Network Anomaly Identification using Supervised Classifier

Prasanta Gogoi, B. Borah and D. K. Bhattacharyya
Department of Computer Science and Engineering
Tezpur University, Napam, Tezpur-784028, India
{prasant,bgb, dkb}@tezu.ernet.in and www.tezu.ernet.in

Keywords: supervise, incremental, clustering, anomaly, DoS, classification, PCC

Received: July 31, 2012

In this paper we present a clustering based classification method and apply it in network anomaly detection. A set of labeled training data consisting of normal and attack instances are divided into clusters which are represented by their representative profiles consisting of attribute-value pairs for selected subset of attributes. Each category of attack and normal instances are broken down into a set of clusters using a training algorithm based on supervised classification algorithm. The cluster profiles together with their class label form rules for labeling unseen test instances. Methods for clustering, training and prediction are provided. The proposed method is evaluated using real life TUIDS Intrusion datasets. Evaluation results on KDD 1999 datasets showed good performance in comparison to results produced by decision tree counterparts. The method presented can be utilized for classification jobs in any other domain.

Povzetek: Predstavljena je metoda za identifikacijo anomalij v omrežju.

1 Introduction

A network intrusion can be any exploit of a network that compromises its stability or the security of information stored on computers connected to it. An intrusion detection system gathers relevant data from computers or the network and analyzes them for signs of intrusion. Different methods such as statistical, pattern matching, machine learning, and data mining are used for intrusion detection. Some approaches are online, that is, they detect attacks in progress in real time, while offline approaches such as data mining provide after-the-fact clues about the attacks to help reduce the possibilities of future attacks of the same type. In general there are two types of approaches [1] for network intrusion detection: misuse detection and anomaly detection. Misuse detection searches for specific signatures to match, signaling previously known attacks without generating a large number of false alarms. Such methods fail to detect new types of attacks as their signatures are not known. Anomaly detection builds models for normal behavior and significant deviations from it are flagged as attack. Supervised or unsupervised learning can be used for anomaly detection. In a supervised approach, the model is developed based on labeled training data. Unsupervised approaches work without any training data or they may use training with unlabeled data. The main advantage of anomaly detection is that it can detect previously unknown attacks if their behavior is significantly different from what is considered to be normal. False alarm rates tend to be higher for anomaly detection methods.

Data mining methods including classification, association analysis, clustering and outlier detection are being used in network intrusion detection. Anomaly detection often tries to cluster test datasets into groups of similar instances which may be either attacks or normal data. Intrusion detection problem is then reduced to the problem of labeling the clusters as intrusive or normal traffic. When doing unsupervised anomaly detection a model based on clusters of data is trained using unlabeled data, normal as well as attacks. Supervised anomaly detection methods such as classification algorithms need to be presented with both normal and known attack data for training.

In this paper we present a supervised classification method and use it for network anomaly detection. Labeled training dataset is broken down into clusters belonging to normal and attack categories and the clusters are represented by their representative profiles, which together with category labels form the classification rules. During detection, the cluster profiles collect similar testing instances which are then labeled as belonging to the same category as the label of the profiles.

Rest of the paper is organized as follows. In Section 2, some related work regarding supervised anomaly detection methods are presented. The proposed classification algorithm including a training method and a prediction method is presented in Section 3. Application of the method to network anomaly detection is presented in Section 4. Experimental results for datasets TUIDS intrusion, KDD 1999, and NSL-KDD datasets are reported in Section 5. Finally, Section 6 concludes providing a few possible future extensions.

2 Related work

Classification is an important supervised learning method that has been applied to anomaly detection. Lee and Stolfo [1] proposed a systematic framework employing data mining methods for intrusion detection. This framework consists of classification, association rules and frequent episodes algorithms that can be used to construct detection models. The authors in [2] presented PNrule, for multi-class classification problem. The key idea used in PNrule is learning a rule-based model in two stages: first find P-rules to predict presence of a class and then find N-rules to predict absence of a class. The scoring mechanism used in PNrule allows one to tune selectively the effect of each N-rule on a given P-rule. ADWICE [3] uses extended BIRCH [4] clustering algorithm to implement a fast, scalable and adaptive anomaly detection scheme. They apply clustering as a method for training of the normality model. Several soft computing paradigms, viz., fuzzy rule-based classifiers, support vector machines, linear genetic programming and an ensemble method to model fast and efficient intrusion detection systems were investigated in [5]. Empirical results clearly show that soft computing approach could play a major role for intrusion detection. Yang et al. [6] present an anomaly detection approach based on clustering and classification for intrusion detection. They perform clustering to group training data points into clusters, from which they select some clusters as normal and create known-attack profiles according to certain criteria. The training data excluded from the profile are used to build a specific classifier. During the testing stage, they use an influence-based classification algorithm to classify network behaviors. A comparative study of several supervised probabilistic and predictive machine learning methods for intrusion detection is reported in [7]. Two probabilistic methods, Naive Bayes and Gaussian and two predictive methods, Decision Tree [8] and Random Forests [9], are used. The ability of each method to detect four attack categories (DoS, Probe, R2L and U2R) has been compared. A new supervised intrusion detection method based on TCM-KNN (Transductive Confidence Machine for K-Nearest Neighbors) algorithm has been presented in [10]. This algorithm works well even if sufficient attack data are not available for training. A review of the most well known anomaly-based intrusion detection methods is provided by [11, 12, 13]. Available platforms, systems under development and research projects in the area are also presented. This paper also outlines the main challenges to be dealt with for the wide scale deployment of anomaly-based intrusion detectors, with special emphasis on assessment issues. Beghdad [14] presents a study of the use of some supervised learning methods to predict intrusions. The performances of six machine learning algorithms involving C4.5, ID3, Classification and Regression Tree (CART), Multinomial Logistic Regression (MLR), Bayesian Networks (BN), and CN2 rule-based algorithm are investigated. KDD 1999 datasets were used to evaluate the considered algorithms. An SVM-based intrusion detection system is found in [15], one which combines a hierarchical clustering algorithm (BIRCH [4]), a simple feature selection procedure, and the SVM (support vector machines) method. This method is also evaluated using on the KDD 1999 datasets. For these evaluations, three cases were considered: the whole attacks case, the five behaviors classes case, and the two behaviors classes case. The performances of each method were compared.

3 Proposed supervised method

The proposed supervised method uses a training algorithm to create a set of representative clusters from the available labeled training objects. Unlabeled test objects are then inserted in these representative clusters based on similarity calculations and thus get labels of the clusters in which they are inserted. We first present the basics of the clustering algorithm, and then present the training and prediction algorithms.

3.1 Background

The dataset to be clustered contains n objects, each described by d attributes $A_1,\ A_2,\ \cdots,\ A_d$ having finite discrete valued domains $D_1,\ D_2,\ \cdots,\ D_d$, respectively. A data object can be represented as $X = \{x_1,\ x_2,\ \cdots,\ x_d\}$. The j-th component of object X is x_j and it takes one of the possible values defined in domain D_j of attribute A_j. Referring to each object by its serial number, the dataset can be represented by the set $N = \{1, 2, \cdots, n\}$. Similarly, the attributes are represented by the set $M = \{1,\ 2,\ \cdots,\ d\}$.

3.1.1 Similarity function between two objects

The similarity between two data objects X and Y is the sum of per attribute similarity for all the attributes. It is computed as $sim(X, Y)$,

$$sim(X,\ Y) = \sum_{j=1}^{d} s(x_j,\ y_j), \tag{1}$$

where $s(x_j, y_j)$ is the similarity of the j-th attribute defined as

$$s(x_j,\ y_j) = \begin{cases} 1 & \text{if } |x_j - y_j| \leq \delta_j \\ 0 & \text{otherwise}, \end{cases} \tag{2}$$

where δ_j is the similarity threshold for the j-th attribute. For categorical attributes δ_j=0 and for numeric attributes $\delta_j \geq 0$.

3.1.2 Similarity between cluster and object

A subspace based incremental clustering method is used here. A cluster is a set of objects which are similar considering a subset of attributes only. The minimum size of the subset of attributes required to form a cluster is defined by the threshold $MinAtt$. Let the subset

of defining attributes be represented by $Dattributes = \{a_1, a_2, \cdots, a_{noAttributes}\}$ such that $Dattributes \subseteq M$ and $noAttributes$ is the size of $Dattributes$. A cluster is represented by its profile, somewhat like an object. All objects in a cluster are similar to the profile. The cluster profile is defined by a set of values, $Values = \{v_1, v_2, \cdots, v_{noAttributes}\}$ considering the attributes in $Dattributes$. That is $v_1 \in D_{a_1}$ is the value for attribute $a_1 \in M$, $v_2 \in D_{a_2}$ is the value for attribute $a_2 \in M$, and so on. Thus, the cluster profile is defined by

$$Profile = \{noAttributes, Dattributes, Values\}. \quad (3)$$

Let $Olist \subseteq N$ be the list of data objects in the cluster. A cluster C is completely defined by its $Profile$ and $Olist$

$$C = \{Olist, Profile\}. \quad (4)$$

Our incremental clustering algorithm inserts an object in any one of the clusters existing at a particular moment. So, the similarity between a cluster and a data object needs to be computed. Obviously, the cluster profile is used for computing this similarity. As the similarity is computed over the set of attributes in $Dattributes$ only, the similarity function between a cluster C and an object X becomes $sim(C, X)$

$$sim(C, X) = \sum_{j=1}^{noAttributes} s(v_j, x_{a_j}), \quad (5)$$

where $s(v_j, x_{a_j})$ is the similarity of the j-th attribute defined as

$$s(v_j, x_{a_j}) = \begin{cases} 1 & \text{if } |v_j - x_{a_j}| \leq \delta_{a_j} \\ 0 & \text{otherwise .} \end{cases} \quad (6)$$

Example: Consider a small dataset shown in Table 1 with seven objects defined using five attributes A_1, A_2, A_3, A_4 and A_5. The domains for the attributes are respectively, $D_1 = \{a1, a2, a3\}$, $D_2 = \{b1, b2\}$, $D_3 = \{c1, c2, c4\}$, $D_4 = \{d1, d2, d3\}$ and $D_5 = \{e1, e2\}$.

Clusters C_1 and C_2 can be identified in the dataset with $MinAtt = 3$ and $\delta_{a_j} = 0$ for $a_j \in Dattributes$:
$C_1 = \{Olist = \{1, 2, 4\}, noAttributes = 3,$
$Dattributes = \{2, 3, 5\}, Values = \{b2, c4, e2\}\}$, and
$C_2 = \{Olist = \{3, 5, 7\}, noAttributes = 4,$
$Dattributes = \{1, 2, 3, 5\}, Values = \{a3, b1, c2, e1\}\}$.

3.1.3 Our supervised clustering algorithm

The clustering algorithm starts with an initially empty set of clusters. It reads each object X_i sequentially, inserts it in an existing cluster based upon the similarity between X_i and the clusters or a new cluster is created with X_i if it is not similar enough, as defined by the threshold $MinAtt$ for insertion in an existing cluster. The search for a cluster for inserting the present object is started with the last cluster created and moves towards the first cluster until the search is successful. If successful, the object is inserted in the cluster found and the search is terminated. At the time of inserting the object in the found cluster C, the number of defining attributes of the cluster ($C.noAttributes$) is set according to the computed similarity measure between the cluster and the object and the sets of $C.Dattributes$ along with $C.Values$ are updated. If the search is not successful a new cluster is created and the object itself made the representative object of the cluster, i.e., the full set of attributes becomes the $Dattributes$ while full set of values of the object becomes corresponding $Values$ of the new cluster profile.

The classification method begins with a fixed number of existing clusters (as defined by their profiles) with no objects present in them. Objects are incrementally inserted in any of the clusters and no new clusters are created. The description of notations used in the algorithm is given in *Table 2*

Table 1: A sample dataset

Serial no.	A_1	A_2	A_3	A_4	A_5
1	a3	b2	c4	d1	e2
2	a2	b2	c4	d3	e2
3	a3	b1	c2	d1	e1
4	a2	b2	c4	d1	e2
5	a3	b1	c2	d3	e1
6	a1	b2	c1	d2	e2
7	a3	b1	c2	d2	e1

Table 2: Algorithm Notations

Symbol	*Description*
$MinAtt$	threshold value for minimum attribute.
$minSize$	minimum objects in a cluster threshold value.
$sim(X, Y)$	similarity value between two objects.
$sim(C, X)$	similarity value between a cluster and an object.
$Olist$	object list in a cluster.
$Profile$	profile of a cluster.
$noAttribtes$	total number of attributes in a cluster profile.
$Dattributes$	distribution of attributes in a cluster profile.
$Values$	values of attributes in a cluster profile.

3.2 Training method

Given a set of labeled training data, a combination of unsupervised incremental clustering and supervised classification methods are applied to create and refine a set of clusters from which profiles are extracted for use in the test object labeling process. The unsupervised clustering algorithm decides cluster membership immediately as the objects arrive sequentially without considering subsequent objects that are yet to be seen. Therefore, refinement of the created clusters is performed using a subsequent supervised classification step that allow possible realignment of the data objects in the clusters.

In the beginning of the training algorithm, all objects are marked unprocessed. Similarity thresholds $minAtt$ (the

minimum number of attributes) and $minSize$ (the minimum number of objects in a cluster) are set high and they are gradually decreased in steps. In each iteration, the remaining unprocessed objects are clustered using a combination of supervised classification and unsupervised clustering. If the supervised classification process fails to insert an object in any of the preexisting (created in the previous iteration) clusters (SCs), then the unsupervised clustering process inserts it in any of the unsupervised clusters (UCs) created in the present iteration or a new unsupervised cluster is created with the object. When the clustering process ends in the present iteration, cluster profiles are extracted from each of the SCs with at least $minSize$ objects in it and the objects in such a cluster are marked processed. All SCs are then deleted. The UCs may also include some insignificant clusters whose sizes are less than $minSize$. Such clusters are deleted. Remaining UCs are made SCs for next iteration after making them empty by deleting their object lists. Then the threshold values $minSize$ and $minAtt$ are reduced so that the next iteration can create larger clusters instead of fragmented clusters. Reducing the thresholds allows more generalization. The algorithm iterates so long as there are unprocessed objects. To ensure termination of the algorithm $minSize$ is reduced to $minSize$/2 so that the ultimate value of $minSize$ becomes 1, beyond which no objects remain unprocessed. The threshold $minAtt$ is loosened by setting $minAtt$=$minAtt - \alpha$, where α is a small integral constant such as 1 or 2. Reduction of $minAtt$ below a certain level (MIN) is not allowed, after which remains constant at MIN. Generalization beyond MIN makes data objects from two different classes indistinguishable. When training terminates, the set of profiles found in the profile file becomes the final cluster profiles for use in the prediction process. The training algorithm is given as Algorithm 2.

3.2.1 Effective profile searching

Two-dimensional link list (2-DLL) structure (as found in [16]) is used to store the profiles in memory. It enables to update the profile base dynamically. The 2-DLL structure updates row wise with insertion of new parameter in a profile and column wise with the insertion of a new protocol type. It enables to search the profile base protocol wise, viz., TCP, ICMP, UDP, "other" (which profiles are not included the protocols viz., TCP, ICMP or UDP. Once the respective protocol is identified in this 2-DLL structure, further traversal is supported for other parameters matching by following the link list structure.

3.3 Prediction method

Once the set of cluster profiles is ready, labeling test objects becomes simple. Supervised clustering is performed with the objects. The label of the cluster profile becomes the label of each object inserted in the cluster. The objects

Algorithm 1 Training algorithm

Input: Training Dataset;
Output: $file1$=TrainProfile;
read (n, d, $minAtt$, $minSize$) ;
for ($i = 0; i < n; i = i + 1$) **do**
  read ($X[i]$, $recordLabel[i]$) ;
  processed[i]=0 ;
end for
$totalProfile = 0$;
$top = 0$;
while ($minSize > 0$) **do**
  for ($i = 0; i < n; i = i + 1$) **do**
    $found = FALSE$; //supervised clustering
    for ($j = 0; j < totTrain; j = j + 1$) **do**
      if ($recordLabel[i] == clusterLabel[j]$) **then**
        if ($sim(SC[j],\ X[i]) == SC[j].noAttributes$) **then**
          $append(i,\ SC[j].oList)$;
          $found = TRUE$;
          break ;
        end if
      end if
    end for //end supervised clustering
    if ($found == TRUE$) **then**
      continue;
    end if
    for ($j = top; j >= 0; j = j - 1$) **do**
      if ($recordLabel[i] == clusterLabel[j]$) **then**
        if ($sim(UC[j],\ X[i]) >= minAtt$) **then**
          $found = TRUE$;
          $update(UC[j],\ i,\ X[i])$;
          $break$;
        end if
      end if
    end for
    if ($found == FALSE$) **then**
      $top = top + 1$; //create new cluster
      $UC[top]$ =new $cluster$;
      $append(i,\ UC[top].oList)$;
      $UC[top].noAttributes = d$;
      for ($j = 0; j < d; + + j$) **do**
        $UC[top].a[j] = j$;
        $UC[top].v[j] = X[i][j]$;
      end for
      $clusterLabel[top] = recordLabel[i]$;
    end if
  end for//end new cluster
  for ($i = 0; i < totTrain; i = i + 1$) **do**
    m=sizeof($SC[i].oList$)
    if ($m >= minSize$) **then**
      $k = SC[i].noAttributes$;
      for ($j = 0; j < k; j = j + 1$) **do**
        write($file1$, $SC[i].a[j]$) ;
      end for
      for ($j = 0; j < k; j = j + 1$) **do**
        write($file1$, $SC[i].v[j]$) ;
      end for
      $p = SC[i].oList$;
      for ($j = 0; j < m; j = j + 1$) **do**
        $k = p.getdata(j)$;
        processed[k]=1 ;
        p=p.next;
      end for
      delete SC[i] ;
    end if
    $totTrain = 0$;
    for ($i = 0; i < top; i = i + 1$) **do**
      m=sizeof($UC[i].oList$)
      if ($m >= minSize$) **then**
        $SC[totTrain] = UC.[i]$;
        delete $UC[i].oList$;
        $UC[i].oList = NULL$;

2 Training algorithm (continued)

```
69:                 totTrain = totTrain + 1 ;
70:             else
71:                 delete UC[i] ;
72:             end if
73:         end for
74:         minSize = minSize/2 ;
75:         MinAtt = minAtt − α ;
76:         if (MinAtt < MIN) then
77:             minAtt = MIN ;
78:         end if
79:     end for
80: end while
81: int function sim(cluster C, object X)
82: int scount = 0 ;
83: k = C.noAttributes ;
84: for (j = 0; j < k; j = j + 1) do
85:     l = C.a[j] ;
86:     if (abs(C.v[j] − x[l]) <= δ[l]) then
87:         scount = scount + 1 ;
88:     end if
89: end for
90: return scount ;
91: end function
92: function update(cluster C, int r, object X)
93: append(r, C.Olist) ;
94: int count = 1;
95: m = C.noAttributes;
96: for (j = 0; j < m; + + j) do
97:     l = C.a[j] ;
98:     if (abs(s(C.v[j] − x[l])) <= δ[l]) then
99:         count = count + 1 ;
100:        C.v[count] = C.v[j] ;
101:        C.a[count] = C.a[j] ;
102:    end if
103:    C.noAttributes = count ;
104: end for
105: end function
```

Algorithm 3 Prediction algorithm

```
Input: Dataset file1=TrainProfile, file2=DataFile;
Output: Object cluster label of file2 ;
 1: read(file1, totTrain);
 2: for i = 0; i < totTrain; i = i + 1 do
 3:     read(file1, k);
 4:     C[i].noAttributes = k;
 5:     for (q = 0; q < k; q = q + 1) do
 6:         read(file1, C[i].a[q]);
 7:         for (j = 0; j < k; j = j + 1) do
 8:             read(file1, C[i].v[j]);
 9:             if count == C[j].noAttributes then
10:                 print("Object" i " is of category"
11:                 clusterLabel[j]);
12:                 found = 1; break;
13:             else
14:                 if count > max then
15:                     max = count; position = j;
16:                 end if
17:             end if
18:         end for
19:         if found == 0 then
20:             print("Object" i "is of category";
21:             clusterLabel[position]);
22:         end if
23:     end for
24: end for
```

not inserted in any of the clusters defined by the profiles need special attention. They may be flagged suspicious for anomalies. Another method for dealing with such objects is to insert such an object in a cluster which is most similar to the object and label accordingly. The algorithm for prediction method is given as Algorithm 4.

3.3.1 Complexity analysis

The clustering algorithms require one pass through the set of training examples that currently remain unprocessed. Each training example needs to be compared with existing clusters one after another until it gets inserted in one of the clusters. The similarity computation involves a subset of attributes. Therefore, the clustering process has a complexity $O(ncd)$, where n is the number of training examples, c is the number of clusters and d is the number of attributes. Each of the created clusters needs to be visited to extract its size and profile. Hence, maximum time complexity of one iteration of the training algorithm becomes $O(ncd) + O(c)$. The algorithm performs at most k iterations, where $k = log_2(minSize)$. As $minSize$ is the minimum number of objects for a cluster to be considered significant, it is not large. Overall maximum time complexity of the algorithm is $O(kncd) + O(kc)$.

4 Classification of network anomalies

In this section we apply our classification method to network anomaly detection.

4.1 Dataset Description

The algorithm was evaluated with three real life TUIDS [17] intrusion datasets and two benchmark intrusion datasets, viz., KDD Cup 1999 [18] and NSL-KDD [19] datasets.

4.1.1 TUIDS dataset

The real life TUIDS Intrusion datasets consist of three datasets *Packet level*, *Flow level* and *Portscan* where each dataset consists of attributes (numerical/categorical) of 50(11/39), 24(9/15) and 23(9/14), respectively as given in Table 3.

4.1.2 Benchmark intrusion datasets

The description of two benchmark intrusion datasets, KDD Cup 1999 [18] and NSL-KDD [19] are given in Table 4 and Table 5, respectively. Each record of the datasets represents a connection between two network hosts according to some well defined network protocol and is described by 41 attributes (38 continuous or discrete numerical attributes and 3 categorical attributes). Each record is labeled as either normal or one specific kind of attack. The attacks fall

Table 3: TUIDS Intrusion Datasets

Data sets	*Connection Type*	*Data set type*	
		Training	*Testing*
Packet Level	**Normal**	35043	27895
	Attack	397832	138370
	Total	432875	166265
Flow Level	**Normal**	36402	16770
	Attack	363729	123955
	Total	400131	140725
Portscan	**Normal**	2445	1300
	Attack	39215	28615
	Total	41660	29915

Table 4: KDD Cup 1999 datasets

Data sets	*DoS*	*U2R*	*R2L*	*Probe*	*Normal*	*Total*
Corrected KDD	229853	70	16347	4166	60593	311029
10-percent KDD	391458	52	1126	4107	97278	494021

Table 5: NSL-KDD datasets

Data sets	*DoS*	*U2R*	*R2L*	*Probe*	*Normal*	*Total*
$KDDTrain^+$	45927	52	995	11656	67343	125973
$KDDTest^+$	7458	67	2887	2422	9710	22544

in one of the four categories: User to Root ($U2R$), Remote to local ($R2L$), Denial of Service (DoS) and $Probe$.

- Denial of Service(DoS): Attacker tries to prevent legitimate users from using a service. For example, SYN flood, smurf, teardrop etc.
- Remote to Local ($R2L$): Attackers try to gain access to victim machine without having an account on it. For example, guessing password.
- User to Root ($U2R$): Attackers have local access to the victim machine and tries to gain super user privilege. For example, buffer overflow attacks.
- $Probe$: Attacker tries to gain information about the target host. For example, Port-scan, ping-sweep etc.

Number of samples of each category of attack in $Corrected\ KDD$ dataset and 10 $percent\ KDD$ dataset of KDD 1999 are shown in Table 4. The attack distribution in $KDDTrain^+$ and $KDDTest^+$ of NSL-KDD are shown in Table 5.

4.2 Data preprocessing

We have discretized continuous valued attributes by taking logarithm to the base 2 and then converting to integer. This is done for each attribute value z using the computation: $if\ (z > 2)\ z = int(log_2(z)) + 1$. Before taking logarithm, the attributes which take fractional values in the range [0, 1] are multiplied by 100 so that they take values in the range [0, 100]. Nominal valued attributes are mapped to discrete numeric codes which are nothing but serial numbers beginning with zero for the unique attribute values in the order in which they appear in the dataset. The class label attribute is removed from the dataset and stored separately in a different file. The class labels are used for training and evaluating the detection performance of the algorithm.

4.3 Feature selection

Information gain [20] is computed for each of the discretized attributes. Six attributes (attribute numbers 7, 9, 15, 18, 20 and 21 in KDD Cup 1999 and NSL-KDD datasets) corresponding to very low information gain are removed from the dataset. It reduces computation time.

4.4 Parameter selection

The training algorithm has 3 parameters to be given as inputs: $minAtt$, $minSize$ and MIN. The parameters $minAtt$ takes values in the range [1,d], where d is the number of attributes. Higher values of $minAtt$ corresponds to more specialization leading to a larger number of rules and a lower value of $minAtt$ corresponds to more generalization leading to fewer number of rules. But over generalization can make normal and attack data indistinguishable resulting in more false positives or false negatives. So, a lower limit for $minAtt$ is specified using the parameter MIN. Now, the possible range of values for $minAtt$ becomes [MIN,d]. To determine the value of MIN, information gain [20] is calculated for each attribute in the training dataset. Attributes with very low information gain can be ignored and the number of remaining attributes is set as the value for MIN. The parameter $minSize$ is the minimum number of data elements received to form a cluster. In some cases, at least one training example for an attack should be considered significant. So, the minimum value for $minSize$ is 1. Its value should increase with increase in $minAtt$ value. The maximum value for $minSize$ can be set to be any multiple of $log_2(n)$, where n is the number of training examples.

In our experiments, $MIN = 27$. The value for $MinAtt$ is set to 33, which is gradually reduced to $MIN = 27$ in steps of -1. The starting value for $MinSize$ is set to $2log_2(n)$. The similarity thresholds for all attributes are set to zero, $\delta_i = 0, i = 1, \cdots, d$.

4.5 Performance measures

Evaluation of performance of a classification model is based on the counts of test records correctly and incorrectly predicted by the model. These counts are tabulated in a confusion matrix [21]. A confusion matrix that summarizes the number of instances predicted correctly or incorrectly by a classification model is shown in Table 6.

Table 6: Confusion matrix for binary classification

		Predicted Class	
		+	-
Actual Class	+	$f_{++}\ (TP)$	$f_{+-}\ (FN)$
	-	$f_{-+}\ (FP)$	$f_{--}\ (TN)$

Recall and Precision are two widely used metrics employed in applications where successful detection of one of the classes is considered more significant than detection of the other classes. A formal definition of the metrics is given below.

$$Precision,\ p = \frac{TP}{TP + FP} \tag{7}$$

$$Recall,\ r = \frac{TP}{TP + FN} \tag{8}$$

Building a model that maximizes both precision and recall is the key challenge for classification algorithms. Another measure of accuracy is PCC (percentage of correct classification). A formal definition of PCC is given below.

$$PCC = \frac{TP + TN}{TP + FN + FP + TN} \tag{9}$$

A high value of PCC ensures reasonably correct classification.

5 Experimental results

The experiments were performed on a 3 GHz HP dc 7000 series desktop with 2 GB RAM, 250 GB HDD. C++ programs were used in a LINUX environment. We provide the cross evaluation result performing training and testing of the method with corresponding training and testing datasets.

5.1 Results on TUIDS Intrusion Datasets

Two categories of experiments 2-class ($normal$ and $attack$) and all-attacks behaviors are carried out.

5.1.1 2-class prediction results

The confusion matrices for the 2-class behavioral categories on the *Packet Level*, *Flow Level* and *Portscan* datasets are shown in Table 7. Classification rates with *PCC* 99.72%, 99.70 and 98.31% for *Packet Level*, *Flow Level* and *Portscan* datasets, respectively indicate good performance for our method.

5.1.2 All-attacks prediction results

The confusion matrices for all-attacks categories on the *Packet level*, *Flow level* and*Portscan* datasets are shown in Table 8. Classification rates of *PCC* are 99.42%, 99.01% and 98.31% for *Packet level*, *Flow level* and*Portscan* datasets, respectively. The average execution time of classification for *Packet level*, *Flow level* and *Portscan* datasets are 0.39 minute, 0.17 minute and 0.14 minute, respectively.

Table 7: 2-class confusion matrix for TUIDS datasets

Data set	Actual Class	Predicted Class				
		$Normal$	$Attack$	Sum	Recall	1-Prc*
Packet level	$Normal$	34790	253	35043	0.9928	0.0267
	$Attack$	955	396877	397832	0.9976	0.0006
	Sum	35745	397130	432875		
	Re-substitution error =0.0028				PCC=99.72%	
		Predicted Class				
		$Normal$	$Attack$	Sum	Recall	1-Prc*
Flow level	$Normal$	36107	295	36402	0.9919	0.0246
	$Attack$	910	362819	363729	0.9975	0.0008
	Sum	37017	363114	400131		
	Re-substitution error =0.0030				PCC=99.70%	
		Predicted Class				
		$Normal$	$Attack$	Sum	Recall	1-Prc*
Portscan	$Normal$	2414	31	2445	0.9876	0.2175
	$Attack$	671	38544	39215	0.9829	0.0009
	Sum	3085	38575	41660		
	Re-substitution error =0.0169				PCC=98.31%	

Note- *1-Precision

5.2 Results on KDD Cup 1999 dataset

We perform three different experiments which are categorized into 2-class ($normal$ and $attack$), 5-class ($normal$, $R2L$, DoS, $Probe$ and $U2R$) and all-attacks behaviors.

5.2.1 2-class prediction results

The confusion matrix for the 2-class behavioral categories on the $Corrected\ KDD$ is shown in Table 9. Classification model is trained with 10-$percentKDD$ dataset. The classification rate is $PCC = 93.40\%$. The performance degrades due to the fact that no examples are present in the training set corresponding to the 15 categories of attacks that are present in the testing set. Most of these attacks are misclassified as normal by our algorithm.

5.2.2 5-class prediction results

The confusion matrix for the 5-class behavioral categories on the $Corrected\ KDD$ datasets is shown in Table 10. The classification model is trained with the 10-$percentKDD$ dataset. In this case most of the unseen attacks are misclassified as normal category and PCC reduced to 92.39%.

5.2.3 All-attacks prediction results

The confusion matrix for the all-attacks categories on the $Corrected\ KDD$ is shown in Table 11. The classification model is trained with $10\%KDD$ dataset that includes

Table 10: 5-class confusion matrix results of $Corrected\ KDD$ dataset training with $10\%KDD$ dataset

		Predicted Class							
		$Normal$	$R2L$	DoS	$Probe$	$U2R$	Sum	Recall	1-Prc*
Actual class	$Normal$	60211	21	73	284	4	60593	0.9937	0.7386
	$R2L$	13868	1115	3	1356	5	16347	0.0682	0.9378
	DoS	6360	42	223349	102	0	229853	0.9717	0.9977
	$Probe$	1036	0	443	2687	0	4166	0.6450	0.6067
	$U2R$	49	11	0	0	10	70	0.1429	0.5263
	Sum	81524	1189	223868	4429	19	311029		
Re-substitution error=0.0761								PCC=92.39%	

Note- *1-Precision

22 attack classes. The testing dataset $Corrected\ KDD$ include 37 attack classes. Since no examples are present in the training set, objects belonging to the 15 categories of attacks cannot be classified correctly and hence we exclude them (18729 objects) from the testing dataset. We see from the results that DoS and $Probe$ attacks are well detected but $R2L$ and $U2R$ attacks are detected very poorly. It is apparent from that there are very few examples in the training set corresponding to attacks belonging to $R2L$ and $U2R$ categories compared to DoS, $Probe$ and normal categories. The average execution time of classification for Corrected KDD dataset is 1 minute.

5.3 Results on NSL-KDD dataset

NSL-KDD [19] is a data set for network-based intrusion detection systems. It is the new version of KDD Cup 1999 intrusion detection benchmark dataset. In the KDD Cup dataset, there are huge number of redundant records, which can cause the learning algorithms to be biased towards the frequent records. To solve this issue, one copy of each record is kept in the NSL-KDD data set. Though, this dataset is not the perfect representative of real networks, still, it can be applied as an effective benchmark dataset to compare different intrusion detection methods. NSL-KDD included two datasets $KDDTrain^+$ and $KDDTest^+$ are shown in Table 5.

5.3.1 2-class prediction results

The confusion matrix for the 2-class behavior category on the $KDDTest^+$ datasets is shown in Table 12. The classification model is trained with $KDDTrain^+$ dataset. Classification rate of $PCC = 98.34\%$ in Table 12 indicate good performance for our method. The performance for the datasets is better compared to the KDD Cup dataset for 2-class classification.

5.3.2 5-class prediction results

The confusion matrix for the 5-class behavioral category of $KDDTest^+$ dataset is shown in Table 16. The classification model is trained with $KDDTrain^+$ dataset. Classification rate of $PCC = 98.39\%$ in Table 16 indicate good performance for our method datasets.

5.3.3 All-attacks prediction results

The confusion matrix for the all-attacks categories on $KDDTest^+$ dataset is shown in Table 13. Here, the classification model is trained with $KDDTrain^+$ dataset. The testing dataset $KDDTest^+$ includes 37 attack classes and training dataset $KDDTrain^+$ includes 15 attack classes. Objects belonging to the 15 categories of attacks cannot be classified correctly since no examples are present in the training set and hence they are excluded (3751 objects) from the testing dataset. Classification rate of $PCC = 98.94\%$ in Table 13 indicates good performance for our method. The classification rates still remain better compared to the KDD Cup dataset. The average execution time of classification for KDDTest+ dataset is 0.07 minute.

5.3.4 Performance comparisons

The algorithms $C4.5$ [22], $CART$ [23], $BayesianNetwork(BN)$ [24] and $CN2$ [25] rule-based algorithm are executed for TUIDS dataset using Weka [26]. Their results are compared with method as given in Table 17 for TUIDS datasets. The performance results of our method outperforms the other methods.

The performance results of our method for *Corrected KDD* dataset is compared with the experiment results provided in [14] for methods $C4.5$ [14, 22], $CART$ [14, 23], $BayesianNetwork(BN)$ [14, 24] and $CN2$ [14, 25] rule-based algorithm as given in Tables 18-20. We see that in terms of PCC our algorithm performs better than these algorithms in most of the cases.

The performance rates of SVM-based IDS [15], a combined method of hierarchical clustering and SVM method, are shown in Tables 14 and 15 using *Corrected KDD* dataset. The evaluation result using the $Corrected\ KDD$ dataset, shows that our algorithm performance rates are distinctly higher in the 5-class attack behavior and in new attack detection. The accuracy rate of detection for DoS attacks is 99.99% as shown in Table 14. In the detection of $snmpgetattack$ and $snmpguess$ attacks, our method detects 6457 and 2360 records respectively, whereas the other method detected 0 and 1 record respectively as shown in Table 15.

Table 16: 5-class confusion matrix of $KDDTest^+$ dataset trined with $KDDTrain^+$ dataset

		Predicted Class							
		$Normal$	$R2L$	DoS	$Probe$	$U2R$	Sum	Recall	1-Prc*
Actual class	$Normal$	9556	125	5	24	0	9710	0.9841	0.0195
	$R2L$	157	2727	0	0	3	2887	0.9446	0.0482
	DoS	14	1	7443	0	0	7458	0.9980	0.0009
	$Probe$	16	5	2	2399	0	2422	0.9905	0.0099
	$U2R$	3	7	0	0	57	67	0.8507	0.0500
	Sum	9746	2865	7450	2423	60	22544		
Re-substitution error=0.0161								PCC=98.39%	

Note- *1-Precision

Table 17: Comparison among CART, C4.5, CN2, BN for all Attacks of TUIDS datasets

Data set	Attack values	CART		C4.5		CN2		BN		Our Algorithm	
		Recall	**1-Prc***	**Recall**	**1-Prc***	**Recall**	**1-Prc***	**Recall**	**1-Prc***	**Recall**	**1-Prc***
Packet level	$normal$	0.9430	0.0540	0.9436	0.0507	0.9822	0.1442	0.7855	0.0215	**0.9928**	0.0014
	$smurf$	0.6360	0.3862	0.6384	0.3828	0.0026	0.0000	0.9503	0.6135	**0.9981**	0.0403
	1234	0.7289	0.1721	0.2353	0.5000	0.7900	0.2410	0.8400	0.2510	**0.9883**	0.0550
	$bonk$	0.8600	0.3156	0.7500	0.2100	0.8300	1.0000	0.8950	0.2400	0.5714	0.2000
	$fraggle$	1.0000	0.0000	1.0000	0.0000	1.0000	0.0011	0.9995	0.0012	**1.0000**	0.0000
	$jolt$	0.9641	0.0199	0.9902	0.0000	0.9248	0.0752	0.9739	0.2905	**0.9998**	0.0424
	$nestea$	0.7694	0.2310	0.0556	0.0000	6830	0.3100	0.8450	0.2010	**0.9993**	0.0027
	$newtear$	0.6829	0.2390	0.6300	0.2190	0.5910	0.2110	0.9300	0.3400	**0.9900**	0.0000
	$oshare$	0.7590	0.3270	0.8200	0.1040	0.6703	0.1401	0.8700	0.1031	**1.0000**	0.0000
	$saihyousen$	0.8900	0.1590	0.2353	0.5000	0.7810	0.2110	0.8505	0.2030	**0.9500**	0.0952
	$syndrop$	0.5910	0.3410	0.4550	0.1060	0.6900	0.1030	0.5450	0.1300	**1.0000**	0.0331
	syn	**1.0000**	0.0000	**1.0000**	0.0000	**1.0000**	0.0011	0.9995	0.0012	0.9931	0.0014
	$teardrop$	0.9641	0.0199	**0.9902**	0.0000	0.9248	0.0752	0.9739	0.2905	0.9759	0.0714
	$window$	0.7910	0.3200	0.0556	0.0000	0.7500	0.2100	0.7480	0.2041	**0.9881**	0.2727
	$winnuke$	0.6830	0.1520	0.5800	0.2170	0.5950	0.2140	0.6200	0.2100	**1.0000**	0.0000
	$xmas$	0.7420	0.1320	0.7300	0.2380	0.6350	0.2100	0.7900	0.2100	**0.9911**	0.0632
		PCC	98.30%		98.65%		95.67%		94.15%		99.42%
Flow level	$normal$	0.9500	0.2310	0.8850	0.2200	0.8350	0.2101	0.7950	0.0067	**0.9918**	0.0018
	$smurf$	0.8507	0.2030	0.7900	0.1220	0.8900	0.2020	0.8105	0.0821	0.6000	0.0103
	1234	0.9863	0.0137	0.9944	0.0293	0.8546	0.1428	0.9850	0.4125	**0.9999**	0.0150
	$bonk$	0.9800	0.1040	0.8700	0.0098	0.9300	0.2055	0.8600	0.0025	**0.9961**	0.0340
	$fraggle$	0.9724	0.2917	0.8598	0.0628	0.8898	0.1113	0.8677	0.1160	**0.9987**	0.0029
	$jolt$	0.8500	0.2010	0.2308	0.0000	0.8250	0.0712	0.9100	0.1090	0.8486	0.0474
	$land$	0.9706	0.0404	0.8927	0.0389	0.8965	0.0268	0.9924	0.0905	**0.9998**	0.0000
	$nestea$	**0.9750**	0.0250	0.9789	0.0000	0.8762	0.1086	0.9657	0.0639	0.5217	0.0021
	$newtear$	0.7809	0.3105	0.8700	0.2110	0.9700	0.2510	0.6950	0.0910	0.7956	0.0062
	$saihyousen$	0.8510	0.1700	0.9881	0.3197	**1.0000**	0.0000	**1.0000**	0.6147	0.7817	0.0059
	$syndrop$	0.9205	0.1850	0.9769	0.0000	0.8900	0.0072	0.6900	0.2900	0.6515	0.0371
	syn	**0.9990**	0.0016	0.9978	0.0011	0.9994	0.0011	0.9923	0.0000	0.9775	0.0214
	$teardrop$	0.4500	0.0230	0.5000	0.0000	0.4900	0.0087	0.7800	0.0426	0.5769	0.0314
	$window$	0.8950	0.0089	0.9400	0.2330	0.7800	0.2600	0.8100	0.1100	**0.9832**	0.3728
	$winnuke$	**1.0000**	0.4583	0.9545	0.0132	0.7951	0.1164	0.9791	0.0835	0.9740	0.0000
	$xmas$	0.7025	0.5088	0.8038	0.3553	0.8987	0.1744	0.9494	0.4700	**0.9720**	0.0538
		PCC	98.69%		97.19%		95.26%		95.15%		99.01%
Portscan	$Normal$	0.9342	0.3527	0.5000	0.0078	0.7263	0.4219	0.8621	0.0069	**0.9876**	0.0024
	SYN	0.8458	0.3910	0.8600	0.7610	0.6500	0.4510	0.5500	0.3290	**0.9859**	0.0040
	ACK	**1.0000**	0.4583	0.9545	0.0132	0.8252	0.1164	0.9791	0.0835	0.9930	0.0252
	FIN	0.7536	0.5088	0.8038	0.3553	0.9393	0.1744	0.9494	0.4700	**0.9766**	0.0153
	$xmas$	0.8813	0.4510	0.6800	0.24100	0.7502	0.3160	0.6578	0.2715	**0.9761**	0.0143
		PCC	87.45%		82.37%		78.76%		78.99%		98.31%

Note- *1-Precision

Table 18: Comparison among CART, C4.5, CN2, BN for two-class on $Corrected\ KDD$ dataset

	CART		C4.5		CN2		BN		Our Algorithm	
Values	**Recall**	**1-Prc***	**Recall**	**1-Prc***	**Recall**	**1-Prc***	**Recall**	**1-Prc***	**Recall**	**1-Prc***
$Normal$	0.9297	0.0511	0.9297	0.0511	0.8917	0.0640	0.9869	0.1835	0.9001	0.0269
$Attack$	0.9879	0.0169	0.9879	0.0145	0.9852	0.0259	0.9463	0.0033	**0.9940**	0.0237
PCC		97.65%		97.85%		96.69%		95.42%		97.57%

Note- *1-Precision

Table 19: Comparison among CART, C4.5, CN2, BN for five-class on $Corrected\ KDD$ dataset

	CART		C4.5		CN2		BN		Our Algorithm	
Values	**Recall**	**1-Prc***	**Recall**	**1-Prc***	**Recall**	**1-Prc***	**Recall**	**1-Prc***	**Recall**	**1-Prc***
$Normal$	0.9379	0.0526	0.9442	0.0526	0.8708	0.0490	0.7946	0.0624	0.9007	0.0273
$R2L$	0.7813	0.1919	0.8153	0.1927	0.8451	0.3561	0.8871	0.4554	**0.9110**	0.2873
DoS	0.9984	0.0027	0.9997	0.0011	0.9993	0.0015	0.9811	0.0022	**0.9999**	0.0001
$Probe$	0.9081	0.2525	0.9482	0.0403	0.9585	0.0230	0.8778	0.3036	**0.9875**	0.0044
$U2R$	0.5526	0.4545	0.6711	0.0192	0.6754	0.12	0.7281	0.9195	**0.7857**	0.0517
PCC		97.37%		97.83%		96.54%		93.83%		97.57%

Note- *1-Precision

Table 20: Comparison among CART, C4.5, CN2, BN for all Attacks on $Corrected\ KDD$ dataset

Attack values	CART		C4.5		CN2		BN		Our Algorithm	
	Recall	**1-Prc***	**Recall**	**1-Prc***	**Recall**	**1-Prc***	**Recall**	**1-Prc***	**Recall**	**1-Prc***
$normal$	0.9430	0.0540	**0.9436**	0.0507	0.9822	0.1442	0.7855	0.0215	0.9016	0.0257
$snmpgetattack$	0.6360	0.3862	0.6384	0.3828	0.0026	0.0000	**0.9503**	0.6135	0.8341	0.4792
$named$	0.0000	1.0000	0.2353	0.5000	0.0000	1.0000	0.0000	1.0000	**0.8824**	0.3750
$xlock$	0.0000	1.0000	0.0000	1.0000	0.0000	1.0000	0.0000	1.0000	**0.8889**	0.1111
$smurf$	1.0000	0.0000	1.0000	0.0000	1.0000	0.0011	0.9995	0.0012	**1.0000**	0.0000
$ipsweep$	0.9641	0.0199	0.9902	0.0000	0.9248	0.0752	0.9739	0.2905	**0.9869**	0.0033
$multihop$	0.0000	1.0000	0.0556	0.0000	0.0000	1.0000	0.0000	1.0000	**0.7222**	0.0714
$xsnoop$	0.0000	1.0000	0.0000	1.0000	0.0000	1.0000	0.0000	1.0000	**0.7500**	0.0000
$sendmail$	0.0000	1.0000	0.0000	1.0000	0.0000	1.0000	0.0000	1.0000	**0.9412**	0.2000
guess_passwd	0.9968	0.0566	0.9863	0.0242	0.9725	0.0270	0.9679	0.0404	**0.9979**	0.0002
$saint$	0.1236	0.0000	0.8302	0.2382	0.8003	0.2209	0.7024	0.3277	**0.9402**	0.6121
buffer_overflow	0.0000	1.0000	0.4545	0.4118	0.0000	1.0000	0.0000	1.0000	**0.9545**	0.0455
$portsweep$	0.8362	0.1111	0.9463	0.1604	0.7260	0.1376	**0.9915**	0.2252	0.9746	0.0000
pod	0.8391	0.0000	**1.0000**	0.4082	0.8391	0.0000	0.7701	0.4071	0.9655	0.0118
$apache2$	0.0000	1.0000	0.9937	0.1841	0.8476	0.0399	0.9786	0.0152	**0.9987**	0.0000
phf	0.0000	1.0000	0.0000	1.0000	0.0000	1.0000	0.0000	1.0000	**1.0000**	0.0000
$udpstorm$	0.0000	1.0000	0.0000	1.0000	0.0000	1.0000	0.0000	1.0000	**1.0000**	0.0000
$warezmaster$	0.9863	0.0137	**0.9944**	0.0293	0.8546	0.1428	0.9850	0.4125	0.9744	0.0013
$perl$	0.0000	1.0000	0.0000	1.0000	0.0000	1.0000	0.0000	1.0000	**1.0000**	0.0000
$satan$	**0.9724**	0.2917	0.8598	0.0628	0.8898	0.1113	0.8677	0.1160	0.3270	0.0582
$xterm$	0.0000	1.0000	0.2308	0.0000	0.0000	1.0000	0.0000	1.0000	**0.7692**	0.0000
$mscan$	0.9706	0.0404	0.8927	0.0389	0.8965	0.0268	0.9924	0.0905	**1.0000**	0.0000
$processtable$	0.9750	0.0250	0.9789	0.0000	0.8762	0.1086	0.9657	0.0639	**1.0000**	0.0000
ps	0.0000	1.0000	0.0000	1.0000	0.0000	1.0000	0.0000	1.0000	**0.8125**	0.0000
$nmap$	0.0000	1.0000	0.9881	0.3197	1.0000	0.0000	1.0000	0.6147	**1.0000**	0.0000
$rootkit$	0.0000	1.0000	0.0769	0.0000	0.0000	1.0000	0.0000	1.0000	**0.6923**	0.1818
$neptune$	0.9990	0.0016	0.9978	0.0011	0.9994	0.0011	0.9923	0.0000	**1.0000**	0.0000
$loadmodule$	0.0000	1.0000	0.5000	0.0000	0.0000	1.0000	0.0000	1.0000	**1.0000**	0.0000
$imap$	0.0000	1.0000	0.0000	1.0000	0.0000	1.0000	0.0000	1.0000	**1.0000**	0.0000
$back$	1.0000	0.4583	0.9545	0.0132	0.7951	0.1164	0.9791	0.0835	**1.0000**	0.0000
$httptunnel$	0.7025	0.5088	0.8038	0.3553	0.8987	0.1744	0.9494	0.4700	**0.9873**	0.0000
$worm$	0.0000	1.0000	0.0000	1.0000	0.0000	1.0000	0.0000	1.0000	0.0000	1.0000
$mailbomb$	0.9516	0.0000	0.9982	0.0062	**0.9998**	0.0161	0.9992	0.0046	**0.9998**	0.0000
ftp_write	0.0000	1.0000	0.0000	1.0000	0.0000	1.0000	0.0000	**1.0000**	**1.0000**	0.0000
$teardrop$	0.0000	1.0000	0.0000	1.0000	0.0000	1.0000	0.0000	1.0000	**0.4167**	0.1667
$land$	0.0000	1.0000	0.0000	1.0000	0.0000	1.0000	0.0000	1.0000	**1.0000**	0.0000
$sqlattack$	0.0000	1.0000	0.0000	1.0000	0.0000	1.0000	0.0000	1.0000	**1.0000**	0.0000
$snmpguess$	0.9909	0.0004	**0.9983**	0.0144	0.3603	0.3812	0.3624	0.4718	0.9809	0.0000
PCC		97.25%		97.69%		96.16%		94.75%		97.25%

Note- *1-Precision

Table 8: All-attacks confusion matrix for TUIDS datasets

Data set	Actual Class	Predicted Class: Detected	Present	Recall	1-Prc*
Packet level	*normal*	34790	35043	0.9928	0.0014
	smurf	38652	38726	0.9981	0.0403
	1234	19679	19902	0.9883	0.0550
	bonk	387	678	0.5714	0.2000
	fraggle	8622	8624	1.0000	0.0000
	jolt	27659	27661	0.9998	0.0424
	nestea	23758	23775	0.9993	0.0027
	newtear	23537	23775	0.9900	0.0000
	oshare	20000	20000	1.0000	0.0000
	saihyousen	4338	4567	0.9500	0.0952
	syndrop	53623	53627	1.0000	0.0331
	syn	20230	20371	0.9931	0.0014
	teardrop	8171	8373	0.9759	0.0714
	window	39908	40386	0.9881	0.2727
	winnuke	68248	68249	1.0000	0.0000
	xmas	38771	39118	0.9911	0.0632
	Sum	430373	432875		
	Re-substitution error=0.0058			PCC=99.42%	

Flow level	Actual Class	Predicted Class: Detected	Present	Recall	1-Prc*
	normal	36103	36402	0.9918	0.0018
	smurf	18	30	0.6000	0.0103
	1234	163408	163412	0.9999	0.0150
	bonk	27196	27303	0.9961	0.0340
	fraggle	38238	38288	0.9987	0.0029
	jolt	1183	1394	0.8486	0.0474
	land	2	2	0.9998	0.0000
	nestea	48	92	0.5217	0.0021
	newtear	109	137	0.7956	0.0062
	saihyousen	197	252	0.7817	0.0059
	syndrop	43	66	0.6515	0.0371
	syn	20330	20797	0.9775	0.0214
	teardrop	75	130	0.5769	0.0314
	window	40947	41646	0.9832	0.3728
	winnuke	29915	30714	0.9740	0.0000
	xmas	38361	39465	0.9720	0.0538
	Sum	396173	400131		
	Re-substitution error=0.0099			PCC=99.01%	

Portscan	Actual Class	Predicted Class: Detected	Present	Recall	1-Prc*
	Normal	2414	2445	0.9876	0.0024
	SYN	9612	9750	0.9859	0.0040
	ACK	9875	9945	0.9930	0.0252
	FIN	9551	9780	0.9766	0.0153
	xmas	9505	9740	0.9761	0.0143
	Sum	40957	41660		
	Re-substitution error=0.0169			PCC=98.31%	

Note- *1-Precision

Table 9: 2-class Confusion matrix of $Corrected\ KDD$ with training $10\%KDD$ dataset

		Predicted Class: *Normal*	*Attack*	Sum	Recall	1-Prc*
Actual class	*Normal*	60215	378	60593	0.9938	0.2508
	Attack	20155	230281	250436	0.9195	0.0016
	Sum	80370	230659	311029		
	Re-substitution error=0.0660				PCC=93.40%	

Note- *1-Precision

Table 11: All-attacks confusion matrix results of $Corrected\ KDD$ dataset training with $10\%KDD$

Actual class	Predicted Class: Detected	Present	Recall	1-Prc*
normal	60207	60593	0.9936	0.0790
smurf	164089	164091	1.0000	0.0002
ipsweep	298	306	0.9739	0.0418
multihop	0	18	0.0000	1.0000
guess_passwd	3	4367	0.0007	0.2500
buffer_overflow	2	22	0.0909	0.5000
portsweep	340	354	0.9605	0.2075
pod	82	87	0.9425	0.1546
phf	1	2	0.5000	0.0000
warezmaster	2	1602	0.0012	0.0000
perl	0	2	0.0000	1.0000
satan	1280	1633	0.7838	0.1551
nmap	84	84	1.0000	0.0455
rootkit	2	13	0.1538	0.9962
neptune	57971	58001	0.9995	0.0004
loadmodule	0	2	0.0000	1.0000
imap	0	1	0.0000	1.0000
back	1068	1098	0.9727	0.0000
ftp_write	0	3	0.0000	1.0000
teardrop	12	12	1.0000	0.7857
land	6	9	1.6667	0.6667
warezclient	0	0	0.0000	1.0000
spy	0	0	0.0000	1.0000
Sum	285447	292300		
Re-substitution error=0.0234			PCC=97.66%	

Note- *1-Precision

Table 12: 2-class confusion matrix of $KDDTest^{+}$ dataset training with $KDDTrain^{+}$

		Predicted Class: *Normal*	*Attack*	Sum	Recall	1-Prc*
Actual class	*Normal*	9531	179	9710	0.9816	0.0200
	Attack	195	12639	12834	0.9848	0.0140
	Sum	9726	12818	22544		
	Re-substitution error =0.0166				PCC=98.34%	

Note- *1-Precision

Table 13: All attacks confusion matrix of $KDDTest^{+}$ dataset

		Predicted Class			
		Detected	Present	Recall	1-Prc*
Actual class	$normal$	9609	9710	0.9895	0.0125
	$smurf$	605	665	0.9098	0.0529
	$ipsweep$	136	141	0.9845	0.0178
	$multihop$	17	18	0.9444	0.2273
	$guess\_passwd$	1130	1231	0.9180	0.0238
	$buffer\_overflow$	20	20	1.0000	0.0000
	$portsweep$	149	157	0.9490	0.0688
	pod	39	41	0.9512	0.0000
	phf	2	2	1.0000	0.0000
	$warezmaster$	902	944	0.9555	0.0075
	$perl$	2	2	1.0000	0.0000
	$satan$	716	735	0.9741	0.1836
	$nmap$	73	73	1.0000	0.0000
	$rootkit$	12	13	0.9231	0.2500
	$neptune$	4626	4657	0.9933	0.0008
	$loadmodule$	2	2	1.0000	0.0000
	$imap$	1	1	1.0000	0.0000
	$back$	339	359	0.9443	0.3416
	ftp_write	3	3	1.0000	0.0000
	$teardrop$	5	12	0.4167	0.2857
	$land$	7	7	1.0000	0.0000
	Sum	18593	18793		
Re-substitution error=0.0181				PCC=98.94%	

Note- *1-Precision

Table 14: Comparison with SVM-based IDS for 5-class over $Corrected\ KDD$ dataset

Type of Traffic	**SVM-based IDS**			**Accuracy of Our Algorithm (%)**
	Correctly Detected	**Miss Detected**	**Accuracy (%)**	
$Normal$	60166	427	99.29	90.07
DoS	228769	1084	99.53	**99.99**
$Probe$	4064	102	97.55	**98.75**
$U2R$	45	183	19.73	78.57
$R2L$	4664	11525	28.81	**91.10**
Overall	297708	13321	95.72	**97.57**

Table 15: Comparison with SVM-based IDS for attack detection over $Corrected\ KDD$ dataset

Attack Name	**Attack present**	**Detection**	
		SVM-based IDS	**Our Algorithm**
$apache2$	794	536	793
$mailbomb$	5000	4459	4999
$processtable$	759	578	759
$mscan$	1053	981	1053
$saint$	736	724	692
$httptunnel$	158	15	156
ps	16	3	13
$sqlattack$	2	2	2
$xterm$	13	5	10
$sendmail$	17	2	**16**
$named$	17	3	**15**
$snmpgetattack$	7741	0	**6457**
$snmpguess$	2406	1	**2360**
$xlock$	9	2	**8**
$xsnoop$	4	0	**3**
$worm$	2	0	0
Total	18729	39.04%	**92.56%**

6 Conclusion and future works

In this paper we provide a clustering based classification method and applied it in network anomaly detection. We have developed a subspace based incremental clustering method which forms the basis for the classification method. A training algorithm with a combination of unsupervised incremental clustering and supervised classification algorithm clusters a labeled training dataset into different clusters which are then represented by their profiles. These profiles together with the class label behave as classification rules. Prediction is done using a supervised classification algorithm that matches testing objects with the cluster profiles for labeling them. The simple classification method provided is effective in network anomaly detection as indicated by the evaluation results on real life TUIDS intrusion dataset and the benchmark intrusion datasets. The method can be applied for other classification jobs as well.

At present, rules (profiles) are stored in a flat file. A decision tree can be constructed based on the derived rules to reduce search time. Investigation for dealing with test instances that fit poorly with the supervised profiles may increase the performance of the algorithm. Another possible area is rule refinement that may cause reduction in number of attributes in each rule and also reduction of some rules themselves. For instance, KDD 1999 data set contains many similar training examples having different labels. In such situation, similar rules may be derived to detect more than one category of attacks.

Acknowledgement

This work is an outcome of a research project funded by MCIT, New Delhi.

References

[1] W. Lee and S. J. Stolfo, "Data mining approaches for intrusion detection," in *Proc. of the 7th conf. on USENIX Security Symposium - Volume 7*. San Antonio, Texas, USA: USENIX, 26-29, January 1998, pp. 1–16.

[2] R. Agarwal and M. V. Joshi, "PNrule: A New Framework for Learning Classifier Models in Data Mining (A Case-Study in Network Intrusion Detection)," Department of Computer Science, University of Minnesota, Tech. Rep. TR 00-015, 2000.

[3] K. Burbeck and S. Nadjm-Tehrani, "ADWICE - Anomaly Detection with Real-Time Incremental Clustering," in *Proc. of Information Security and Cryptology -ICISC 2004*, vol. 3506/2005. Berlin, Germany: Springer, May 2005, pp. 407–424.

[4] T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: an effective data clustering method for very large databases," *SIGMOID Record 1996 ACM SIGMOID*

Int'nl Conf. on Management of Data, vol. 25, pp. 103–114, 1996.

[5] A. Abraham and R. Jain, "Soft Computing Models for Network Intrusion Detection Systems," *Studies in Computational Intelligence*, vol. 16, pp. 191–211, 2005.

[6] H. Yang, F. Xie, and Y. Lu, "Clustering and Classification Based Anomaly Detection," *Fuzzy Systems and Knowledge Discovery*, vol. 4223/2006, pp. 1082–1091, 2006.

[7] A. Lazarevic, L. Ertoz, V. Kumar, A. Ozgur, and J. Srivastava, "A Comparative Study of Supervised Machine Learning Techniques for Intrusion Detection," in *Proc. of the 3rd SIAM Int'nl Conf. on Data Mining*. San Francisco, CA: SIAM, May 1-3 2003, pp. 25–36.

[8] G. Stein, B. Chen, A. S. Wu, and K. A. Hua, "Decision tree classifier for network intrusion detection with GA-based feature selection," in *Proc. of the 43rd Annual Southeast Regional Conference, ACM-SE 43*, 2005, pp. 136–141.

[9] J. Zhang and M. Zulkernine, "Network intrusion detection using random forest," in *Proc. of the 3rd Annual Conf. on Privacy, Security and Trust*, 2005, pp. 53–61.

[10] Y. Li, B.-X. Fang, L. Guo, and Y. Chen, "TCM-KNN Algorithm for Supervised Network Intrusion Detection," *LNCS*, vol. 4430/2007, pp. 141–151, 2007.

[11] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Macia-Fernandez, and E. Vazquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *Computers & Security*, vol. 28, pp. 18–28, 2009.

[12] P. Gogoi, B. Borah, and D. K. Bhattacharyya, "Anomaly Detection Analysis of Intrusion Data using Supervised & Unsupervised Approach," *Journal of Convergence Information Technology*, vol. 5(1), pp. 95–110, Feb. 2010.

[13] P. Gogoi, D. K. Bhattacharyya, B. Borah, and J. K. Kalita, "A Survey of Outlier Detection Methods in Network Anomaly Identification," *The Computer Journal*, vol. 54(4), pp. 570–588, 2011.

[14] R. Beghdad, "Critical Study of Supervised Learning Techniques in Predicting Attacks," *Information Security Journal: A Global Perspective*, vol. 19, pp. 22–35, 2010.

[15] S.-J. Horng, M.-Y. Su, Y.-H. Chen, T.-W. Kao, R.-J. Chen, J.-L. Lai, and C. D. Perkasa, "A novel intrusion detection system based on hierarchical clustering and support vector machines," *Expert Systems with Applications*, vol. 38, pp. 306–313, 2011.

[16] M. Roesch, "Snort-lightweight intrusion detection for networks," in *Proc. of the 13th USENIX conf. on System administration*. Seattle, Washington: USENIX, Nov. 1999, pp. 229–238.

[17] P. Gogoi, M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, "Packet and Flow Based Network Intrusion Dataset," in *LNCS Proc. of IC3 2012*, ser. CCIS, vol. 306. Berlin Heidelberg: Springer, August 6-8 2012, pp. 322–334.

[18] University of California, "KDD Cup 1999 Data," http://kdd.ics.uci.edu/∼/kddcup99.html, Irvine, 1999.

[19] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A Detailed Analysis of the KDD CUP 99 Data Set," in *Proc. of the 2009 IEEE Symposium on Computational Intelligence in Security and Defence Applications (CISDA 2009)*. Ottawa, Canada: IEEEXplore, 8-10 July 2009, pp. 1–6.

[20] H. G. Kayacik, A. N. Z. Heywood, and M. I. Heywood, "Selecting Features for Intrusion Detection: A Feature Relevance Analysis on KDD 99 Intrusion Detection Datasets," in *Proc. of the 3rd Annual Conf. on Privacy, Security and Trust*. Halifax, Dalhousie University, October 2005.

[21] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*. Pearson Education, Inc., 2009.

[22] J. R. Quinlan, "C4.5: Programs for Machine Learning," Morgan Kaufman, New York, USA, 1993.

[23] L. Brieman, J. H. Friedman, R. A. Olshen, and C. Stone, "Classification and reregression trees," Wadsworth & Brooks, Monterey, CA, 1984.

[24] F. V. Jesen, "Introduction to Bayesien networks," UCL Press, 1996.

[25] P. Clark and T. Niblett, "The CN2 induction algorithm," *Machine Learning*, vol. 3, pp. 261–283, 1989.

[26] University of Waikato, "Weka 3: Data Mining Software in Java," http://www.cs.waikato.ac.nz/ml/weka, New Zealand, 1999.