Expert Systems with Applications 38 (2011) 306–313

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

A novel intrusion detection system based on hierarchical clustering and support vector machines

Shi-Jinn Horng[a,b,*], Ming-Yang Su[c], Yuan-Hsin Chen[b], Tzong-Wann Kao[d], Rong-Jian Chen[b], Jui-Lin Lai[b], Citra Dwi Perkasa[a]

[a] *Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology, 43, Sec/4. Kee-Lung Road, 106 Taipei, Taiwan*
[b] *Department of Electronic Engineering, National United University, Miaoli, Taiwan*
[c] *Department of Computer Science and Information Engineering, Ming Chuan University, Taoyuan, Taiwan*
[d] *Department of Electronic Engineering, Northern Taiwan Institute of Science and Technology, Taipei, Taiwan*

ARTICLE INFO

Keywords:
Network intrusion detection system (NIDS)
Support vector machines (SVMs)
Hierarchical clustering algorithm
KDD Cup 1999
Network security
Data mining

ABSTRACT

This study proposed an SVM-based intrusion detection system, which combines a hierarchical clustering algorithm, a simple feature selection procedure, and the SVM technique. The hierarchical clustering algorithm provided the SVM with fewer, abstracted, and higher-qualified training instances that are derived from the KDD Cup 1999 training set. It was able to greatly shorten the training time, but also improve the performance of resultant SVM. The simple feature selection procedure was applied to eliminate unimportant features from the training set so the obtained SVM model could classify the network traffic data more accurately. The famous KDD Cup 1999 dataset was used to evaluate the proposed system. Compared with other intrusion detection systems that are based on the same dataset, this system showed better performance in the detection of DoS and Probe attacks, and the beset performance in overall accuracy.

© 2010 Elsevier Ltd. All rights reserved.

1. Introduction

Due to the advancement of the Internet technology, numerous e-commerce transactions are completed online. However, given the weakness of servers (Vigna, Robertson, Kher, & Kemmerer, 2003; Yu, Tsai, & Weigert, 2007), hackers often intrude upon these Internet transactions, and use DoS/DDoS attacks to prevent these servers from providing services. Network intrusion detection system (NIDS), as an important link in the network security infrastructures, aims to detect malicious activities, such as denial of service attacks, port scans, or even attempts to crack into computers by monitoring network traffic. In addition to inspecting incoming network traffic, NIDS can also obtain valuable information on an ongoing intrusion from outgoing or local traffic. NIDS is often not a standalone system, but works with other systems as a firewall. If necessary, it can update the blacklist of the firewall in real-time to block the suspicious connections.

A common problem of NIDS is that it specifically detects known service or network attacks only, which is called misuse detection, by using pattern matching approaches. On the other hand, an anomaly detection system detects attacks by building profiles of normal behaviors first, and then identifies potential attacks when their behaviors are significantly deviated from the normal profiles. Many methods have been proposed in the past few years on the design of effective NIDSs, among which, decision tree has been proven to have good performance. Bagged boosting (Pfahringer, 2000), based on C5 decision trees is a method used by the KDD Cup 1999 winner. In the KDD Cup (1999), the approaches of the top three winners were based on the tree methods with some variations. The runner-up entry used a variant of decision tree proposed by Kernel Miner (Levin, 2000), which is a data mining tool for building the optimal decision forest.

Some NIDSs are based on the fuzzy set theory. For example, fuzzy rough C-means (FRCM), proposed by Chimphlee, Abdullah, Md Sap, Srinoy, and Chimphlee (2006), utilized the advantage of fuzzy set theory and rough set theory for network intrusion detection. Another fuzzy approach, proposed by Toosi and Kahani (2007), combined the neuro-fuzzy network, fuzzy inference approach, and genetic algorithms to design their NIDS, and was evaluated by the KDD Cup 1999 dataset. Novikov, Yampolskiy, and Reznik (2006), the combination of a radial basis function (RBF) neural network and a multi layer perception (MLP) neural network was used to design an intrusion detection system. Sabhnani and Serpen (2003) analyzed the performances of a comprehensive set of pattern recognitions and machine learning algorithms. Their system

* Corresponding author at: Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology, 43, Sec/4. Kee-Lung Road, 106 Taipei, Taiwan. Tel.: +886 2 27376700; fax: +886 2 27301081.
E-mail addresses: horngsj@yahoo.com.tw (S.-J. Horng), minysu@mail.mcu.edu.tw (M.-Y. Su), yschen@nuu.edu.tw (Y.-H. Chen), tkao@tsint.edu.tw (T.-W. Kao), rjchen@nuu.edu.tw (R.-J. Chen), jllai@nuu.edu.tw (J.-L. Lai), liemcici@gmail.com (C.D. Perkasa).

0957-4174/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2010.06.066

in Sabhnani and Serpen (2003), which outperformed the KDD Cup 1999 winner's system, combined several classifiers, one designated for one type of attacks in the KDD Cup 1999 dataset. Xuren, Famei, and Rongsheng (2006), an association rule discovering system was applied in a rough set theory framework to design an NIDS.

Many researches have applied data mining techniques in the design of NIDS. One of the promising techniques is support vector machine (SVM), which solid mathematical foundations (Khan, Awad, & Thuraisingham, 2007; Yu, Yang, Han, & Li, 2003) have provided satisfying results. SVM separates data into multiple classes (at least two) by a hyperplane, and simultaneously minimizes the empirical classification error and maximizes the geometric margin. Thus, it is also known as maximum margin classifiers.

Although SVMs have shown good results in data classification, they are not favorable for large-scale dataset because the training complexity is very dependent on the amount of data in the training set. Larger amount of data would lead to higher training complexity. However, many data mining applications involve millions or even billions of pieces of data records. For example, in the KDD Cup 1999 dataset, there are more than 4 million and 3 million instances in the training set and test set, respectively. The SVM technique is unable to operate at such a large dataset due to system failures caused by insufficient memory, or may take too long to finish the training. Since this study used the KDD Cup 1999 dataset, to reduce the amount of data, a hierarchical clustering method was applied to preprocess the dataset before SVM training. The clustering method could produce high quality dataset with far less instances that sufficiently represent all of the instances in the original dataset.

This study proposed an SVM-based intrusion detection system based on a hierarchical clustering algorithm to preprocess the KDD Cup 1999 dataset before SVM training. The hierarchical clustering algorithm was used to provide a high quality, abstracted, and reduced dataset for the SVM training, instead of the originally enormous dataset. Thus, the system could greatly shorten the training time, and also achieve better detection performance in the resultant SVM classifier. The remainder of this paper is organized as follows. Section 2 provides an overview of the hierarchical clustering algorithm proposed by Zhang, Ramakrishnan, and Livny (1996) and the SVMs. Section 3 describes the proposed system. Section 4 presents experimental results to demonstrate that the performance of the proposed system is better than others. Finally concluding remarks are given in Section 5.

2. Background

This section will give a detailed description about the balanced iterative reducing and clustering using hierarchies (BIRCH) hierarchical clustering algorithm that is used to produce fewer significant instances from a very large dataset. With fewer significant instances, the support vector machines (SVMs) can achieve shorter training time and better classification performance. A brief introduction of the SVM is also given in this section.

2.1. BIRCH hierarchical clustering algorithm

The BIRCH hierarchical clustering algorithm applied in this system was originally proposed by Zhang et al. (1996). The concept of BIRCH is different from other clustering algorithms, such as CURE (Guha, Rastogi, & Shim, 1998), ROCK (Guha, Rastogi, & Shim, 1999), and Chameleon (Karypis, Han, & Kumar, 1999), because it stores fewer abstracted data points than the whole dataset. Each abstracted point represents the centroid of a cluster of data points. Compared to CURE, ROCK, and Chameleon, the BIRCH clustering algorithm can achieve high quality clustering with lower processing cost. The advantages of BIRCH are as follows:

- Constructs a tree, called a clustering feature (CF) tree, by only one scan of dataset using an incremental clustering technique.
- Able to handle noise effectively.
- Memory-efficient because BIRCH only stores a few abstracted data points instead of the whole dataset.

2.1.1. Clustering feature (CF)

The concept of a clustering feature (CF) tree is at the core of BIRCH's incremental clustering algorithm. Nodes in the CF tree are composed of clustering features. A CF is a triplet, which summarizes the information of a cluster.

Definition 1 (*Clustering feature (Zhang et al., 1996)*). Given n d-dimensional data points in a cluster $\{x_i\}$, where $i = 1, 2, \ldots, n$, the clustering feature (CF) of the cluster is a 3-tuple, denoted as CF = (n, LS, SS), where n is the number of data points in the cluster, LS is the linear sum of the data points, i.e., $\sum_{i=1}^{n} x_i$, and SS is the square sum of the data points, i.e., $\sum_{i=1}^{n} x_i^2$.

Theorem 1 (CF addition theorem (Zhang et al., 1996)). *Assume that $CF_1 = (n_1, LS_1, SS_1)$ and $CF_2 = (n_2, LS_2, SS_2)$ are the CFs of two disjoint clusters. Then the CF of the new cluster, as formed by merging the two disjoint clusters is*

$$\mathrm{CF}_1 + \mathrm{CF}_2 = (n_1 + n_2, LS_1 + LS_2, SS_1 + SS_2) \tag{3}$$

For example, suppose there are three points (2, 3), (4, 5), (5, 6) in cluster C1, then the CF of C1 is

$$\begin{aligned}\mathrm{CF}_1 &= \left\{3, (2+4+5, 3+5+6), \left(2^2+4^2+5^2, 3^2+5^2+6^2\right)\right\}\\ &= \{3, (11, 14), (45, 70)\}.\end{aligned}$$

Suppose that there is another cluster C2 with $\mathrm{CF}_2 = \{4, (40, 42), (100, 101)\}$. Then the CF of the new cluster formed by merging cluster C1 and C2 is

$$\begin{aligned}\mathrm{CF}_3 &= \{3+4, (11+40, 14+42), (45+100, 70+101)\}\\ &= \{7, (51, 56), (145, 171)\}.\end{aligned}$$

By Definition 1 and Theorem 1, the CFs of clusters can be stored and calculated incrementally and accurately as clusters are merged. Based on the information stored in CF, the centroid C and radius R of a cluster can be easily computed. The definitions of C and R of a cluster are given as follows. Given n d-dimensional data points, say $\{x_i\}$ and i = 1, 2, ..., n, in a cluster:

$$\text{the centroid} \quad C = \frac{\sum_{i=1}^{n} x_i}{n}, \quad \text{and} \tag{1}$$

$$\text{the radius} \quad R = \frac{\sum_{i=1}^{n} \|x_i - C\|^2}{n}. \tag{2}$$

where, R denotes the average distance of all member points to the centroid. As mentioned earlier, CF stores only the abstracted data point, i.e., statistically summary of data points that belong to the same cluster. After a data point is added into a cluster, the detail information of the data point itself is missing. Therefore, this approach can save space significantly for densely packed data points, especially when the size of the dataset is large.

2.1.2. CF tree

A CF tree is a height-balanced tree with two parameters, branching factor B and radius threshold T. Each non-leaf node in a CF tree contains the most B entries of the form (CF_i, $child_i$), where $1 \leqslant i \leqslant B$ and $child_i$ is a pointer to its ith child node, and CF_i is the CF

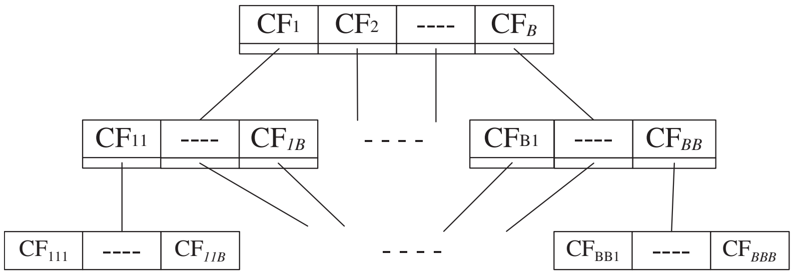

Fig. 1. A CF tree with height $h = 3$.

of a cluster pointed by the *child*$_i$. An example of a CF tree with height $h = 3$ is shown in Fig. 1.

Each node represents a cluster made up of sub-clusters, which represents its entries. It is different from a non-leaf node is that a leaf node has no pointer to link to other nodes, and contains at most B entries. The CF at any particular node contains information for all data points in that node's sub-trees. A leaf node must satisfy the threshold requirement, that is, every entry in every leaf node must have its radiuses less than threshold T.

A CF tree is a compact representation of a dataset, each entry in a leaf node represents a cluster that absorbs many data points within its radius of T or less. A CF tree can be built dynamically as new data points are inserted. The insertion procedure is similar to that of a B+-tree to insert a new data to its correct position in the sorting algorithm. The insertion procedure of CF tree has the following three steps.

Step 1: Identify the appropriate leaf
To identify the appropriate leaf node to insert a new entry, the algorithm starts from the root and traverses the CF tree recursively down to the leaf level by choosing the child node, whose centroid is closest at each level to the new entry. An example of a new entry is shown in Fig. 2, where CF_X, is inserted into the given CF tree. Starting from the root, the algorithm computes the distances between CF_X and each entry of $CF_1/CF_2/CF_3$ in the root; suppose their distances are $D1 = 0.52$, $D2 = 0.15$, and $D3 = 0.23$, respectively. The algorithm chooses the entry with the smallest distance, i.e., $D2$, and then traverses down to the child of CF_2. The algorithm repeats the process of computing the distances between CF_X of each entry in the node, with $D4 = 0.35$ and $D5 = 0.25$. Since the current node is a leaf node and $D5$ is smaller than $D4$, the appropriate place to insert the new entry is entry CF_{22}.

Step 2: Modify the leaf
When the closest entry in a leaf node is identified, there are three possible options. First, if the leaf can absorb the new entry without violating the radius threshold T condition, then the algorithm only needs to update the CF of the leaf entry, and then terminate (Fig. 3(a)). Second, if the leaf entry cannot absorb the new entry without violating T threshold, it will add a new leaf entry for the new entry (Fig. 3(b)). In this case, if adding a new entry violates the branching factor B threshold (i.e., too many children or entries), the leaf node has to split by choosing the farthest pair of entries as seeds, and redistribute the remaining entries based on its closeness to the seeds (Fig. 3(c)). In the example of Fig. 3(c), a leaf node has three entries (Fig. 3(c) upper part), and the new entry CF_4 is added into that node. Suppose that there is a new entry because the entry (say entry CF_3) cannot absorb CF_4 without violating T threshold, and the branching factor $B = 3$, thus, splitting is required in the leaf node. First, the distances between each entry pair in the leaf node (CF_1 vs. CF_2, CF_1 vs. CF_3, and CF_2 vs.

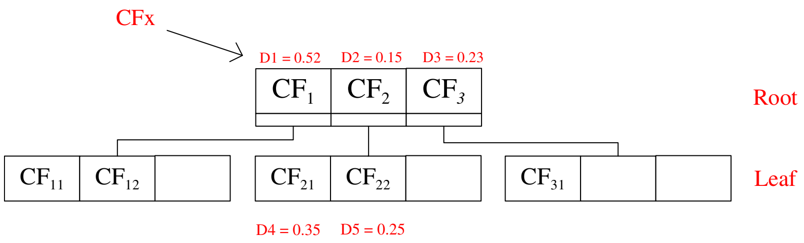

Fig. 2. Identifying the appropriate leaf entry.

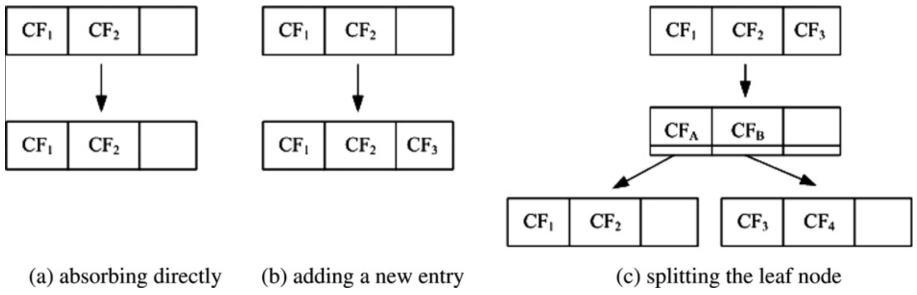

Fig. 3. Update of the leaf node.

CF_3) is computed, and the farthest pair is chosen. For example, if the farthest pair is CF_1 and CF_3, then these two entries are split into two different nodes, and redistribute the remaining entries, CF_2 and CF_4, based on their closeness to the current centroid of the new entries. Finally, the algorithm generates a parent node with two entries, CF_A and CF_B, to link the two leaf nodes.
Step 3: Modify entries on the path to the leaf
After inserting the new entry into a leaf node, the algorithm needs to update the CF information for each non-leaf entry, along the path backwards to the root. If no leaf node splitting is invoked, the algorithm simply adds the CF to reflect the addition of the new entry. If leaf node splitting is invoked, the algorithm checks whether the parent node meets the branching factor constraint. If the parent node violates the B threshold, it is split and recursively traversed back to the root, while performing the same checks.

There are several methods to compute the distance between two points, such as Euclidean distance and Manhattan distance. The simplest one and the one used in this study is Euclidean distance, described as follows:

$$D = ((x_1 - x_2)^2)^{1/2}, \tag{3}$$

where x_1 and x_2 are two data points.

An important parameter in building the CF tree is the T threshold because it determines the size of the tree, so that the tree fits into the memory. If T is too small, the number of node will be too large, thus, the program will run out of memory before scanning all of the data points. The original BIRCH algorithm initially sets a very low T, and iteratively increases it, until the tree fits into the memory. However, the problem is when changing the value of T, the tree must be rebuilt. Rebuilding a tree is an expensive process because it requires a re-scan of the presently inserted data, and at most h extra pages of memory, where h is the height of the tree (Zhang et al., 1996). In this study, the value T is used intuitively based on the number of data points, dimensionality, and value range of each dimensionality.

2.2. Support vector machines

An SVM is a supervised learning method (Hsu, Chang, & Lin, xxxx; Vapnik, 1995). It performs classification by constructing an N-dimensional hyperplane that optimally separates the data into different categories. In the basic classification, SVM classifies the data into two categories. Given a training set of instances, labeled pairs $\{(x, y)\}$, where y is the label of instance x, SVM works by maximizing the *margin* to obtain the best performance in classification, as shown in Fig. 4.

In SVM, the problem of computing a margin-maximizing boundary function is specified by the following quadratic programming (QP) problem:

$$\text{Minimize} \quad W(\alpha) = -\sum_{i=1}^{l} \alpha_i + \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l} y_i y_j \alpha_i \alpha_j k(x_i x_j)$$

$$\text{Subject to} \quad \forall i: 0 \leqslant \alpha_i \leqslant C, \quad \text{and} \quad \sum_{i=1}^{l} \alpha_i y_i = 0,$$

where l is the number of training data, α is a vector of l variables, in which each component α_i corresponds to the training data x_i, and C is the soft margin parameter, which controls the influence of the outliers (or noise) in the training data. The boundary function of SVM is described by *support vectors*, which are the data points located closest to the class boundary. The above quadratic programming problem computes vector α, where each element specifies a weight of each data point. Those data points, whose α_i's value are greater than 0, are the support vectors; and data points with α_i value equal to zero are the non-support vectors. For the boundary function, only support vectors are useful and considered. In the SVM, all training data points are mapped into a higher dimensional space. Then, SVM can find a separating hyperplane with a maximal margin in this higher dimensional space.

In the SVM, there are kernels, as listed below, and any of those can be chosen to achieve the boundary function. Their detailed usages and descriptions, including parameters definitions, can be found in Hsu et al. (xxxx):

- Linear kernel: $k(x_i, x_j) = x_i \cdot x_j$
- Polynomial kernel: $k(x_i, x_j) = (\gamma x_i^T x_j + r^d)^2$
- RBF kernel: $k(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2)$
- Sigmoid kernel: $k(x_i, x_j) = \tanh(\gamma x_i^T x_j + r)$.

As mentioned in Section 1, although SVM is a promising method for data classification, it has some drawbacks. Its main problem is that the training complexity is very dependent on the size of the dataset, it is known to be at least quadratic to the number of data points. Since the dataset applied in this study has more than 4 million data points, if SVM is used directly with all of the data points, it would be unable to finish the training process, or may take too long to finish the training. Therefore, a hierarchical clustering algorithm is applied in this paper to reduce the number of data points, from the original 4 million data points to hundreds of abstracted data points.

3. SVM with hierarchical clustering

This section will describe the proposed method, which is a combination of support vector machine and hierarchical clustering

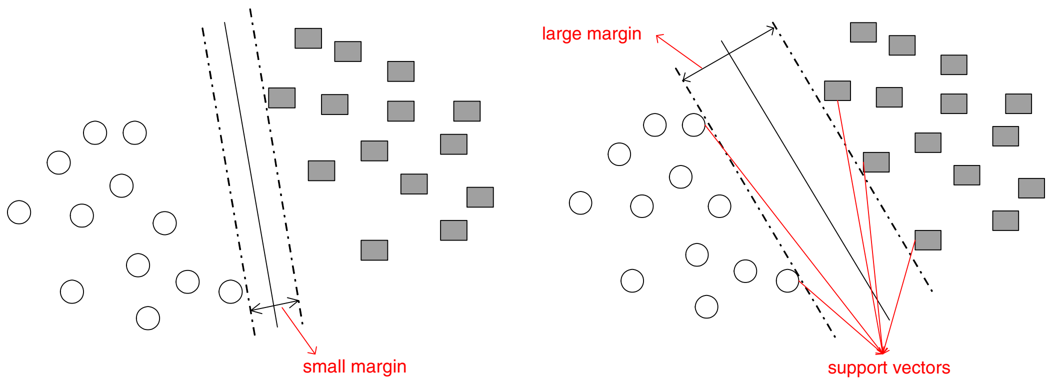

Fig. 4. Support vectors and margin maximizing.

(BIRCH), to build an intrusion detection system. The KDD Cup (1999) dataset, which contains more than 4 million data points, is applied in this paper. Since such a large dataset cannot be fed directly to SVM in the training phase, the BIRCH algorithm has to transform the KDD Cup 1999 dataset to a smaller sized dataset, and then the reduced dataset with abstracted data points can be used to train SVM classifiers for intrusion detection.

The key to the performance of the SVM classifier is the clustering algorithm. In order to let the SVM classifier having a high detection rate and a low false positive rate, the clustering algorithm must provide the SVM with "high quality" data points as its training set. The term "high quality" means that the abstracted data points in the reduced dataset must represent all data points in the original dataset, and the number of abstracted data points cannot be too small or too large. With a very small number of data points, SVM cannot generate a good SVM classifier. On the other hand, if the number of data points is too large, SVM cannot finish training and build the classifier (or it takes a very long time to finish the training).

In the paper, BIRCH clustering algorithm is first used to produce a reduced and high quality dataset from the original KDD Cup 1999 dataset before SVM training. The flow process of the proposed system is described as follows:

(1) Transform non-continuous attributes into continuous attributes, and then scale the whole dataset to let all feature values fall in the interval [0, 1].
(2) Construct four CF trees for DoS, U2R, R2L, and Probe (KDD Cup, 1999) attacks, respectively, and one CF tree for normal packets.
(3) Do feature selection for each type of attacks.
(4) Train four SVM classifiers separately corresponding to the four kinds of attacks, by the centroid of all entries in leaf nodes of CF trees.
(5) Combine the four SVMs classifiers to build an intrusion detection system.

Since step 4 follows the SVM procedure and step 5 is trivial, the first three procedures are described below.

3.1. Data transformation and scaling

SVM requires each data point to be represented as a vector of real numbers. Hence, every non-numerical attribute has to be transformed into numerical data first. The method is simply by replacing the values of the categorical attributes with numeric values. For example, the *protocol_type* attribute in KDD Cup 1999, thus, the value *tcp* is changed with 0, *udp* with 1, and *icmp* with 2.

The important step after transformation is scaling. Data scaling can avoid attributes with greater values dominating those attributes with smaller values, and also avoid numerical problems in computation. In this paper, each attribute is called with linear scaling to the range of [0, 1] by dividing every attribute value by its own maximum value.

3.2. CF trees construction

After data transformation and scaling, the next step is to construct the CF trees. The CF trees can be constructed with a single scan of the entire dataset. Since there are four kinds of attacks in KDD Cup 1999, i.e., DoS, R2L, U2R, and probing (KDD Cup, 1999), the four CF trees must be built separately corresponding to the four types of attack traffic. In addition, one CF tree for normal traffic is also required. The proposed intrusion detection system is composed of the four SVM classifiers.

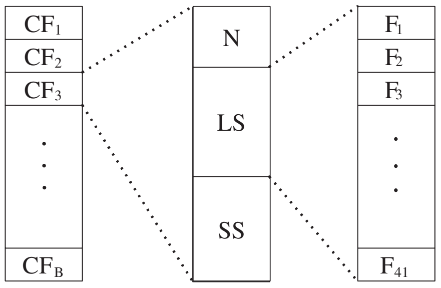

Fig. 5. Leaf node structure in CF tree for KDD Cup 1999 data set.

In the KDD Cup 1999 data set (KDD Cup, 1999), there are 41 features (listed in Appendix Table A1) suggested for each record. As mentioned in Section 2, each non-leaf node in CF tree contains at most *B* entries of the form (CF_i, $child_i$), and each leaf node contains *B* entries at most without pointers to other nodes, and each entry has the constraint of radius threshold *T*. In this paper, a node is represented as an object, which contains an array of CF, the actual size of the node (actual number of entries in that node), and pointers to child nodes (for non-leaf node only). Fig. 5 illustrates the leaf node structure in the implementation. The leftmost part in the figure is a leaf node, which contains at most *B* entries of CF. A CF entry (middle part of the figure) contains information about the number of data points (*N*), the linear sum (*LS*) of the data points, and square sum (*SS*) of the data points. The structure of *LS* and *SS* are the same, in which they contain the sum and square sum value of each of the features, respectively (right part of the figure).

Once the four CF trees for the four types of attacks and one CF tree for normal traffic are constructed, four training sets for SVMs can be built from these trees. Each training set can be derived from one attack CF tree for providing attack instances, and the normal CF tree for providing normal instances. Every instance, either attack or normal instance, is obtained from one entry of leaf node in the CF tree; one entry in the leaf node represents one cluster. The centroid of a single cluster at leaf level is considered as one instance in the training set. Fig. 6 shows an example, in which there are five leaf nodes, including 11 entries. Therefore, the CF tree provides 11 instances for the training set.

3.3. Feature selection

Although the KDD Cup 1999 suggested 41 features for each network connection in the dataset, in fact, not all features are needed in the design of a network intrusion detection system. It is critical to identify important features of network traffic data, in order to achieve maximal performance. In this study, whether a feature is important is determined based on the accuracy and the number of false positives of the system, with and without the feature. In other words, the feature selection of this study is "leave-one-out"; remove one feature from the original dataset, redo the experiment, then compare the new results with the original result, if any case of the following cases occurs. The feature is regarded as important; otherwise it is regarded as unimportant. Since there are 41 features suggested in the KDD Cup 1999, as shown in Appendix Table A1, the experiment is repeated 41 times to ensure that each feature is either important or unimportant.

Case 1: If the *accuracy* decreased and the *false positive* decreased, then the feature is important.
Case 2: If the *accuracy* decreased and the *false positive* increased, then the feature is important.
Case 3: If the *accuracy* unchanged and the *false positive* increased, then the feature is important.

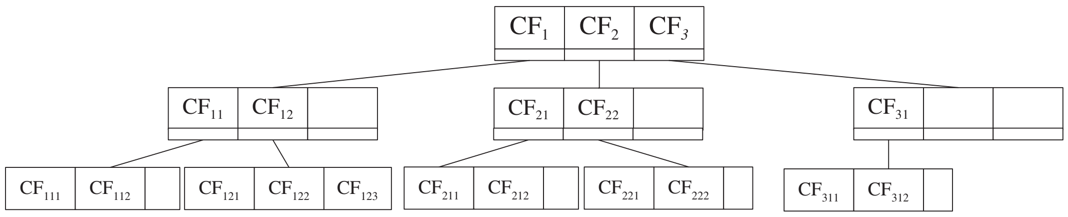

Fig. 6. Centroids of leaf node entries as training instances.

Table 1
Important features out of the 41 features.

Attack type	Important feature #
DoS	2, 4, 8, 10, 14, 17, 19, 22, 24, 25, 26, 27, 28, 30, 31, 33, 34, 35, 39
Probe	3, 4, 12, 22, 23, 24, 25, 26, 27, 29, 30, 31, 34, 36, 37, 39, 40
U2R	1, 2, 3, 9, 10, 11, 12, 13, 14, 15, 17, 18, 21, 22, 25, 29, 30, 31, 32, 36, 38, 39, 40, 41
R2L	1, 2, 3, 4, 7, 11, 12, 15, 17, 18, 19, 22, 23, 24, 26, 27, 29, 31, 32, 34, 35, 37, 38, 40

Case 4: If the *accuracy* increased and the *false positive* increased, then the feature is important.

According to the methodology, the important features for each kind of attack are listed in Table 1. Once the important features are identified, the SVM classifiers for the intrusion detection system can be trained and tested using only these important features.

4. Experimental results

KDD Cup (1999) data was used for the Third International Knowledge Discovery and Data Mining Tools Competition, which was held in conjunction with the Fifth International Conference on Knowledge Discovery and Data Mining. This database contained a wide variety of intrusions simulated in a military network environment. The KDD Cup 1999 provided both training dataset and test dataset. The test dataset included some specific attacks that did not appear in the training dataset to make the task more difficult and realistic. Some intrusion experts suggested that most novel attacks are variants of known attacks, and the "signature" of known attacks can be sufficient to catch novel variants. The datasets contained 24 training attack types, with additional 14 types in the test data only.

The KDD Cup 1999 contained 4,898,431 and 311,029 records in the training set and test set, respectively. All attack records were classified into four kinds of attacks, DoS, U2R, R2L, and Probe. For the training set, only 19.85% (972,781 records) were normal traffic and the remaining were attack traffic. For the test set, 19.48% (60,593 records) were normal traffic and the remaining were attack traffic. Each record in the KDD Cup 1999 data set contained 41 various quantitative and qualitative features (KDD Cup, 1999).

In this study, the whole training data and test data were applied. Every data instance in the training data or test data consisted

Table 2
System performance.

Type of traffic	Correctly detected	Miss detected	Accuracy (%)
Normal	60,166	427	99.29
Denial of service	228,769	1,084	99.53
Probe	4,064	102	97.55
U2R	45	183	19.73
R2L	4,664	11,525	28.81
Overall	297,708	13,321	95.72

Table 3
Numbers of instances in the training set.

Data type	DoS	Probe	U2R	R2L	Normal
Original data set	3,883,370	41,102	52	1,126	972,781
CF tree ($T = 0.1$)	700	722	33	76	13,121
CF tree ($T = 0.2$)	271	343	26	45	4,057
CF tree ($T = 0.3$)	133	273	20	35	1,507

Table 5
System performance on new attack.

Attack type	Attack name	Number of occurrences on test data	Number of occurrences detected
DoS	Apache2	794	536
	Mailbomb	5,000	4,459
	Processtable	759	578
Probing	Mscan	1,053	981
	Saint	736	724
U2R	Httptunnel	158	15
	Ps	16	3
	Sqlattack	2	2
	Xterm	13	5
R2L	Sendmail	17	2
	Named	17	3
	Snmpgetattack	7,741	0
	Snmpguess	2,406	1
	Xlock	9	2
	Xsnoop	4	0
	Worm	2	0
Total		18, 729	7, 311 (39.04%)

Table 4
Comparisons with other works by detection rate, accuracy, and false positive rate.

Method	Normal	DoS	Probe	U2R	R2L	Accuracy	FP
This method	99.3	99.5	97.5	19.7	28.8	95.7	0.7
ESC–IDS (Toosi and Kahani, 2007)	98.2	99.5	84.1	14.1	31.5	95.3	1.9
KDD'99 winner (Pfahringer, 2000)	99.5	97.1	83.3	13.2	8.4	91.8	0.6
KDD'99 runner-up (Levin, 2000)	99.4	97.5	84.5	11.8	7.3	91.5	0.6
Multi-classifier (Sabhnani and Serpen, 2003)	n/r	97.3	88.7	29.8	9.6	N/A	N/A
Association rule (Xuren et al., 2006)	99.5	96.8	74.9	3.8	7.9	N/A	N/A

of 41 features. First, each feature value was normalized to the range of [0, 1], by dividing their maximum value, and removing the non-important features, i.e., leaving only the important features, as listed in Table 1, for each SVM classifier. Then the four CF trees for these four kinds of attacks and one CF tree for normal traffic were constructed. The centroids of the leaf node entries (one entry represents one cluster) in the CF trees were taken as instances for the SVM training set. The implementation was coded by Java language, and C-SCV with RBF kernel of LIBSVM (version 2.84, Java version) (http://www.csie.ntu.edu.tw/~cjlin/libsvm) was applied.

The best performance of this system, in terms of accuracy, was 95.72%, with only a 0.73% false positive rate. The detailed values are shown in Table 2. This result was obtained as the CF tree's radius threshold T set to 0.2 and branching factor B set to 100. The number of instances could be adjusted by these two parameters so that: (1) it is not too small (lose of information), or too large (training time is too long or unending), and (2) a balance could be retained between the normal instances and attack instances. The original numbers of instance for each kind of attack and the numbers of instances by CF trees with different threshold T are given in Table 3. See the DoS column for example. In the original dataset, there were 3,883,370 instances. However, by this CF tree, with radius threshold $T = 0.2$, the number of leaf node entries was 271, each entry represented a cluster, and the centroid of a cluster denoted an instance. The system suffered from both U2R and R2L attacks because the numbers of instances for these two attacks were too small in the original KDD Cup 1999 dataset.

As shown in Table 4, compared to the KDD Cup 1999 winner's system and other researches, which also used the KDD Cup 1999 dataset, this system provided the best detection rate for DoS and Probe attacks. The ESC-IDS (Toosi & Kahani, 2007) showed the best detection rate for the R2L attack, and Multi-classifier (Sabhnani & Serpen, 2003) showed the best detection rate for the U2R attack. Although the KDD'99 winner (Pfahringer, 2000) and the KDD'99 runner-up (Levin, 2000) both showed better performance than that

Table 6
Snmpgetattack being the same as normal connection in the KDD Cup 1999 (Bouzida and Cuppens, 2006).

0, udp, snmp, SF, 105, 146, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0.00, 0.00, 0.00, 0.00, 1.00, 0.00, 0.00, 255, 254, 1.00, 0.01, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, snmpgetattack
0, udp, snmp, SF, 105, 146, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0.00, 0.00, 0.00, 0.00, 1.00, 0.00, 0.00, 255, 254, 1.00, 0.01, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, normal

Table A1
Features description of KDD Cup 1999 data set.

No.	Feature name	Description	Type
1	Duration	Length of the connection (second)	Continuous
2	Protocol_type	Type of protocol, e.g. tcp, udp, etc.	Discrete
3	Service	Network service on the destination, e.g., http, telnet, etc.	Discrete
4	Flag	Normal or error status of the connection	Discrete
5	Src_bytes	Number of data bytes from source to destination	Continuous
6	Dst_bytes	Number of data bytes from destination to source	Continuous
7	Land	1 if connection is from/to the same host/port; 0 otherwise	Discrete
8	Wrong_fragment	Number of "wrong" fragments	Continuous
9	Urgent	Number of urgent packets	Discrete
10	Hot	Number of "hot" indicators	Discrete
11	Num_failed_logins	Number of failed login attempts	Discrete
12	Logged_in	1 if successfully logged in; 0 otherwise	Discrete
13	Num_compromised	Number of compromised condition	Discrete
14	Root_shell	1 if root shell is obtained; 0 otherwise	Discrete
15	Su_attempted	1 if "su root" command attempted; 0 otherwise	Discrete
16	Num_root	Number of "root" accesses	Discrete
17	Num_file_creations	Number of file creation operations	Discrete
18	Num_shells	Number of shell prompts	Discrete
19	Num_access_files	Number of operations on access control files	Discrete
20	Num_outbound_cmds	Number of outbound commands in an ftp session	Discrete
21	Is_host_login	1 if the login belongs to the "hot" list; 0 otherwise	Discrete
22	Is_guest_login	1 if the login is a "guest"login; 0 otherwise	Discrete
23	Count	number of connections to the same host as the current connection in the past two seconds	Discrete
24	Srv_count	Number of connections to the same service as the current connection in the past two seconds	Discrete
25	Serror_rate	% of connections that have "SYN" errors	Discrete
26	Srv_serror_rate	% of connections that have "SYN" errors	Discrete
27	Rerror_rate	% of connections that have "REJ" errors	Discrete
28	Srv_rerror_rate	% of connections that have "REJ" errors	Discrete
29	Same_srv_rate	% of connections to the same services	Discrete
30	Diff_srv_rate	% of connections to different services	Discrete
31	Srv_diff_host_rate	% of connections to different hosts	Discrete
32	Dst_host_count	Count for destination host	Discrete
33	Dst_host_srv_count	Srv_count for destination host	Discrete
34	Dst_host_same_srv_rate	Same_srv_rate for destination host	Discrete
35	Dst_host_diff_srv_rate	Diff_srv_rate for destination host	Discrete
36	Dst_host_same_src_port_rate	Same_src_port_rate for destination host	Discrete
37	Dst_host_srv_diff_host_rate	Diff_host_rate for destination host	Discrete
38	Dst_host_serror_rate	Serror_rate for destination host	Discrete
39	Dst_host_srv_serror_rate	Srv_serror_rate for destination host	Discrete
40	Dst_host_rerror_rate	Rerror_rate for destination host	Discrete
41	Dst_host_srv_rerror_rate	Srv_serror_rate for destination host	Discrete

of this study in terms of false positive rates, the difference was insignificant. Overall, in terms of accuracy, this system could achieve the best performance.

Some intrusion detection researches used only part of the KDD Cup 1999 data set for evaluation, which makes the comparison difficult. (Abraham, Grosan, & Martin-Vide, 2007) used three variants of genetic programming techniques, and reported that the method could achieve a detection rate of over 99% for all kinds of attack and normal data. However, since (Abraham et al., 2007) only used part of the KDD Cup 1999 data set for the training and test set, it was unfair to compare between the result of Abraham et al. (2007) and this ssystem. Moreover, unlike this system, which is built to detect both known and previously unseen attacks, the system proposed by Abraham et al. (2007) could only detect known attacks.

The KDD Cup 1999 dataset is a very famous dataset, however, there are many new types of attack instances, which had never appeared in the training set. There were 18, 729 records of various new attacks appearing only in the test data, which makes an intrusion detection system trained by a training set hard to achieve good performance by test set. As shown in Table 5, the IDS detection rate of this study for new attacks was only 39.04%, and the worst detection was on new R2L attacks. None of *snmpgetattack*, *xsnoop*, or *xlock* attacks were detected, and only one instance of *snmpguess* was detected as an attack. It shows that this system could not make the distinction of a new R2L instance from normal instances. In fact, as shown in Table 6 (Bouzida & Cuppens, 2006), the features of *snmpgetattack* and normal connection were the same, which makes it impossible for the system to differentiate. The *snmpgetattack* traffic was recognized as normal because the attacker guessed the password, and logged in as a non-malicious user.

5. Conclusion

Many researches concerning NIDSs applied SVMs because SVMs are well known for their generalization performances. In this study, in addition to a simple feature selection method, it proposed an SVM-based network intrusion detection system with BIRCH hierarchical clustering for data preprocessing. The BIRCH hierarchical clustering could provide highly qualified, abstracted and reduced datasets, instead of original large dataset, to the SVM training. Thus, in addition to a significant reduction of the training time, the resultant SVM classifiers showed better performance than the SVM classifiers using the originally redundant dataset.

According to the experiment on the KDD Cup 1999, the proposed system could reach an accuracy of 95.72% with a false positive rate of 0.7%. Compared with other NIDSs that also applied KDD Cup 1999 as a dataset, this system showed superior performance in DoS and Probe attacks, though it was not the best for U2R and R2L attacks. However, in terms of accuracy, the proposed system could obtain the best performance at 95.72%. This experiment was performed on the whole KDD Cup 1999 dataset without sampling. Some new attack instances in the test dataset, which never appeared in training, could also be detected by this system.

Acknowledgement

This work was supported in part by the National Science Council under Contract No. NSC 96-2918-I-011-002, 96-2221-E-011-022, 95-2221-E-011-032-MY3.

Appendix A. Appendix

See Table A1.

References

Abraham, A., Grosan, C., & Martin-Vide, C. (2007). Evolutionary design of intrusion detection programs. *International Journal of Network Security, 4*(3), 328–339.

Bouzida, Y., & Cuppens, F. (2006). Neural networks vs. decision trees for intrusion detection. <http://www.rennes.enst-bretagne.fr/~fcuppens/articles/monam06.pdf>.

Chimphlee, W., Abdullah, A. H., Md Sap, M. N., Srinoy, S., & Chimphlee, S. (2006) Anomaly-based intrusion detection using fuzzy rough clustering. In *Proceedings of the international conference on hybrid information technology (ICHIT'06).*

Guha, S., Rastogi, R., & Shim, K. (1999). Rock: A robust clustering algorithm for categorical attributes. In *Proceedings of the international conference on data engineering (ICDE'99)* (pp. 512–521).

Guha, S., Rastogi, R., & Shim, K. (1998). Cure: An efficient clustering algorithm for large databases. In *Proceedings of the ACM SIGMOD (SIGMOD'98)* (pp. 73–84).

Hsu, C. -W., Chang, C. -C., & Lin, C. -J., (xxxx). *A practical guide to support vector classification*. <http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf>.

Karypis, G., Han, E.-H., & Kumar, V. (1999). Chameleon: A hierarchical clustering algorithm using dynamic modeling. *Computer, 32*, 68–75.

KDD Cup, (1999). *Intrusion detection data set*. <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>.

Khan, L., Awad, M., & Thuraisingham, B. (2007). A new intrusion detection system using support vector machines and hierarchical clustering. *The International Journal on Very Large Data Bases, 16*(4), 507–521.

Levin, I. (2000). KDD-99 classifier learning contest LLSoft's results overview. *SIGKDD Explorations, 1*(2), 67–75.

Novikov, D., Yampolskiy, R. V., & Reznik, L. (2006). Anomaly detection based intrusion detection. In *Proceedings of the third international conference on information technology: New generations (ITNG'06).*

Pfahringer, B. (2000). Winning the KDD99 classification cup: Bagged boosting. *SIGKDD Explorations, 1*(2), 65–66.

Sabhnani, M. R., & Serpen, G. (2003). Application of machine learning algorithms to KDD intrusion detection dataset with in misuse detection context. In *Proceedings of the international conference on machine learning: Models, technologies, and applications* (pp. 209–215).

Toosi, A. N., & Kahani, M. (2007). A new approach to intrusion detection based on an evolutionary soft computing model using neuro-fuzzy classifiers. *Computer Communications, 30*, 2201–2212.

Vapnik, V. (1995). *The nature of statistical learning theory*. New York, NY: Springer-Verlag.

Vigna, G., Robertson, W., Kher, V., & Kemmerer, R. A. (2003). A stateful intrusion detection system for world-wide web servers. In *Proceedings of the 19th annual computer security applications conference, December 8–12.*

Xuren, W., Famei, H., & Rongsheng, X. (2006). Modeling intrusion detection system by discovering association rule in rough set theory framework. In *Proceedings of the international conference on computational intelligence for modelling control and automation, and international conference on intelligent agents*. Web Technologies and Internet Commerce (CIMCA-IAWTIC'06).

Yu, H., Yang, J., Han, J., & Li, X., (2003). Classifying large data sets using SVM with hierarchical clusters. In *Proceedings of the international conference on knowledge discovery in databases (KDD'03).*

Yu, Z., Tsai, J. J. P., & Weigert, T. (2007). An automatically tuning intrusion detection system. *IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics, 37*(2), 373–384.

Zhang, T., Ramakrishnan, R., & Livny, M., (1996). BIRCH: An efficient data clustering method for very large databases. In *Proceedings of the ACM SIGMOD (SIGMOD'96)* (pp. 103–114).