A Geometric Framework for Unsupervised Anomaly Detection: Detecting Intrusions in Unlabeled Data

Eleazar Eskin, Andrew Arnold, Michael Prerau, Leonid Portnoy, Sal Stolfo
Department of Computer Science
Columbia University
New York, NY 10027
{eeskin,aoa5,mjp61,lp178,sal}@cs.columbia.edu

Abstract

Most current intrusion detection systems employ signature-based methods or data mining-based methods which rely on labeled training data. This training data is typically expensive to produce. We present a new geometric framework for *unsupervised anomaly detection*, which are algorithms that are designed to process unlabeled data. In our framework, data elements are mapped to a feature space which is typically a vector space $\Re^d$. Anomalies are detected by determining which points lies in sparse regions of the feature space. We present two feature maps for mapping data elements to a feature space. Our first map is a data-dependent normalization feature map which we apply to network connections. Our second feature map is a spectrum kernel which we apply to system call traces. We present three algorithms for detecting which points lie in sparse regions of the feature space. We evaluate our methods by performing experiments over network records from the KDD CUP 1999 data set and system call traces from the 1999 Lincoln Labs DARPA evaluation.

1 Introduction

Intrusion detection systems (IDSs) are an integral part of any complete security package of a modern, well managed network system. The most widely deployed and commercially available methods for intrusion detection employ signature-based detection. These methods extract features from various audit streams, and detect intrusions by comparing the feature values to a set of attack signatures provided by human experts. Such methods can only detect previously known intrusions since these intrusions have a corresponding signature. The signature database has to be manually revised for each new type of attack that is discovered and until this revision, systems are vulnerable to these attacks. Because of this, there has been a lot of research in the use of data mining and machine learning algorithms which train on labeled (i.e. with instances preclassified as being an attack or not) data to detect intrusions. These approaches leverage the generalization ability of data mining methods in order to detect new attacks.

There are two major paradigms for training data mining-based intrusion detection systems: *misuse detection* and *anomaly detection*. In misuse detection approaches, each instance in a set of data is labeled as normal or intrusion and a machine learning algorithm is trained over the labeled data. For example, the MADAM/ID system [21] extracts features from network connections and builds detection models over connection records that represent a summary of the traffic from a given network connection. The detection models are generalized rules that classify the data with the values of the extracted features. These approaches have the advantage of being able to automatically retrain intrusion detection models on different input data that include new types of attacks.

Traditional anomaly detection approaches build models of normal data and detect deviations from the normal model in observed data. Anomaly detection applied to intrusion detection and computer security has been an active area of research since it was originally proposed by Denning [6]. Anomaly detection algorithms have the advantage that they can detect new types of intrusions, because these new intrusions, by assumption, will deviate from normal usage [6, 15]. In this problem, given a set of normal data to train from, and given a new piece of data, the goal of the algorithm is to determine whether or not that piece of data is "normal" or is an anomaly. The notion of "normal" depends on the specific application, but without loss of generality, we can assume that this means stemming from the same distribution. We refer to this problem for reasons that will become clear below as *supervised anomaly detection*.

Most research in supervised anomaly detection can loosely be termed as performing generative modeling. These approaches build some kind of a model over the normal data and then check to see how well new data fits into that model. A survey of these techniques is given in [29]. An approach for modeling normal sequences using look ahead pairs and contiguous sequences is presented in [14], and a statistical method to determine sequences which occur more frequently in intrusion data as opposed to normal data is presented in [13]. One approach uses a prediction model obtained by training decision trees over normal data [19], while another one uses neural networks to obtain the model [11]. Ensemble-based approaches are presented in [9]. Lane and Brodley [18] evaluated unlabeled data for anomaly detection by looking at user profiles and comparing the activity during an intrusion to the activity during normal use. A technique developed at SRI in the EMERALD system [15] uses historical records as its normal training data. It then compares distributions of new data to the distributions obtained from those historical records and differences between the distributions indicate an intrusion. Recent works such as [31] and [8] estimate parameters of a probabilistic model over the normal data and compute how well new data fits into the model.

Supervised anomaly detection algorithms require a set of purely normal data from which they train their model. If the data contains some intrusions buried within the training data, the algorithm may not detect future instances of these attacks because it will assume that they are normal.

However, in practice, we do not have either labeled or purely normal data readily available. This makes the use of the traditional data mining-based approaches impractical. Generally, we must deal with very large volumes of audit data, and thus it is prohibitively expensive to classify it manually. We can obtain labeled data by simulating intrusions, but

then we would be limited to the set of known attacks that we were able to simulate and new types of attacks occurring in the future will not be reflected in the training data. Even with manual classification, we are still limited to identifying only the known (at classification time) types of attacks, thus restricting our detection system to identifying only those types. In addition, if we collect raw data from a network environment, it is very hard to guarantee that there are no attacks during the time we are collecting the data.

Recently there has been work on a third paradigm of intrusion detection algorithms, *unsupervised anomaly detection* (also known as anomaly detection over noisy data [7]), to address these problems [26]. These algorithms takes as input a set of unlabeled data and attempt to find intrusions buried within the data. In the unsupervised anomaly detection problem, we are given a set of data where it is unknown which are the normal elements and which are the anomalous elements. The goal is to recover the anomalous elements. Unsupervised anomaly detection is a variant of the classical outlier detection problem. After these anomalies or intrusions are detected and removed, we can then train a misuse detection algorithm or a traditional anomaly detection algorithm over the data.

In practice, unsupervised anomaly detection has many advantages over supervised anomaly detection. The main advantage is that they do not require a purely normal training set. Unsupervised anomaly detection algorithms can be performed over *unlabeled* data, which is easy to obtain since it is simply raw audit data collected from a system. In addition, unsupervised anomaly detection algorithms can be used to analyze historical data to use for forensic analysis.

In this paper, we present a geometric framework for unsupervised anomaly detection. Our frameworks maps the data to a feature space which are points in $\Re^d$. We then determine what points are outliers based on the position of the points in the feature space. We label points that are in sparse regions of the feature space as anomalies. Exactly how we determine which points are in a sparse region of the feature space is dependent on the specific algorithm within our framework that we are using. However, in general, our algorithms will detect anomalies because they will tend to be distant from other points.

A major advantage of our framework is its flexibility. We can define our mappings to feature spaces that better capture intrusions as outliers in the feature space. We can define mappings over different types of data such as network connection records and system call traces. Once we perform the mapping to the feature space, we can apply the same algorithm to these different kinds of data. For network data, we present a data-dependent normalization feature map specifically designed for outlier detection. For system call traces, we apply a spectrum kernel feature map. Using these feature maps, we are able to process both network data which is a vector of features and system call traces which are sequences of system calls using the same algorithms.

We present three algorithms for detecting outliers in the feature space. All of the algorithms are very efficient and can deal with high dimensional data. This is required for the application of intrusion detection. Our first algorithm is a variant of the cluster-based algorithm presented in Portnoy et al. [26]. The second algorithm is a k-nearest neighbor based algorithm. The third algorithm is a Support Vector Machine-based algorithm.

We evaluated our three unsupervised anomaly detection algorithms over two types of data sets, a set of network connections and sets of system call traces. The network data the

we examined was from the KDD CUP 99 data [1], which is a very popular and widely used intrusion attack data set. The system call data set was obtained from the 1999 Lincoln Labs DARPA intrusion detection evaluation [22]. Over both data sets, the methods show very promising results.

2 Unsupervised Anomaly Detection

In the unsupervised anomaly detection problem, we are given a large data set where most of the elements are normal, and there are intrusions buried within the data set [26]. Unsupervised anomaly detection algorithms have the major advantage of being able to process unlabeled data and detect intrusions that otherwise could not be detected. In addition, these types of algorithms can semi-automate the manual inspection of data in forensic analysis by helping analysts focus on the suspicious elements of the data.

Unsupervised anomaly detection algorithms make two assumptions about the data which motivate the general approach. The first assumption is that the number of normal instances vastly outnumbers the number of anomalies. The second assumption is that the anomalies themselves are qualitatively different from the normal instances. The basic idea is that since the anomalies are both different from normal and are rare, they will appear as outliers in the data which can be detected. An example of an intrusion that an unsupervised algorithm will have a difficulty detecting is a *syn-flood* DoS attack. The reason is that often under such an attack there are so many instances of the intrusion that it occurs in a similar number to normal instances. Our algorithms may not label these instances as an attack because the region of the feature space where they occur may be as dense as the normal regions of the feature space.

Unsupervised anomaly detection algorithms are limited to being able to detect attacks only when the assumptions hold over that data which is not always the case. For example, these algorithms will not be able to detect the malicious intent of someone who is authorized to use the network and who uses it in a seemingly legitimate way. The reason is that this intrusion is not qualitatively different from normal instances of the user. In our framework, these instances would be mapped very close to each other in the feature space and the intrusion would be undetectable.

A previous approach to unsupervised anomaly detection involves building probabilistic models from the training data and then using them to determine whether a given network data instance is an anomaly or not [7]. In this algorithm, a mixture model for explaining the presence of anomalies is presented, and machine learning techniques are used to estimate the probability distributions of the mixture to detect the anomalies.

There is recent work in distance based outliers which is similar to our approach to unsupervised anomaly detection [16, 17, 3]. These approaches examine inter-point distances between instances in the data to determine which points are outliers. A difference between these approaches and the problem of unsupervised anomaly detection is that the nature of the outliers are different. Often in network data, the same intrusion occurs multiple times which means there are many similar instances in the data. However, the number of instances of this intrusion is significantly smaller than the typical cluster of normal instances.

A problem related to unsupervised anomaly detection is the study of outliers in the field of statistics. Various techniques have been developed for detecting outliers in univariate, multivariate and structured data, using a given probability distribution. A survey of outliers in statistics is given by [2].

3 A Geometric Framework for Unsupervised Anomaly Detection

The key to our framework is mapping the records from the audit stream to a feature space. The feature space is a vector space typically of high dimension. Inside this feature space, we assume that some probability distribution generated the data. We wish to label the elements that are in low density regions of the probability distribution as anomalies. However, in general we do not know the probability distribution. Instead we label as anomalies points that are in sparse regions of the feature space. For each point, we examine the point's location within the feature space and determine whether or not it lies in a sparse region of the feature space. Exactly how we determine this depends on the algorithm that we are using.

The choice of algorithm to determine which points lie in sparse regions and the choice of the feature map is application dependent. However, critical to the practical use of these algorithms for intrusion detection is the efficiency of the algorithms. This is because the data sets in intrusion detection are typically very large.

3.1 Feature Spaces

Our data is collected from some audit stream of the system. Without loss of generality, this data is split into data elements $x_1, ..., x_l$. We define the space of all possible data elements as the *input (instance) space* X. Exactly what our input space is depends on the type of data that is being analyzed. Our input space can be the space of all possible network connection records, event logs, system call traces, etc.

We map elements of our input space to points in a *feature space* Y. In our framework a feature space is typically a real vector space of some high dimension d, $\Re^d$, or more generally a Hilbert space. The main requirement for our applications in the feature space is that we can define a dot product between elements of the feature space.

We define a *feature map* to be a function that takes as input an element in the input space and maps it to a point in the feature space. In general, we use ϕ to define a feature map and we get

$$\phi : X \to Y \quad . \tag{1}$$

We use the term *image* of a data element x to denote the point in the feature space $\phi(x)$.

Since our feature space is a Hilbert space, for any points y_1 and y_2 their dot product $< y_1, y_2 >$ is defined. Any time we have a dot product, we can also define a norm on the space as well as a distance.

The norm of a point y in the feature space $||y||$ is simply the square root of the dot product of the point with itself, $||y|| = \sqrt{< y, y >}$. Using this and the fact that a dot

product is a symmetric bilinear form we can define the distance between two elements of the feature space y_1 and y_2 with

$$\begin{aligned} ||y_1 - y_2|| &= \sqrt{< y_1 - y_2, y_1 - y_2 >} \\ &= \sqrt{< y_1, y_1 > -2 < y_1, y_2 > + < y_2, y_2 >} \quad . \end{aligned}$$

Using our framework, we can use the feature map to define relations between elements of the input space. Given two elements in the input space x_1 and x_2, we can use the feature map to define a distance between the two elements as the distance between their corresponding images in the feature space. We can define our distance function d_ϕ as

$$\begin{aligned} d_\phi(x_1, x_2) &= ||\phi(x_1) - \phi(x_2)|| \\ &= \sqrt{< \phi(x_1), \phi(x_1) > -2 < \phi(x_1), \phi(x_2) > + < \phi(x_2), \phi(x_2) >} \quad . \end{aligned} \tag{2}$$

For notational convenience, we often drop the subscript from d_ϕ.

If the feature space is $\Re^d$, this distance corresponds to standard Euclidean distance in that space.

3.2 Kernel Functions

In many cases, it is difficult to explicitly map a data instance to a point in its feature space. One reason is that the feature space has a very high dimension which makes it difficult to explicitly store the points in the feature space because of memory considerations. In some cases, the explicit map may be very difficult to determine.

However, for our purposes, we are interested in the dot products of points in the feature space. If have a mechanism for computing the dot product of the images of two data elements, we do not need to explicitly map the data elements to their images.

We can define a *kernel function* to compute these dot products in the feature space. A kernel function is defined over a pair of elements in the feature space and returns the dot product between the images of those elements in the feature space. More formally

$$K_\phi(x_1, x_2) =< \phi(x_1), \phi(x_2) > \quad . \tag{3}$$

We can redefine our distance measure (2) through a kernel function as

$$d_\phi(x_1, x_2) = \sqrt{K_\phi(x_1, x_1) - 2K_\phi(x_1, x_2) + K_\phi(x_2, x_2)} \quad . \tag{4}$$

In many cases, the kernel function can be computed efficiently without explicitly mapping the elements from the input space to their images. A function is a kernel function if there exists a feature space which is a Hilbert space and for which the kernel function corresponds to a dot product. There are conditions on whether or not a function is a kernel which are described in detail in [5].

An example of a kernel that performs the mapping implicitly is the radial basis kernel. The radial basis kernel is a function of the form

$$K_{\mathrm{rb}}(x_1, x_2) = e^{-\frac{||x_1 - x_2||^2}{\sigma^2}} \quad . \tag{5}$$

The radial basis kernel corresponds to an infinite dimensional feature space [5].

In addition to the computational advantages of kernels, we can define kernels to take advantage of our intuitions about the application. We can easily weight various features higher or lower depending on domain knowledge.

3.3 Convolution Kernels

Although the examples of kernels we have given up to this point have been defined over input spaces which are vector spaces, we can define kernels over arbitrary input spaces. These kinds of kernels are referred to as *convolution kernels* [12, 30].

In this framework, we can define our kernels directly over our audit data without needing to first convert our data into a vector in $\Re^n$. In addition, since kernels can be defined on not only numerical features, but other types of structures such as sequences, we can define kernels to handle many different types data such as sequences of system calls and event logs. This allows us to handle different kinds of data in a consistent framework using different kernels but using the same algorithms which are defined in terms of kernels.

4 Detecting Outliers in Feature Spaces

After mapping our data to points in the feature space, we can more easily formalize the problem of unsupervised anomaly detection. The main idea is that we want to detect points that are distant from most other points or in relatively sparse regions of the feature space. This is similar to the problem of outlier detection.

We present three algorithms for detecting anomalies in the feature space. All of the algorithms can be implemented in terms of dot products of the input elements which allows us to use kernel functions to perform implicit mappings to the feature space. Each algorithm detects points that lie in sparse regions in a different way.

The first algorithm is a variant of the cluster-based algorithm presented in [26]. For each point, the algorithm approximates the density of points near the given point. The algorithm makes this approximation by counting the number of points that are within a sphere of radius w around the point. Points that are in a dense region of the feature space and contain many points within the ball are considered normal. Points that are in a sparse region of the feature space and contain few points within the ball are considered anomalies. We implement an efficient approximation to this algorithm. We first perform fixed-width clustering over the points with a radius of w. We then sort the clusters based on the size. The points in the small clusters are labeled anomalous. We present details of the efficient algorithm below.

The second algorithm detects anomalies based on computing the k-nearest neighbors of each point. If the sum of the distances to the k-nearest neighbors is greater than a threshold, we consider the point an anomaly. We present below an efficient algorithm to detect outliers which uses a fixed-width clustering algorithm to significantly speed up the computation of the k-nearest neighbors.

The third algorithm is a support vector machine based algorithm that identifies low support regions of a probability distribution by solving a convex optimization problem [28]. The points in the feature space are further mapped into another feature space using a

Gaussian kernel. In this second feature space, a hyperplane is drawn to separate the majority of the points away from the origin. The remaining points represent the outliers or anomalies.

Each of these three algorithms is optimizing a clearly defined objective function over points in the feature space. We can present efficient algorithms to solve these problems given their formulation.

5 Algorithm 1: Cluster-based Estimation

The goal for our first algorithm is to compute how many points are "near" each point in the feature space. A parameter to the algorithm is a radius w also referred to as the cluster width. For any pair of points x_1 and x_2 we consider the two points "near" each other if their distance is less that or equal to w, $d(x_1, x_2) \leq w$ with distance defined as in equation (2).

For each point x, we define $N(x)$ to be the number of points that are within w of point x. More formally we define

$$N(x) = |\{s|d(x,s) \leq w\}| \quad . \tag{6}$$

The straightforward computation of $N(x)$ for all points has a complexity of $O(n^2)$ where n is the number of points. The reason is that we have to compute the pairwise distances between all points.

However, since we are really only interested in determining what the outliers are, we can effectively approximate the computation as follows.

We first perform fixed-width clustering over the entire data set with cluster width w. Then we label the points in the small clusters as anomalies.

A fixed width clustering algorithm is as follows. The first point is center of the first cluster. For every subsequent point, if it is within w of a cluster center, it is a added to that cluster. Otherwise it is a center of a new cluster. Note that some points may be added to multiple clusters. The fixed width clustering algorithm requires only one pass through the data. The complexity of the algorithm is $O(cn)$ where c is the number of clusters and n is the number of data points. For a reasonable w, c will be significantly smaller than n.

Note that by the definition in (6), for each cluster, the number of points near the cluster center, $N(c)$, is the number of points in the cluster c. For each point x, not a center of a cluster, we approximate $N(x)$, by $N(c)$ for the cluster c that contains x. For points in very dense regions where there is a lot of overlap between clusters, this will be an inaccurate estimate. However, for points that are outliers, there will be relatively few overlapping clusters in these regions and $N(c)$ will be an accurate approximation of $N(x)$. Since we are only interested in the points that are outliers, the points in the dense regions will be higher than the threshold anyway. Thus the approximation is reasonable in our case.

With the efficient approximation algorithm, we are able to process significantly larger data sets then possible with the straightforward algorithm because we do not need to perform a pairwise comparison of points.

6 Algorithm 2: K-nearest neighbor

Our second algorithm determines whether or not a point lies in a sparse region of the feature space by computing the sum of the the distances to the k-nearest neighbors of the point. We refer to this quantity as the k-NN score for a point.

Intuitively, the points in dense regions will have many points near them and will have a small k-NN score. If the size of k exceeds the frequency of any given attack type in the data set and the images of the attack elements are far from the images of the normal elements, then the k-NN score is useful for detecting these attacks.

The main problem with computing the k-NN score is that it is computationally expensive to compute the k-nearest neighbors of each point. The complexity of this computation is $O(n^2)$ which is impractical for intrusion detection applications.

Since we are interested in only the k-nearest points to a given point, we can significantly speed up the algorithm by using a technique similar to canopy clustering [23]. Canopy clustering is used as a means of breaking down the space into smaller subsets so as to remove the necessity of checking every data point. We use the clusters as a tool to reduce the time of finding the k-nearest neighbors.

We first cluster the data using the fixed-width clustering algorithm of the previous section with a variation where we place each element into only one cluster. Once the data is clustered with width w, we can compute the k-nearest neighbors for a given point x by taking advantage of the following properties.

We denote as $c(x)$ the point which is the center of the cluster that contains a given point x. For a cluster c and a point x we use the notation $d(x, c)$ to denote the distance between the point and the cluster center. For any two points x_1 and x_2, if the points are in the same cluster

$$d_\phi(x_1, x_2) \leq 2w \tag{7}$$

and in all cases

$$d_\phi(x_1, x_2) \leq d_\phi(x_1, c(x_2)) + w \tag{8}$$

$$d_\phi(x_1, x_2) \geq d_\phi(x_1, c(x_2)) - w \quad . \tag{9}$$

The algorithm uses these three inequalities to determine the k-nearest neighbors of a point x.

Let C be a set of clusters. Initially C contains all of the clusters in the data. At any step in the algorithm, we have a set of points which are potentially among the k-nearest neighbor points. We denote this set P. We also have a set of points that are in fact among the k-nearest neighbor points. We denote this set K. Initially K and P are empty. We precompute the distance from x to each cluster. For the cluster with center closest to x, we remove it from C and add all of its points to P. We refer to this operation as "opening" the cluster. The key to the algorithm is that we can obtain a lower bound the distance from all points in the clusters in set C using equation (9). We define

$$d_{\min} = \min_{c \in C} d(x, c) - w \quad . \tag{10}$$

The algorithm performs the following. For each point in $x_i \in P$, we compute $d(x, x_i)$. If $d(x, x_i) < d_{\min}$, we can guarantee that x_i is closer point to x then all of the points in the

clusters in C. In this case, we remove x_i from P and add it to K. If we can not guarantee this for any element of P (including the case that if P is empty), then we "open" the closest cluster by adding all of its points to P and remove that cluster from C. Notice that when we remove the cluster from C, $d_{\min}$ will increase. Once K has k elements, we terminate.

Most of the computation is spent checking the distance between points in D to the cluster centers. This is significantly more efficient than computing the pairwise distances between all points.

The choice of width w does not affect the k-NN score, but instead only affects the efficiency of computing the score. Intuitively, we want to choose a w that splits the data into reasonably sized clusters.

7 Algorithm 3: One Class SVM

Our third algorithm uses an algorithm presented in [28] to estimate the region of the feature space where most of the data occurs. We first map our feature space into a second feature space with a radial basis kernel, equation (5), and then work in the new feature space.

The standard SVM algorithm is a supervised learning algorithm. It requires labeled training data to create its classification rule. In [28], the SVM algorithm was adapted into an unsupervised learning algorithm. This unsupervised variant does not require its training set to be labeled to determine a decision surface.

Whereas the supervised version of SVM tries to maximally separate two classes of data in feature space by a hyperplane, the unsupervised version instead attempts to separate the entire set of training data from the origin. This is done by solving a quadratic program that penalizes any points not separated from the origin while simultaneously trying to maximize the distance of this hyperplane from the origin. At the end of this optimization, this hyperplane then acts as our decision function, with those points that it separates from the origin classified as normal, and those which are on the other side of the hyperplane, we classify as anomalous.

The algorithm is similar to the standard SVM algorithm in that it uses kernel functions to perform implicit mappings and dot products. It also uses the same kind of hyperplane for the decision surface. The solution is only dependent on the support vectors as well. However, the support vectors are determined in a different way. This algorithm attempts to find a small region where most of the data lies and label points in that region as class $+1$. Points in other regions are labeled as class -1. The main idea is that the algorithm attempts to find the hyperplane that separates the data points from the origin with *maximal margin*. The decision surface that is chosen is determined by solving an optimization problem that determines the "best" hyperplane under a set of criteria which is beyond the scope of this paper and described fully in [5].

The specific optimization that is solved for estimating the hyperplane specified by the hyperplane's normal vector in the feature space w and offset from the origin ρ is

$$\min_{w \in Y, \zeta_i \in \Re, \rho \in \Re} \quad \tfrac{1}{2}||w||^2 + \tfrac{1}{vl}\textstyle\sum_i^l \zeta_i - \rho \tag{11}$$

$$\text{subject to:} \quad (w \cdot \phi(x_i)) \geq \rho - \zeta_i, \zeta_i \geq 0 \tag{12}$$

where $0 < v < 1$ is a parameter that controls the trade off between maximizing the distance from the origin and containing most of the data in the region created by the hyperplane and corresponds to the ratio of expected anomalies in the data set. ζ_i are slack variables that penalize the objective function but allow some of the points to be on the other wrong side of the hyperplane.

After we solve the optimization problem, the decision function for each point x is

$$f(x) = \text{sgn}((w \cdot \phi(x)) - \rho) \quad . \tag{13}$$

If we introduce a Lagrangian and rewrite this optimization in terms of the Lagrange multipliers α_i we can represent the optimization as

$$\text{minimize: } \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j K_\phi(x_i, x_j)$$

$$\text{subject to: } 0 \leq \alpha_i \leq \frac{1}{vl}, \sum_i \alpha_i = 1$$

at the optimum, ρ can be computed from the Lagrange multipliers for any x_i such that the corresponding Lagrange multiplier α_i satisfies $0 < \alpha_i < \frac{1}{vl}$

$$\rho = \sum_j \alpha_j K_\phi(x_j, x_i)$$

In terms of the Lagrange multipliers, the decision function is

$$f(x) = \text{sgn}\left(\sum_i \alpha_i K_\phi(x_i, x) - \rho\right) \quad . \tag{14}$$

One property of the optimization is that for the majority of the data points, α_i will be 0 which makes the decision function efficient to compute.

The optimization is solved with a variant of the Sequential Minimal Optimization algorithm [25]. Details on the optimization, the theory behind the relation of this algorithm and the estimation of the probability density of the original feature space, and details of the algorithm are fully described in [28].

8 Feature Spaces for Intrusion Detection

The choice of feature space for unsupervised anomaly detection is application specific. The performance greatly depends on the ability of the feature space to capture information relevant to the application. For optimal performance, it is best to analyze the specific application and choose a feature space accordingly.

In our experiments, we analyze two data sets. The first data set is a set of network connection records. This data set contains records which contain 41 features describing a

network connection. The second data set is a set of system call traces. Each entry is a sequence of all of the system calls that a specific process makes during its execution.

We use two different feature maps for the different kinds of data that we analyze. For data which are records, we use a *data-dependent normalization kernel* to implicitly define the feature map. This feature map takes into account how abnormal a specific feature in a record is when it performs the mapping. We have found that this significantly improves detection performance.

For system call data where each trace is a sequence of system calls, we apply a string kernel over these sequences. The kernel we use is called a *spectrum kernel* which was previously used to analyze biological sequences [4]. The spectrum kernel maps short sub-sequences of the string into the feature space. This is consistent with our intuitions for intrusion detection because short sub-sequences have been the primary basis for analysis of system call sequences previously [10, 20, 8].

8.1 Data-dependent Normalization Kernels

For data which is a network connection record, we use a data-dependent normalization feature map. This feature map takes into account the variability of each feature in the mapping in such a way that normalizes the relative distances between feature values in the feature space.

There are two types of attributes in a connection record. There are either numerical attributes or discrete attributes. Examples of numerical attributes in connection records are the number of bytes in a connection or the amount of connections to the same port. An example of discrete attributes in network connection records is the type of protocol used for the connection. Even some attributes that are numerical are in fact discrete values, such as the destination port of a connection. We handle discrete and numerical attributes differently in our kernel mapping.

One problem with the straightforward mapping of the numerical attributes is that they are all on different scales. If a certain attribute is a hundred times larger than another attribute, it will dominate the second attribute.

We normalize all of our attributes to the number of standard deviations away from the mean. This scales our distances based on the likelihood of the attribute values. This feature map is data dependent because the distance between two points depends on the mean and standard deviation of of the attributes which in turn depends on the distribution of attribute values over all of the data.

For discrete values, we use a similar data dependent idea. Let Σ_i be the set of possible values for discrete attribute i. For each discrete attribute we have $|\Sigma_i|$ coordinates in the feature space corresponding to this attribute. There is one coordinate for every possible value of the attribute. A specific value of the attribute gets mapped to the feature space as follows. The coordinate corresponding to the attribute value has a positive value $\frac{1}{|\Sigma_i|}$ and the remaining coordinates corresponding to the feature have a value of 0. The distance between two vectors is weighted by the size of the range of values of the discrete attributes. A different value for attribute i between two records will contribute $\frac{2}{|\Sigma_i|^2}$ to the square of the norm between the two vectors.

8.2 Kernels for Sequences: The Spectrum Kernel

Convolution kernels can be defined over arbitrary input spaces. We use a kernel defined over sequences to model sequences of system calls. The specific kernel we use is a *spectrum kernel* which has been successfully applied to modeling biological sequences [4].

The spectrum kernel is defined over an input space of sequences. These sequences can be an arbitrary long sequence of elements from an alphabet Σ. For any $k > 0$, we define the feature space of the k-spectrum kernel as follows. The feature space is a $|\Sigma|^k$ dimensional space where each coordinate corresponds to a specific k length sub-sequence. For a given sequence, the value of a specific coordinate of the feature space is the count of the number of times the corresponding sub-sequence occurs in the sequence. These sub-sequences are extracted from the sequence by using a sliding window of length k.

The dimension of the feature space is exponential in k which makes it impractical to store the feature space explicitly. Note that the feature vectors corresponding to a sequence are extremely sparse. We can take advantage of this fact and efficiently compute the kernels between sequences using an efficient data structure as described in [4]. This is critical because in one of our experiments we consider 26 possible system calls and sub-sequences of length 4 which gives a dimension of the feature space of close to $500,000$.

9 Experiments

We perform experiments over two different types of data. We analyze network connection records, and system call traces.

9.1 Performance measures

To evaluate our system we were interested in two major indicators of performance: the *detection rate* and the *false positive* rate. The detection rate is defined as the number of intrusion instances detected by the system divided by the total number of intrusion instances present in the test set. The false positive rate is defined as the total number of normal instances that were (incorrectly) classified as intrusions divided by the total number of normal instances. These are good indicators of performance, since they measure what percentage of intrusions the system is able to detect and how many incorrect classifications it makes in the process. We calculate these values over the labeled data to measure performance.

The trade-off between the false positive and detection rates is inherently present in many machine learning methods. By comparing these quantities against each other we can evaluate the performance invariant of the bias in the distribution of labels in the data. This is especially important in intrusion detection problems because the normal data outnumbers the intrusion data by a factor of $100 : 1$. The classical accuracy measure is misleading because a system that always classifies all data as normal would have a 99% accuracy.

We plot ROC (Receiver Operating Characteristic) [27] curves depicting the relationship between false positive and detection rates for one fixed training/test set combination. ROC curves are a way of visualizing the trade-offs between detection and false positive rates.

9.2 Data Set Descriptions

The network connection records we used was the KDD Cup 1999 Data [1], which contained a wide variety of intrusions simulated in a military network environment. It consisted of approximately 4,900,000 data instances, each of which is a vector of extracted feature values from a connection record obtained from the raw network data gathered during the simulated intrusions. A connection is a sequence of TCP packets to and from some IP addresses. The TCP packers were assembled into connection records using the Bro program [24] modified for use with MADAM/ID [21]. Each connection was labeled as either normal or as exactly one specific kind of attack. All labels are assumed to be correct.

The simulated attacks fell in one of the following four categories : DOS - Denial of Service (e.g. a syn flood), R2L - Unauthorized access from a remote machine (e.g. password guessing), U2R - unauthorized access to superuser or root functions (e.g. a buffer overflow attack), and Probing - surveillance and other probing for vulnerabilities (e.g. port scanning). There were a total of 24 attack types.

The extracted features included the basic features of an individual TCP connection such as its duration, protocol type, number of bytes transferred, and the flag indicating the normal or error status of the connection. Other features of an individual connection were obtained using some domain knowledge, and included the number of file creation operations, number of failed login attempts, whether root shell was obtained, and others. Finally, there were a number of features computed using a two-second time window. These included - the number of connections to the same host as the current connection within the past two seconds, percent of connections that have "SYN" and "REJ" errors, and the number of connections to the same service as the current connection within the past two seconds. In total, there are 41 features, with most of them taking on continuous values.

The KDD data set was obtained by simulating a large number of different types of attacks, with normal activity in the background. The goal was to produce a good training set for learning methods that use labeled data. As a result, the proportion of attack instances to normal ones in the KDD training data set is very large as compared to data that we would expect to observe in practice.

Unsupervised anomaly detection algorithms are sensitive to the ratio of intrusions in the data set. If the number of intrusions is too high, each intrusion will not show up as anomalous. In order to make the data set more realistic we filtered many of the attacks so that the resulting data set consisted of 1 to 1.5% attack and 98.5 to 99% normal instances.

The system call data is from the BSM (Basic Security Module) data portion of the 1999 DARPA Intrusion Detection Evaluation data created by MIT Lincoln Labs [22]. The data consists of 5 weeks of BSM data of all processes run on a Solaris machine. We examined three weeks of traces of the programs which were attacked during that time. The programs we examined were *eject*, and *ps*.

Each of the attacks that occurred correspond to one or more process traces. An attack can correspond to multiple process traces because a malicious process can spawn other processes. We consider the attack detected if one of the processes that correspond to the attack is detected.

Tables 1 summarize the system call trace data sets and list the number of system calls

Table 1: Lincoln Labs Data Summary

Program Name	Total # of Attacks	# Intrusion Traces	# Intrusion System Calls	# Normal Traces	# Normal System Calls	% Intrusion Traces
ps	3	21	996	208	35092	2.7%
eject	3	6	726	7	1278	36.3%

and traces for each program.

9.3 Experimental Setup

For each of our data sets, we split the data into two portions. One portion, the training set, was used to set parameters values for our algorithms and the second, the test set, was used for evaluation.

We set parameters based on the training set. Then for each of the methods over each of the data sets, we varied the detection threshold and at each threshold computed the detection rate and false positive rate. For each algorithm over each data set we obtained a ROC curve.

Our parameter settings are as follows. For the cluster-based algorithm presented in Section 5, when processing the network connection data, we set the width of the fixed-width clustering to be 40 in the feature space. For the *eject* system call traces, the width was set to be 5. For the *ps* traces, the width was set to be 10.

For the k-nearest neighbor-based algorithm presented in Section 6, for the KDD cup data, the value of k was set to $10,000$ of the data set. For the *eject* data set, $k = 2$ and for the *ps* data set, $k = 15$. The k is adjusted to the overall size of the data.

For the SVM-based algorithm in Section 7, for the KDD cup data, we set $v = .01$ and $\sigma^2 = 12$. For the system call data sets, we used a value of $v = .05$ and $\sigma^2 = 1$.

9.4 Experimental Results

Our approach to unsupervised anomaly detection performed very well over both types of data.

In the case of the system call data, the each of the algorithms performed perfectly. What this means is that at a certain threshold, there was at least one process trace from each of the attacks identified as being malicious without any false positives. An explanation for these results can be obtained by looking at exactly what the feature space is encoding. Each system call trace is mapped to a feature space using the spectrum kernel that contains a coordinate for each possible sub-sequence. Process traces that contain many of the same sub-sequences of system calls are closer together than process traces that contain fewer sub-sequences of system calls.

The results are not surprising when we consider previous approaches to anomaly detection over system call traces. In the original work in this area [10], sub-sequences of system calls

Algorithm	Detection rate	False positive rate
Cluster	93%	10%
Cluster	66%	2%
Cluster	47%	1%
Cluster	28%	.5%
K-NN	91%	8%
K-NN	23%	6%
K-NN	11%	4%
K-NN	5%	2%
SVM	98%	10%
SVM	91%	6%
SVM	67%	4%
SVM	5%	3%

Table 2: Selected points from the ROC curves of the performance of each algorithm over the KDD Cup 1999 Data.

were the basis of the detection algorithm. The supervised anomaly detection algorithm presented in [10] recorded a set of sub-sequences that occurred in a normal training set and then detected anomalies in a test set by counting the number of unknown sub-sequences that occur within a window of 20 consecutive sub-sequences. If the number of previously unseen sub-sequences is above a threshold, then the method would determine that the process corresponds to an intrusion. The results of their experiments suggest that there a many sub-sequences that occur in malicious processes that do not occur in normal processes. This explains why in our feature space defined by the spectrum kernel, the intrusion processes are significantly far away from the normal processes. Also, in our experiments, the normal processes clumped together in the feature space. This is why the intrusion processes were easily detected as outliers.

For the network connections, the data is not nearly as regular as the system call traces. From our experiments, we found that there were some types of attacks that we were able to detect well and other types of attacks that we were not able to detect. This is reasonable because some of the attacks using our feature space were in the same region as normal data. Although the detection rates are lower than what is typically obtained for either misuse or supervised anomaly detection, the problem of unsupervised anomaly detection is significantly harder because we do not have access to the labels or have a guaranteed clean training set.

Figure 1 shows the performance of the three algorithms over the KDD Cup 1999 data. Table 2 shows the Detection Rate and False Positive Rate for some selected points from the ROC curves. All three algorithms perform relatively close to each other.

10 Discussion

This paper presents a geometric framework for unsupervised anomaly detection. The framework maps data elements to a feature space and then detects anomalies by determining

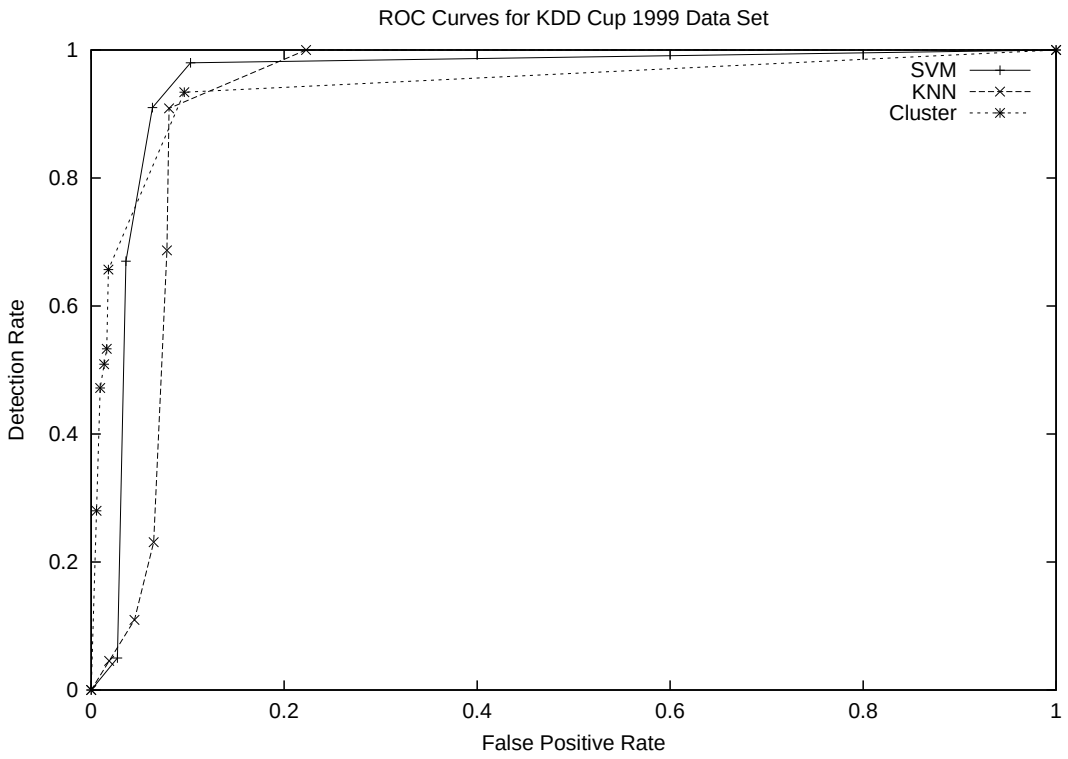

Figure 1. ROC curves showing the performance of the 3 algorithms over the KDD data set. The curves are obtained by varying the threshold.

which points lie in sparse regions of the feature space. We presented two feature maps. The first feature map is a data-dependent normalization feature map which is applied to network connection records. The second feature map is a spectrum kernel which is applied to system call traces. We also presented three algorithms for detecting points in the sparse regions of the feature space. The first algorithm is a cluster-based approach which is a variant of [26]. The second algorithm is a k-nearest neighbor-based approach. The third algorithm is a SVM-based approach. Each of these algorithms can be applied to any feature space which when combined with the different feature maps, allows us to apply the same algorithms to model different kinds of data.

We evaluated our methods over both a collection of network records and a set of system call traces. In both cases, our algorithms were able to detect intrusions over the unlabeled data. In practice, detecting intrusions over unlabeled data is important, because these algorithms can be applied to raw collected system data and do not need to be manually labeled which can be an expensive process.

Future work involves defining more feature maps over different kinds of data and performing more extensive experiments evaluating the methods presented in this paper.

References

[1] The third international knowledge discovery and data mining tools competition dataset KDD99-Cup `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`, 1999.

[2] V. Barnett and T. Lewis. *Outliers in Statistical Data.* John Wiley and Sons, 1994.

[3] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jorg Sander. LOF: identifying density-based local outliers. In *ACM SIGMOD Int. Conf. on Management of Data*, pages 93–104, 2000.

[4] Eleazar Eskin Christina Leslie and William Stafford Noble. The spectrum kernel: A string kernel for SVM protein classification. In *Proceedings of the Pacific Symposium on Biocomputing (PSB-2002)*, Kaua'i, Hawaii, 2002.

[5] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines.* Cambridge University Press, Cambridge, UK, 2000.

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

[7] E. Eskin. Anomaly detection over noisy data using learned probability distributions. In *Proceedings of the International Conference on Machine Learning*, 2000.

[8] Eleazar Eskin, Wenke Lee, and Salvatore J. Stolfo. Modeling system calls for intrusion detection with dynamic window sizes. In *Proceedings of DARPA Information Survivabilty Conference and Exposition II (DISCEX II)*, Anaheim, CA, 2001.

[9] W. Fan and S. Stolfo. Ensemble-based adaptive intrusion detection. In *Proceedings of 2002 SIAM International Conference on Data Mining*, Arlington, VA, 2002.

[10] Stephanie Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. A sense of self for unix processes. In *1996 IEEE Symposium on Security and Privacy*, pages 120–128. IEEE Computer Society, 1996.

[11] A. Ghosh and A. Schwartzbard. A study in using neural networks for anomaly and misuse detection. In *Proceedings of the 8th USENIX Security Symposium*, 1999.

[12] D. Haussler. Convolution kernels on discrete structures. Technical Report UCS-CRL-99-10, UC Santa Cruz, 1999.

[13] P. Helman and J. Bhangoo. A statistically base system for prioritizing information exploration under uncertainty. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 27(4):449–466, 1997.

[14] S. A. Hofmeyr, Stephanie Forrest, and A. Somayaji. Intrusion detect using sequences of system calls. *Journal of Computer Security*, 6:151–180, 1998.

[15] H. S. Javitz and A. Valdes. The NIDES statistical component: description and justification. In *Technical Report, Computer Science Laboratory, SRI International*, 1993.

[16] Edwin M. Knorr and Raymond T. Ng. Algorithms for mining distance-based outliers in large datasets. In *Proc. 24th Int. Conf. Very Large Data Bases, VLDB*, pages 392–403, 24–27 1998.

[17] Edwin M. Knorr and Raymond T. Ng. Finding intentional knowledge of distance-based outliers. *The VLDB Journal*, pages 211–222, 1999.

[18] T. Lane and C. E. Brodley. Sequence matching and learning in anomaly detection for computer security. In *AAAI Workshop: AI Approaches to Fraud Detection and Risk Management*, pages 43–49. AAAI Press, 1997.

[19] W. Lee and S. J. Stolfo. Data mining approaches for intrusion detection. In *Proceedings of the 1998 USENIX Security Symposium*, 1998.

[20] W. Lee, S. J. Stolfo, and P. K. Chan. Learning patterns from unix processes execution traces for intrusion detection. In *AAAI Workshop on AI Approaches to Fraud Detection and Risk Management*, pages 50–56. AAAI Press, 1997.

[21] W. Lee, S. J. Stolfo, and K. Mok. Data mining in work flow environments: Experiences in intrusion detection. In *Proceedings of the 1999 Conference on Knowledge Discovery and Data Mining (KDD-99)*, 1999.

[22] R. P. Lippmann, R. K. Cunningham, D. J. Fried, I. Graf, K. R. Kendall, S. W. Webster, and M. Zissman. Results of the 1999 darpa off-line intrusion detection evaluation. In *Second International Workshop on Recent Advances in Intrusion Detection (RAID 1999)*, West Lafayette, IN, 1999.

[23] Andrew McCallum, Kamal Nigam, and Lyle H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *Knowledge Discovery and Data Mining*, pages 169–178, 2000.

[24] V. Paxson. Bro: A system for detecting network intruders in real-time. In *Proceedings of the 7th USENIX Security Symposium*, San Antonio, TX, 1998.

[25] J. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 185–208, Cambridge, MA, 1999. MIT Press.

[26] Leonid Portnoy, Eleazar Eskin, and Salvatore J. Stolfo. Intrusion detection with unlabeled data using clustering. In *Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA-2001)*, Philadelphia, PA, 2001.

[27] Foster Provost, Tom Fawcett, and Ron Kohavi. The case against accuracy estimation for comparing induction algorithms. In *Proceedings of the Fifteenth International Conference on Machine Learning*, July 1998.

[28] B. Schölkopf, J. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson. Estimating the support of a high-dimensional distribution. Technical Report 99-87, Microsoft Research, 1999. To appear in *Neural Computation*, 2001.

[29] Christina Warrender, Stephanie Forrest, and Barak Pearlmutter. Detecting intrusions using system calls: alternative data models. In *1999 IEEE Symposium on Security and Privacy*, pages 133–145. IEEE Computer Society, 1999.

[30] C. Watkins. Dynamic alignment kernels. In A.J. Smola, P.L. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 39–50, Cambridge, MA, 2000. MIT Press.

[31] Nong Ye. A markov chain model of temporal behavior for anomaly detection,. In *Proceedings of the 2000 IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop*, 2000.