An Efficient Local Region and Clustering-Based Ensemble System for Intrusion Detection

Huu Hoa Nguyen, Nouria Harbi, Jérôme Darmont

An Efficient Local Region and Clustering-Based Ensemble System for Intrusion Detection

Huu Hoa Nguyen
Université de Lyon (ERIC Lyon 2)
5 Pierre Mendès-France, France
nhhoa@eric.univ-lyon2.fr

Nouria Harbi
Université de Lyon (ERIC Lyon 2)
5 Pierre Mendès-France, France
nouria.harbi@univ-lyon2.fr

Jérôme Darmont
Université de Lyon (ERIC Lyon 2)
5 Pierre Mendès-France, France
jerome.darmont@univ-lyon2.fr

ABSTRACT
The dramatic proliferation of sophisticated cyber attacks, in conjunction with the ever growing use of Internet-based services and applications, is nowadays becoming a great concern in any organization. Among many efficient security solutions proposed in the literature to deal with this evolving threat, ensemble approaches, a particular family of data mining, have proven very successful in designing high performance intrusion detection systems (IDSs) resting on the mutual combination of multiple classifiers. However, the strength of ensemble systems depends heavily on the methods to generate and combine individual classifiers. In this thread, we propose a novel design method to generate a robust ensemble-based IDS. In our approach, individual classifiers are built using both the input feature space and additional features exploited from *k*-means clustering. In addition, the ensemble combination is calculated based on the classification ability of classifiers on different local data regions defined in form of *k*-means clustering. Experimental results prove that our solution is superior to several well-known methods.

Categories and Subject Descriptors
H.2.8 [**Database Application**], H.6.5 [**Security and Protection**]

Keywords
Data mining, ensemble system, intrusion detection, cyber attack.

1. INTRODUCTION
Recent years have seen an increasing awareness of the risk of cyber attacks not only coming from outside but also inside organizations. The vast spectrum of security breaches, resulting from the popularity of Internet-based services and applications, makes organizational information systems more and more subject to potential vulnerabilities by intruders. For years, there has been a considerable increase in both types and complexities of attacks that are difficult to detect by using traditional methods (e.g., signature-based techniques). Such security-related concerns have motivated researchers to propose many solutions in the literature to face with this dramatically growing threat. Among them, data mining, by regarding the task of detecting cyber attacks as a classification problem, has brought on a noticeable success in developing intrusion detection systems (IDSs) that efficiently detect various types of cyber attacks ranging from virus infections to phishing scams. The strong point of this approach lies in its good generalization abilities to correctly classify (or detect) both known and unknown attacks. However, different inducers (learning algorithms), or even the same inducer but with different input parameters, often exhibit different pros and cons. Hence, it is not easy to determine a single perfect classifier for the target classification model. An efficient approach to avoid mistaken risks in choosing a single poor or inappropriate classifier is to employ an ensemble system (or multiple classifier system) resting on the mutual combination of multiple classifiers for the classification problem. Ensemble systems exploit the mutual complementary decision boundaries of individual classifiers to improve performance of the whole. However, the effectiveness of ensemble systems heavily depends on the ways to generate and combine individual classifiers. As a whole, individual classifiers are required to have the diversity of decision boundaries. Classifier diversity can be achieved in several ways ranging from the sampling techniques based on instance space (e.g., bagging [1]) and feature space (e.g., random subspace [2]) to the adjustment of inducer parameters and/or paradigm architectures. In addition, the combination of classifier outputs can be grounded under different techniques (e.g., voting, decision template).

In this thread, we propose a novel design approach to generate a high performance ensemble system. In our approach, individual classifiers are trained using both the input feature space and additional features exploited from *k*-means clustering. The objective is to enhance both strength and diversity of individual classifiers. Furthermore, the selection and combination of ensemble members are calculated according to their classification abilities over different local data regions defined in form of *k*-means clustering on a separate validation set. Such a selection and combination method expresses the particular potentials of classifiers more appropriately, and thus produces a higher performance. To this goal, we first present cornerstones to generate and combine individual classifiers. Then, we describe a concrete algorithm for creating a high performance ensemble system. Eventually, we experimentally show that our solution outperforms several well-known methods.

The rest of this paper is organized as follows. Section 2 surveys some main studies related to our work. We present methods to generate ensemble classifiers in Section 3. Section 4 comes up with a technique to combine ensemble classifiers. Section 5 presents the algorithm we propose for creating an ensemble-based IDS. Section 6 portrays the experimental results of our algorithm. The paper finally gives a conclusion of our approach in section 7.

2. RELATED WORK

Over the past decade, many ensemble design methods have been proposed for various application domains. This section provides a succinct survey of some noticeable studies in ensemble systems for the intrusion detection application.

Giacinto et al. present a multiple classifier system that uses the KDD99 dataset [4] for intrusion detection [5]. The system consists of various modules, where each is designed for a specific service group. In other words, the instance space of the training set is decomposed with respect to different service groups defined by prior knowledge. For example, the Mail module-based classifier is trained from those instances related to email services, i.e., SMTP, POP2, POP3, NNTP, and IMAP4. This system uses three different learning algorithms (i.e., Parzen, *k*-means, and *v*-SVC) to induce ensemble classifiers.

Zhang and Zulkernine construct an ensemble-based IDS using an adaptive random forest algorithm [3]. The system produces a forest of classification trees in which each tree is built from a different bootstrap sample. The proposed method only uses the attribute *service type* (e.g., HTTP) as the target of classification. In misuse detection, a given instance is passed through the trees and then a majority voting mechanism is applied to label this instance. For anomaly detection, the general idea is that if an instance is classified as the one that is different from its own service type, then this instance is regarded as an anomaly. For example, if an HTTP instance is classified as FTP service type, this instance is determined as an anomaly.

Roli and Giacinto propose a multiple classifier system comprising three different groups of classifiers that correspond to three feature subsets predefined by domain knowledge [7]. Specifically, each group of classifiers is trained from one out of the three predefined feature subsets (i.e., intrinsic, traffic, and content feature subsets). A subsequent work of the same authors describes an ensemble architecture including multiple one-class *k*-means classifiers [8]. Each classifier is built from an instance subspace related to a specific attack type (e.g., Neptune is one of twenty three attack types and belongs to DoS attack class).

Several studies have also applied Soft Computing algorithms for building ensemble systems for the intrusion detection problem. For example, Abadeh et al. propose a parallel genetic local search algorithm to generate fuzzy rule sets for each class label in the training set [11]. Each of these rule sets is utilized to build a fuzzy classifier. Then, a decision fusion procedure is in charge of determining a class label for a given instance. Likewise, Zainal et al. describe an ensemble model that uses three different learning algorithms to build three classifiers, i.e., Linear Genetic Programming (LGP), Neural Fuzzy Inference, and Random Forest (RF) [12]. Each classifier is trained on the same training set and assigned to a weight calculated based on classifier performance.

Finally, apart from other methods that build classifiers from network packet headers, Perdisci et al. introduce a multiple classifier system for anomaly intrusion detection given network packet payloads [14]. This system applies a dimensionality reduction algorithm to retrieve several compact representations of payload in different feature spaces. Then, each representation of payload is used to train a one-class SVM classifier.

By contrast to the surveyed methods, our approach, on one hand, exploits supplemental cluster features resulting from k-means clustering for generating the diversity of classifiers. On the other hand, the selection and combination of classifiers are based on dynamic local validation regions with respect to a given testing instance. Intuitively, such factors all together contribute to the performance improvement of the whole system.

3. GENERATION OF CLUSTER FEATURES AND INDIVIDUAL CLASSIFIERS

3.1 Generation of Cluster Features

Clustering aims to organize data into groups (clusters) according to their similarities measured by some concepts. In metric spaces, similarity is often defined in term of a distance norm measured between data vectors. The potentiality of clustering is to express the latently natural relationships between data points. Clusters are represented by their centers (or centroids) that are found in the partitioning process of a clustering algorithm. In our approach, the *k*-means clustering algorithm [15] is used for constructing additional cluster features. These features are then incorporated into the input space for building individual classifiers.

Let us first denote $\mathbf{S}=\{\mathbf{X},\mathbf{Y}\}$ the original training set of n data points $\mathbf{X}=\{x_1,\ldots,x_n\}$, where each point x_i is an m-dimensional vector $(x_{i1},\ldots,x_{im})$ assigned to a label $y_i \in \mathbf{Y}$ belonging one of the c classes $\mathbf{\Omega}=\{\omega_1, \ldots,\omega_c\}$. Let us also denote $\mathbf{V}$ the set of k centroids obtained by portioning $\mathbf{X}$ into k clusters, using k-means clustering. Let $\mathbf{W}=\{w_{ij} \mid w_{ij}\in[0,1], i=1\ldots n, j=1\ldots k\}$ be an cluster membership matrix, where w_{ij} is a membership weight that data point x_i belongs to cluster j. The matrix $\mathbf{W}$ is calculated by Formula 1.

$$w_{ij} = \left(\frac{1}{d(x_i, v_j)^2}\right) \Big/ \sum_{q=1}^{k}\left(\frac{1}{d(x_i, v_q)^2}\right) \quad (1)$$

where k is the number of clusters, and $d(x_i,v_j)$ is the distance from data point $x_i \in \mathbf{X}$ to centroid $v_j \in \mathbf{V}$.

Let $\mathbf{U}=\{u_i \mid u_i = \max(w_{ij}), i=1\ldots n, j=1\ldots k\}$ hold the maximum membership weight of each data point x_i, and $\mathbf{Z}=\{z_i \mid z_i = \text{argmax}_j(w_{ij}), i=1\ldots n, j=1\ldots k\}$ contains the cluster (symbolic) number assigned to each data point x_i. For conciseness, we term two column matrices $\mathbf{Z}$ and $\mathbf{U}$ as "cluster features". We also term the training set added with cluster features $\{\mathbf{X}, \mathbf{Z}, \mathbf{U}, \mathbf{Y}\}$ as a "manipulated training set". These notations and terminologies are depicted in Figure 1. Here, cluster features are exploited by partitioning n training data points into four clusters, where x_1 belongs to cluster '4' with a membership weight equal to 0.4, and so on. The manipulated training set is employed to build an individual classifier.

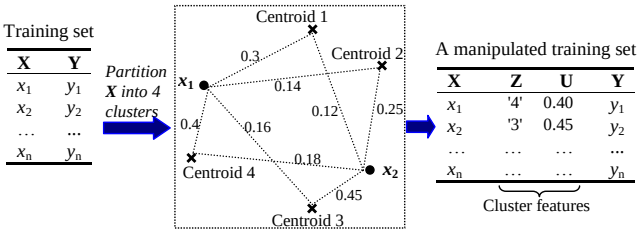


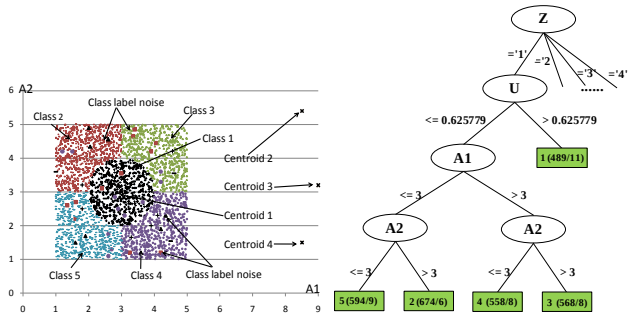
Figure 1. A manipulated training set, resulting from adding cluster features into the input space.

3.2 Generation of Individual Classifiers

Obviously, by changing parameter k (the number of clusters in k-means clustering), we receive different values of cluster features, and hence different classifiers trained by a given inducer. More

precisely, we generate an ensemble system of t classifiers induced from the training set manipulated with values k ranging from 2 to t+1. In other words, each individual classifier is trained using both the input feature space and cluster features exploited from k-means clustering with $k \in \{2,3,\ldots,t+1\}$.

Although it is applicable for any type of inducers, our approach uses Decision Tree (DT) [16] as a base inducer to build individual classifiers for the ensemble. Because the DT inducer is very sensitive to perturbation of the training set, incorporating cluster features into the input space intensifies the diversity of classifiers. Basically, cluster features exploited from relevant values of k often benefit DT inducer in determining more appropriate splits, thus outputting more efficient classification trees. For example, let us examine an artificial dataset **S** partially plotted in Figure 2(a). This dataset comprises 12,000 data points, where the square-shaped region contains 3,000 uniformly-distributed data points (plotted) and the outside region includes 9,000 points (not plotted). A decision tree DT^* built from both the initial features ($\mathbf{A}_1$, $\mathbf{A}_2$) and cluster features (**Z**, **U**, with k=4) of the dataset **S** is partially depicted in Figure 2(b). Noticeably, for the square-shaped data region, DT^* only contains 9 nodes (4 leaves) but attains a 99.41% accuracy, whereas the tree (not shown) generated from the initial features ($\mathbf{A}_1$, $\mathbf{A}_2$) comprises 51 nodes (26 leaves) but only achieves a 97.24% accuracy (10-fold cross valuation).



(a) **(b)**

Figure 2. (a) The 2D scatter plot of a dataset S; (b) A decision tree DT^* generated using both initial and cluster features.

4. ENSEMBLE COMBINATION

In the ensemble context, there are two basic approaches, i.e., combination and selection, to classify unforeseen instances. The first approach fuses (or combines) the output of ensemble members in some fashion (e.g., majority voting) to achieve a final decision, whereas the second approach attempts to dynamically select one best ensemble member with respect to some criteria for classifying a given testing instance. Our method follows the first approach, because the second is known to be sensitive to determining an appropriate classifier for a given testing instance.

Typically, ensemble members are often weighted in some manners to further improve the overall performance of the final model. Such weighting indicates the influence degree of each ensemble member to the final decision. One common way to weight classifiers is to estimate their performance on a full validation or training set. However, such estimates are insufficient to reflect particular abilities of individual classifiers, especially in application domains where the distribution of target classes is heavily biased. This is because, for example, a classifier exhibits its high performance on the global region but very poor performance on certain local regions. For solving this problem, some researchers define a dynamic local region surrounding an unlabeled test instance in terms of k-nearest neighbors in the training set, and then estimate the accuracy of classifiers in this region to determine the best one for classifying the test instance [17]. In our opinion, the estimation of such k-nearest-based local accuracy, although efficient, is somewhat unfit for those application domains where classification is required to be instant or *online*, such as intrusion detection. This is because, in case the training set is large, calculating the k-nearest neighbors for each incoming test instance in the operation phase is time-consuming and hence hardly applicable in this situation.

Hence, we propose a combination method by weighting the performance of ensemble members based on dynamic local regions defined in terms of cluster prototypes (or centroids) in the validation set. The basic idea is to estimate the accuracy of each ensemble member in a cluster (region) whose centroid is closest to the test instance needed to be classified. The keystone is to intensify correct decisions and reduce incorrect decisions of each classifier in local regions surrounding the test instance. One obvious advantage to our combination solution is that, in the operation phase, identifying the local regions in the validation set is considered to be instantaneous. Another applicability is that the method is little sensitive to selecting the number of local regions (clusters). This is because it attempts to weight classifier abilities in different local regions for combination purpose rather than dynamic classifier selection. The ensemble combination method we propose is formally defined as follows.

Let $E = \{C_2,\ldots,C_{t+1}\}$ be the set of t classifiers, where each C_k is induced from the training set added with cluster features resulting from a k-means clustering with k clusters. Suppose that each classifier produces its outputs in form of the estimate of posterior probabilities, where $P(C_k, \omega_j \mid x)$ represents the support degree produced by classifier C_k to class ω_j for a given test instance x. Let us also assume that the instance space of the validation set is divided into q regions (clusters) $\{\mathbf{R}_1,\ldots,\mathbf{R}_q\}$ by a k-means clustering, and $\mathbf{R}^* \in \{\mathbf{R}_1,\ldots,\mathbf{R}_q\}$ is the region whose centroid is closest to the test instance (x) needed to be classified. Then, the overall accuracy of classifier C_k on region $\mathbf{R}^*$, which is denoted by $OA(C_k \mid \mathbf{R}^*)$, is formulated in Formula 2.

$$OA(C_k \mid \mathbf{R}^*) = \frac{\text{Nr of instances in } \mathbf{R}^* \text{ correctly classified by } C_k}{\text{Total number of instances in } \mathbf{R}^*} \quad (2)$$

According to the Bayes theorem, the posterior probability estimate that the test instance x belongs to class ω_j given by the ensemble E is provided in Formula 3. Note that in case the overall accuracy of all classifiers on region $\mathbf{R}^*$ equals to zero (i.e., $OA(C_k \mid \mathbf{R}^*)$=0, $\forall\ C_k \in E$), the posterior probability given by the ensemble E is calculated from Formula 4. By such a formulation, the final prediction of the test instance x is given in Formula 5.

$$P_E(\omega_i \mid x) = \frac{\sum_{C_k \in E} OA(C_k \mid \mathbf{R}^*) \cdot P(C_k, \omega_i \mid x)}{\sum_{\omega_j \in \Omega} \left(\sum_{C_k \in E} OA(C_k \mid \mathbf{R}^*) \cdot P(C_k, \omega_j \mid x) \right)} \quad (3)$$

$$P_E(\omega_i \mid x) = \frac{\sum_{C_k \in E} P(C_k, \omega_i \mid x)}{\sum_{\omega_j \in \Omega} \left(\sum_{C_k \in E} P(C_k, \omega_j \mid x) \right)} \quad (4)$$

$$\text{Label}(x) \leftarrow \underset{\omega_i}{\text{argmax}}\ P_E(\omega_i \mid x), \quad i = 1 \ldots c \quad (5)$$

With respect to generating local validation regions, the number of cluster prototypes (q) is basically determined by experiment. Usually, the value of parameter q is much smaller than the total number of instances (n'') in the validation set. However, if q is too small, the concept of "local region" is no longer applicable and thus the particular ability of each classifier is not exploited either. When q is too large, it takes more time for determining region $\mathbf{R}^*$. Moreover, in this case, region $\mathbf{R}^*$ may comprise only a few instances, which is sensitive to or insufficient for evaluating the classification ability of classifiers. Normally, a reasonable value of q can be found by some prior knowledge such as subclasses (e.g., attack types) or visualization analyses.

5. ENSEMBLE-BASED IDS

5.1 Global Approach

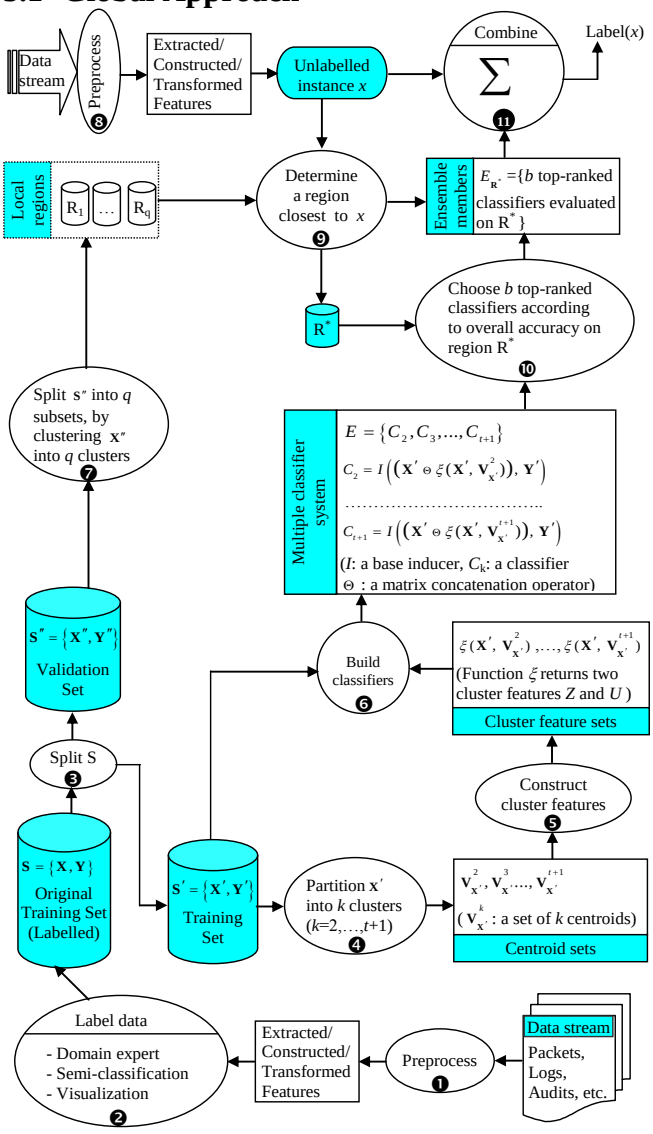


Figure 3. Ensemble-based intrusion detection approach

Our approach for generating an ensemble-based IDS is graphically described in Figure 3. Since the task of intrusion detection is regarded as a classification problem, it is first required to have a well-labeled IDS dataset. In general, data can be gathered from various sources such as network traffic packets, operation system audits, system log files (e.g., system calls, shell commands), and application log files. These heterogeneous data are then preprocessed (e.g., data integration, feature extraction, construction, and/or transformation) to establish a well-formed dataset (Step 1). Subsequently, derived data are labeled using domain expert knowledge and/or other supporting techniques (e.g., semi-classification, data visualization) to form an original IDS training set (Step 2). Next, the original training set is split into two separate sets (i.e., a training set and a validation set) by a stratification sampling based on classes or subclasses (Step 3). Subsequently, a multiple classifier system is built using both initial and cluster features by the method described in Section 3 (Steps 4-6). Then, local regions are created by splitting the validation set into q subsets, using k-means clustering (Step 7).

An incoming event to be classified is first preprocessed and transformed into a testing record (instance) as those used for constructing the original training set (Step 8). This unlabeled testing instance is then compared to the prototypes of local regions, by a distance metric, to determine one nearest region $\mathbf{R}^*$ (Step 9). Because our approach employs the so-called *overproduction* method to generate a large amount of classifiers (i.e., t classifiers), the next step is to choose b top-ranked (highest performance) classifiers ($b \le t$), evaluated on region $\mathbf{R}^*$, for classifying the testing instance (Step 10). Finally, the testing instance is inputted to b top-ranked classifiers for ensemble combination as described in Section 4 (Step 11).

For calculating distances between data points, we employ the heterogeneous distance metric weighed by information gain [6]. Furthermore, to tackle the wide dispersion in different ranges, continuous features are also normalized by *min_max* scaling [15].

5.2 Algorithm for Generating an Ensemble-based IDS

Our algorithm for generating an ensemble system of classifiers, called *CBE*, is depicted by the pseudo-code from Figure 4. Built-in functions are shown in Figure 5, whereas related notations are indicated in Table 1.

Table 1. Notations used in Figures 3-5

Notation	Description
$\mathbf{V}_{\mathbf{D}}^{k}$	A set of k centroids, $\mathbf{V}_{\mathbf{D}}^{k} = \{v_j \mid j = 1...k\}$, derived by clustering a given dataset **D** into k clusters.
ξ	A function to calculate cluster features for a given dataset **D**. This function is described in Figure 5.
Θ	A vertical concatenation operator between two matrices.
E	A set of t classifiers, $E = \{C_2, C_3, ..., C_{t+1}\}$, where each classifier C_k is trained from the training set concatenated with cluster features resulting from clustering the training set into k clusters.
V	A set of t centroid sets, $V = \left\{\mathbf{V}_{\mathbf{X}'}^{2}, \mathbf{V}_{\mathbf{X}'}^{3}, ..., \mathbf{V}_{\mathbf{X}'}^{t+1}\right\}$
$E_{\mathbf{R}}$	A set of q classifier sets, $E_{\mathbf{R}} = \left\{E_{\mathbf{R}_i} \mid i = 1...q\right\}$, where each $E_{\mathbf{R}_i}$ consists of b top-ranked classifiers evaluated on region $\mathbf{R}_i$

TRAINING PHASE

Input: $\mathbf{S}=\{\mathbf{X},\mathbf{Y}\}$: an original training set; *I*: a base inducer; *t*: total number of classifiers; *b*: number of top-ranked classifiers ($b \le t$); *q*: number of local regions.

Output: E, V, and $E_{\mathbf{R}}$

***Initialization*:** $\{\mathbf{S}', \mathbf{S}''\} \leftarrow \text{Split}(\mathbf{S})$ // $\mathbf{S}' \cap \mathbf{S}'' = \varnothing$

where $\mathbf{S}' = \{\mathbf{X}', \mathbf{Y}'\}$ is the training set of n' instances,

$\mathbf{S}'' = \{\mathbf{X}'', \mathbf{Y}''\}$ is the validation set of n'' instances

Step1: Generate a multiple classifier system

1: *For k*=2 to *t+1 do*

2: $\mathbf{V}^{k}_{\mathbf{X}'} \leftarrow k\text{-Means}(\mathbf{X}', k)$ *// Partition* $\mathbf{X}'$ *into k clusters [15]*

3: $C_k \leftarrow I\left((\mathbf{X}' \Theta \xi(\mathbf{X}', \mathbf{V}^{k}_{\mathbf{X}'})), \mathbf{Y}'\right)$ *// Build* C_k *using inducer I*

4: *End For*

5: $E \leftarrow \{C_2, C_3, ..., C_{t+1}\}$ *// Store the generated classifiers*

6: $V \leftarrow \left\{\mathbf{V}^{2}_{\mathbf{X}'}, \mathbf{V}^{3}_{\mathbf{X}'}, ..., \mathbf{V}^{t+1}_{\mathbf{X}'}\right\}$ *// Store the generated centroid sets*

Step2: Create q local validation regions

7: $\{\mathbf{R}_1, \mathbf{R}_2, ..., \mathbf{R}_q\} \leftarrow \text{CreateLocalRegions}(\mathbf{S}'', q)$ (Fig. 5)

where $\mathbf{R}_i = \{\mathbf{X}''_i, \mathbf{Y}''_i\}$, $i = 1...q$

Step3: Evaluate the performance (overall accuracy) of each classifier on each local validation region

8: *For each pair* $(C_k \in E, \mathbf{V}^{k}_{\mathbf{X}'} \in V)$ and *each* $\mathbf{R}_i$ $(i = 1...q)$ *do*

9: $OA(C_k \mid \mathbf{R}_i) \leftarrow C_k\left((\mathbf{X}''_i \Theta \xi(\mathbf{X}''_i, \mathbf{V}^{k}_{\mathbf{X}'})), \mathbf{Y}''_i\right)$ (Formula 2)

Step4: Determine *b* top-ranked classifiers (highest performance classifiers evaluated based on $OA(C_k \mid \mathbf{R}_i)$) for each region

10: *For each* $\mathbf{R}_i$ $(i = 1...q)$ *do*

11: $E_{\mathbf{R}_i} \leftarrow \left\{C_{k^*} \,\middle|\, C_{k^*} \in E \ \wedge \underset{OA(C_{k^*} \mid \mathbf{R}_i)}{\text{Rank}}\left(C_{k^*}\right) \le b\right\}$

12: $E_{\mathbf{R}} \leftarrow \left\{E_{\mathbf{R}_i} \mid i = 1...q\right\}$ *// Store the top-ranked classifier sets*

OPERATION PHASE

For a given unlabelled test instance *x*,

13: Determine one region $\mathbf{R}^* \in \{\mathbf{R}_1, ..., \mathbf{R}_q\}$ such that its centroid is closest to *x*.

14: $\text{Label}(x) \leftarrow \text{Combine}\left(\left\{C_k \in E_{\mathbf{R}^*}\right\}, x \Theta \xi(x, \mathbf{V}^{k}_{\mathbf{X}'})\right)$ (Fig. 5)

where $\begin{cases} E_{\mathbf{R}^*} \text{ is the set of } b \text{ top-ranked classifiers on } \mathbf{R}^*, (E_{\mathbf{R}^*} \in E_{\mathbf{R}}) \\ \mathbf{V}^{k}_{\mathbf{X}'} \text{ is the centroid set used to build } C_k, (\mathbf{V}^{k}_{\mathbf{X}'} \in V) \end{cases}$

Figure 4. Algorithm *CBE*

BUILT-IN FUNCTIONS

Function $\xi(\mathbf{D}, \mathbf{V}^{k}_{\mathbf{X}'})$

// Construct cluster features for n_d *data points of a given data set* $\mathbf{D}$

Compute w_{ij}, $\forall(x_i \in \mathbf{D} \wedge v_j \in \mathbf{V}^{k}_{\mathbf{X}'})$, using Formula 1

$u_i \leftarrow \max(w_{ij}), \quad j = 1...k, \ i = 1...n_d$

$z_i \leftarrow \arg\max_j(w_{ij}), \quad j = 1...k, \ i = 1...n_d$

Return $\{(z_i, u_i) \mid i = 1...n_d\}$

End Function

Function $\text{CreateLocalRegions}(\mathbf{S}'', q)$ // $\mathbf{S}'' = \{\mathbf{X}'', \mathbf{Y}''\}$ *: Validation set*

$\mathbf{V}^{k}_{\mathbf{X}''} \leftarrow k\text{-Means}(\mathbf{X}'', q)$ *//Partition* $\mathbf{X}''$ *into q clusters [15]*

Compute w_{ij}, $\forall(x_i \in \mathbf{X}'' \wedge v_j \in \mathbf{V}^{k}_{\mathbf{X}''})$, using Formula 1

$\mathbf{L} \leftarrow \left\{l_i \,\middle|\, l_i = \arg\max_j(w_{ij}), \ j = 1...q, \ i = 1...n''\right\}$

Split $\mathbf{S}''$ into q regions $\{\mathbf{R}_i = \{\mathbf{X}''_i, \mathbf{Y}''_i\}\}$ according to $\mathbf{L}$, (i=1..q)

Return $\left\{\mathbf{R}_i = \{\mathbf{X}''_i, \mathbf{Y}''_i\} \,\middle|\, i = 1...q\right\}$

End function

Function $\text{Combine}\left(\left\{C_k \in E_{\mathbf{R}^*}\right\}, \hat{x}\right)$

// $\hat{x}$ is a unlabeled testing instance concatenated with cluster features

If $OA(C_k \mid \mathbf{R}^*) == 0$, $\forall C_k \in E_{\mathbf{R}^*}$ *then*

Compute $P_E(\omega_i \mid \hat{x})$, $i = 1...c$, using Formula 4.

Else Compute $P_E(\omega_i \mid \hat{x})$, $i = 1...c$, using Formula 3.

$\text{Label}(\hat{x}) \leftarrow \underset{\omega_i}{\text{argmax}} P_E(\omega_i \mid \hat{x}), \quad i = 1...c$

End function

Figure 5. Built-in functions called by the *CBE* algorithm

In the training phase, the algorithm initially splits the original training set into a training set and a validation set. The splitting ratio is basically around 7:3, depending on the size of the original training set. Then, it *overproduces* a system of *t* classifiers; each built from both the original features and the cluster features resulting from *k*-means clustering (Lines 1-5). The algorithm also stores the generated centroid sets that are later used to construct cluster features for validation and testing data points (Line 6). Then, it creates *q* local regions by partitioning the validation set into *q* clusters (Line 7). Subsequently, it evaluates the performance (overall accuracy) of each classifier on each region (Lines 8, 9). Here, cluster features of local regions are calculated based on the corresponding centroid sets exploited from the training set. Finally, the algorithm chooses *b* top-ranked (highest performance) classifiers for each local region (Lines 10-12).

In the operation phase, for an unlabeled testing instance *x*, the algorithm first determines one local region $\mathbf{R}^*$ whose centroid is closest to *x* (Line 13). Then, it uses *b* top-ranked classifiers on this region for ensemble combination to produce a final prediction for the test instance (Line 14). Note that cluster features of the testing instance (*x*) are calculated based on those centroid sets employed to build *b* top-ranked classifiers.

6. EXPERIMENTS

6.1 Dataset

Our experiments are conducted on the intrusion detection dataset KDD99 [4]. Although the dataset has been criticized by the IDS community mainly because it is unrepresentative of a real-life network scenario [10], so far it is still the only intrusion detection evaluation benchmark for research purposes. The KDD99 dataset comprises 494,021 training instances and 311,029 testing instances. Due to data volume, the research community mostly uses small subsets of the dataset for evaluating IDS methods. Each instance in the dataset represents a network connection, i.e., a sequence of network packets starting and ending at some well

defined times, between which data flows to and from a source IP address to a target IP address. Such a connection instance is described by a 41-dimensional feature vector and labeled with respect to 5 classes: *Normal*, *Probe*, *DoS* (denial of service), *R2L* (remote to local), and *U2R* (user to root).

To facilitate experiments without losing generality, we only use a smaller set of the KDD99 dataset for the purpose of evaluating and comparing our method to others. In particular, the training and testing sets used in our experiments are made up of 33,016 instances and 169,687 instances that are selectively extracted from the KDD99 training and testing sets, respectively. The principle for forming such reduced sets is to get all instances in each small group (attack type), but only a limited amount of instances in each large group, from both the KDD99 training and testing sets. More explicitly, for forming the reduced training set, we randomly select five percent of each large group *Neptune*, *smurf*, and *normal*, while gathering all instances in the remaining groups from the KDD99 training set. For sampling the reduced testing set, we randomly select 50 percent of each large group *Neptune*, *smurf*, and *normal*, while collecting all instances in the remaining groups from the KDD99 testing set. Class distribution of these two reduced sets is shown in Table 2.

Table 2. Class distribution of the reduced datasets

Class	Training Set	Testing Set
DoS	22,867	118,807
Probe	4,107	4,166
R2L	1,126	16,347
U2R	52	70
Normal	4,864	30,297
Total	**33,016**	**169,687**

6.2 Experimental Results

In our experiments, the reduced training set of 33,016 instances, also termed the original training set, is disjointedly split into a training set and a validation set with a fraction of 75% and 25%, respectively. Intuitively, such a splitting fraction is reasonable to the size of the original training set. In addition, the split is carried out by randomly sampling the original training set (without replacement) based on each stratification of normal traffic and attack types. This is to ensure sufficient representative of each subclass in both the training and validation sets. The other input parameters of our algorithm (*CBE*) are established as follows.

The total number of classifiers (parameter t) is set to 150. The reason for choosing such a large number is because, firstly, the training phase of Decision Tree (i.e., base inducer for *CBE*) is not time-consuming, compared to that of many other inducers such as Support Vector Machines or Artificial Neural Networks. Secondly, as an inherent essence of the *overproduction*-based ensemble approach, it is necessary to generate a large amount of different classifiers. This not only aims to increase the diversity of classifiers but also the possibility of selecting best classifiers for the ensemble combination. However, when t is too large, it becomes much more time-consuming in the training phase.

The number of top-ranked classifiers (parameter b) is set to 10. This setting is just like many other popular methods (e.g., boosting, bagging) that typically use 10 classifiers for the ensemble combination. Besides, the number of local validation regions (parameter q) is basically set to 23 (i.e., the total number of normal traffic and attack types or subclasses in the KDD99 intrusion detection dataset). Nevertheless, to have a wider comparative view, we experimentally run the *CBE* algorithm with some more values of parameter q, i.e., 50, 100, 150, and 200.

The experimental comparison of our method (implemented in Matlab) to other seven well-known algorithms is presented in Table 3. All the compared ensemble methods (Rows 2-12) employ the same base inducer (i.e., DT) and the same number of base classifiers (i.e, 10). To avoid biases related to sampling and randomization factors , we run our *CBE* algorithms (Rows 8-12) as well as the other six ensemble algorithms (Rows 2-7) ten times, and then get the average result of runs. The performance of classifiers (algorithms) is evaluated by True Positive Rates (TPRs) and False Positive Rates (FPRs). The TPR of a class ω_c is the ratio of "the number of correctly classified instances in the class ω_c" to "the total number of instances in the class ω_c". The FPR of a class ω_c is the ratio of "the number of instances that do not belong to the class ω_c but are classified as ω_c" to "the total number of instances that do not belong to the class ω_c".

Table 3. TPRs and FPRs (%) of classifiers (algorithms)

Algorithms		DoS	Prob	R2L	U2R	Normal	Avg
1. DT	TP	94.67	76.12	6.01	50.00	97.70	86.20
	FP	5.51	0.41	0.05	0.01	14.24	6.41
2. Boosting	TP	95.48	89.46	6.82	47.14	99.28	87.45
	FP	0.41	0.32	0.01	0.01	14.91	2.96
3. Bagging	TP	94.76	77.04	6.24	37.14	98.36	86.42
	FP	3.02	0.45	0.04	0.00	14.96	4.80
4. Decorate	TP	94.92	75.69	5.42	48.57	97.95	86.35
	FP	4.25	0.47	0.10	0.01	14.37	5.56
5. RC	TP	94.42	83.81	11.41	51.43	98.43	86.86
	FP	0.59	0.32	0.47	0.04	14.69	3.09
6. Dagging	TP	95.29	79.85	10.08	48.57	97.61	87.10
	FP	0.83	0.34	0.08	0.01	14.84	3.25
7. RS	TP	95.18	76.34	7.71	18.57	98.85	86.91
	FP	3.87	0.39	0.03	0.00	14.01	5.22
8. *CBE* (q=23)	TP	98.12	95.08	25.03	88.57	99.41	91.23
	FP	0.38	0.31	0.02	0.00	10.08	2.08
9. *CBE* (q=50)	TP	98.63	96.62	26.29	81.43	99.49	91.76
	FP	0.36	0.19	0.01	0.00	9.66	1.98
10. *CBE* (q=100)	TP	97.59	94.39	22.39	78.57	99.36	90.58
	FP	0.78	0.32	0.02	0.00	10.71	2.47
11. *CBE* (q=150)	TP	97.16	95.94	22.51	85.71	99.27	90.31
	FP	1.02	0.29	0.02	0.00	10.97	2.68
12. *CBE* (q=200)	TP	97.02	95.17	22.12	87.14	99.65	90.22
	FP	1.13	0.35	0.01	0.00	10.94	2.75

Note:
- DT refers to Decision Tree [16] with input parameters established as follows: Pruning method=*pessimistic pruning*, Confidence=0.25, the minimum number of instances per leaf=6, and Counting at leaves is smoothed based on Laplace.
- Classifiers 2-7 uses the following algorithms:
  Boosting, Bagging [1], Decorate [9], Dagging [18],
  RC (Random Committee), and RS (Random Subspace) [2].
- Classifiers 8-12 are built from our algorithm parameterized with: *CBE*(I=DT, t=150, b=10, q={23, 50, 100, 150, 200}).
- Column *Avg* is the average weighted by the number of instances on each class.

As shown in Table 3, in general, while producing a lower FPR, our method (*CBE*) outperforms the other seven classifiers with respect to TPR in all five classes. Particularly, *CBE* is considerably superior to all the others in detecting (classifying) *R2L*, *U2R*, and *Probe* attacks. Regarding *R2L* (a 'hard' class), on

the whole average, *CBE* gives a 23.67% TPR (with a 0.02% FPR), whereas the classifier having highest performance on this class (i.e., Random Committee) produces an 11.44% TPR (with a 0.46% FPR). Likewise, for 'hard' class *U2R*, on the whole average, our method produces an 84.29% TPR (with a 0% FPR), while the best classifier on this class (i.e., Random Committee) gives a 51.43% TPR (with a 0.04% FPR). With respect to class *Probe*, the average TPR of five *CBE* classifiers is 15.68% higher than that of the other seven classifiers. The experiment results also tell that the average performance of five *CBE* classifiers reaches 90.82% TPR (2.39% FPR), whereas performance of the other seven classifiers fluctuates from 86.20% TPR (6.41% FPR) to 87.45% TPR (2.96% FPR).

Although reaching a very high TPR, all the tested fifteen classifiers incur a very high FPR on class *Normal*. This is mainly because there is a large mount of *R2L* instances misclassified as *Normal* instances. In fact, *R2L* instances are very similar to *Normal* instances in the KDD99 dataset, thus many classification methods proposed in the literature (e.g., KDD99 winners [19]) incur a very high FPR on class *Normal*.

It is also noticed that, as the main drawback of overproduction-based ensemble approaches, our method takes more time in the training phase than the others. This is because it overproduces a large pool of candidate classifiers and then chooses some competent classifiers from the pool for ensemble combination. Nevertheless, the detection (classification) time of our method is comparable to that of the others, because choosing 10 top-ranked classifiers for each local validation region is carried out in the training phase. In addition, since the number of local validation regions is often small (e.g., 50), the time to determine region $\mathbf{R}^*$ with respect to a testing instance in the detection phase is negligible. The average running time of all tested algorithms (executed on a computer with 2.8GHz CPU, 4GB RAM) is shown in Table 4.

Table 4. Training time and detection time (in second)

Algorithms	Nr of generated individual classifiers	Training time	Detection time
DT		07.02	02.15
Boosting	10	46.45	04.49
Bagging	10	39.09	03.89
Decorate	10	447.76	04.65
RC	10	03.79	03.81
Dagging	10	15.12	03.55
RS	10	23.88	04.37
CBE	150	11,852.6	04.76

7. CONCLUSION

We propose in this paper an efficient design approach for generating a robust ensemble system for intrusion detection. The use of cluster features exploited from different clustering processes intensifies both performance and diversity of individual classifiers. In addition, the selection and combination of ensemble members, based on different local validation regions, express the particular classification abilities of classifiers more appropriately, and accordingly further enhance performance of the whole system. The tactic to achieve a robust ensemble-based IDS is concretely formulated via a set of algorithms. Moreover, we experimentally show that our method clearly outperforms all the tested seven methods. In particular, our method is considerably superior to all evaluated methods with respect to *R2L*, *U2R*, and *Probe* classes. Although the experiments are conducted on the KDD99 intrusion detection dataset, the approach we propose can be generally used to improve classification in other application domains. However, to be more objective in evaluating any data mining solution as well as overcome criticized drawbacks of the KDD99 dataset, our future work will be to test the proposed algorithms on other real datasets. In particular, our current effort is fulfilling a honeypot system for the goal of gathering both real intrusion and normal traffic activities. Such a real dataset will then be employed to evaluate the algorithms we proposed.

8. REFERENCES

[1] Breiman, L. 1996. Bagging predictors. *Machine Learning*. 24, 2, 123-140.

[2] Ho. T. K. 1998. The random subspace method for constructing decision forests. *IEEE Trans. Pattern Anal. Mach. Intell.* 20(8), 832–844 .

[3] Zhang, J. and Zulkernine, M. 2006. Anomaly based network intrusion detection with unsupervised outlier detection. In *Proceedings of ICC'06.* Istanbul, 2388-2393.

[4] KDD99 data. http://kdd.ics.uci.edu/databases/kddcup99

[5] Giacinto, G., Perdisci, R., Del Rio, M., and Roli, F. 2008. Intrusion detection in computer networks by a modular ensemble of one-class classifiers. *Inform. Fusion*, 1, 69-82.

[6] Nguyen, H.H., Harbi, N., and Darmont, J. 2011. An efficient fuzzy clustering-based approach for intrusion detection. In *Proceedings of ADBIS 2011*. Vienna, Austria.

[7] Giacinto, G. and Roli, F. 2002. Intrusion detection in computer networks by multiple classifier systems, In *Proceedings of Pattern Recognition*, 390-393.

[8] Giacinto, G., Perdisci, R., and Roli, F. 2005. Network intrusion detection by combining one-class classifiers. *Image Analysis and Processing*. LNCS 3617, 58-65.

[9] Melville, P. and Mooney, R.J. 2005. Creating diversity in ensembles using artificial data. *Inform. Fusion*, 6, 1, 99-111.

[10] Vigna, G., Jonsson, E., and Krügel, C. 2003. An analysis of the 1999 Darpa/Lincoln laboratory evaluation data for network anomaly detection. *RAID 2003*. 2820, 220-237.

[11] Saniee, A., Habibi, J., Barzegar, Z., and Sergi, M. 2007. A parallel genetic local search algorithm for intrusion detection in computer networks. *Eng. Appl. Artif. Intell.* 20, 1058-1069

[12] Zainal, A., Maarof, M.A, Shamsuddin, S.M., and Abraham, A. 2008. Ensemble of one-class classifiers for network intrusion detection system. *ISIAS' 08*. 180-185.

[13] Kohavi. R. 1996. Scaling up the accuracy of naive-bayes classifiers: a decision-tree hybrid. *KDD 96*. 202-207.

[14] Perdisci, R., Ariu, D., Fogla, P., Giacinto, G., and Lee, W. 2009. McPAD: A multiple classifier system for accurate payload based anomaly detection. *Comput. Net.* 53, 864-881.

[15] Lloyd. S.P. 1982. Least square quantization in PCM. *IEEE Trans. Inform. Theory.* 28, 2, 129-137.

[16] Quinlan. R. 1993. C4.5: *Programs for Machine Learning*.

[17] Giacinto, G. and Roli, F. 2001. Dynamic classifier selection based on multiple classifier behavior. *Pattern Recogn*. 34.

[18] Ting, K.M. and Witten, I.H. 1997. Stacking bagged and dagged models. *ICML 97* (San Francisco, CA). 367-375.

[19] Elkan, C. 2000. Results of the KDD'99 classifier learning. *SIGKDD Explorations*. 1, 2, 63-64.