Intrusion Detection with Unlabeled Data Using Clustering

Leonid Portnoy, **Eleazar Eskin** and **Sal Stolfo**
Department of Computer Science
Columbia University
New York, NY 10027
{lp178,eeskin,sal}@cs.columbia.edu

Contact Author: Eleazar Eskin (eeskin@cs.columbia.edu)
Keywords: intrusion detection, anomaly detection, clustering, unlabeled data

Abstract

Intrusions pose a serious security risk in a network environment. Although systems can be hardened against many types of intrusions, often intrusions are successful making systems for detecting these intrusions critical to the security of these system. New intrusion types, of which detection systems are unaware, are the most difficult to detect. Current signature based methods and learning algorithms which rely on labeled data to train, generally can not detect these new intrusions. In addition, labeled training data in order to train misuse and anomaly detection systems is typically very expensive. We present a new type of clustering-based intrusion detection algorithm, *unsupervised anomaly detection*, which trains on unlabeled data in order to detect new intrusions. In our system, no manually or otherwise classified data is necessary for training. Our method is able to detect many different types of intrusions, while maintaining a low false positive rate as verified over the KDD CUP 1999 dataset..

1 Introduction

A network intrusion attack can be any use of a network that compromises its stability or the security of information that is stored on computers connected to it. A very wide range of activity falls under this definition, including attempts to destabilize the network as a whole, gain unauthorized access to files or privileges, or simply mishandling and misuse of software. Added security measures can not stop all such attacks. The goal of intrusion detection is to build a system which would automatically scan network activity and detect such intrusion attacks. Once an attack is detected, the system administrator is informed and can take corrective action.

Traditionally, signature based automatic detection methods have been used for this task. These methods extract features from the network data, and detect intrusions by comparing the feature values to a set of attack signatures provided by human experts. Obviously, such methods can not detect new types of intrusions because these intrusions do have have a corresponding signature. The signature database has to be manually revised for each new type of attack that is discovered. Other approaches use data mining and machine learning algorithms to train on labeled (i.e. with instances preclassified as being an attack or not) network data. These approaches the generalization ability of data mining methods in order to attempt 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. An example of a data mining-based misuse detection system is the MADAM/ID system [19], which extracted features from network connections and built detection models over connection records that represented a summary of the traffic from a given network connection. These detection models are generalized rules that classify the data using 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. We would have to insert many labeled instances of these new attacks into the dataset, and the method would readjust its rule sets to detect them.

Anomaly detection approaches build models of normal data and then attempts to detect deviations from the normal model in observed data. Anomaly detection algorithms have the advantage that they can detect new types of intrusions, because these new intrusions, by assumption, will deviate from normal network usage [5, 13]. Traditional 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, more often than not, we do not have either labeled or purely normal data readily available. Generally, we must deal with very large volumes of network data, and thus it is difficult and tiresome 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. Generating purely normal data is also very difficult in practice. 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.

In this paper, we present a new type of intrusion detection algorithm, *unsupervised anomaly detection* (also known as anomaly detection over noisy data [6]), to address these problems. This algorithm takes as inputs a set of unlabeled data and attempts to find intrusions buried within the data. After these intrusions are detected, we can apply train a misuse detection algorithm or a traditional anomaly detection algorithm over 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 intrusions. The second assumption is that the intrusions themselves are qualitatively different from the normal instances. The basic idea is that since the intrusions are both different from normal and rare, they will appear as outliers in the data which can be detected. Despite these inherent limitations, unsupervised anomaly detection algorithms have the major advantage of being able to process unlabeled data and detect some of the intrusions. In addition, these types of algorithms are useful for semi-automated detection in helping analysts focus on suspicious instances.

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 [6]. In that approach, the data was modeled using a probabilistic model that was known to perform well for that kind of data. In our current work, we drop the requirement of a probabilistic model and instead use inter-point distances to motivate our algorithm.

The approach we used and describe below, clusters the data instances together into clusters using a simple distance-based metric. This clustering is performed on unlabeled data, requiring only feature vectors without labels to be presented. Once the data is clustered, we label as anomalies all of the instances that appear in small clusters. The reason that this method works can be explained using the assumptions that we made about the data for unsupervised anomaly detection. Under the first assumption, the number of normal instances vastly outnumber the number of intrusion instances. This implies that the normal instances should form large clusters compared to the intrusions. Under the second assumption, since the intrusions and normal instances are qualitatively different, they will not fall into the same clusters.

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. Our algorithm may cluster these instances together and the intrusion would be undetectable. Another example is that the algorithm will have a difficulty detecting 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 algorithm may not label these instances as an attack because the size of the cluster may be as large as typical clusters of normal instances.

We evaluated our cluster-based unsupervised anomaly detection method over real network data. Both the training and testing was done using (different subsets of) KDD CUP 99 data [14], which is a very popular and widely used intrusion attack dataset. Various combinations of subsets of this dataset were used for training and testing, using standard cross validation techniques, each combination yielding slightly different results. On average, the detection rate was around 40%-55% with a 1.3%-2.3% false positive rate. Given the advantages of our method over traditional approaches, that the data was unlabeled, and our method uses almost no domain knowledge about security, these results indicate that this approach to unsupervised anomaly detection is promising.

1.1 Related work

Clustering is a well known and studied problem. It has been studied in many fields including statistics [24], machine learning [23], databases [11], and visualization. Basic methods for clustering include the Linkage based [3] and K-means [8] techniques. K-means makes several passes through the training data and on each pass shifts cluster centers to the mean of the data points assigned to that cluster. It then re-assigns data points to the nearest prototype, and continues iterating in this manner until no significant changes in cluster center positions occur. The K-means method generally produces a more accurate clustering than linkage based methods, but it has a greater time complexity and this becomes an extremely important factor in network intrusion detection due to very large dataset sizes. Although some optimizations of K-means for very large datasets exist , they still do not perform sufficiently fast for datasets with high dimensionality. Some other techniques for clustering include Clarans [20], Birch[26], density based methods such as Dbscan [7], and AI methods like Self-Organizing Maps [23]and Growing Networks [1].

Anomaly detection is a widely used method in the field of computer security, and there are approaches that utilize it for detecting intrusions [5]. Various techniques for modeling anomalous and normal data have been developed for intrusion detection. A survey of these techniques is given in [25]. An approach for modeling normal sequences using look ahead pairs and contiguous sequences is presented in [12], and a statistical method to determine sequences which occur more frequently in intrusion data as opposed to normal data is presented in [10]. One approach use a prediction model obtained by training decision trees over normal data [18], while another one uses neural networks to obtain the model [9]. Lane and Brodley [17] 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 [13] 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. The problem with this approach, however, is that if the historical distributions contain intrusions, the system may not be able to detect similar intrusions in the new instances.

Another algorithm for unsupervised anomaly detection is presented in [6]. 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 that is similar to our approach [15, 16, 4]. 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 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].

2 Methodology

In this section we describe the dataset and how it is used to build clusters and detect intrusions. We first examine what type of data was present in the dataset, what features were extracted, and what intrusion types were represented. Then, we discuss how the data was normalized based on the standard deviation of the training set, so that the system would be able to create clusters with data coming from different distributions. A description of the metric and the clustering algorithm follows, and finally the methods for labeling clusters and classifying unseen instances are discussed.

2.1 Dataset Description

The dataset used was the KDD Cup 1999 Data [14], 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 [21] modified for use with MADAM/ID [19]. 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 were 41 features, with most of them taking on continuous values.

2.2 Normalization

Since our algorithm is designed to be general, it must be able to create clusters given a dataset from an arbitrary distribution. A problem with typical data is that different features are on different scales. This causes bias toward some features over other features.

As an example, consider two 3-feature vectors, each set coming from different distributions : $\{(1, 3000, 2), (1, 4000, 3)\}$. Under an Euclidean metric, the squared distance between feature vectors will be $(1-1)^2 + (3000-4000)^2 + (2-3)^2$ which is dominated by the second column.

To solve this problem, we convert the data instances to a standard form based on the training dataset's distribution. That is, we make the assumption that the training dataset accurately reflects the range and deviation of feature values of the entire distribution. Then, we can normalize all data instances to a fixed range of our choosing, and hard code the cluster width based on this fixed range.

Given a training dataset, the average and standard deviation feature vectors are calculated :

$$avg\_vector[j] = \frac{1}{N}\sum_{i=1}^{N} instance_i[j]$$

$$std\_vector[j] = (\frac{1}{N-1}\sum_{i=1}^{N}(instance_i[j] - avg\_vector[j])^2)^{1/2}$$

where $vector[j]$ is the jth element (feature) of the vector.

Then each instance (feature vector) in the training set is converted as follows :

$$new\_instance[j] = \frac{instance[j] - avg\_vector[j]}{std\_vector[j]}$$

In other words, for every feature value we calculate how many standard deviations it is away from the average, and that result becomes the new value for that feature. Only continuous features were converted; symbolic ones were preserved as they were.

In effect this is a transformation of an instance from its own space to our standardized space, based on statistical information retrieved from the training set.

2.3 Metric

One of the main assumptions made was that data instances having the same label will tend to be closer together than instances with different labels under some metric. Therefore, finding or constructing an appropriate metric is critical to the performance of the method.

The particular choice of metric is likely to be dictated by the domain. In detecting network intrusions, it seemed at first that some features of the data instances would be important (have greater weight) than others, and thus differences in the values of those features should have a greater contribution to the overall distance. Therefore, we experimented with several weighted metrics, with higher weights assigned to different subsets of features.

However, in the end we used a standard Euclidean metric, with equally weighted features. One reason for this was that while the weighted metric did show some increase in performance, it was not a significant amount. But more importantly, tuning the metric's parameters to achieve maximum performance for a particular domain, data distribution, and feature set would undermine the system's generality and would contribute to over fitting.

Some features took on discrete values, and so there was an issue of how to factor them into the metric. The metric we used added a constant value to the squared distance between two instances for every discrete feature where they had two distinct values. This is equivalent to treating each different value as being orthologous in the feature space.

2.4 Clustering

To create clusters from the input data instances, we used a simple variant of single-linkage clustering. Although this is not the most effective clustering algorithm, it has the advantage of working in near linear time. The algorithm starts with an empty set of clusters, and generates the clusters with a single pass through the dataset. For each new data instance retrieved from the normalized training set, it computes the distance between it and each of the centroids of the clusters in the cluster set so far. The cluster with the shortest distance is selected, and if that distance is less than some constant W (cluster width) then the instance is assigned to that cluster. Otherwise, a new cluster is created with the instance as its center. More formally, the algorithm proceeds as follows :

Assume we have fixed a metric M, and a constant cluster width W. Let $dist(C, d)$ where C is a cluster and d is an instance, be the distance under the metric M, between C's defining instance and d. The defining instance of a cluster is the feature vector that defines the center (in feature space) of that cluster. We refer to this defining instance as the centroid.

1. Initialize the set of clusters, S, to the empty set.

2. Obtain a data instance (feature vector) d from the training set. If S is empty, then create a cluster with d as the defining instance, and add it to S. Otherwise, find the cluster in S that is closest to this instance. In other words, find a cluster C in S, such that for all C_1 in S, $dist(C, d) <= dist(C_1, d)$.

3. If $dist(C, d) <= W$, then associate d with the cluster C. Otherwise, d is more than W away from any cluster in S, and so a new cluster must be created for it : $S \leftarrow \ S \cup \{C_n\}$ where C_n is a cluster with d as its defining instance.

4. Repeat steps 2 and 3, until no instances are left in the training set.

2.5 Labeling clusters

Our hope is that under our metric, instances with the same classification are close together and those with different classifications are far apart. If an appropriate cluster width W was chosen, then after clustering we obtain a set of clusters with instances of a single type in each of them. This corresponds to our second assumption about the data that the normal and intrusion instances are qualitatively different.

Since we are dealing with unlabeled data, we do not have access to labels during training. Therefore, it is necessary to find some other way to determine which clusters contain normal instances and which contain attacks (anomalies). Our first assumption about the data is that normal instances constitute an overwhelmingly large portion ($> 98\%$) of the training dataset. Under this assumption it is highly probable that clusters containing normal data will have a much larger number of instances associated with them then would clusters containing anomalies. We therefore label some percentage N of the clusters containing the largest number of instances associated with them as 'normal'. The rest of the clusters are labeled as 'anomalous' and are considered to contain attacks.

A problem may arise with this approach, however, depending on how many sub-types of normal instances there are in the training set. For example, there may be many different kinds of normal network activity, such as using different protocols - ftp, telnet, www, etc. Each of these uses might have its own distinct point in feature space where network data instances for that use will tend to cluster around. This, in turn, might produce a large number of such 'normal' clusters, one for each type of normal use of the network. Each of these clusters will then have a relatively small number of instances associated with it - less than some clusters containing attack instances. Then these normal clusters will be incorrectly labeled as anomalous. To prevent this problem, we need to insure that the percentage of normal instances in the training set is indeed extremely large in relation to attacks. Then, it is very likely that each type of normal network use will have adequate (and larger) representation than each type or sub-type of attack.

2.6 Detection

Once the clusters are created from a training set, the system is ready to perform detection of intrusions. Given an instance d, classification proceeds as follows :

1. Convert d based on the statistical information of the training set from which the clusters were created. Let d' be the instance after conversion.

2. Find a cluster which is closest to d' under the metric M (i.e. a cluster C in the cluster set, such that for all C' in S, $dist(C, d') <= dist(C', d')$.

3. Classify d' according to the label of C (either normal or anomalous).

In other words, we simply find the cluster that is closest to d (converted) and give it that cluster's classification.

3 System evaluation and results

3.1 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.

3.2 Filtering the training dataset

The KDD dataset 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 dataset is very large as compared to data that we would expect to observe in practice.

Our second major assumption, however, states that the training set should represent normal network activity, where attacks are very rare and most of the data represents normal operation. Therefore, the raw KDD dataset obviously does not satisfy this condition. We trained the system with this raw set and obtained very poor performance, as was to be expected. To meet the requirement, we generated training sets from KDD data by filtering it for attacks. It was filtered such that the resulting training set consisted of 1 to 1.5% attack and 98.5 to 99% normal instances.

3.3 Parameter Estimation

There were two main parameters whose values needed to be fixed before performance could be measured. The first one is the cluster width for doing clustering, which determines how close two instances have to be to be assigned to the same cluster. The second is the percentage of the largest clusters N that would be labeled 'normal' during the detection phase. The goal was to set values for these two variables such that the performance over the entire domain would be maximized.

In this section we report results over the same dataset to give intuitions of how the dynamics of the parameters behave. In the following section we present results of testing over separate data sets to give a more accurate measure of the performance. We used a single subset (around 10%) of the KDD data to run a series of tests with different values for these two variables, measuring the resulting performance. A hazard is that the training set might represent a narrow spectrum of the domain and we might over fit the values of the two variables to that spectrum. However, the subset that we chose was representative of the entire KDD dataset, as it contained many instances of each type of attack.

Once we found the values for cluster width and the N that maximized results for that set, those values were fixed for all the subsequent experiments over different datasets. The two parameters are set to compare the best values over this type of data. Cluster width is a measure indicating the average radius in feature

Width	N	Detection rate	False positive rate
20	15%	35.7%	1.44%
20	7%	66.2%	2.7%
20	2%	88.%	8.14%

Table 1: These are the results of some tests to obtain the value of N (percentage of largest clusters to label as normal during detection). The cluster width was fixed for these tests.

Width	N	Detection rate	False positive rate
30	15%	28.1%	1.07%
40	15%	30.77%	0.84%
60	15%	31.9.%	0.7%
80	15%	22.84%	0.6%

Table 2: These are the results of some tests to obtain the value of the cluster width variable. Cluster width of 40 was chosen for subsequent tests.

space of a cluster containing instances of the same type. This is a particular property of the domain - network connection records. The N is also a property of the network - it attempts to measure the ratio of the number of sub-types of normal instances to the total number of different sub-types.

When fixing the values of the cluster width and percentage of largest clusters variables, and measuring performance on the single training/test set, the results are shown in Table 3.3.

We decided to use 15% as the value for N in subsequent tests, since it produced an acceptable false positive rate, without sacrificing too much detection rate. To find the value for cluster width we conducted several tests on the same training/test set combination, and with a fixed value for N The results of some of these tests are shown in Table 2.

Cluster width of 40 was chosen even though width=60 produced a slightly higher detection rate and a false positive rate. The difference was minor however, and tests on different datasets indicated that with width=60 performance was worse than with width=40.

Figure 1 shows an ROC (Receiver Operating Characteristic) [22] curve 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.

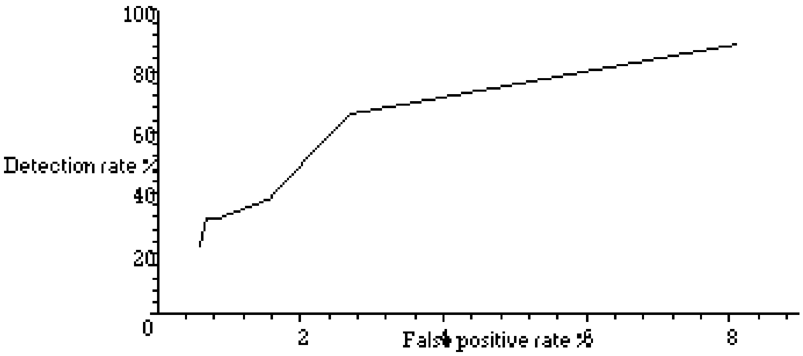

Figure 1. The ROC curve of false positive vs. detection rate, for a fixed training and test set combination.

3.4 Cross validation testing

Finally, after all parameters were specified, we evaluated the system by using a variant of the cross validation method. Cross validation is the standard technique used to obtain an estimation of a method's performance over unseen data.We partitioned the entire KDD dataset into ten subsets, each containing approximately 490,000 instances or 10% of the data. Unfortunately, the distribution of the attacks in the KDD dataset is very uneven which made cross validation very difficult. Many of these subsets contained instances of only a single type. For example, the 4th, 5th, 6th, and 7th 10% portions of the full dataset contained only SMURF attacks, and the data instances in the 8th were almost entirely NEPTUNE intrusions. Since we require that all intrusion (and normal) sub-types should be represented at least to some degree in the training dataset, we did not use these subsets because they failed to meet this requirement. For cross validation training only four of the ten subsets were selected. These four subsets contained a good mix of various intrusion types, and conformed to our necessary assumptions about the data. They were likely to produce a clustering that would be representative of many intrusions.

Each of these four subsets was then selected, and filtered such that the intrusion would constitute 1% of the resulting dataset. The system was trained on this filtered data, and the cluster set that resulted was saved. We then evaluated system performance of this cluster set over each of these four subsets, this time used as test sets. This process was repeated several times, with a different subset selected for training each time. The results are shown in Table 3.5.

The test sets were also filtered to contain approximately equal number of instances of each type of attack. This was necessary in order to have a meaningful measure of performance, since for example if 80% of intrusions in the test set were of a single type, then a detection rate of 81% would indicate that the system is well suited for detecting only this particular type of attack. If, however, the test sets contain an equal percentage of different types of instances, then an 81% detection rate would show the system as capable of detecting several different types of intrusions.

3.5 Variations to clustering and detection

In addition to the experiments with the cluster width and the constant indicating the percent of largest clusters to be labeled normal, we explored some variations to the clustering and detection methods, and the evaluated the performance over the single training and test sets.

The clustering method was altered by allowing multiple (two in the version we used) passes for the creation and assignment of instances to clusters. Previously, only one pass was made, during which for every instance a cluster nearest to it was found in the set of currently existing clusters, and the instance was assigned to that cluster if it was less than cluster width away (under the metric). If it was farther away, a new cluster was created for that instance. In this scheme the instances which appeared earlier in the training dataset had a smaller set of existing clusters to compare distance to. It was thought that this might have possibly resulted in a non-optimal assignment of an instance to a cluster, in the sense that if it was closest to some type or sub-type of instances and the cluster representing them was not yet present in the set, it would have been assigned (if it was within cluster width) to the closest cluster that was in the set at the time that instance was considered. That cluster would be a non-optimal choice, as it might represent a different type or sub-type than that of the instance which was assigned to it. To prevent this from occurring, we implemented a double pass method where we would first only create the clusters without assigning instances to them, and then during a second pass through the training set assign instances based on the closest cluster in this complete set.

The performance of the system with this change is shown in Table 3.5. Another variation was changing the clustering method. The performance obtained from changing the clustering method to use two passes was the same or worse than the performance of clustering with one pass.

The second variation was applied to the detection method, where instead of choosing the closest cluster to the presented instance and assigning it that cluster's classification (either normal or anomalous), we chose

Training set	Test set	Detection rate	False positive rate
P10	P1	55.7%	.99%
P10	P2	51.04%	1.58%
P10	P3	53.01%	1.67%
P10	P10	53.39%	1.04%
P2	P1	46.3%	.46%
P2	P2	22.0%	.70%
P2	P3	29.3%	2.35%
P2	P10	23.0%	9.83%
P1	P1	28.3%	4.5%
P1	P2	50.5%	1.26%
P1	P3	38.5%	3.45%
P1	P10	50.4%	11.37%
P3	P1	56.25%	.3%
P3	P2	18.56%	.6%
P3	P3	18.75%	.74%
P3	P10	23.0%	1.31%

Table 3: Performance of the system under various training and test set combinations. P1, P2, P3, and P10 represent the first, second, third, and the tenth 10% partitions of the 4,000,000 KDD CUP 99 dataset, respectively. Cluster width was set to 40, and 20% of largest clusters were marked as normal. Both the training and test sets were filtered prior to their use. The training was done over only 10% of the total data since there was enough data in this subset to for good clusters.

N	Detection rate	False positive rate
2	28.5%	.56%
3	51.3%	1.21%
4	47.2%	.93%
5	53.3%	1.61%
6	50.9%	1.36%
7	65.7%	1.78%

Table 4: Results for the labeling by majority variation to the detection method.

N closest clusters to that instance and assigned it the majority's classification (i.e. if a larger number of those N clusters were labeled anomalous then the instance was classified correspondingly, and as normal otherwise).

After experimenting with these changes and evaluating their performance on a test set as described below, we concluded that they did not improve detection accuracy and in some cases decreased the detection accuracy.

4 Analysis

The results from cross validation show that performance of our system depends heavily on which training set was used. In fact, it depends on how well the training set meets the requirement of representing a wide variety of intrusion and normal sub-types. As Table 4 shows, training on sets P2 or P1 resulted in a very high false positive rate compared to the other sets. A closer examination of those datasets revealed that they contained a smaller number of different normal sub-types than the other two sets. This resulted in the failure to create clusters for many normal regions of the feature space, and therefore data instances from those regions were assigned to incorrect clusters, possibly to those marked as anomalous. This may have caused the high false positive rate.

The training set P10 showed the best performance across all four of the test sets, with a high detection and a low false positive rate. When training on P10 and testing on the P3 sets, 53.01% detection and 1.67% false positive rates were obtained. On the other hand, when we reversed the situation by training on P3 and testing on P10, only a 23% detection rate was obtained (with a similar false positive rate). This can again be explained by the fact that in the P10 dataset more different types of intrusions were represented than in the P3 set, and therefore training on P10 resulted in a better cluster set than training on P3, which in turn manifested itself in the increased detection rate.

In an actual application of the system, the expected performance greatly depends on the composition of the data as shown with the variability of the detection rate over the different subsets. However, in all of these datasets, we have a significant detection rate with a low false positive which suggests that the method will be able to detect some of the attacks successfully.

4.1 Detection vs. false positive rates

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.

In our system, the false positive vs. detection rate trade-off was very apparent. As the percent of largest clusters to be labeled normal was decreased, detection rate increased substantially since a larger number of clusters were now labeled anomalous. The intrusion instances which were assigned to those clusters but were previously classified as normal (because those clusters were labeled normal), now were classified correctly as intrusions. However, at the same time the false positive rate also increased because all the normal instances assigned to clusters that were previously labeled normal and that now were labeled anomalous, were classified as intrusions as well. If those clusters indeed represented anomalous regions in the feature space, then those normal instances were assigned to them incorrectly, perhaps due to an sub-optimal metric or because the assumption that instances of the same type or sub-type will cluster together was not satisfied. However, the negative effect of this mis-assignment on performance could have been avoided if the percent of largest clusters to be labeled normal was not decreased.

To successfully utilize the system, then, a suitable value for that percentage must be found, one that would yield a high detection rate while keeping the false positive rate within a tolerable low value. If we assume that no mis-assignment of instances to clusters occurs, then this essentially amounts to measuring a property of the domain - the ratio of the number of sub-types of normal instances to the total number of different sub-types. This ratio will be reflected in the number of clusters representing normal regions of the feature space relative to the number of clusters representing all regions. In reality, when our assumptions are not met and mis-assignment occur, that ratio can be estimated indirectly, by noting the value for the percentage of largest clusters to be labeled normal which makes the false positive and detection rate combination most favorable. For example, we could choose a value which minimizes their sum (possibly weighted).

4.2 Variations to the Algorithms

It was concluded that the changes made to the clustering algorithm and to the detection method did not increase performance for several reasons. Changing the detection method to perform classification based on the majority of k nearest clusters' labels showed improved results for the single training and test set that were used to measure performance. The detection rate was generally higher for values of $k > 1$ than when k was equal to 1, while still keeping the false positive rate relatively low. However, the results varied greatly with k, with no apparent pattern as k increased. This lead us to suspect that the value of k which produced the best results was related to the particular training/test set that was used, and that it did not represent a value that increased performance over the entire domain. In other words, the number (k) of nearest clusters to be considered that yielded the best results, was in reality dependent on the training/test set combination and the portion of the domain it represented. Using this value for training on different sets might give different, less favorable, results. This suspicion of over fitting to the single training/test set was confirmed when we tested the labeling by majority method on other training and test set combinations. Results for those tests indicated that the method did not improve, and in some cases lowered, the performance.

The idea of changing clustering to use the double pass method was discarded immediately, after the results with that variation used were obtained. They showed that detection rate was about the same with false positive rate remaining the same or even slightly higher when using the double pass method than without using it. One possible explanation for the increased false positive rate is that with the double pass method less instances were being assigned to each cluster on average (because instances were now more evenly distributed across clusters). This could have led to the inability to differentiate between anomalous and normal clusters during the detection phase, since due to the more even distribution some truly normal clusters now had less instances assigned to them than previously. Therefore they might have been labeled as anomalous, and this increased the false positive rate.

5 Conclusion

The contribution that we presented in this paper was a method for detecting intrusions based on feature vectors collected from the network, without being given any information about classifications of these vectors. We designed a system that implemented this method, and it was able to detect a large number of intrusions while keeping the false positive rate reasonably low. There are two primary advantages of this system over signature based classifiers or learning algorithms that require labeled data in their training sets. The first is that no manual classification of training data needs to be done. The second is that we do not have to be aware of new types of intrusions in order for the system to be able to detect them. All that is required is that the data conform to several assumptions. The system then will try to automatically determine which data instances fall into the normal class and which ones are intrusions.

Even though the detection rate of the system we implemented is not as high as of those using algorithms relying on labeled data, our system is still very useful. Since no prior classification is required on the training data, and no knowledge is needed about new attacks, we can automate the process of training and creating new cluster sets. In practice, this would mean periodically (every 2 weeks for example) collecting raw data from the network, extracting feature values from it, and training on the resulting set of feature vectors. This will help detect new and yet unknown attacks. In addition, the method can be used for semi-automated detection by helping analysts focus on portions of the data that are more likely to contain intrusions.

5.1 Future work

Future work involves possible extensions or modifications to our method to achieve better performance and a better degree of automation. Currently, during detection clusters are labeled as either anomalous or normal according to the relative number of instances they contain. Another possibility would be to label clusters which are outliers in the feature space as anomalous, and all others as normal. This involves making the assumption that normal data of different sub-types will be clustered together, while sub-types of intrusion data will not be near the normal region of feature space.

To achieve a greater degree of automation, we can also determine the value for the percentage of largest clusters labeled normal N variable automatically, perhaps based on the standard deviation and average values of the number of instances in clusters. In that scheme, clusters containing only a 'small' (some fixed standard deviations lower than the mean) number of instances will be labeled anomalous. The advantage to this method is that in the current system as more new and unknown attacks are introduced into the network environment, the ratio of the number of normal sub-types to the total number of sub-types will decrease. Having a fixed value for N which does not reflect the decreased ratio will therefore cause the algorithm to label more clusters as normal, some of which should have really been labeled as anomalous. As a result, detection rate will decrease as more new intrusion types are introduced, and therefore periodic manual updates of the value for N will be required. If the system determines the value for N automatically, however, then no manual intervention will be required even over long periods of time.

References

[1] D. Touretzky B. Fritzke and T. Leen. A growing neural gas network learns topologies. *Advances in Neural Information Processing Systems*, 7:625–632, 1995.

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

[3] H.H. Bock. *Automatic Classification*. Vandenhoeck and Ruprecht, 1974.

[4] 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.

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

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

[7] M. Ester, H. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, pages 226–231, 1996.

[8] K. Fukunaga. *Introduction to Statistical Pattern Recognition, Second Edition*. Academic Press, Boston, MA, 1990.

[9] 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.

[10] P. Helman and J. Bhangoo. A stiatistically 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.

[11] Alexander Hinneburg and Daniel A. Keim. Clustering methods for large databases: From the past to the future. In Alex Delis, Christos Faloutsos, and Shahram Ghandeharizadeh, editors, *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadephia, Pennsylvania,USA*. ACM Press, 1999.

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

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

[14] KDD99. KDD99 cup dataset `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`, 1999.

[15] 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.

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

[17] 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.

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

[19] 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.

[20] R. Ng and J. Han. Efficient and effective methods for spatial data mining. In *Proceedings of Very Large Data Bases*, 1994.

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

[22] 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.

[23] R. Rojas. *Neural Networks - A systematic introduction*. Springer, Berlin, 1996.

[24] P. Schnell. A method for discovering data-groups. *Biometrica*, 6:47–48, 1964.

[25] 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.

[26] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: An efficient data clustering method for very large databases. In *Proc. 1996 ACM SIGMOD International Conference on Management of Data*, 1996.