The VLDB Journal (2007) 16: 507–521
DOI 10.1007/s00778-006-0002-5

REGULAR PAPER

Latifur Khan · Mamoun Awad · Bhavani Thuraisingham

A new intrusion detection system using support vector machines and hierarchical clustering

Received: 13 January 2005 / Revised: 10 June 2005 / Accepted: 21 July 2005 / Published online: 31 August 2006

Abstract Whenever an intrusion occurs, the security and value of a computer system is compromised. Network-based attacks make it difficult for legitimate users to access various network services by purposely occupying or sabotaging network resources and services. This can be done by sending large amounts of network traffic, exploiting well-known faults in networking services, and by overloading network hosts. Intrusion Detection attempts to detect computer attacks by examining various data records observed in processes on the network and it is split into two groups, anomaly detection systems and misuse detection systems. Anomaly detection is an attempt to search for malicious behavior that deviates from established normal patterns. Misuse detection is used to identify intrusions that match known attack scenarios. Our interest here is in anomaly detection and our proposed method is a scalable solution for detecting network-based anomalies. We use Support Vector Machines (SVM) for classification. The SVM is one of the most successful classification algorithms in the data mining area, but its long training time limits its use. This paper presents a study for enhancing the training time of SVM, specifically when dealing with large data sets, using hierarchical clustering analysis. We use the Dynamically Growing Self-Organizing Tree (DGSOT) algorithm for clustering because it has proved to overcome the drawbacks of traditional hierarchical clustering algorithms (e.g., hierarchical agglomerative clustering). Clustering analysis helps find the boundary points, which are the most qualified data points to train SVM, between two classes. We present a new approach of combination of SVM and DGSOT, which starts with an initial training set and expands it gradually using the clustering structure produced by the DGSOT algorithm. We compare our approach with the Rocchio Bundling technique and random selection in terms of accuracy loss and training time gain using a single benchmark real data set. We show that our proposed variations contribute significantly in improving the training process of SVM with high generalization accuracy and outperform the Rocchio Bundling technique.

L. Khan (✉) · M. Awad · B. Thuraisingham
University of Texas at Dallas, Dallas, TX, USA.
E-mail: {lkhan, maa013600, bxt043000}@utdallas.edu

1 Introduction

Since the tragic events of September 11, 2001, insuring the integrity of computer networks, both in relation to security and with regard to the institutional life of the nation in general, is a growing concern. Security and defense networks, proprietary research, intellectual property, data-based market mechanisms which depend on unimpeded and undistorted access, can all be severely compromised by intrusions. We need to find the best way to protect these systems.

An intrusion can be defined [3, 9] as "any set of actions that attempts to compromise the integrity, confidentiality, or availability of a resource". User authentication (e.g., using passwords or biometrics), avoiding programming errors, and information protection (e.g., encryption) have all been used to protect computer systems. As systems become more complex, there are always exploitable weaknesses due to design and programming errors, or through the use of various "socially engineered" penetration techniques. For example, exploitable "buffer overflow" still exists in some recent system software because of programming errors. Elements central to intrusion detection are resources to be protected in a target system, i.e., user accounts, file systems, system kernels, etc.; models that characterize the "normal" or "legitimate" behavior of these resources; techniques that compare the actual system activities with the established models identifying those that are "abnormal" or "intrusive". In pursuit of a secure system, different measures of system behavior have been proposed, on the basis of an ad hoc presumption that normalcy and anomaly (or illegitimacy) will be accurately manifested in the chosen set of system features.

Intrusion Detection attempts to detect computer attacks by examining various data records observed through processes on the same network. These attacks are split into two categories, host-based attacks [2, 3, 13] and network-based attacks [18, 26, 27]. Host-based attacks target a machine and try to gain access to privileged services or resources on that machine. Host-based detection usually uses routines that obtain system call data from an audit-process which tracks all system calls made on behalf of each user. Network-based attacks make it difficult for legitimate users to access various network services by purposely occupying or sabotaging network resources and services. This can be done by sending large amounts of network traffic, exploiting well-known faults in networking services, overloading network hosts, etc. Network-based attack detection uses network traffic data (i.e., tcpdump) to look at traffic addressed to the machines being monitored. Intrusion detection systems are split into two groups, anomaly detection systems and misuse detection systems. Anomaly detection is the attempt to identify malicious traffic based on deviations from established normal network traffic patterns [27, 28]. Misuse detection is the ability to identify intrusions based on a known pattern for the malicious activity [18, 26]. These known patterns are referred to as signatures. Anomaly detection is capable of catching new attacks. However, new legitimate behavior can also be falsely identified as an attack, resulting in a false positive. Our research will focus on network level systems. The problem with current state-of-the-art is to reduce false negative and false positive rate (i.e., we wish to minimize "abnormal normal" behavior). At the same time, a real-time intrusion detection system should be considered. It is difficult to achieve both.

The SVM is one of the most successful classification algorithms in the data mining area, but its *long training time limits its use*. Many applications, such as Data Mining and Bio-Informatics, require the processing of huge data sets. The training time of SVM is a serious obstacle in the processing of such data sets. According to [41], it would take years to train SVM on a data set consisting of one million records. Many proposals have been submitted to enhance SVM in order to increase its training performance [1, 8], either through random selection or approximation of the marginal classifier [14]. However, such approaches are still not feasible with large data sets where even multiple scans of entire data set are too expensive to perform, or result in the loss through over-simplification of any benefit to be gained through the use of SVM [41].

This paper proposes a new approach for enhancing the training process of SVM when dealing with large training data sets. It is based on the combination of SVM and clustering analysis. The idea is as follows: SVM computes the maximal margin separating data points; hence, only those patterns closest to the margin can affect the computations of that margin, while other points can be discarded without affecting the final result. Those points lying close to the margin are called support vectors (see Sect. 3 for more details). We try to approximate these points by applying clustering analysis.

In general, using hierarchical clustering analysis based on dynamically growing self-organizing tree (DGSOT) involves expensive computations, especially if the set of training data is large. However, in our approach, we control the growth of the hierarchical tree by allowing tree nodes (support vector nodes) close to the marginal area to grow, while halting distant ones. Therefore, the computations of SVM and further clustering analysis will be reduced dramatically. Also, to avoid the cost of computations involved in clustering analysis, we train SVM on the nodes of the tree after each phase/iteration, in which few nodes are added to the tree. Each iteration involves growing the hierarchical tree by adding new children to the tree. This could cause a degradation of the accuracy of the resulting classifier. However, we use the support vector set as a priori knowledge to instruct the clustering algorithm to grow support vector nodes and to stop growing non-support vector nodes. By applying this procedure, the accuracy of the classifier improves and the size of the training set is kept to a minimum.

We report results here with one benchmark data set, the 1998 DARPA [29]. Also, we compare our approaches with the Rocchio Bundling algorithm, recently proposed for classifying documents by reducing the number of data points [37]. Note that the Rocchio Bundling method reduces the number of data points before feeding those data points as support vectors to SVM for training. On the other hand, our clustering approaches intertwined with SVM. We have observed that our approaches outperform Pure SVM and the Rocchio Bundling technique in terms of accuracy, false positive (FP) rate, false negative (FN) rate, and processing time.

The main contributions of this work are as follows: First, to reduce the training time of SVM, we propose a new support vector selection technique using clustering analysis. Here, we combine the clustering analysis and SVM training phases. Second, we show analytically the degree to which our approach is asympotatically quicker than pure SVM, and validate this claim with experimental results. Finally, we compare our approaches with random selection, and Rocchio Bundling on a benchmark data set, and demonstrate impressive results in terms of training time, FP rate, FN rate, and accuracy.

The paper is organized as follows: In Sect. 2, we discuss the related work. In Sect. 3, we present our approach employing clustering algorithms. In Sect. 4, we present motivation behind our clustering algorithm. In Sect. 5, experimental setup of our systems is described. In Sect. 6, we present Rocchio Bundling algorithm which has been proposed recently. In Sect. 7, we present experimental results of our approaches, pure SVM, random selection, and Rocchio Bundling algorithms. In Sect. 8, we summarize the paper and outline some future research.

2 Related work

Here, first, we present related work relevant to intrusion detection (extensive survey can be found in [3, 9]), and next, we present related work for the reduction of training time of

SVM. In particular, we will present various clustering techniques as data reduction mechanisms.

With regard to intrusion detection, as noted earlier, there are two different approaches to intrusion detection system (IDS): misuse detection and anomaly detection. Misuse detection is the ability to identify intrusions based on a known pattern for the malicious activity. These known patterns are referred to as signatures. The second approach, anomaly detection, is the attempt to identify malicious traffic based on deviations from established normal network traffic patterns. *"A State Transition Analysis Tool for Intrusion Detection" (STAT)* [18] and *"Intrusion Detection in Our Time" (IDIOT)* [22] are misuse detection systems that use the signatures of known attacks. Lee et al. [24] propose a data mining framework for intrusion detection which is misuse detection. Their goal is to automatically generate misuse detection signatures from classified network traffic. Anomaly detection [39] is capable of catching new attacks. However, new legitimate behavior can also be falsely identified as an attack, resulting in a false positive. In recent years, there have been several learning-based or data mining-based research [33] efforts in intrusion detection.

Instead of network level data, researchers may also concentrate on user command-level data [2, 5, 6, 12, 13, 32]. For example, *"Anomaly-Based Data Mining for Intrusions," ADMIT* [32] is a user profile dependent, temporal sequence clustering based real-time intrusion detection system with host-based data collection and processing. In this effort *"Next Generation Intrusion Detection Expert System"*, NIDES [2] and *"Event Monitoring Enabling Responses to Anomalous Live Disturbances," EMERALD* [12] create user profile based on statistical method. A few other groups [7, 31] advocate the use of neural networks in intrusion detection. Some of them rely on a keyword count for a misuse detection system, along with neural networks. Attack specific keyword counts in network traffic are fed as neural network input. Ghosh et al. [15] use a neural network to extract program behavior profiles instead of user behavior profiles, and later compare these with the current system behavior. Self-organizing maps (SOM) and support vector machine [8, 28, 40] have also been used as anomaly intrusion detectors. An SOM is used to cluster and then graphically display the network data for the user to determine which clusters contained attacks [16]. SVM is also used for an intrusion detection system. Wang et al. [40] use "one class SVM" based on one set of examples belonging to a particular class and no negative examples rather than using positive and negative examples. Neither of these approaches addresses the reduction of the training time of SVM, which is what prohibits real-time usage of these approaches.

With regard to the training time of SVM, random sampling has been used to enhance the training of SVM [38]. Sub-sampling speeds up a classifier by randomly removing training points. Balacazar et al. [4] use random sampling successfully to train SVM in many applications in the data mining field. Shih et al. [37] use sub-sampling in classification using a Rocchio Algorithm along with other data reduction techniques. Sub-sampling surprisingly has led to an accurate classification in their experiments with several data sets. However, Yu et al. [41] show that random sampling could hurt the training process of SVM, especially when the probability distribution of training and testing data were different.

Yu et al. [41] use the idea of clustering, using BIRCH [42], to fit a huge data set in the computer memory and train SVM on the tree's nodes. The training process of SVM starts at the end of building the hierarchical tree causing expensive computations, especially when the data cannot fit in the computer memory or the data is not distributed uniformly. The main objective of the clustering algorithm is to reduce the expensive disk access cost; on the other hand, our main focus is to approximate support vectors in advance. Furthermore, our use of clustering analysis goes in parallel with training SVM, i.e., we do not wait until the end of building the hierarchical tree in order to start training SVM.

It is a legitimate question to ask why we use hierarchical clustering rather than partitioning/flat clustering (e.g., K-means). Partitioning/flat clustering directly seeks a partition of the data which optimizes a predefined numerical measure. In partitioning clustering, the number of clusters is predefined, and determining the optimal number of clusters may involve more computational cost than that of the clustering itself. Furthermore, a priori knowledge may be necessary for initialization and the avoidance of local minima. Hierarchical clustering, on the other hand, does not require a predefined number of clusters or a priori knowledge. Hence, we favor hierarchical clustering over flat clustering. Hierarchical clustering employs a process of successively merging smaller clusters into larger ones (agglomerative, bottom-up) [35], or successively splitting larger clusters (divisive, top-down) [11, 20, 21]. Agglomerative algorithms are more expensive than divisive algorithms and, since we need a clustering algorithm that grows in a top-down fashion and is computationally less expensive, we favor the use of divisive hierarchal clustering over agglomerative algorithms.

3 SVM with clustering for training

In this section, we first present the basics of SVM. Next, we present how clustering can be used as a data reduction technique to find support vectors.

3.1 SVM basics

SVM are learning systems that use a hypothesis space of linear functions in a high dimensional feature space, trained with a learning algorithm from optimization theory. SVM is based on the idea of a hyper-plane classifier, or linearly separability. Suppose we have N training data points $\{(x_1, y_1), (x_2,y_2), (x_3, y_3), \ldots, (x_N, y_N)\}$, where $x_i \in R^d$ and $y_i \in \{+1, -1\}$. Consider a hyper-plane defined by (w, b), where w is a weight vector and b is a bias. Details of SVM can be

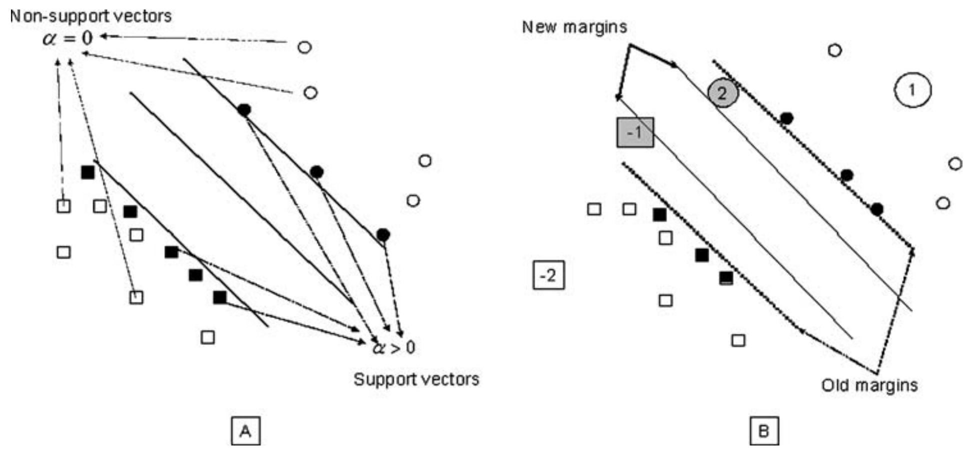

Fig. 1 **A** Value of α_i for support vectors and non-support vectors. **B** The effect of adding new data points on the margins

found in [34]. We can classify a new object x with

$$f(x) = \text{sign}(w \cdot x + b) = \text{sign}\left(\sum_i^N \alpha_i y_i (x_i \cdot x) + b\right) \quad (1)$$

Note that the training vectors x_i occur only in the form of a dot product; there is a Lagrangian multiplier α_i for each training point. The Lagrangian multiplier values α_i reflect the importance of each data point. When the maximal margin hyper-plane is found, only points that lie closest to the hyper-plane will have $\alpha_i > 0$ and these points are called support vectors. All other points will have $\alpha_i = 0$ (see Fig. 1A. This means that only those points that lie closest to the hyper-plane give the representation of the hypothesis/classifier. These most important data points serve as support vectors. Their values can also be used to give an independent boundary with regard to the reliability of the hypothesis/classifier. Figure 1A shows two classes and their boundaries, i.e., margins. The support vectors are represented by solid objects, while the empty objects are non-support vectors. Notice that the margins are only affected by the support vectors, i.e., if we remove or add empty objects, the margins will not change. Meanwhile any change in the solid objects, either adding or removing objects, could change the margins. Figure 1B shows the effects of adding objects in the margin area. As we can see, adding or removing objects far from the margins, e.g., data point 1 or −2, does not change the margins. However, adding and/or removing objects near the margins, e.g., data point 2 and/or −1, has created new margins.

3.2 Data reduction using hierarchy

Our approach for enhancing the training process of SVM is based on the combination of clustering and SVM to find relevant support vectors. For this, we present an approach, namely, Clustering Tree based on SVM, CT-SVM.

3.2.1 Clustering tree based on SVM, CT-SVM

In this approach, we build a hierarchical clustering tree for each class in the data set (for simplicity and without loss of generality, we assume binary classification) using the DGSOT algorithm. The DGSOT algorithm, top-down clustering, builds the hierarchical tree iteratively in several epochs. After each epoch, new nodes are added to the tree based on a learning process. To avoid the computation overhead of building the tree, we do not build the entire hierarchical trees. Instead, after each epoch we train SVM on the nodes of both trees. We use the support vectors of the classifier as prior knowledge for the succeeding epoch in order to control the tree growth. Specifically, support vectors are allowed to grow, while non-support vectors are stopped. This has the impact of adding nodes in the boundary areas between the two classes, while eliminating distant nodes from the boundaries.

Figure 2 outlines the steps of this approach. First, assuming binary classification, we generate a hierarchical tree for each class in the data set. Initially, we allow the two trees to grow until a certain size of the trees is reached. Basically, we want to start with a reasonable number of nodes. First, if a tree exhibits convergence earlier (i.e., fewer number of nodes), one option is to train SVM with these existing nodes. If the result is unsatisfactory, we will adjust the threshold (profile and radius thresholds) (see Sect. 4). Reducing thresholds may increase number of clusters and nodes. Second, we will train SVM on the nodes of the trees, and compute the support vectors. Third, on the basis of stopping criteria, we either stop the algorithm or continue growing the hierarchical trees. In the case of growing the tree, we use prior knowledge, which consists of the computed support vector nodes, to instruct the DGSOT algorithm to grow support vector nodes, while non-support vector nodes are not grown. By growing a node in the DGSOT algorithm, we mean that we create two children nodes for each support vector node [25]. They imply prior knowledge to be used in the next iteration of our algorithm, especially to direct the

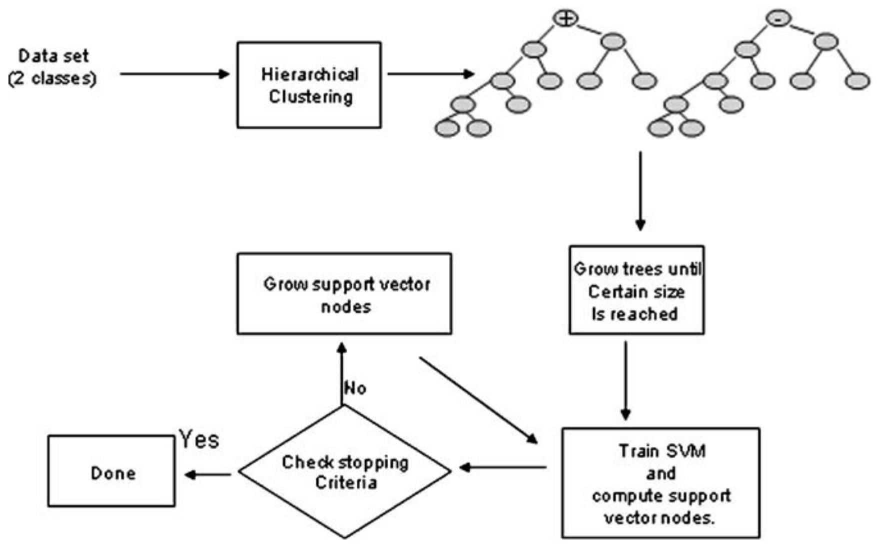
Data set
(2 classes)
Hierarchical Clustering
Grow trees until Certain size Is reached
Grow support vector nodes
No
Yes
Done
Check stopping Criteria
Train SVM and compute support vector nodes.

Fig. 2 Flow diagram of clustering trees

growth of the hierarchical trees. This process has the impact of growing the tree only in the boundary area between the two classes, while stopping the tree from growing in other areas. Hence, we save extensive computations that would have been carried out without any purpose or utility.

3.2.2 Stopping criteria

There are several ways to set the stopping criteria. For example, we can stop at a certain size/level of tree, upon reaching a certain number of nodes/support vectors, and/or when a certain accuracy level is attained. For accuracy level, we can stop the algorithm if the generalization accuracy over a validation set exceeds a specific value (say 98%). For this accuracy estimation, support vectors will be tested with the existing training set based on proximity. In our implementation, we adopt the second strategy (i.e., a certain number of support vectors) so that we can compare our procedure with the Rocchio Bundling algorithm which reduces the data set into a specific size.

One subtlety of using clustering analysis is that clustering might take long time to complete, which undoes any benefits from improvements in SVM. Our strategy here is that we do not wait until DGSOT finishes. Instead, after each epoch/iteration of the DGSOT algorithm, we train the SVM on the generated nodes. After each training process we can control the growth of the hierarchical tree from top to bottom because non-support vector nodes will be stopped from growing, and only support vector nodes will be allowed to grow. Figure 3 shows the growth of one of the hierarchical trees during this approach. The bold nodes represent the support vector nodes. Notice that nodes 1, 2, 3, 5, 6, and 9 are allowed to expand because they are support vector nodes. Meanwhile, we stop nodes 4, 8, 7, and 10 from growing because they are not support vector nodes.

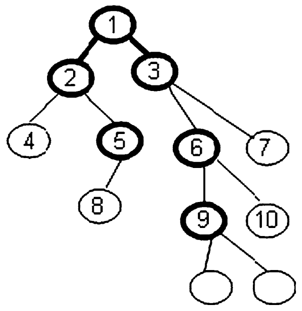

Fig. 3 Only support vectors are allowed to grow

Growing the tree is very important in order to increase the number of points in the data set so as to obtain a more accurate classifier. Figure 4 shows an illustrative example of growing the tree up to a certain size/level and in which we have the training set (+3, +4, +5, +6 + 7, −3, −4, −5, −6, −7). The dashed nodes represent the clusters' references which are not support vectors. The bold nodes represent the support vector references. Hence, we add the children of the support vector nodes +4 and +7 to the training set. The same applies to the support vector nodes −5 and −7. The new data set now is (+3, +11, +12, +5, +6, +14, +15, −3, −4, −10, −11, −6, −12, −13). Notice that expanded nodes, such as +4 and +7, are excluded because their children nodes are added to the training set. Part A of Fig. 5 shows the layout of those clusters around the boundaries, and Part B of Fig. 5 shows the effect of growing the trees on the classifiers. Note that adding new nodes in the boundary area of the old classifier, represented by dashed lines, is corrected and the new classifier, solid lines, is more accurate now than the old one. By doing so, the classifier is adjusted accordingly, creating a more accurate classifier.

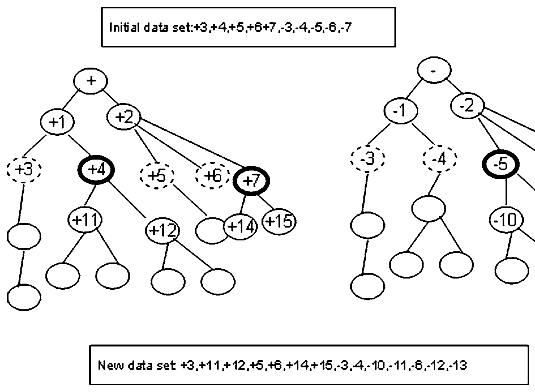
Initial data set:+3,+4,+5,+6+7,-3,-4,-5,-6,-7
+
+1
+2
+3
+4
+5
+6
+7
+11
+12
+14
+15
-
-1
-2
-3
-4
-5
-6
-7
-10
-11
-12
-13
New data set +3,+11,+12,+5,+6,+14,+15,-3,-4,-10,-11,-6,-12,-13

Fig. 4 Updating the training set after growing SVM

4 Hierarchy construction using a dynamic growing self-organizing tree algorithm (DGSOT)

We would like to organize a set of data/records into a number of clusters where a cluster may contain more than one record. Furthermore, we would like to extend our hierarchy onto several levels. For this, several existing techniques are available, such as the hierarchical agglomerative clustering algorithm (HAC) [35], the self-organizing map (SOM) [21], the self-organizing tree algorithm (SOTA) [11], and so on. In this research, we will exploit a new algorithm, a new hierarchical, dynamically growing self-organizing tree (DGSOT) algorithm, to construct a hierarchy from top to bottom rather than bottom up, as in HAC. We have observed that this algorithm constructs a hierarchy with better precision and recall than a hierarchical agglomerative clustering algorithm [20, 25]. Hence, we believe that false positives and false negatives (similar to precision and recall area in IR) will be lowered using our new algorithm as compared to HAC. Furthermore, it works top-down fashion. We can stop tree growing earlier and it can be entirely mixed with SVM training.

The DGSOT is a tree structure self-organizing neural network. It is designed to discover the correct hierarchical structure in an underlying data set. The DGSOT grows in two directions: vertical and horizontal. In the direction of vertical growth, the DGSOT adds descendents. In the vertical growth of a node (x), only two children are added to the node. The need is to determine whether these two children, added to the node x, are adequate to represent the proper hierarchical structure of that node. In horizontal growth, we strive to determine the number of siblings of these two children needed to represent data associated with the node, x. Thus, the DGSOT chooses the right number of children (sub-clusters) of each node at each hierarchical level during the tree construction process. During each vertical and horizontal growth a learning process is invoked in which data will be distributed among newly created children of node, x (see Sect. 4.2). The pseudo code of DGSOT is shown in Fig. 6

In Fig. 6, at lines 1–4, initialization is done with only one node, the root node (see Fig. 7A). All input data belong to the root node and the reference vector of the root node is initialized with the centroid of the data. At lines 5–29, the vertical growing strategy is invoked. For this, first we select a victim leaf for expansion based on heterogeneity. Since, we have only one node which is both leaf and root, two children will be added (at lines 7–9). In CT-SVM case, at line 7, we need to do additional check. Recall that support vector nodes will be candidate for vertical expansion. The reference vector of each of these children will be initialized to the root's reference vector. Now, all input data associated with the root will be distributed between these children by invoking a learning strategy (see Sect. 4.2). The tree at this stage will be shown in Fig. 7B. Now, we need to determine the right number of children for the root by investigating horizontal growth (see Sect. 4.3; lines 16–28).

At line 19, we check whether the horizontal growing stop rule has been reached (see Sect. 4.3.1). If it has, we remove

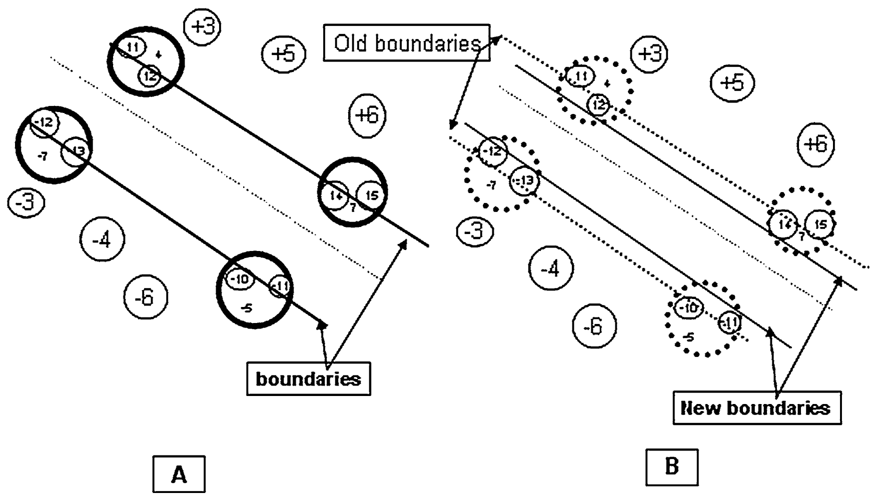

Fig. 5 Adjusting the classifiers as a result of growing the support vector nodes

```
1 /*Initialization*/
2       Create a tree has only one root node. The reference vector of the root node is
3       initialized with the centroid of the entire data and all data will be associated with
4       the root.
5 /*Vertical Growing*/
6 Do
7       For any leaf which is heterogeneous (See Section 4.1)
8       Changes the leaf to a node, x and create two descendent leaves. The reference
9       vector of a new leaf is initialized with the node's reference vector
10              /*Learning*/
11              Do
12                      For each input data of node, x
13                      Find winner (using KLD, see Section 4.4), and update
14                      reference vectors of winner and its neighborhood
15              While the relative error of the entire tree is less than error threshold, ε
16      /*Horizontal Growing*/
17      Do
18 If the horizontal growing stop rule (see Section 4.3.1) is unsatisfied
19      Add a child leaf to this node, x
20                      /*Learning*/
21                      Do
22                         For each input data of node, x
23                  Find winner (using KLD, see Section 4.4), and update
24                        reference vectors of winner and its neighborhood
25                      While the relative error of the entire tree is less than ε
26                      Else
27                                      Delete a child leaf to this node, x
28      While the node, x is unsatisfied with the horizontal growing stop rule
29 While there are more level/node necessary in the hierarchy
```

Fig. 6 DGSOT Algorithm

the last child and exit from the horizontal growth loop at line 28. If the stop rule has not been reached, we add one more child to the root, and the learning strategy is invoked to distribute the data of the root among all children. After adding two children (i.e., total four children) to the root one by one, we notice that horizontal stop rule has been reached and then we delete the last child from the root (at line 27). At this point the tree will be shown in Fig. 7C. After this, at line 29, we check whether or not we need to expand to another level. If the answer is yes, a vertical growing strategy is invoked. In addition, in CT-SVM case, to incorporate intrusion detection facility with SVM, stopping criterion will be checked (see Sect. 3.2.2). For this, first, we determine the heterogeneous leaf node (leaf node $N1$ in Fig. 7D) at line 7, add two children to this heterogeneous node; and distribute the data of $N1$ within its children using learning strategy. Next, horizontal growth is undertaken for this node, $N1$, and so on (see Fig. 7E).

4.1 Vertical growing

In DGSOT, a process of non-greedy vertical growth is used to determine the victim leaf node for further expansion in the vertical direction. During vertical growth any leaf whose heterogeneity is greater than that of a given threshold is changed to a node and 2 descendent leaves are created. The reference vectors of these leaves will be initialized by the parent. There are several ways to determine the heterogeneity of a leaf. One simple way is to use the number of data associated with a leaf to determine heterogeneity, what we call profile heterogeneity, T_P. This simple approach controls the number of data that appears in each leaf node. Here, data will be evenly distributed among the leaves. This may generate too many tiny fragmented clusters where data points are densely placed. Hence, the alternative is based on the radius of clusters, what we call radius heterogeneity. For this case, if a cluster radius exceeds a threshold, T_R we split the node. Hence, it is possible that data for a cluster may be sparsely distributed. In our implementation, we combine both approaches. We choose a node to split whose profile heterogeneity and radius heterogeneity exceed their thresholds simultaneously. Thus, we guarantee that clusters will be compact, while at the same time, on average, clusters will be in similar shapes. It is also possible that a node with too few points sparsely distributed may not be selected for expansion.

4.2 Learning process

In DGSOT, the learning process consists of a series of procedures to distribute all the data to leaves and update the reference vectors. Each procedure is called a *cycle*. Each *cycle* contains series of *epochs*. Each *epoch* consists of a presentation of input data with each presentation having two steps: finding the best match node and updating the reference vector. The input data is only compared to the leaf nodes bounded by a sub-tree based on KLD mechanism (see Sect. 4.4) in order to find the best match node, which is known as the *winner*. The leaf node, c, that has the mini-

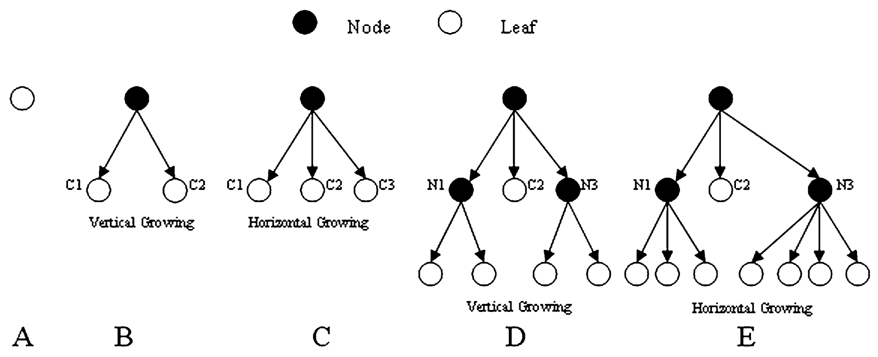

Fig. 7 Illustration of DGSOT algorithm

mum distance to the input data, x, is the best match node (i.e., winner).

$$c : \| x - n_c \| = \min\{\| x - n_i \|\} \tag{2}$$

After a *winner* c is found, the reference vectors of the winner and its neighborhood will be updated using the following function:

$$\Delta n_i = \phi(t) \times (x - n_i) \tag{3}$$

where $\phi(t)$ is the neighborhood function:

$$\phi(t) = \alpha \times \eta(t), \quad 0 < \eta(t) \le 1 \tag{4}$$

$\eta(t)$ is the learning rate function, α is the learning constant, and t is the time parameter. The convergence of the algorithm depends on a proper choice of α and $\eta(t)$. We would like to make sure that the reference vector of the winner will be updated quickly as compared to the reference vector of the sibling. For this, the $\eta(t)$ of winner will hold a larger value compared to the $\eta(t)$ of siblings. We define a Gaussian neighborhood function in such a manner that for a sibling node j, $\eta(t)$ will be calculated in the following way:

$$\eta(t) = \exp\left(- \frac{\| h(i, j) \|^2}{2\sigma^2(t)} \right) \tag{5}$$

where σ gives the width of the Gaussian kernel function, and $h(i, j)$ is the number of hops between the winner i and node j. Here the number of hops represents the length of the shortest acyclic path that connects two sibling nodes. Figure 8 shows the path that connects two sibling nodes assuming that node 4 is the winner. It also shows the distance between each sibling. Here, two siblings are any two nodes that have a common parent. Notice that from Eq. (5), the learning rate to update node 9 is less than the learning rate of updating node 8 because the length of the path, from the winner node 4 to node 8, is shorter. Therefore, the closest sibling of a winner will receive a higher value $\eta(t)$ in relation to the most distant sibling node.

The Error of the tree, which is defined as the summation of the distance of each input data to the corresponding *winner*, is used to monitor the convergence of the learning process. A learning process is converged when the relative increase of Error of the tree falls below a given threshold for average distortion

$$\left| \frac{\text{Error}_{t+1} - \text{Error}_t}{\text{Error}_t} \right| < \varepsilon \tag{6}$$

It is easier to avoid over training the tree in the early stages by controlling the value of the threshold during training.

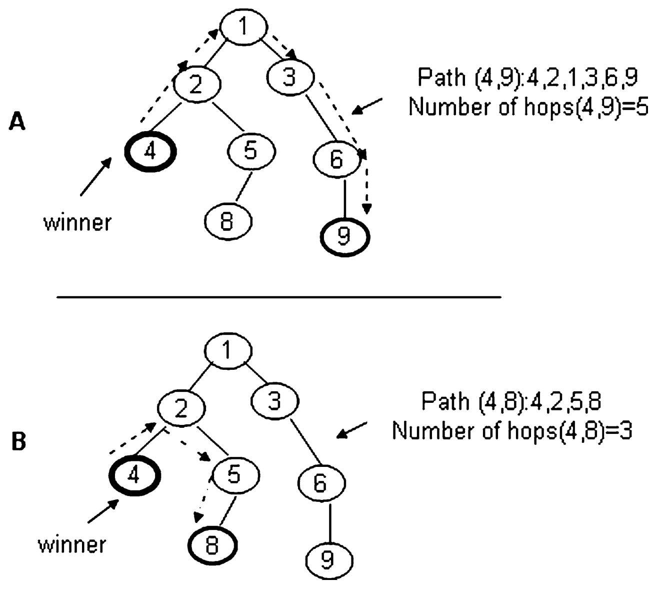

Fig. 8 The Hop-Distance between siblings

4.3 Horizontal growing

In each stage of vertical growth, only two leaves are created for a growing node. In each stage of horizontal growth, the DGSOT tries to find an optimal number of the leaf nodes (sub-cluster) of a node to represent the clusters in each node's expansion phase. Therefore, DGSOT adopts a dynamically growing scheme in horizontal growing stage. Recall that in vertical growing, a leaf with most heterogeneous will be determined, and two children will be created to this heterogeneous node. Now, in horizontal growing, a new child (sub-cluster) is added to the existing children of this heterogeneous node. This process continues until a certain stopping rule is reached. Once the stopping rule (see Sect. 4.3.1) is reached, the number of children node is optimized. After each addition/deletion of a node, a process of learning takes place (see Sect. 4.2).

4.3.1 Stopping rule for horizontal growing

To stop the horizontal growth of a node, we need to know the total number of clusters/children of that node. For this, we can apply the cluster validation approach. Since the DGSOT algorithm tries to optimize the number of clusters for a node in each expansion phase, cluster validation is used heavily. Therefore, the validation algorithms used in DGSOT must have a light computational cost and must be easily evaluated. A simple method is suggested for the DGSOT here, the measures of average distortion. However, cluster scattering measure [30] can be used to minimize the intra-cluster distance and maximize inter-cluster distance.

4.3.2 Average distortion (AD)

AD is used to minimize the intra-cluster distance. The average distortion of a sub-tree with j children is defined as

$$\mathrm{AD}_j = \frac{1}{N}\sum_{i=1}^{N} \mid d(x_i, n_k) \mid^2 \tag{7}$$

where N is the total number of input data assigned in the sub-tree, and n_k is the reference vector of the winner of input data x_i. AD_j is the average distance between input data and its winner. During DGSOT learning, the total distortion is already calculated and the AD measure is easily computed after the learning process is finished. If AD versus the number of clusters is plotted, the curve is monotonically decreasing. There will be a much smaller drop after the number of clusters exceeds the "true" number of clusters, because once we have crossed this point we will be adding more clusters simply to partitions within rather than between true clusters. Thus, the AD can be used as a criterion for choosing the optimal number of clusters. In the horizontal growth phase, if the relative value of AD after adding a new sibling is less than a threshold ε (Eq. (8)), then the new sibling will be deleted and horizontal growth will stop:

$$\left|\frac{\mathrm{AD}_{j+1} - \mathrm{AD}_j}{\mathrm{AD}_j}\right| < \varepsilon \tag{8}$$

where j is the number of children in a sub-tree and ε is a small value, generally less than 0.1.

4.4 K-level up distribution (KLD)

Clustering in a self-organizing neural network is distortion-based competitive learning. The nearest neighbor rule is used to make the clustering decision. In SOTA, data associated with the parent node will only be distributed between its children. If data are incorrectly clustered in the early stage, these errors cannot be corrected in the later learning process. To improve the cluster result, we propose a new distribution approach called K-level up distribution (KLD). Data associated with a parent node will be distributed not only to its children leaves but also to its neighboring leaves. The following is the KLD strategy:

1 For a selected node, its K level ancestor node is determined.
2 The sub-tree rooted by the ancestor node is determined.
3 Data assigned to the selected node will be distributed among all leaves of the sub-tree.

For example, Fig. 9 shows the scope of $K = 1$. Now, the data associated with node M needs to be distributed to the newly created leaves. For $K = 1$, the immediate ancestor of M will be determined, which is node A. The data associated with node M will be distributed to leaves B, C, D, and E of the sub-tree rooted by A. For each data, the winning leaf will be determined among B, C, D, and E using Eq. (2). Note that if $K = 0$, the data of node M will be distributed between leaves B and C.

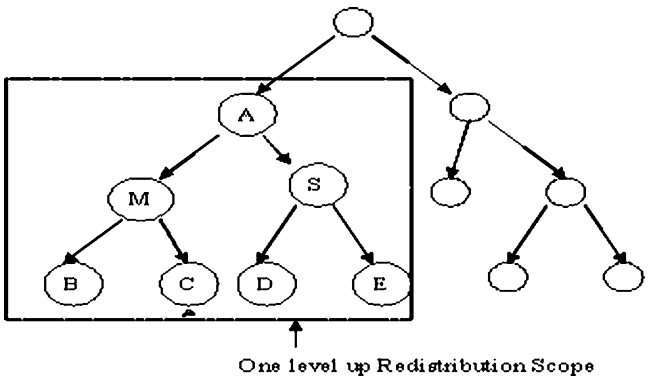

Fig. 9 One level up distribution scope ($K = 1$)

Note that vertical growing is a non-greedy expansion. Therefore, in vertical growing if heterogeneities of two leaves are greater than ϵ, then both will be the victim and expanded simultaneously.

5 Complexity and analysis

In this section, we present an analysis of the running time for the CT-SVM algorithm. Building the hierarchical tree is the most expensive computation in CT-SVM. However, we try to reduce this cost by growing the tree only in the marginal area. In this analysis, we consider the worst case scenario for building the hierarchical tree and the overhead of training SVM. Specifically, the time complexity of building the DGSOT tree is $O(\log_d M \times J \times N)$, where d is the branch factor of the DGSOT, J is the average number of learning iterations needed to expand the tree one more level, N is the size of the data set, and M is the number of nodes in the tree. Notice that, in the worse case scenario, M is close to the number of data points in the data set, N; hence, $O(\log_d M \times J \times N) \approx O(\log_d N \times J \times N)$ (see Table 1 for symbols and definitions).

The key factor to determine the complexity of SVM is the size of the training data set. In the following, we try to approximate the size of the tree in iteration i, approximate the running time of iteration i, and compute the running time of the CT-SVM algorithm.

Let us assume $t(\mathrm{SVM}) = O(N^2)$ where $t(\Psi)$ is the training time of algorithm Ψ (notice that $t(\mathrm{SVM})$ is known to be at least quadratic to N and linear to the number of dimensions). Let support entries be the support vectors when the training data are entries in some nodes. Assume that r is the average rate of the number of the support entries among the training entries. Namely, $r=s/b$ where b is the average number of training entries and s is the average number of support vector entries among the training entries, e.g., $r = 0.01$ for $s = 10$ and $b = 1000$. Normally, $s \ll b$ and $0 < r \ll 1$ for standard SVM with large datasets [41].

For simplicity, we will consider the size of only one hierarchical tree. Therefore, at iteration i the number of nodes

Table 1 Symbols used in this paper

Symbols	Descriptions
DGSOT symbols	
$\eta(t)$	Learning rate function
α	Learning constant
$\phi(t)$	Learning function
n_i	Reference vector of leaf node i
$d(x_j, n_i)$	Distance between data x_j and leaf node i
ε	Threshold for average distortion
e_i	Average error of node i
T_{R}	Radius heterogeneity
T_{P}	Profile heterogeneity
Complexity and analysis symbols	
N	The size of the data set
M	The number of nodes in the tree
J	The average number of learning iterations needed to expand the tree one more level
D	The branch factor for the DGSOT
r	The average rate of the number of the support entries among the training entries
B	The average number of training entries in each node
$t(\Psi)$	Training time of algorithm Ψ
s	The average number of support vector entries among the training entries
N_i	The number of nodes for iteration i

is given as follows:

$$\begin{aligned} N_i &= b - s + bs - s^2 + \cdots + bs^{i-2} - s^{i-1} + bs^{i-1} \\ &= (b-s)(1 + s + s^2 + s^3 + \cdots + s^{i-2} + bs^{i-1}) \\ &= (b-s)\frac{s^{i-1}-1}{s-1} + bs^{i-1} \end{aligned} \tag{9}$$

where b is the average number of data points in a node, and s is the number of the support vectors among the data. Figure 10 illustrates the computation of N_i for iterations 1, 2, 3, and 4. The size of training set is b at the beginning (at iteration 1). In iteration 2, we will have s support vectors after training SVM for each node in b; hence, we expand the support vector nodes s and add bs nodes to the training set. We exclude from the training set the nodes we expand,

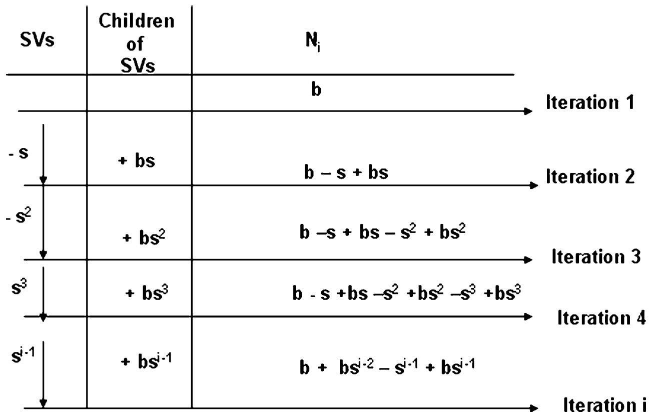

Fig. 10 Illustrative example of computing N_i for iterations 1, 2, 3, and 4

and since we have s support nodes expanded, hence we subtract s. In iteration 3, we expand s^2 support vectors, add bs^2 nodes, and subtract s^2. The left column of Fig. 10 shows that the number of support vectors after each iteration increases by the factor of s; hence, the number of children added also increases by the factor of s. Now, to approximate the running time of an iteration i, we, not only, compute the running time of the DGSOT algorithm iteration i, but also, we compute the running time of training SVM in that iteration. If we assume $t(\mathrm{SVM}) = \mathrm{O}(N^2)$, by approximating $s - 1 \approx s$ in the denominator, Eq. (9) becomes

$$\begin{aligned} N_i^{'} &= b\left(\frac{s^{i-1}-1}{s-1}\right) - s\left(\frac{s^{i-1}-1}{s-1}\right) + bs^{i-1} \\ &= \frac{bs^{i-1}-b}{s} - \frac{ss^{i-1}-s}{s} + bs^{i-1} \\ &= bs^{i-2} + 1 - \frac{b}{s} - s^{i-1} + bs^{i-1} \end{aligned}$$

The approximate running time for an iteration i (t_i(CT-SVM)) is

$$\begin{aligned} t_i(\text{CT-SVM}) &= \text{running time of SVM for the } i\text{th iteration} \\ &\quad + \text{running time for the DGSOT for the} \\ &\quad\ i\text{th iteration} \\ t_i(\text{CT-SVM}) &= \mathrm{O}\Big(\left[bs^{i-2} + 1 - \frac{b}{s} - s^{i-1} + bs^{i-1}\right]^2 \\ &\quad + j \cdot \log_d \left[bs^{i-2} + 1 - \frac{b}{s} - s^{i-1} + bs^{i-1}\right]\Big) \\ &= \mathrm{O}([bs^{i-1}]^2 + j \cdot \log_d [bs^{i-1}]) \end{aligned}$$

If we accumulate the running time of all iterations,

$$
\begin{aligned}
t_i(\text{CT-SVM}) &= \mathrm{O}\left(\sum_{i=1}^{h}([bs^{i-1}]^2 + j \cdot \log_d[bs^{i-1}])\right)\\
&= \mathrm{O}\left(\sum_{i=1}^{h}([bs^{i-1}]^2) + h \cdot j \cdot \log_d[bs^{h-1}]\right)\\
&= \mathrm{O}\left(\left[b^2 \sum_{i=0}^{h-1} s^{2i}\right] + [h \cdot j \cdot \log_d[bs^{h-1}]]\right)\\
&= \mathrm{O}\left(\frac{b^2(s^{2h}-1)}{s^2-1} + h \cdot j \cdot \log_d[bs^{h-1}]\right)
\end{aligned}
$$

Notice that, for simplicity, we assume that the running time for creating each level of the tree is the same, and the total running time of creating the whole tree is $h \cdot j \cdot \log_d[bs^{h-1}]$. Also, we have assumed that $s - 1 \approx s$. If we replace s with rb since $r = s/b$, and only focus on training time of SVM

$$t(\text{CT-SVM}) = \mathrm{O}([b^h r^{h-1}]^2) = \mathrm{O}(b^{2h} r^{2h-2})$$

Therefore, t(CT-SVM) approach trains asymptotically $1/r^{2h-2}$ times faster than t(SVM) which is $\mathrm{O}(b^{2h})$ for $N = b^h$

$$\frac{t(\text{CT-SVM})}{t(\text{SVM})} = \mathrm{O}(r^{2h-2}) \tag{10}$$

6 Rocchio Bundling algorithm

In this section, we describe Rocchio Bundling algorithm for data reduction and classification [37].

6.1 Rocchio Bundling technique

The Rocchio Bundling algorithm is a data reduction technique. The Rocchio Bundling preserves a set of k user-chosen statistics, $s = (s_1, \ldots, s_k)$,where s_i is a function that maps a set of data to a single value. Rocchio Bundling works as follows: We assume that there is a set of training examples for each class. Bundling is applied separately to each class, so we will only discuss what needs to be done for a single class. We select a statistic to preserve and in our experiments we choose to preserve the mean statistic for each feature. Then we partition the full data set D into m equal-sized partitions $P_1, \ldots, P_m$. Each partition becomes a single data point in the bundled data set $D' = \{D_1', \ldots, D_m'\}$. Each element D_i' is a vector of the mean of the data in partition P_i.

Now, we will discuss how bundling works. For this, first, we calculate the Rocchio score which reflects the distance of each point from the Rocchio decision boundary created as will be described in Sect. 6.2. After computing the Rocchio score for each data point in the class, second, we sort the data points according to their scores. This means that points close to the Rocchio decision boundary will be placed close to each other in the list. Third, we partition this sorted list to n/m partitions. Fourth, we compute the mean of each partition as a representative of that partition. Finally, the reduced data set will be the list of all representatives/means of all partitions and will be used for training SVM.

6.2 Rocchio decision boundary

Consider a binary classification problem. Simply put, Rocchio selects a decision boundary (plane) that is perpendicular to a vector connecting two class centroids. Let $\{x_{11}, \ldots, x_{1n}\}$ and $\{x_{21}, \ldots, x_{2n}\}$ be sets of training data for positive and negative classes, respectively. Let $c_1 = (1/n)\sum_i x_{1i}$ and $c_2 = (1/m)\sum_i x_{2i}$ be the centroids for the positive and negative classes, respectively. Then we define the Rocchio score of an example x as in Eq. (11). One selects a threshold value, b, which may be used to make the decision boundary closer to the positive or negative class centroid. Then an example is labeled according to the sign of the score minus the threshold value as in Eq. (12). Figure 11 explains the idea of the Rocchio Classification boundary in the binary classification case. Positive and negative points are presented in oval and square objects, respectively. Points C_1 and C_2 represent the centroids of positive and negative classes, respectively. The bold line is the Rocchio classifier, and it is clearly perpendicular to the line connecting the two class centroids. The Rocchio score represents the distance of a point from the classifier (bold line), for example, Point p_1 has Rocchio score equal to the distance represented by the line connecting it to the Rocchio classifier (bold line).

$$\text{Rocchio score}(x_i) = x_i(c_1 - c_2) \tag{11}$$

$$l(x) = \text{sign}(\text{Rocchio score}(x) - b) \tag{12}$$

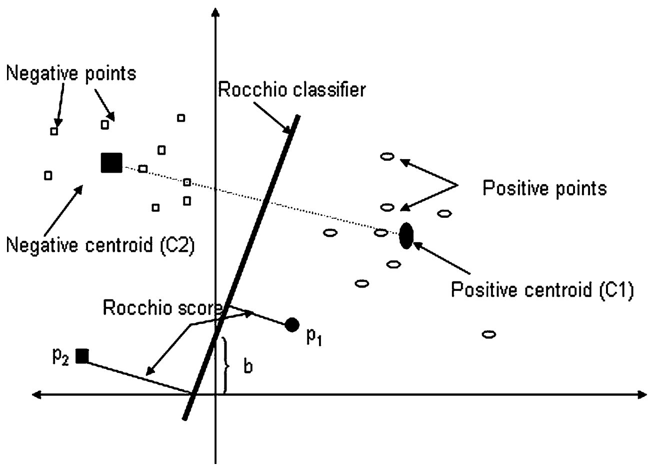

Fig. 11 Rocchio Classifier

7 Evaluation

7.1 Dataset

Our system focuses on network-based anomaly detection. We have applied our algorithm to a set of standard benchmark data, namely the 1998 DARPA data [29] that originated from the MIT Lincoln Lab. Network traffic can be captured using packet-capturing utilities (e.g., libpcap [27]) or operating system utilities at the system call level (e.g., BSM) and then stored as a collection of records in the audit file.

Each data point represents either an attack or a normal connection. There are four categories of attacks: denial-of-service (Dos), surveillance (Probe), remote-to-local (R2L), and user-to-root (U2R). Therefore, overall training and testing data will be categorized into these five classes, i.e., a normal and four attack categories. As a training set, we have total 11,32,365 data points; distribution of data among these normal, Dos, U2R, R2L, and Probe classes are as follows: 830405, 287088, 22, 999, and 13851, respectively. Note that the U2R and R2L classes do not have sufficient training examples as compared to others. Distribution of test data is as follows: normal, Dos, U2R, R2L, and Probe classes, each with 47913, 21720, 17, 2328, and 1269 data points, respectively. Note that here we make sure that the test data is not from the same probability distribution as the training data. In other words, the testing set also includes novel attacks that are variants of known attacks.

Each data point is described by 41 features. Each item of record will be represented by a vector. Note that some attributes are continuous and some are nominal. Since the clustering and classification algorithms require continuous values, these nominal values will be first converted to continuous. We used a bit vector to represent each nominal value.

We use the LIBSVM for SVSM implementation and use the ν-SVM with RBF kernel. In our experiments, we set ν very low (0.001). Since, we address the problem of multi-class classification (i.e., five classes), we implement the "one-vs.-one" scheme due to its reduction of training time over "one-vs.-all" classification.

With regard to setting parameters for the DGSOT, ε is set to 0.08, with α sets to 0.01 K, T_R and T_P are set to 3, 100, and 300, respectively. The parameter settings are purely based on observation. A higher value of K produces a good quality of cluster in relation to the cost of running time. Hence, we choose a moderate value for K. A larger value of ε causes quick convergence for learning. Hence, we set a lower value of ε in order to achieve smooth convergence. T_R and T_P are set in such a way that we will get densely populated uniform clusters.

7.2 Results

Random selection has been used in many applications to reduce the size of the data set. In this purpose, we select randomly 14% from the overall training data set. Here, we report the results by applying a random selection technique to reduce the training set of SVM (see Table 2). In this table, along with Tables 3, 4, and 5, diagonal values represent the number that is correctly classified. For example, out of 47,913 data points 47137, 86, 2, 6, and 682 data points are classified as normal, Dos, U2R, R2L, and Probe, respectively. In Tables 2, 3, 4, and 5 the last column represents false negative (FN) rate and the last row represents false positive (FP) rate, respectively. Accuracy rate for normal, Dos, U2R, R2L, and Probe is 98, 39, 23, 15, and 88%, respectively. FP rate is 5, 1, 100, 2, and 92%, respectively. Note that the accuracy rate is low for U2R and R2L classes. This is a result of a shortage of training set for these classes. In Table 6, we report results based on accuracy. In Table 6, we observe that random selection, pure SVM, SVM + Rocchio Bundling and SVM + DGSOT observe 52, 57.6, 51.6, and 69.8% accuracy, respectively. Note that the total training time is higher for Pure SVM (17.34 hr) and SVM + Rocchio

Table 2 Results of random selection

	Normal	DOS	U2R	R2L	Probe	Accuracy (%)	FP (%)	FN (%)
Normal	47,137	86	2	6	682	98	5	2
DOS	303	8,491	0	0	12,926	39	1	61
U2R	13	0	4	0	0	23	100	76
R2L	1,955	2	1	368	0	15	2	84
Probe	148	1	0	0	1,120	88	92	12

Table 3 Results of pure SVM

	Normal	DOS	U2R	R2L	Probe	Accuracy (%)	FP (%)	FN (%)
Normal	47,180	138	4	7	584	98	7	2
DOS	1,510	18,437	0	0	1,773	84	1	15
U2R	16	0	0	1	0	0	100	100
R2L	1,888	0	8	431	1	18	2	81
Probe	144	3	0	0	1,122	88	68	12

Table 4 Results of SVM + Rocchio Bundling

	Normal	DOS	U2R	R2L	Probe	Accuracy (%)	FP (%)	FN (%)
Normal	47,044	67	3	130	669	98	9	2
DOS	2,896	7,530	0	34	11,260	34	1	65
U2R	14	0	2	1	0	11	100	88
R2L	1,685	2	0	640	1	27	20	73
Probe	148	0	0	0	1,121	88	91	12

Table 5 Results of SVM + DGSOT

	Normal	DOS	U2R	R2L	Probe	Accuracy (%)	FP (%)	FN (%)
Normal	45,616	541	92	267	1,397	95	3	5
DOS	316	21,145	0	48	211	97	2	3
U2R	4	0	4	9	0	23	100	76
R2L	793	0	408	1,024	103	43	24	56
Probe	112	1	0	0	1,156	91	60	9

Table 6 Training time, average accuracy, FP and FN rates of various methods

Method	Average accuracy(%)	Training time (h)	Average FP (%)	Average FN (%)
Random selection	52	0.44	40	47
Pure SVM	57.6	17.34	35.5	42
SVM + Rocchio	51.6	26.7	44.2	48
CT-SVM	69.8	13.18	37.8	29.8

Bundling (26.7 hr). Although SVM + Rocchio Bundling reduces data set first but it incurs significant training time due to preprocessing of dataset. On the other hand, the average training time of SVM + DGSOT (13.18 hr) is lower than for pure SVM with improved accuracy. This is the lowest training time as compared to other methods except random selection. Furthermore, SVM + DGSOT time includes clustering time and SVM training time.

7.2.1 Complexity validation

We notice that 99.8% of total training time was used for DGSOT, and 0.2% time (2.14 min) was used for training SVM. Recall that DGSOT deals with the entire training dataset; however, for later SVM training we only use a smaller subset of data from this dataset. In experimental results, we gathered parameters, b, s, and h for time complexity analysis. In particular, for a dominating normal class we got $b = 209.33$, $s = 11.953$, and $h = 28$. On the other hand, for a DOS class, we got $b = 604.39$, $s = 8.89$, and $h = 13$. After taking the average, we have $b = 406.86$, $s = 10.4215$, and $h = 20.5$, and using these values in Eq. (10) t(CT-SVM) is asymptotically 1.1749×10^{62} times faster than t(SVM). With our experimental results, we observed t(CT-SVM) is 361 times faster than t(SVM) and this validates our claim that with complexity analysis training rate over pure SVM is increased. It is obvious from Table 6, for SVM + DGSOT accuracy rate is highest, FN rate is the lowest and FP rate is as good as pure SVM. This demonstrates that SVM + DGSOT can be deployed as a real time modality for intrusion detection.

7.3 Discussion

Random selection proves, at times, to be a successful technique to save training time. However, in general this may not hold. This is because the distribution of the training data is different from the testing data results. Furthermore, some intrusion experts believe that most novel attacks are variants of known attacks and the "signature" of known attacks can be sufficient to catch novel variants. This exactly happens with SVM + DGSOT. We notice that accuracy rate of our SVM + DGSOT is the best and it is better as compared to pure SVM. It is possible that pure SVM *training* accuracy for testing set may be good but generalization accuracy may not be as good as the accuracy of SVM + DGSOT. DGSOT provides a set of supporting vectors and using SVM we generate hyperplane that would be more accurate for the testing set as compared to the hyperplane generated by pure SVM from entire training data set. Furthermore, FN is lowest for SVM + DGSOT and FP rate is as low as pure SVM. Performance of Rocchio Bundling is the worst in terms of training time, FP and FN rate. Therefore, from the above discussion we notice the superiority of our SVM + DGSOT over other methods.

8 Conclusions and future work

In this paper, we have applied reduction techniques using clustering analysis to approximate support vectors in order to speed up the training process of SVM. We have proposed a method, namely, Clustering Trees based on SVM (CT-SVM), to reduce the training set and approximate support vectors. We exploit clustering analysis to generate support vectors to improve the accuracy of the classifier.

Our approaches proved to work well and to outperform all other techniques in terms of accuracy, false positive rate, and false negative rate. In the case of Rocchio Bundling, the partitions produced were not good representatives as support vectors of the whole data set; hence, the results demonstrated only fair accuracy. Therefore, we observe that our approaches outperform the Rocchio Bundling algorithm on standard benchmark data, namely the 1998 DARPA, data originated at the MIT Lincoln Lab.

We would like to extend this work in the following directions. First, we would like to investigate the effectiveness of our algorithm in other datasets such as DEFCON and UCSB, as well as noisy and incomplete datasets. Second, we would like to address the intrusion detection problem in the domain of mobile wireless networks. Finally, we would like to propose a framework first, and then study various classification algorithms (including ours) using the framework as a test bed, reporting the results in terms of accuracy and efficiency.

References

1. Agarwal, D.K.: Shrinkage estimator generalizations of proximal support vector machines, In: Proceedings of the 8th International Conference Knowledge Discovery and Data Mining, pp. 173–182. Edmonton, Canada (2002)
2. Anderson, D., Frivold, T., Valdes, A.: Next-generation intrusion detection expert system (NIDES): a summary. Technical Report SRI-CSL-95-07. Computer Science Laboratory, SRI International, Menlo Park, CA (May 1995)
3. Axelsson, S.: Research in intrusion detection systems: a survey. Technical Report TR 98-17 (revised in 1999). Chalmers University of Technology, Goteborg, Sweden (1999)
4. Balcazar, J.L., Dai, Y., Watanabe, O.: A random sampling technique for training support vector machines for primal-form maximal-margin classifiers, algorithmic learning theory. In: Proceedings of the 12th International Conference, ALT 2001, p. 119. Washington, DC (2001)
5. Bivens, A., Palagiri, C., Smith, R., Szymanski, B., Embrechts, M.: Intelligent engineering systems through artificial neural networks. In: Proceedings of the ANNIE-2002, vol. 12, pp. 579–584. ASME Press, New York (2002)
6. Branch, J., Bivens, A., Chan, C.-Y., Lee, T.-K., Szymanski, B.: Denial of service intrusion detection using time dependent deterministic finite automata. In: Proceedings of the Research Conference. RPI, Troy, NY (2002)
7. Cannady, J.: Artificial neural networks for misuse detection. In: Proceedings of the National Information Systems Security Conference (NISSC98), pp. 443–456. Arlington, VA (1998)
8. Cauwenberghs, G., Poggio, T.: Incremental and decremental support vector machine learning. In: Proceedings of the Advances in Neural Information Processing Systems, pp. 409–415. Vancouver, Canada (2000)
9. Debar, H., Dacier, M., Wespi, A.: A revised taxonomy for intrusion detection systems. Ann. Télécommun. **55**(7/8), 361–378 (2000)
10. Denning, D.E.: An intrusion detection model. IEEE Trans. Software Eng. **13**(2), 222–232 (1987)
11. Dopazo, J., Carazo, J.M.: Phylogenetic reconstruction using an unsupervised growing neural network that adopts the topology of a phylogenetic tree. J. Mol. Evol. **44**, 226–233 (1997)
12. Forras, P.A., Neumann, F.G.: EMERALD: event monitoring enabling response to anomalous live disturbances. In: Proceedings of the 20th National Information Systems Security Conference, pp. 353–365 (1997)
13. Freeman, S., Bivens, A., Branch, J., Szymanski, B.: Host-based intrusion detection using user signatures. In: Proceedings of the Research Conference. RPI, Troy, NY (2002)
14. Feng, G., Mangasarian, O.L.: Semi-supervised support vector machines for unlabeled data classification. Optimization Methods Software **15**, 29–44 (2001)
15. Ghosh, A., Schwartzbard, A., Shatz, M.: Learning program behavior profiles for intrusion detection. In: Proceedings of the First USENIX Workshop on Intrusion Detection and Network Monitoring, pp. 51–62. Santa Clara, CA (1999)
16. Girardin, L., Brodbeck, D.: A visual approach or monitoring logs. In: Proceedings of the 12th System Administration Conference (LISA 98), pp. 299–308. Boston, MA (1998) (ISBN: 1–880446–40–5)
17. Hu, W., Liao, Y., Vemuri, V.R.: Robust support vector machines for anomaly detection in computer security. In: Proceedings of the 2003 International Conference on Machine Learning and Applications (ICMLA'03). Los Angeles, CA (2003)
18. Ilgun, K., Kemmerer, R.A., Porras, P.A.: State transition analysis: A rule-based intrusion detection approach. IEEE Trans. Software Eng. **21**(3), 181–199 (1995)
19. Joshi, M., Agrawal, R.: PNrule: a new framework for learning classifier models in data mining (a case-study in network intrusion detection) (2001). In: Proceedings of the First SIAM International Conference on Data Mining. Chicago (2001)
20. Khan, L., Luo, F.: Hierarchical clustering for complex data, in press. Int. J. Artif. Intell. Tools. World Scientific
21. Kohonen, T.: Self-Organizing Maps, Springer Series. Springer Berlin Heidelberg New York (1995)
22. Kumar, S., Spafford, E.H.: A software architecture to support misuse intrusion detection. In: Proceedings of the 18th National Information Security Conference, pp. 194–204. (1995)
23. Lane, T., Brodley, C.E.: Temporal sequence earning and data reduction for anomaly detection. ACM Trans. Inform. Syst. Security **2**(3), 295–331 (1999)
24. Lee, W., Stolfo, S.J.: A framework for constructing features and models for intrusion detection systems. ACM Trans. Inform. Syst. Security **3**(4), 227–261 (2000)
25. Luo, F., Khan, L., Bastani, F.B., Yen, I.L., Zhou, J.: A dynamically growing self-organizing tree (DGSOT) for hierarchical clustering gene expression profiles. Bioinformatics **20**(16), 2605–2617 (2004)
26. Marchette, D.: A statistical method for profiling network traffic. In: Proceedings of the First USENIX Workshop on Intrusion Detection and Network Monitoring, pp. 119–128. Santa Clara, CA (1999)
27. McCanne, S., Leres, C., Jacobson, V.: Libpcap, available via anonymous ftp at ftp://ftp.ee.lbl.gov/ (1989)
28. Mukkamala, S., Janoski, G., Sung, A.: Intrusion detection: support vector machines and neural networks. In: Proceedings of the IEEE International Joint Conference on Neural Networks (ANNIE), pp. 1702–1707. St. Louis, MO (2002)
29. Lippmann, R., Graf, I., Wyschogrod, D., Webster, S.E., Weber, D.J., Gorton, S.: The 1998 DARPA/AFRL off-line intrusion detection evaluation. In: Proceedings of the First International Workshop on Recent Advances in Intrusion Detection (RAID). Louvain-la-Neuve, Belgium (1998)
30. Ray, S., Turi, R.H.: Determination of number of clusters in k-means clustering and application in color image segmentation. In: Proceedings of the 4th International Conference on Advances in Pattern Recognition and Digital Techniques (ICAPRDT'99), pp. 137–143. Calcutta, India (1999)
31. Ryan, J., Lin, M., Mikkulainen, R.: Intrusion detection with neural networks. In: Advances in Neural Information Processing Systems, vol. 10, pp. 943–949. MIT Press, Cambridge, MA (1998)
32. Sequeira, K., Zaki, M.J.: ADMIT: anomaly-base data mining for intrusions. In: Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 386–395 (2002)

33. Stolfo, S.J., Lee, W., Chan, P.K., Fan, W., Eskin, E.: Data mining-based intrusion detectors: an overview of the Columbia IDS project. ACM SIGMOD Record **30**(4), 5–14 (2001)
34. Vapnik, V.N.: The Nature of Statistical Learning Theory. Springer Berlin Heidelberg New York (1995)
35. Voorhees, E.M.: Implementing agglomerative hierarchic clustering algorithms for use in document retrieval. Inform. Process. Manage. **22**(6), 465–476 (1986)
36. Warrender, C., Forrest, S., Pearlmutter, B.: Detecting intrusions using system calls: Alternative data models. In: Proceedings of the 1999 IEEE Symposium on Security and Privacy, pp. 133–145. (1999)
37. Shih, L., Rennie, Y.D.M., Chang, Y., Karger, D.R.: Text bundling: statistics-based data reduction. In: Proceedings of the 20th International Conference on Machine Learning (ICML), pp. 696–703. Washington DC (2003)
38. Tufis, D., Popescu, C., Rosu, R.: Automatic classification of documents by random sampling. Proc. Romanian Acad. Ser. **1**(2), 117–127 (2000)
39. Upadhyaya, S., Chinchani, R., Kwiat, K.: An analytical framework for reasoning about intrusions. In: Proceedings of the IEEE Symposium on Reliable Distributed Systems, pp. 99–108. New Orleans, LA (2001)
40. Wang, K., Stolfo, S.J.: One class training for masquerade detection. In: Proceedings of the 3rd IEEE Conference, Data Mining Workshop on Data Mining for Computer Security. Florida (2003)
41. Yu, H., Yang, J., Han, J.: Classifying large data sets using SVM with hierarchical clusters. In: Proceedings of the SIGKDD 2003, pp. 306–315. Washington, DC (2003)
42. Zhang, T., Ramakrishnan, R., Livny, M.: BIRCH: an efficient data clustering method for very large databases. In: Proceedings of the SIGMOD Conference, pp. 103–114 (1996)