousing and Mining (IJDWM)*, 3(3):1–13, 2007.

[67] J.D. Tygar. Adversarial machine learning. *IEEE Internet Computing*, 15(5):4–6, 2011.

[68] Vladimir Naumovich Vapnik. *The Nature of Statistical Learning Theory*. Springer, 2000.

[69] Deepak Verma and Marina Meila. A comparison of spectral clustering algorithms. Technical report, 2003.

[70] Paul A. Viola and Michael J. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57(2):137–154, 2004.

[71] Christina Warrender, Stephanie Forrest, and Barak Pearlmutter. Detecting intrusions using system calls: Alternative data modelsy. In *Proceedings of IEEE Symposium on Security and Privacy*, 1999.

[72] Charles V. Wright, Fabian Monrose, and Gerald M. Masson. On inferring application protocol behaviors in encrypted network traffic. *Journal of Machine Learning Research*, 6:2745–2769, 2006.

[73] Yi-Ren Yeh, Su-Yun Huang, and Yuh-Jye Lee. Nonlinear dimension reduction with kernel sliced inverse regression. *IEEE Transactions on Knowledge and Data Engineering*, 21(11):1590–1603, 2009.

[74] Dit-Yan Yeung and Yuxin Ding. Host-based intrusion detection using dynamic and static behavioral models. *Pattern Recognition*, 36(1):229–243, 2003.

[75] Harry Zhang. The optimality of naïve bayes. In *FLAIRS Conference*, pages 562–567, 2004.

[76] Yang Zhou. Structure learning of probabilistic graphical models: A comprehensive survey. *CoRR*, abs/1111.6925, 2011.