Intrusion detection with unlabeled data using clustering

Leonid Portnoy
Data Mining Lab
Department of Computer Science
Columbia University
lp178@columbia.edu

Abstract

Intrusions pose a serious security threat in a network environment, and therefore need to be promptly detected and dealt with. New intrusion types, of which detection systems may not even be aware, 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. We present a framework for automatically detecting intrusions, new or otherwise, even if they are yet unknown to the system. 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.

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 de-stabilize the network as a whole, gain unauthorized access to files or priviledges, 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

IPR2023-00556
CrowdStrike EX1011 Page 1

and detect such intrusion attacks. Once an attack is detected, the system administrator could be informed and thus 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 using a preset hard-coded algorithm provided by human experts. Obviously, such methods can not adapt to new types of intrusions quickly, and the preset algorithm has to be manually revised for each new type of attack that is invented. Other approaches use data mining and machine learning algorithms to train on labeled (i.e. with instances pre-classified as being an attack or not) network data. An example of this would be a program called RIPPER, which when given a set of feature vectors with labels, will create generalized rules that best seem to classify the data using those features. These approaches have the advantage of being able to automatically retrain intrusion detection models on different input data when we need new types of attacks to be detected. 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 as well.

However, more often than not, we do not have labeled 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 occuring in the future will not be reflected in the training data. Therefore, in the end our intrusion detection system will not be able to detect new attacks. 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.

To solve these difficulties, we need a technique for detecting intrusions when our training data is unlabeled, as well as for detecting new and unknown types of intrusions. A method that offers promise in this task is anomaly detection. Anomaly detection detects anomalies in the data (i.e. data instances in the data that deviate from normal or regular ones). It also allows us to detect new types of intrusions, because these new types will, by assumption, be deviations from the normal network usage, just like the other types of intrusions.

There are several approaches to anomaly detection. Some use data known to be normal and use it as as a reference for detecting anomalous data [12]. This approach is an example of supervised anomaly detection, since the clas-

IPR2023-00556
CrowdStrike EX1011 Page 2

sification of the data must be known prior to training on it. Methods for unsupervised anomaly detection do not assume that the data is labeled or somehow otherwise sorted according to classification. One method 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 [5]. The approach we used and describe below, clusters similar data instances together into clusters and uses distance metrics on clusters to determine what is an anomaly. This clustering can be performed on unlabeled data, requiring only feature vectors without labels to be presented. There are several primary assumptions that this method relies upon. First, data instances having the same classification (type of attack or normal) should be close to each other in feature space under some reasonable metric, while instances with different classifications will be far apart. Also, the amount of instances in the training set that represent normal network activity will be overwhelmingly larger than the amount of intrusion instances.

It is very difficult, if not impossible, to detect malicious intent of someone who is authorized to use the network and who uses it in a seemingly legitimate way. For example, there is probably no highly reliable way to know whether someone who correctly logged into a system is the intended user of that system, or if the password was stolen. Our focus then is to be able to detect a particular subset of intrusion attacks - ones that can be objectively identified on the basis of network flow data alone. For instance, a 'denial of service (DOS)' attack can be argued to have a distinct set of features and patterns that manifest themselves when examining packets on the network.

Under these assumptions we built a system which created clusters from its input data, then automatically labeled clusters as containing either normal or anomalous data instances, and finally used these clusters to classify unseen network data instances as either normal or anomalous. Both the training and testing was done using (different subsets of) KDD CUP 99 data [13], 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, and that the data was unlabeled, these results are indicate good performance.

IPR2023-00556
CrowdStrike EX1011 Page 3

1.1 Related work

Clustering is a well known and studied problem. It has been studied in many fields including statistics [18], machine learning [17], databases [10], and visualization. Basic methods for clustering include the Linkage based [3] and K-means [7] 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 [16], Birch[20], density based methods such as Dbscan [6], and AI methods like Self-Organizing Maps [17]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 [4].

Various techniques for modeling anomalous and normal data have been developed for intrusion detection. A survey of these techniques is given in [19]. A method that is closely related to ours, assumes a data set containing large number of normal elements and relatively few anomalies [5]. A mixture model for explaining the presence of anomalies is presented, and machine learning techniques are used to estimate the probability distribution. An approach for modeling normal sequences using look ahead pairs and contiguous sequences is presented in [11], and a statistical method to determine sequences which occur more frequently in intrusion data as opposed to normal data is presented in [9]. One approach use a prediction model obtained by training on a decision tree applied over normal data [15], while another one uses neural networks to obtain the model [8]. Lane and Brodley [14] 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 [12] uses historical records as its normal training data. It then compares distributions of new data to the distributions obtained from those historical records and differ-

IPR2023-00556
CrowdStrike EX1011 Page 4

ences 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.

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 [13], 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, starting and ending at some well defined times. 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.

IPR2023-00556
CrowdStrike EX1011 Page 5

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 the system was designed to be general, it must be able to create clusters given a dataset from an arbitrary distribution. However, if we assign data instances to a cluster if they are closer than some constant distance (cluster width) to that cluster's center, a difficulty will arise. If this width is hardcoded into the algorithm, then it will be used for all datasets, possibly from different distributions. They will then have widely varying distances between instances (depending on the distribution they come from) and a constant hard-coded width for all distributions will not correctly determine when two instances are close enough to be put inside the same cluster.

As an example, consider two sets of two 3-feature vectors, each set coming from different distributions :

1. {(1,3,2),(1,4,3)}

2. {(900,1000,700),(1000,1100,800)}

Under an Euclidean metric, the squared distance between feature vectors in the first set will be $(1-1)^2 + (3-4)^2 + (2-3)^2 = 1$, while it will be $30,000$ for the second set. If our cluster width is a hard-coded constant (e.g. 1.5), it will not take into account the distribution that the input dataset is taken from, and will put instances in the first set into the same cluster, and the instances in the second set into different clusters.

IPR2023-00556
CrowdStrike EX1011 Page 6

One possible solution to this is to determine the width dynamically based on the training dataset. We take a different approach however, and make the width a hard-coded constant, but 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 attack type (or if both of them are normal) will be close together under some metric. Therefore, finding or constructing an appropriate metric is essential for clustering.

IPR2023-00556
CrowdStrike EX1011 Page 7

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, several weighted metrics were tried, with higher weights assigned to different subsets of features.

However, in the end it was decided to use 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 overfitting. Finally, an additional assumption made was that the feature set was selected such as to keep the degree of independence between features high. Because of this, and also since the data was transformed into a standardized space, an Euclidean metric was chosen for computing the distance between instances.

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.

2.4 Clustering

To create clusters from the input data instances, we used a simple variant of single-linkage clustering. The algorithm starts with an empty set of clusters, and updates it as it proceeds. 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, the instance is farther away than W from any cluster in the set, and therefore 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

IPR2023-00556
CrowdStrike EX1011 Page 8

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

If, under our metric, instances with the same classification are close together, 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. (Of course, in reality, these assumptions will rarely be fully satisfied).

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). If our second major assumption about normal instances constituting an overwhelmingly large portion ($> 98\%$) of the training dataset is satisfied, then 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'.

A problem might 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

IPR2023-00556
CrowdStrike EX1011 Page 9

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 an anomaly).

In other words, we simply find the cluster that is closest to d (converted) and give it that cluster's classification. Note that in the first step we are effectively transforming d into the space of the training set. This is done so that it would be meaningful to use the metric and find the cluster closest to the instance. According to an assumption stated earlier, the distribution of the training set should adequately represent the distribution and data of the entire domain, and therefore we can use its statistical information to transform new data into a standardized space. If that assumption is not satisfied and d comes from an entirely different distribution than the training set or is of a classification type that is not represented in it, then it would not follow that its closest cluster will contain instances of the same type as d.

IPR2023-00556
CrowdStrike EX1011 Page 10

3 System evaluation and results

In this section we detail how the various free parameters to the system were fixed, and the approach to evaluating its performance. The main free parameters that needed to be set before performance could be evaluated were the cluster width set during clustering, and the percentage of biggest clusters that were labeled normal during detection. Other parameters included details of the clustering and detection methods themselves. Several small variations of them were tried and we determined the ineffectiveness of these variations on improving performance. An additional parameter was the training data set itself, in the sense of deciding which part of and how to use it so that it would satisfy our assumption about the input data. Finally, once the parameters values were set, we used cross-validation testing to measure system performance.

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. To calculate these values, access to labels (classifications) of instances in the test set was required. This was used for measuring performance only, however, and in practical uses of the system the existence of labeled data is not assumed.

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.

IPR2023-00556
CrowdStrike EX1011 Page 11

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 Fixing free parameters

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 (PLC) 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.

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. Both training and testing was done on that same subset. This is justified because no information about the data's classification was used during training and therefore using the training set for testing is essentially the same as using a different set. The only hazard of overfitting is that the training set might represent a narrow spectrum of the domain and we might overfit 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 PLC that maximized results for that set, those values were fixed for all the subsequent experiments. It can be argued that such tuning of free variables will result in overfitting to the particular dataset used to train and measure performance. However, we believe that if that dataset represents the entire domain well, then values found for the free parameters will maximize performance for other datasets as well, and we would only be overfitting to the domain itself. Cluster width is a measure indicating the average radius in feature space of a cluster containing instances of the same type. This is a particular property of the domain - network connection records. The PLC is also a property of the network - it

IPR2023-00556
CrowdStrike EX1011 Page 12

Width	P.L.C.	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 P.L.C. (percentage of largest clusters to label as normal during detection). The cluster width was fixed for these tests.

Width	P.L.C.	Detection rate	False positive rate
40	15%	30.77%	0.84%
30	15%	28.1%	1.07%
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.

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 P.L.C. in subsequent tests (except cross validation), 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 P.L.C. 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. Since there were approximately 40 feature values in each feature vector, a width of 40 indicates that on average each feature value of an instance could differ by no more than 1 standard deviation from the defining instance of a cluster, in order for the instance to

IPR2023-00556
CrowdStrike EX1011 Page 13

be assigned to that cluster.

Figure 1 shows an ROC (Receiver Operating Characteristic) (Receiver Operating Characteristic) 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.

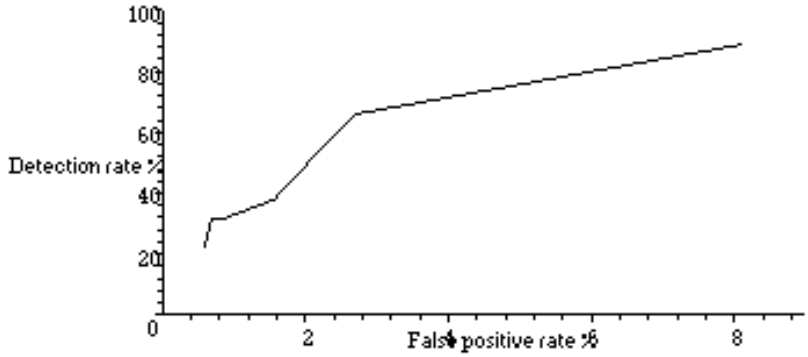

Figure 1. The ROC curve of false positive vs. detection rate, for a fixed training and test set combination.

3.4 Variations to clustering and detection

After the cluster width and the constant indicating the percent of largest clusters to be labeled normal were fixed, some variations on the clustering and detection methods were tried, and the change in performance over the single training and test subset was measured for each variation.

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

IPR2023-00556
CrowdStrike EX1011 Page 14

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

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 occuring, 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 3.4. 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 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 implementing these changes and evaluating their performance on a test set, we concluded that they did not improve detection accuracy by a considerable amount and in some cases were even detrimental. Therefore, these changes were taken out, and we used the original system for cross validation testing.

IPR2023-00556
CrowdStrike EX1011 Page 15

3.5 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 10% percent subsets, each containing approximately 490,000 instances. 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. Training on such subsets would fail to produce clusters for regions of feature space representing intrusions that were not present in the training set. Therefore, for cross validation training only four of the ten subsets were selected. These four subsets contained a good mix of various intrusion types, and were therefore 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 3.4.

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.

IPR2023-00556
CrowdStrike EX1011 Page 16

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 4: 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.

IPR2023-00556
CrowdStrike EX1011 Page 17

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, as more intrusions are introduced into the network and are trained on, the cluster set and detection rate will tend to become better. The results from training on the P10 dataset can be used as an approximation of the system's performance when presented with training sets that adequately represent currently existing intrusions.

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. If some method would classify instances as attacks or normal by simply choosing one of these classifications at random with a 50% probability for each one (i.e. it would guess), then the detection and false positive rates would be roughly 50% as well. Heuristics subsequently added to the method to increase the detection rate, would tend to make it classify more instances as attacks, thereby increasing the chance

IPR2023-00556
CrowdStrike EX1011 Page 18

of incorrectly misclassifying normal instances and therefore increasing false positive rate. In the other direction, if we make the method more biased towards classifying instances as normal, the potential is increased for detection rate to suffer. To keep the false positive rate low while increasing detection rate, requires making the detection method more accurate and making it more informed about the domain.

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 unoptimal metric or because the assumption that instances of the same type or sub-type (and only them) will cluster together was not satisfied. However, the negative effect of this misassignment 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 misassignment 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 misassignment 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).

IPR2023-00556
CrowdStrike EX1011 Page 19

4.2 Variations

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 N 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 $N > 1$ than when N was equal to 1, while still keeping the false positive rate relatively low. However, the results varied greatly with N, with no apparent pattern as N increased. This lead us to suspect that the value of N 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 (N) 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 overfitting 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.

IPR2023-00556
CrowdStrike EX1011 Page 20

4.3 Satisfaction of assumptions by real world data

Ideally, we would like both of our assumptions about the domain to be satisfied. In particular, we would like data instances of the same type or sub-type to cluster together under our metric, and those of different types to be assigned into distinct clusters. In reality, of course, this assumption is not fully satisfied and this is one of the primary reasons for the failure of our method to detect 100% of all attacks. Improving the metric can help increase the accuracy of assignment of instances to proper clusters. However, sometimes several such improvements might not be possible because they would reduce the performance of the overall method. An example of this manifested itself in the course of our research.

In trying to analyze why assignments of instances to clusters of non proper (i.e. non corresponding to the instances') types occured, we discovered that there were several intrusion types which were nearly identical in all feature values except for some minor variations in a few of features. One example of this are the following fragments of two instances :

$$Inst1 \quad : \quad -0.086031 udpprivateSF - 0.019133 - 0.0860130 - 0.036963 - 0.010252$$
$$Inst2 \quad : \quad -0.086031 udpprivateSF - 0.019369 - 0.0906550 - 0.036965 - 0.010249$$

The feature values shown are ones after the statistical conversion, so that -0.086031, for example, means that the feature value of that instance was approximately .086 standard deviations below the mean for that feature in the entire training dataset. As can be seen, all of these feature values are identical except for a few with very minor differences. For example, the difference in the 8th feature is only two hundred thousandth of a standard deviation. However, Inst1 was classified as normal in the (labeled) training set, while Inst2 was classified as an attack (snmpguess). It is of no surprise then, that under our metric which was tuned to tolerate differences much larger than this, these two instances clustered into the same cluster, even though they were of different types. To solve this problem, we might try to improve the metric by making it much more sensitive so that small differences like the ones above would be reflected in the distance returned by the metric. However, if we would do this the number of clusters created would be greatly increased, as a cluster would be created for each small sub-region in the feature space. This in turn would lead to an extremely even distribution of instances among these clusters. Only a handful of them (representing only

IPR2023-00556
CrowdStrike EX1011 Page 21

the ones would in each small sub-region) would be assigned to each cluster, and during detection it would be nearly impossible to differentiate between anomalous and normal clusters and thus label them correspondingly. It seems that a rule based classification method (e.g. decision tree, if-then rule sets, etc.) trained with labeled instances, would be more appropriate to handle these slight variations in the almost identical instances, and would be able to separate them into different classes. With unlabeled data and clustering, however, we are faced with the tradeoff of cluster granularity vs. the ability to distinguish normal from anomalous clusters.

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 new type is represented by its sufficient presence in the training set, whether the administrators of the network are aware of that presence or not. The system then will try to automatically determine which data instances fall into the normal class and which ones are intrusions.

The methods discussed in this paper can be implemented as part of a larger system for detecting intrusions, which will automatically collect data from the network and ask several intrusion detection methods to evaluate that data. A vote could then be taken among these methods regarding each data instance, and the majority's opinion used. If our system is used within such a context, higher weight should be given to its opinion when it classifies an instance as an intrusion. This is necessary because new types of attacks may be detected by our system, of which other (possibly based on labeled data) methods in the voting pool will know nothing about and which they will not be able to detect. In this scenario, only our system will have the opinion that some instance is an intrusion, while all the other ones will consider it

IPR2023-00556
CrowdStrike EX1011 Page 22

normal, and so such instances will never receive a majority of the votes for labeling it as an intrusion. Since the false positive rate in our system is low, giving higher weight to positive decisions (i.e. when something is classified as an intrusion) will not hurt the overall false positive rate of the voting pool, but it will help the detection rate. When our system classifies something as normal, however, then its weight should be equal or less than the weights of other systems in the pool, because of the possibly lower detection rate relative to that of other systems. Of course, our method can also be implemented as a standalone system.

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.

5.1 Future work

Future work involves possible extentions 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 that (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 (PLC) 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 PLC which does not reflect the decreased ratio will therefore cause the algorithm

IPR2023-00556
CrowdStrike EX1011 Page 23

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 PLC will be required. If the system determines the value for PLC automatically, however, then no manual intervention will be required even over long periods of time.

5.2 Acknowledgements

We would like to acknowledge Salvatore J. Stolfo and Eleazar Eskin {sal,eeskin@cs.columbia.edu} at Columbia University's Computer Science department, for their help and numerous contributions that made this work possible.

References

[1] D. Touretzky B. Fritzke and T. Leen. Advances in neural information processing systems, 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] D.E. Denning. An intrusion detection model. *IEEE Transactions on Software Engineering*, SE-13:222–232, 1987.

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

[6] M. Ester, H. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise, 1996.

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

[8] A. Ghosh and A. Schwartzbard. A study in using neural networks for anomaly and misuse detection, 1999.

IPR2023-00556
CrowdStrike EX1011 Page 24

[9] 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.

[10] 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.

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

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

[13] KDD99. Kdd99 cup dataset, 1999.

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

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

[16] R. Ng and J. Han. Efficient and effective methods for spatial data mining, 1994.

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

[18] P. Schnell. A method for discovering data-groups, 1964.

[19] C. Warrender, S. Forrest, and B. Pearlmutter. Detecting intrusions using system calls: Alternative data models, 1999.

[20] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: An efficient data clustering method for very large databases, 1996.

IPR2023-00556
CrowdStrike EX1011 Page 25