Applied Soft Computing 38 (2016) 360–372

Contents lists available at ScienceDirect

Applied Soft Computing

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

A novel SVM-kNN-PSO ensemble method for intrusion detection system

Abdulla Amin Aburomman*, Mamun Bin Ibne Reaz

Department of Electrical, Electronic & Systems Engineering, Faculty of Engineering & Built Environment, National University of Malaysia, 43600 UKM Bangi, Selangor Darul Ehsan, Malaysia

ARTICLE INFO

Article history:
Received 23 August 2014
Received in revised form 6 October 2015
Accepted 6 October 2015
Available online 23 October 2015

Keywords:
Ensemble
k-NN
LUS
PSO
SVM
Weighted majority voting (WMV)

ABSTRACT

In machine learning, a combination of classifiers, known as an ensemble classifier, often outperforms individual ones. While many ensemble approaches exist, it remains, however, a difficult task to find a suitable ensemble configuration for a particular dataset. This paper proposes a novel ensemble construction method that uses PSO generated weights to create ensemble of classifiers with better accuracy for intrusion detection. Local unimodal sampling (LUS) method is used as a meta-optimizer to find better behavioral parameters for PSO. For our empirical study, we took five random subsets from the well-known KDD99 dataset. Ensemble classifiers are created using the new approaches as well as the weighted majority algorithm (WMA) approach. Our experimental results suggest that the new approach can generate ensembles that outperform WMA in terms of classification accuracy.

© 2015 Elsevier B.V. All rights reserved.

1. Introduction

Intrusion detection is a new network security mechanism for detecting, preventing, and repelling unauthorized access to a communication or computer network. Intrusion detection systems (IDS) play a crucial role in maintaining a safe and secure network. The term *anomaly-based intrusion detection* describes a class of techniques that attempt to classify network traffic as either *normal* or *anomalous*. It mainly involves binary classification of selected audit data and other aspects of a system. The success of an intrusion detection system depends on how well it succeeds in maximizing its detection accuracy while minimizing its false alarm rate. Because of their success in practice at spotting new and unfamiliar attacks, anomaly-based intrusion detection systems have remained a heavily researched topic in the IDS community [1].

In recent decades, anomaly-based intrusion detection and many other classification problems have benefited from the idea of combining multiple classifiers. The idea of combining responses produced by multiple classifiers into a single response is known as the *ensemble* approach [2].

Ensemble-based classifiers are well-studied and have been used to improve the accuracy of several classification tasks. Several ensemble methods have been proposed, including mean combiner, median combiner, max combiner, majority voting, and weighed majority voting (WMV). While individual classifiers can be combined using any one of these methods, WMV is by far the most popular among them partly because of its conceptual simplicity, intuitiveness, and its effectiveness in practice [3].

In early research, ensembles were shown empirically and theoretically to possess better accuracy than any single component classifier. An ensemble generated from classifiers trained from the same learning algorithm is termed *homogeneous*, whereas one generated from classifiers trained from different learning algorithms is a *heterogeneous* ensemble. For example, bagging and boosting are often used to generate homogeneous ensembles, whereas stacking can be used to produce heterogeneous ensembles. The success of an ensemble classifier strongly depends on the diversity in the outputs of its component classifiers, as well as on the choice of method to combine these outputs into a single one [4]. Because the selection of a suitable combination method is still poorly understood, several heuristic approaches for combining classifiers have been proposed. Particle swarm optimization (PSO) is one such approach. In 1995, Eberhart and Kennedy proposed the PSO method, which is a stochastic optimization technique that models the analogy of a swarm of birds in flight [5].

In this paper, we define an *expert* as a collection of five binary classifiers that together generate a binary vector of responses. We trained six *k*-nearest neighbor (*k*-NN) and six support vector machine (SVM) experts on the same dataset. We then created three new ensembles using the two new approaches (PSO and meta-optimized PSO) and the weighted majority algorithm (WMA) approach. The ensembles all combine the opinions of the twelve experts to reach the final decision. The twelve experts were created from binary classifiers using different parameters to ensure their diversity. This in turn means that the resulting ensembles inherit this diversity property. We combined the expert opinions in three ways. In this first way, we generate weights using PSO that is constructed with manually selected behavioral parameters. These weights are then used with the weighted majority voting (WMV) to combine the expert opinions. We call this approach the PSO approach. The second way (meta-optimized PSO) is similar to the first approach, except that the PSO behavioral parameters were optimized using Local unimodal sampling (LUS). The third way (WMA approach) is to combine the opinions using the weighted majority algorithm (WMA). Finally, the three approaches were empirically compared. The experimental results based on five randomly selected subsets

* Corresponding author. Tel.: +60 123573020.
E-mail addresses: reoroman@hotmail.com (A.A. Aburomman), mamun.reaz@gmail.com (M.B. Ibne Reaz).

http://dx.doi.org/10.1016/j.asoc.2015.10.011
1568-4946/© 2015 Elsevier B.V. All rights reserved.

of the KDD99 datasets showed that the new method gives better accuracy than weighted majority algorithm (WMA).

The remainder of this paper is organized as follows. Section 2 provides an overview of related work. Section 3 introduces the methodology. Section 4 describes experimental results and discussions. Finally, Section 5 concludes the paper and gives direction for future work.

2. Related work

The application of ensemble techniques to intrusion detection has recently witnessed a surge of research efforts.

Syarif et al. [6] applied bagging, boosting and stacking ensemble techniques to the intrusion detection problem to improve the accuracy and reduce its false positive rates. As base classifiers for these ensemble methods, they used naï ve Bayes, J48 (decision tree), JRip (rule induction) and iBK (nearest neighbor). They report that their approach achieves an accuracy of more than 99% in detecting known intrusions but could only detect novel intrusions with the accuracy rates of around 60%. The use of bagging, boosting, and stacking showed no significant gain in accuracy. While stacking was the only method that led to a significant reduction in false positive rates 46.84%, it also has the longest execution time and thus is too inefficient to be practical for the intrusion detection problem.

Bahri et al. [7] introduced a hybrid approach based on a new ensemble method called Greedy-Boost. They compared experimentally the precision and recall of AdaBoost, C4.5 and Greedy-Boost using the KDD 1999 data set. It was not very clear from the paper what base classifiers were used in the experimental study. Their results indicate that Greedy-Boost outperforms the other algorithms in terms of the precision even for Probe, U2R, and R2L attacks.

Bukhtoyarov et al. [8] applied a probabilistic approach to design base neural network classifiers called probability-based generator of neural networks structures (PGNS) to the network intrusion detection problem. To design neural network ensembles, they employed an approach called Genetic Programming based ENsembling (GPEN). GPEN applies genetic programming operators to find an optimal function for combining the base classifiers into an ensemble. This technique was applied to the KDD Cup 1999 data set with the goal was to classify the input intrusions as PROBE attacks or non-PROBE attacks, using nine of the 41 attributes. They compared the results with those published for other approaches in [9]. The results obtained using their approach show better detection accuracy of PROBE attacks than almost all the competing approaches included in [9]. The only approach that had better detection accuracy and fewer false positives was PSO-RF approach.

Although PSO performs well in many applications, it has not been implemented in solving ensemble configuration problems with respect to intrusion detection before.

Cordeiro and Pappa [10] used the PSO algorithm for weighting classifications coming from different data views. They dubbed this approach PSO-WV. They also modified PSO to provide weights for classes in specific views, by estimating how well a view can predict a class. They called this version PSO-WC. One of the four classification algorithms – KNN, Naive Bayes, Rocchio, and SVM is applied to each view. They conducted experiments using PSO-WV and PSO-WC on two problems: a document classification problem from ACM-DL dataset and classification of users of video social network – YouTube dataset. The comparison of results show that PSO-WV has better results for YouTube dataset, and PSO-WC was better for ACM-DL dataset. Both approaches outperformed a single classifier on both datasets.

Similar approaches have been applied to other problems besides intrusion detection. We found [11] to be the most similar work to ours. The authors proposed an ensemble of classifiers to perform a classification task for four well-known datasets: Hearth, Diabetes, Iris, and Transfusion. An ensemble is generated based on opinions of four expert systems: Linear Discriminant Classifier (LDC), Quadratic Discriminant Classifier (QDC), *k*-nearest neighbor (KNN) and back propagation (BP). The opinion of each expert in the ensemble is weighted by a set of coefficients generated by PSO. The ensemble is created by combining the opinions of the experts, each multiplied by a weight coefficient, with weighted majority voting (WMV). The experimental results indicate the PSO-WMV approach had greater accuracy compared to the other ensemble approaches like Mean, Maximum, Minimum and Median Combiner and simple majority voting. The authors did not provide any implementation details nor specify their PSO objective function.

In this paper, we adopt the same approach for utilizing PSO in majority voting process, but with methodology specifically designed for IDS. We also introduced LUS as a meta-optimizer for PSO, while [11] only used base PSO. Since two papers used different experimental datasets, it is not possible to make a direct comparison of the results.

3. Methodology

3.1. System framework

As stated earlier the objective of this paper is to develop ensemble based classifiers that will improve the accuracy of intrusion detection. For this purpose, we trained and tested twelve experts and then combined them into an ensemble. We used the PSO algorithm to weight the opinion of each expert. Because the quality of the behavioral parameters inserted by the user into PSO strongly affects its effectiveness, we have used the LUS method as a meta-optimizer for finding high-quality parameters. We then used the improved PSO to create new *weights* for each expert. For comparison, we also developed an ensemble classifier with *weights* generated using WMA [12]. Fig. 1 depicts the entire process. For simplicity, the system framework was divided into the following seven stages:

1. Kdd99 data pre-processing.
2. Data classification with six different SVM experts.
3. Data classification with six different *k*-NN experts.
4. Data classification with ensemble classifier based on PSO.
5. Data classification with ensemble classifier based on LUS improvement of PSO.
6. Data classification with ensemble classifier based on WMA.
7. Comparison of results for each approach.

3.2. Dataset used for experiments

We used the knowledge discovery and data mining 1999 (KDD99) dataset [13] in the experiments. The dataset contains hundreds of thousands of connection records. For each TCP/IP connection, 41 different quantitative and qualitative features were extracted. Each single set of 41 features represents one observation that is either in a normal or an intrusion state. Ref. [14] gives a description of all features.

To correctly assess the performance of each classifier we need, at least, two distinct datasets: one for training and one for testing. Optionally a third dataset may be obtained to validate classifiers, and we decided to create a new sub-set especially for validation. All the data comes from [13], and can be divided into three groups:

1. Training data: (kddcup.data_10_ percent.gz), this data is used to train each expert in the ensemble.
2. Testing data: (corrected.gz), this data is used to evaluate the performance of each base classifier in the system, as well as the performance of ensemble classifiers.

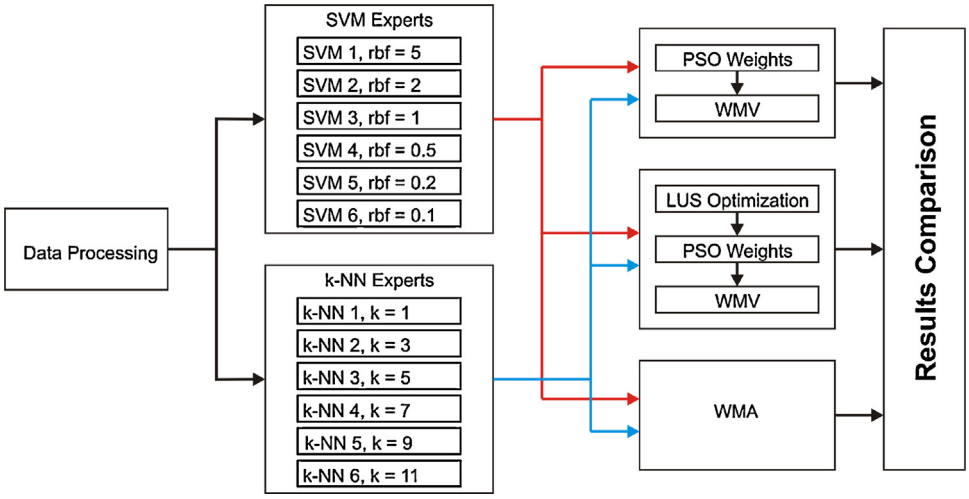

Fig. 1. System framework.

3. Validation data: a unique sub-set created from corrected.gz file. Data used for validation was removed from other datasets, to ensure the independency of the validation process.

The KDD99 dataset is divided into normal traffic and four classes of attacks [15]:

- Denial of Service (DoS): A connections trying to prevent authentic users from getting access to the services in the targeted machine.
- Probe (Scanning): A connections searching potential weaknesses on target machine.
- Remote to Local (R2L): A connections trying to acquire illegal access to a network or computer.
- User to Root (U2R): A connection in which an attacker has already gained access, but is attempting to gain super user privileges.

Table 1 represents the number of observations for each attack sorted in one of the four intrusion states. Testing data introduces some new types of attacks, marked with gray shade. Observations for these attacks were not available during training of each expert. Table 2 provides information about the number of observations in training and testing datasets for each class.

This experiment used five data sets, taken from the training and testing KDD99. As shown in Table 3, all five samples are of the same size, but contain different observations selected at random from the training and testing datasets represented in Table 2.

3.3. Data pre-processing

To use the proposed classification methods we need to ensure that each observation is a set of numeric values. We also need to ensure that each class can be represented as numeric value as well. KDD99 data includes three symbolic features that are incompatible with the proposed classification algorithms:

- Protocol type – this feature represents connection protocol (e.g. tcp, udp,...).
- Service – this feature represents destination service (e.g. telnet, ftp,...).
- Flag – this feature represents status flag of the connection.

We perform data pre-processing in two steps:

- Data mapping: Symbolic values of three features were mapped to numeric values for each observation in training, validation and testing datasets. The value for each of three features is mapped to a numeric value ranging from 1 to N, where N is the total number of symbols for each feature.
- Identification of State. The KDD99 dataset includes a state for each set of features, where the state is either a normal connection or a type of attack as represented in Table 1. This means that each record in the data belongs to one of five major classes: Normal, DoS, Probe, U2R, and R2L. The values for each state are mapped to a numeric value. More specifically the Normal class was mapped to the number 1, Probe to 2, DoS to 3, U2R to 4, and R2L to 5.

3.4. SVM classifier

Support vector machines (SVM) are an effective technique for solving classification and regression problems. SVM is originally an implementation of Vapnik's Structural Risk Minimization (SRM) principle [16], which is known to have low *generalization error* or equivalently does not suffer much from *overfitting* to the training data set. A model is said to overfit or has a high generalization error if it performs poorly on instances not present in the training set. SVM is particularly effective on data sets that are *linearly separable*, i.e. where hyperplane H can be found that partitions the instances into two classes such that instances in one class (almost) entirely fall on one side of H. Since there is an infinite number of candidate hyperplances that can be selected, SVM selects the hyperplane H so that it maximizes its distance to the nearest data points in either class. This is refered to as *margin maximization*. So far, we have only considered the case where the data set is linearly separable. However, for many real-life data sets, such a hyperplane may not exist. In these cases, SVM uses a function to map the data into a different feature space where such separability is then possible. This transformation often comes in the form of mapping to a high-dimensional space. A function used to perform such a transformation is called a *kernel* function. Thus, kernel functions play a pivotal role both in the theory and application of SVM.

The following kernel functions are commonly used along with SVM [17].

- Linear kernel: $k(x_i, x_j) = x_i x_j$
- Polynomial kernel: $k(x_i, x_j) = (y x_i^t x_j + r^d)^2$
- RBF kernel: $k(x_i, x_j) = e^{\gamma \|x_i - x_j\|^2}$
- Sigmoid kernel: $k(x_i, x_j) = \tanh(y x_i^t x_j + r)$

Table 1
Attack distribution.

Class	In training			Testing		Total
	Attack names	Samples	Total	Attack names	Samples	
DOS	teardrop	979	391,458	Apache 2	794	229,853
	smurf	280,790		Back	1098	
	neptune	107,201		land	9	
	Pod	264		mailbomb	5000	
	Back	2203		neptune	58,001	
	Land	21		pod	87	
				processtable	759	
				smurf	164,091	
				teardrop	12	
				udpstorm	2	
Probe	satan	1589	4107	ipsweep	306	4166
	nmap	231		mscan	1053	
	ipsweep	1247		nmap	84	
	portsweep	1040		portsweep	354	
				saint	736	
				satan	1633	
U2R	perl	3	52	buffer_overflow	22	70
	buffer_overflow	30		loadmodule	2	
	rootkit	10		perl	2	
	loadmodule	9		ps	16	
				rootkit	13	
				sqlattack	2	
				xterm	13	
R2L	ftp_write	8	1126	ftp_write	3	16,347
	Warezclient	1020		guess_ passwd	4367	
	Warezmaster	20		imap	1	
	Spy	2		multihop	18	
	guess_ passwd	53		named	17	
	Imap	12		phf	2	
	multihop	7		sendmail	17	
	Phf	4		snmpgetattack	7741	
				Snmpguess	2406	
				warezmaster	1602	
				worm	2	
				xlock	9	
				xsnoop	4	
				httptunnel	158	

To extend SVM to multi-class classification, a set of five binary classifiers are trained, one for each class. Let $i = (1, \ldots, 5)$ be an index into the quintuple T = (Normal, Probe, DoS, U2R, and R2L) and let B_i denote the corresponding binary classifier for the target class i in T. All five binary classifiers were trained using the whole training set, but each for its corresponding target class. In other words, when training the classifier B_i, the label 1 is assigned to observations that belong to class i, and 0 to those that belong to any other class. This is known as the *One-Versus-All* approach for classifying the observations into one of the five classes. To differentiate among the binary classifiers, we introduce term *expert*, to denote one set of five binary classifiers. Fig. 2 illustrates the relationship between binary classifiers and experts and gives the output format of classification for each class.

SVM produces the best results for classification when the RBF kernel function is used [16]. Experimental results have shown that performance of SVM classifiers with RBF kernel function will vary with the selection of RBF function. Therefore, in this paper, we train six different SVM experts with different RBF parameters, to ensure that SVM algorithm is maximally utilized. This approach will also ensure greater diversity of experts in ensemble classifier.

Selected values for RBF parameter are defined by RBF vector with values: RBF = [5 2 1 0.5 0.2 0.1].

Later, it will be demonstrated that accuracy of each binary classifier inside each expert system will vary, according to the selected value in RBF vector. Based on RBF vector six SVM experts are developed as follows:

SVM 1: RBF = 5;
SVM 2: RBF = 2;
SVM 3: RBF = 1;
SVM 4: RBF = 0.5;
SVM 5: RBF = 0.2;
SVM 6: RBF = 0.1.

Table 2
Number of observations in KDD99.

Connection type	Training dataset	Testing dataset
Normal	97,278	60,593
Dos	391,458	229,853
Probe	4107	4166
R2L	1126	16,347
U2R	52	70
Total	494,021	311,029

Table 3
Dataset size used in experiments.

	Normal	DoS	Probe	U2R	R2L	Total
Training	5000	4107	5000	52	1126	15,285
Testing	7500	3124	7500	52	3750	21,926
Validation	2500	1042	2500	18	1250	7310

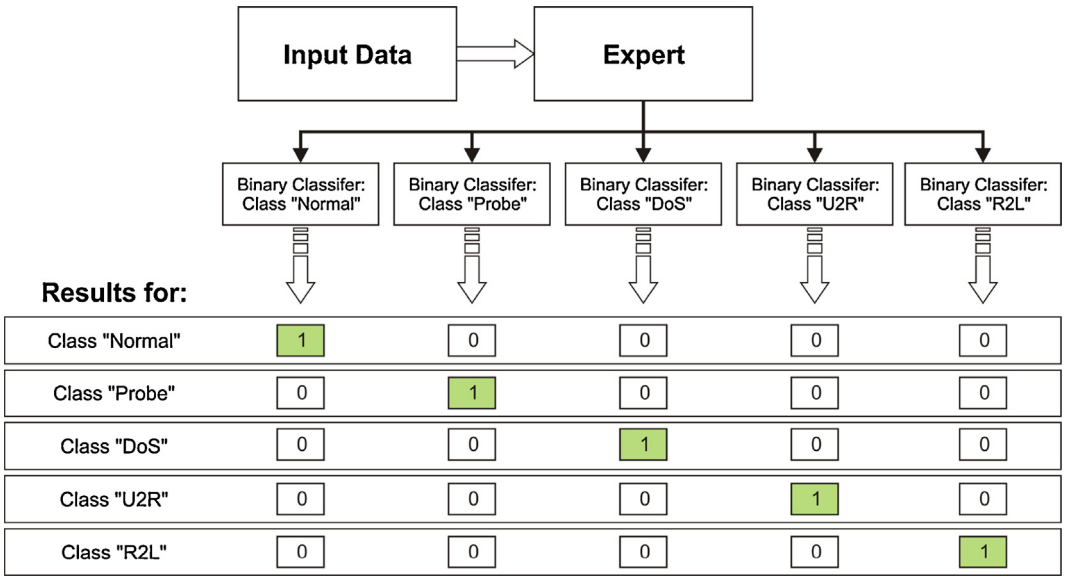

Fig. 2. Structure of an expert system.

3.5. k-NN classifier

The k-nearest neighbour (k-NN) is a simple and effective technique for objects classification according to the closest training examples in the feature space [18]. Consider a set of observations and targets $(x_1, y_1), \ldots, (x_n, y_n)$, where observations $x_i \in \mathbf{R}^d$ and targets $y_i \in \{0, 1\}$; then for a given i, k-NN rates the neighbours of a test sequence among the training sample, and uses the class labels of the nearest neighbours to predict the test vector class. So, k-NN takes the new points and classifies them according to the majority of the votes obtained for the K nearest points in the training data. In k-NN, the Euclidean distance is often used as the distance metric to measure the similarity between two vectors (points):

$$d^2(x_i, x_j) = \|x_i - x_j\|^2 = \sum_{k=1}^{d} (x_{ik} - x_{jk})^2 \quad (1)$$

where $(x_i, x_j) \in \mathbf{R}^d$, $x_i = (x_{i1}, x_{i2}, \ldots, x_{id})$.

Unlike SVM, k-NN classifiers can be used to solve multi-class problems. However, to make k-NN experts and SVM experts compatible, we needed to implement five binary classifiers for this method as well. Therefore, the structure of the k-NN expert system, depicted Fig. 2, is the same as that of the SVM expert. Compatibility between this two approaches enables us to combine both SVM and k-NN experts into an ensemble expert system.

The k parameter of k-NN classifiers represents the number of neighbors in a set of training observations that are nearest to the given observation in validation or testing data set. Variation of this parameter will affect the accuracy of each binary classifier inside an expert.

To ensure greater diversity of classifiers and to maximally utilize potential of k-NN classifier, we have created six k-NN experts with different values of k parameter, defined by k vector:

$$k = [1, 3, 5, 7, 9, 11].$$

By selecting different k parameter we create six k-NN experts as follows:

k-NN 1: $k = 1$;
k-NN 2: $k = 3$;
k-NN 3: $k = 5$;
k-NN 4: $k = 7$;
k-NN 5: $k = 9$;
k-NN 6: $k = 11$.

3.6. Ensemble approach with WMV

The very idea of majority voting is simple and intuitive; first, votes are assigned to each expert's opinion. The opinion with the most votes is accepted as the final decision. Littlestone and Warmuth [12], have shown that we can reduce number of mistakes that ensemble system makes by introducing weights to the majority voting process. A detailed description of methodology and proofs are presented in [12], and we will provide details of WMV implementation for IDS in this section.

In this paper we use voting procedure, as described in [12], to combine opinions of the base experts in an ensemble. Each expert is assigned a weight derived from the expert's accuracy in classifying validation sample. Since each expert consists of five binary classifiers B_i (as shown in Fig. 2), we consider expert's opinion for each class i separately. Based on the output of each binary classifier B_i we can divide expert's opinions in two categories:

- Experts which classify given observation as an instance of class i (output value 1) and
- Experts which claim that given observation belongs to some other class than i (output value 0).

The voting procedure is repeated for each observation x, and for each binary classifier inside expert. As a result, we generate an ensemble expert, with five binary classifiers – one for each class.

This paper presents three methods to generate weight: by using PSO algorithm; by using PSO algorithm with LUS improvements of PSO algorithm and with WMA. PSO and LUS-PSO based ensembles are a novel way to combine expert opinions, for intrusion detection systems.

We define a set of weight coefficients w as a twelve element vector, where each element j represents weight for jth expert in ensemble. i.e. $w = (w_1, w_2, \ldots, w_{12})$.

To define a final decision function, we must first consider how weights are used in the voting process. For a single observation x, we obtain twelve output values $(y_1, y_2 \ldots, y_{12})$, one output value per expert. Each value can be defined as a positive or a negative instance, i.e. $y_j = \{1, -1\}$.where value 1 corresponds to expert's

output 1, and negative value −1 represents expert's output 0. The final decision y is reached by evaluation of Eq. (2).

$$y = sgn\left(\sum_{i=1}^{j=12} w_j \cdot y_j\right) \tag{2}$$

where tie cases, $sgn(0)$, are broken randomly

Each coefficient w_j is multiplied with output from jth expert y_j and final decision is formed by determining the sign of the sum of weight coefficients for all twelve experts.

3.7. PSO algorithm

Particle swarm optimization is a population-based iterative optimization algorithm, formulated by Kennedy and Eberhart [22]. PSO is derivative-free, zero-order method. That means it does not need gradients, so it can be applied to a variety of problems, including those with discontinuous or non-convex and multimodal problems.

The algorithm starts out with a set of agents, called particles, in random positions in the problem space. Each is also assigned random velocity at the outset. A fitness function is defined on a particle's location. The optimization problem to be solved is to find the best position, i.e. the one that minimizes the fitness function. Through each iteration, the algorithm evaluates each particle's fitness, updates its velocity, and computes its new position. A particle's new velocity depends on its current velocity, its distance from its own best position so far and its distance from the populations best position yet.

Compared to genetic algorithms (GA), PSO has no evolution operators such as crossover and mutation which makes it easy to implement with great success to several problems wherever GA can be applied [5].

The mathematical basis of the algorithm [19] is as follows. Let $A \subset \mathbf{R}^n$ be the search space and $f: A \to Y \subseteq \mathbf{R}$ be the objective function. We assume as well that A is also the feasible space of the given problem. In other words there are no more explicit constraints imposed on the candidate solutions. We mentioned earlier that the algorithms analogous to a swarm of solutions which floats through search space. Thus we define the search space to be the set of all the positions of the N particles (candidate solutions) $S = \{s_1, s_2, s_3, \ldots, s_N\}$. Each solution s_i is defined as $s_i = (s_{i1}, s_{i2}, \ldots, s_{in})^T$ for $i = (1, 2, \ldots, N)$. where n is a dimension of the search space.

Indices are arbitrarily assigned to particles, while the number of particles N is a user-defined parameter of the algorithm. The objective function $f(s)$ is assumed to be available for all points in A. Therefore each particle has a unique function value, $f_i = f(s_i) \in Y$.

The particles are assumed to move within the search space A iteratively. This is done by updating their *position* using a proper position shift, named *velocity* and denoted as $v_i = (v_{i1}, v_{i2}, \ldots, v_{in})^T, i = (1, 2, \ldots, N)$.

Velocity is iteratively updated so that every particle can visit any region of A. If t stands for the iteration counter (or time unit), then the current position of the *ith* particle and it's velocity will be denoted as $s_i(t)$ and $v_i(t)$, respectively.

Each *best* position of every particle is stored in a memory array which contains the best positions, $p_i = (p_{i1}, p_{i2}, \ldots, p_{in})^T \in A$, for $i = (1, 2, \ldots, N)$. So in addition to the swarm set S we have the set $P = \{p_1, p_2, \ldots, p_N\}$ for the best positions. These positions are defined as $p_i(t) = \arg\min_t(f_i(t))$. Analogous to the interaction of ants, there is a communication mechanism that shares information of the best position visited by all particles. This position is defined as $p_g = \arg\min_t f(p_i(t))$.

While there are slightly different versions of the PSO update equations, the version we have used in this work is the one defined by the following equations (Eberhart et al. [20]):

$$v_{ij}(t+1) = \omega v_{ij}(t) + \phi_p r_p(p_{ij}(t) - x_{ij}(t)) + \phi_g r_g(p_{gj}(t) - x_{ij}(t)) \tag{3}$$

$$s_{ij}(t+1) = s_{ij}(t) + v_{ij}(t+1) \tag{4}$$

where $i = 1, 2, \ldots, N$ is an index of each particle in the swarm; $j = 1, 2, \ldots, n$ represents jth element of ith particle $s_i(t)$ and velocity $v_i(t)$; r_p, r_g are uniformly distributed stochastic variables; $p_i(t)$ is the particle's best position in previous iteration; $p_g(t)$ is the best position in whole swarm for previous iteration.

Factors: ω, ϕ_p, and ϕ_g are user defined behavioral parameters, and they represent weight or inertia quotient, particle acceleration and swarm acceleration respectively.

The pseudocode of the PSO algorithm is defined as:

Algorithm 1. PSO algorithm

```
procedure PSO
  for particle i ∈ {1, 2, ..., N} do
    for dimension j ∈ {1, 2, ..., n} do
      set s_ij ~ U(lowerBoundary_j, upperBoundary_j)
      set d_j ← |upperBoundary_j − lowerBoundary_j|
      set v_ij ~ U(−d_j, d_j)
      set p_ij ← s_ij
      if f(p_ij) < f(p_gj) then
        set p_gj ← p_ij
  for timestep t ∈ {1, 2, ..., I_max} do
    for particle i ∈ {1, 2, ..., N} do
      set r_p ~ U(0, 1)
      set r_g ~ U(0, 1)
      for dimension j ∈ {1, 2, ..., n} do
        update v_ij, s_ij from (3), (4)
        if f(s_ij) < f(p_ij) then
          set p_ij ← s_ij
          if f(p_ij) < f(p_gj) then
            set p_gj ← p_ij
  print best solution p_gj
```

Stopping criteria is met when iteration t reaches the maximal allowed number of iterations I_{max}. A set of particles $s_g(t = I_{max})$ with minimal value of fitness function is declared to be the optimal solution. In this paper, particle with best fitness $s_g = (s_{g1}, s_{g2}, \ldots, s_{gn})$ will represent the optimal set of weight coefficients $w = (w_1, w_2, \ldots, w_n)$, where n represents number of expert in ensemble. Therefore, each particle s represents a potential set of weight coefficients w.

For each observation x in the sample, we evaluate the Eq. (2) in order to obtain class predicted by voting algorithm y. The correct class (target) T is provided for each observation in the training set. In a training set of a size m, we define a number of correctly classified observations c, as each instance in training sample for which the predicted output y is same as a target T, i.e. $y = T$. The accuracy for generated set of weights $ACC(w)$ is a fraction of correctly classified observations, i.e. $ACC(w) = \frac{c}{m}$, where m is a total number of observations from validation sample.

To achieve better performance of ensemble classifiers we need to maximize accuracy or minimize the error for each particle in the swarm. Therefore we define our fitness function, that is to be minimized, as:

$$ACC(w) = 1 - \frac{c}{m} \tag{5}$$

Weights are generated for each class separately. The ensemble classifier created with PSO weights will have same structure as an expert, illustrated with Fig. 2. Consequently, we need to generate five sets of weights with PSO, one for each binary classifier in base expert.

It is important to note that the validation data is used to generate weights. This is necessary because we are not able to correctly

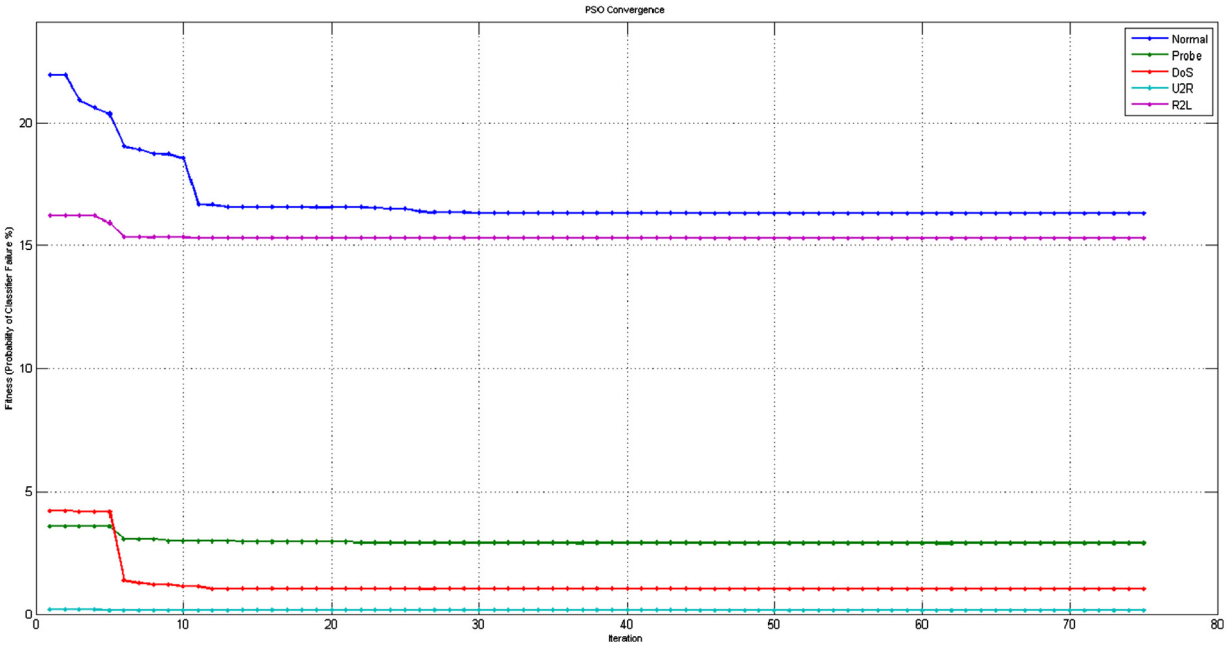

Fig. 3. Convergence of PSO algorithm.

evaluate accuracy of each classifier with the training data. Evaluating model performance with the data used for training is not acceptable because this will result in strongly biased weights and can easily generate overfitted model. To determine fitness value we cannot use testing data either, since testing targets must be used only to evaluate the performance of each expert system, whether it is a base classifier or an ensemble. To generate the validation dataset, we took a subset from the corrected.gz file which is used for testing and this subset was removed from all datasets used in the testing process. In such way we insure the independency of the validation process.

Fig. 3 represents convergence of PSO algorithm towards optimal solution, for each of our five classes.

3.8. LUS algorithm

As stated earlier, the number of PSO parameters are defined by the user. These parameters are:

N – number of particles,
I_{max} – number of iterations of PSO algorithm,
ω – weight or inertia quotient,
ϕ_p – particle acceleration and
ϕ_g – swarm acceleration.

A user may choose to impose limits for the minimal and maximal value of weights and velocities for PSO algorithm. The performance of PSO is strongly influenced by selection of these parameters. Although it is possible to determine them by trial-and-error, it is more effective to select parameters by adding a layer of optimization. The structure of meta-optimization algorithm for PSO is shown in Fig. 4.

Algorithm used as a meta-optimizer is a method called local unimodal sampling (LUS). As explained in Pedersen [21], this method does not require evaluation of the gradient, and it is derivative-free. Instead it uses a sampling method to find the optimum choice of parameters by gradually adapting the search-range through iteration.

The basic idea behind LUS is to decrease iteratively search range d_i for each parameter i until the optimal solution is reached. We define search range as $d = (1, 2, \ldots, n)$, where n is the total number of parameter to be optimized. In our case $n = 5$. Initial search range, at timestep $t = 0$, is derived from user defined upper and lower boundary b_{up} and b_{low}, respectively.

$$\vec{d} \leftarrow \vec{b_{up}} - \vec{b_{low}} \tag{6}$$

The optimal solution s is defined in similar fashion as: $s = (s_1, s_2, \ldots, s_n)$ and, at timestep $t = 0$, value of each parameter is initialized to uniformly distributed random number with value that is inside search range boundaries:

$$\vec{s} \sim U(\vec{b_{low}}, \vec{b_{up}}) \tag{7}$$

At this stage, the fitness function is evaluated for the initial set of parameters s. As with PSO, the objective is to find set of weights that will maximize accuracy for each class in the ensemble. Therefore the fitness function of both PSO and LUS is defined by Eq. (5). Unlike in [21], we decided to set initial fitness value to the fitness of PSO weights that were derived with base PSO, with user set PSO parameters. Thus, we ensure that LUS will generate set of weights with better accuracy than those generated by PSO.

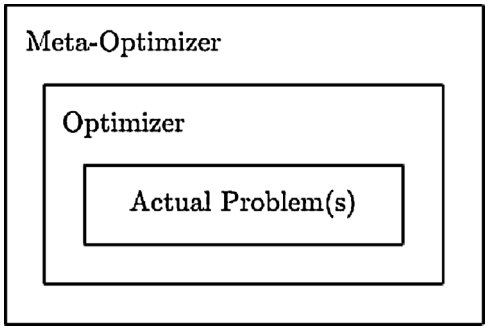

Fig. 4. Structure of the meta-optimization algorithm.

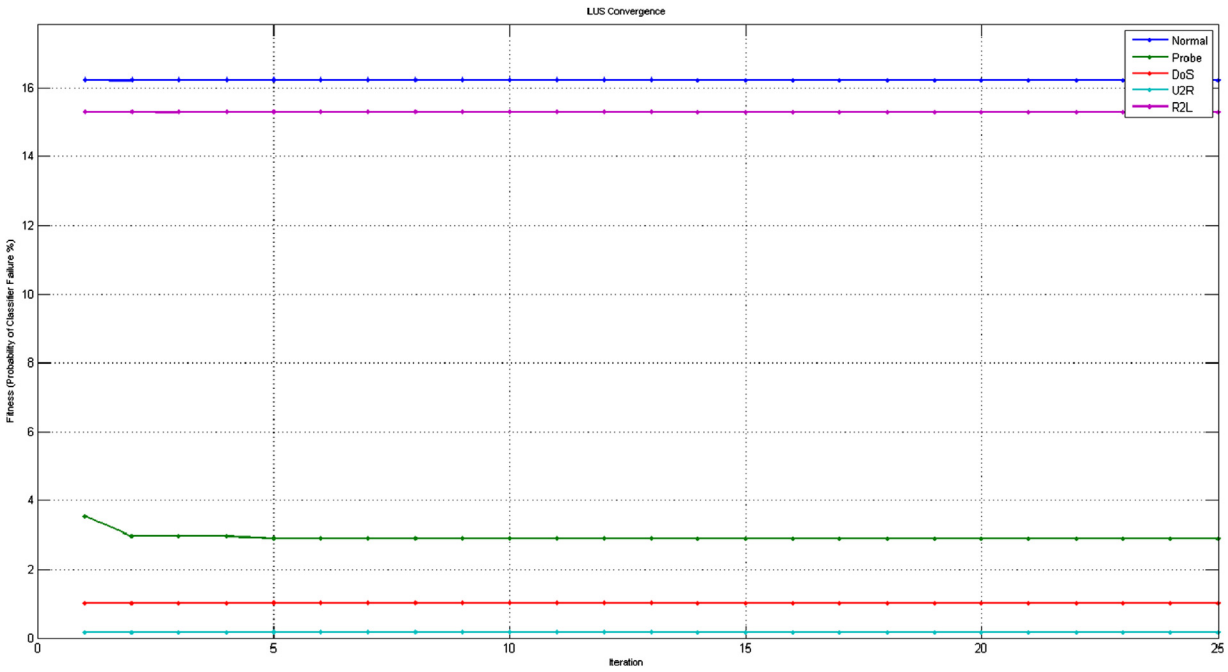

Fig. 5. Convergence of LUS algorithm.

After the initial fitness is evaluated, we generate a new set of PSO parameters s_{new}, as defined by formula (8).

$$\vec{s}_{new} = \vec{s} + \vec{a} \tag{8}$$

where $a = (a_1, a_2, \ldots, a_n)$, is a vector of uniformly distributed random values inside search range d, i.e. $\vec{a} \sim (-\vec{d}, \vec{d})$

A Fitness value is calculated using the new set of parameters s_{new} and value is compared to the initial fitness. If the new set of

Table 4
Experimental results for SVM experts.

Expert	Normal	Probe	DoS	U2R	R2L
Dataset 1					
SVM 1: RBF = 5	75.9418%	96.2191%	93.9706%	98.6819%	84.621%
SVM 2: RBF = 2	76.5347%	95.2066%	95.5943%	99.5166%	83.832%
SVM 3: RBF = 1	81.4056%	94.8144%	98.577%	99.6169%	83.8001%
SVM 4: RBF = 0.5	76.895%	94.3993%	98.3399%	99.6853%	83.4626%
SVM 5: RBF = 0.2	72.9773%	93.6651%	94.7095%	99.74%	83.4215%
SVM 6: RBF = 0.1	69.8531%	92.2284%	93.697%	99.7492%	83.1433%
Dataset 2					
SVM 1: RBF = 5	76.667%	96.438%	94.6547%	98.5451%	84.662%
SVM 2: RBF = 2	76.781%	95.3662%	98.5086%	99.4071%	83.9141%
SVM 3: RBF = 1	80.229%	94.8053%	98.796%	99.5667%	83.8548%
SVM 4: RBF = 0.5	77.2827%	94.4814%	98.7914%	99.6534%	83.4626%
SVM 5: RBF = 0.2	75.2622%	93.7061%	96.9716%	99.7492%	83.4124%
SVM 6: RBF = 0.1	71.7185%	92.3561%	95.2568%	99.7537%	83.2528%
Dataset 3					
SVM 1: RBF = 5	76.0695%	96.1507%	93.9387%	98.4676%	84.4887%
SVM 2: RBF = 2	76.4572%	95.0926%	98.6409%	99.4983%	83.7727%
SVM 3: RBF = 1	78.0398%	94.6776%	98.723%	99.6534%	83.7818%
SVM 4: RBF = 0.5	76.8084%	94.3401%	97.4824%	99.7081%	83.4489%
SVM 5: RBF = 0.2	71.4449%	93.6103%	93.6149%	99.7446%	83.4124%
SVM 6: RBF = 0.1	68.9501%	92.2786%	93.0813%	99.7583%	83.1479%
Dataset 4					
SVM 1: RBF = 5	75.5997%	96.3012%	93.3823%	98.6591%	84.5207%
SVM 2: RBF = 2	75.8597%	95.0424%	94.3674%	99.5029%	83.8457%
SVM 3: RBF = 1	78.6555%	94.6684%	95.5031%	99.6215%	83.7362%
SVM 4: RBF = 0.5	76.5256%	94.3309%	96.4015%	99.6671%	83.4306%
SVM 5: RBF = 0.2	71.3445%	93.6605%	93.4826%	99.7309%	83.385%
SVM 6: RBF = 0.1	69.014%	92.3333%	93.0265%	99.7537%	83.1524%
Dataset 5					
SVM 1: RBF = 5	76.635%	96.5156%	93.9752%	98.5588%	84.4386%
SVM 2: RBF = 2	76.4435%	95.3434%	97.7105%	99.4983%	83.8457%
SVM 3: RBF = 1	79.8778%	94.8645%	98.8279%	99.6123%	83.7955%
SVM 4: RBF = 0.5	77.2553%	94.445%	98.723%	99.6807%	83.4762%
SVM 5: RBF = 0.2	75.3535%	93.7335%	97.2453%	99.7446%	83.4261%
SVM 6: RBF = 0.1	72.0104%	92.347%	95.4255%	99.7492%	83.1752%

Table 5
Experimental results for k-NN experts.

Expert	Normal	Probe	DoS	U2R	R2L
Dataset 1					
k-NN 1: $k=1$	81.1685%	96.3331%	97.861%	99.6944%	83.2756%
k-NN 2: $k=3$	81.0362%	95.8725%	97.8427%	99.7355%	83.2756%
k-NN 3: $k=5$	76.3204%	95.8451%	93.4917%	99.7902%	83.2117%
k-NN 4: $k=7$	76.3705%	95.8816%	92.8715%	99.7856%	82.8195%
k-NN 5: $k=9$	76.3295%	95.9774%	92.9673%	99.7948%	82.8195%
k-NN 6: $k=11$	76.2565%	96.0504%	93.0357%	99.7948%	82.8058%
Dataset 2					
k-NN 1: $k=1$	81.1913%	96.3468%	97.7105%	99.6215%	83.2528%
k-NN 2: $k=3$	80.8401%	95.9546%	97.7333%	99.6944%	83.2938%
k-NN 3: $k=5$	80.4296%	95.7904%	97.4733%	99.7446%	83.2391%
k-NN 4: $k=7$	79.0249%	95.8634%	96.5429%	99.7765%	82.8423%
k-NN 5: $k=9$	79.1526%	95.9135%	93.2774%	99.772%	82.8423%
k-NN 6: $k=11$	80.7352%	96.0732%	92.8167%	99.772%	82.8149%
Dataset 3					
k-NN 1: $k=1$	80.9222%	96.1689%	97.5828%	99.6853%	83.1798%
k-NN 2: $k=3$	80.8811%	95.8269%	97.8245%	99.7172%	83.1935%
k-NN 3: $k=5$	81.0818%	95.7949%	97.2225%	99.7674%	83.1387%
k-NN 4: $k=7$	75.96%	95.8907%	92.2603%	99.7948%	82.7921%
k-NN 5: $k=9$	75.8369%	96.0184%	92.3424%	99.7993%	82.7784%
k-NN 6: $k=11$	75.8551%	96.1689%	92.4792%	99.7993%	82.7693%
Dataset 4					
k-NN 1: $k=1$	81.433%	96.438%	97.3%	99.6944%	83.2026%
k-NN 2: $k=3$	80.2153%	95.8725%	96.2328%	99.7355%	83.2619%
k-NN 3: $k=5$	78.9337%	95.8041%	95.3115%	99.7993%	83.1661%
k-NN 4: $k=7$	75.6773%	95.7813%	92.4154%	99.7993%	82.8012%
k-NN 5: $k=9$	75.7001%	95.9409%	92.5021%	99.7993%	82.8012%
k-NN 6: $k=11$	75.7229%	96.0914%	92.5978%	99.7993%	82.7921%
Dataset 5					
k-NN 1: $k=1$	81.1183%	96.0777%	97.7105%	99.6351%	83.271%
k-NN 2: $k=3$	76.2975%	95.9135%	93.0083%	99.6762%	83.312%
k-NN 3: $k=5$	76.0376%	95.6536%	93.1588%	99.7537%	83.2938%
k-NN 4: $k=7$	76.2109%	95.7858%	92.4884%	99.8039%	82.856%
k-NN 5: $k=9$	76.3477%	95.9455%	92.6252%	99.8084%	82.8377%
k-NN 6: $k=11$	76.2793%	96.1598%	92.6161%	99.8084%	82.8195%

parameters produces weights with greater accuracy, we adopt this set as the current optimal solution, i.e. $s = s_{new}$. Otherwise, we decrease search range with Eq. (9), and determine a new set of parameters s_{new}, for decreased search range d.

$$\vec{d} = q\vec{d} \tag{9}$$

We define decrease factor q as:

$$q = \left(\frac{1}{2}\right)^{\beta/n} \tag{10}$$

where: β is a user defined behavior parameter and n is a number of PSO parameters that are being optimized.

Table 6
Experimental results for ensemble experts.

Expert	Normal	Probe	DoS	U2R	R2L
Dataset 1					
PSO based	83.458%	96.1416%	98.8461%	99.8084%	84.7396%
LUS based	83.5036%	96.8759%	98.8461%	99.8084%	84.7259%
WMA based	65.794%	96.2191%	98.577%	99.7948%	82.897%
Dataset 2					
PSO based	82.8104%	96.9443%	98.9191%	99.7674%	84.9676%
LUS based	83.1114%	96.9078%	98.9145%	99.7674%	84.9676%
WMA based	65.794%	96.3468%	98.796%	99.772%	82.897%
Dataset 3					
PSO based	83.8457%	96.6159%	98.8917%	99.7993%	84.6666%
LUS based	83.8776%	96.7025%	98.8917%	99.7993%	84.6666%
WMA based	65.794%	96.1507%	98.723%	99.7948%	82.897%
Dataset 4					
PSO based	83.9642%	96.6341%	98.6409%	99.8176%	84.7259%
LUS based	83.9642%	96.7527%	98.7184%	99.8176%	84.6894%
WMA based	65.794%	96.438%	97.3%	99.7993%	82.897%
Dataset 5					
PSO based	83.9825%	97.0492%	98.8963%	99.8039%	84.7578%
LUS based	83.9825%	97.0492%	98.8963%	99.8221%	84.7578%
WMA based	65.794%	96.0777%	98.8279%	99.8084%	82.897%

Table 7
Average results for 5 datasets.

Expert	Normal	Probe	DoS	U2R	R2L
SVM 1	76.1826%	96.3249%	93.9843%	98.5825%	84.5462%
SVM 2	76.4152%	95.2102%	96.9643%	99.4846%	83.8420%
SVM 3	79.6415%	94.7660%	98.0854%	99.6142%	83.7937%
SVM 4	76.9534%	94.3993%	97.9476%	99.6789%	83.4562%
SVM 5	73.2765%	93.6751%	95.2048%	99.7419%	83.4115%
SVM 6	70.3092%	92.3087%	94.0974%	99.7528%	83.1743%
k-NN 1	81.1667%	96.2729%	97.6329%	99.666%	83.2364%
k-NN 2	79.8540%	95.8880%	96.5283%	99.7118%	83.2674%
k-NN 3	78.5606%	95.7776%	95.3316%	99.7710%	83.2099%
k-NN 4	76.6487%	95.8407%	93.3157%	99.7920%	82.8222%
k-NN 5	76.6734%	95.9591%	92.7429%	99.7948%	82.8158%
k-NN 6	76.9698%	96.1087%	92.7091%	99.7948%	82.8003%
PSO based	83.6121%	96.6770%	98.8388%	99.7993%	84.7715%
LUS based	83.6878%	96.8576%	98.8534%	99.8029%	84.7615%
WMA based	65.794%	96.2464%	98.4448%	99.7939%	82.897%

The process is repeated iteratively until the maximal number of iterations maxEval for LUS algorithm is reached, or until accuracy for generated set of weights is 100%.

We define the LUS algorithm as:

Algorithm 2. Meta-PSO algorithm

```
procedure LUS
  set q ← 2^(−β/n)
  for dimension j ∈ {1, 2, ..., n} do
    set s_j ~ U(lowerBoundary_j, upperBoundary_j)
    set d_j ← |upperBoundary_j − lowerBoundary_j|
    while eval < maxEval and fitness > acceptFitness do
      set a_j ~ U(−d_j, d_j)
      set y_j ← s_j + a_j
      if f(y_j) < f(s_j) then
        set s_j ← y_j
      else
        set d_j ← q · d_j
```

The LUS algorithm does not generate weights directly. Instead, it uses PSO to generate weights by optimizing the PSO parameters. This means that LUS will produce results in the same format as PSO, i.e. five sets of weights will be generated, one for each class. The convergence of LUS algorithm for each class is presented in Fig. 5.

As with PSO, LUS also uses validation sample to obtain optimal weights.

3.9. Weighted majority algorithm

Weighted majority algorithm was first introduced in [12], and since then it has gained popularly as an effective method for combining classifiers into an ensemble. To compare the efficiency of developed methods, we have also implemented WMA to generate a set of weights in same way as PSO and LUS. Same as with PSO and LUS, WMA uses validation sample to obtain weights for each classifier. All opinions are awarded initial weight with value 1, and opinions are combined with WMV algorithm, as described earlier. The predicted class is then compared with the target provided for the validation process, one observation at a time. All base experts that made a mistake have their initial weight divided by learning factor β. Learning factor β is user defined parameter, and it may take values in range $0 < \beta < 1$. The process is repeated for each observation in validation sample.

It was proved in [12] that after each trial in which a mistake occurs the sum of the weights is at most u times the sum of the weights before the trial W_{init}, for some $u < 1$. If the initial total weight

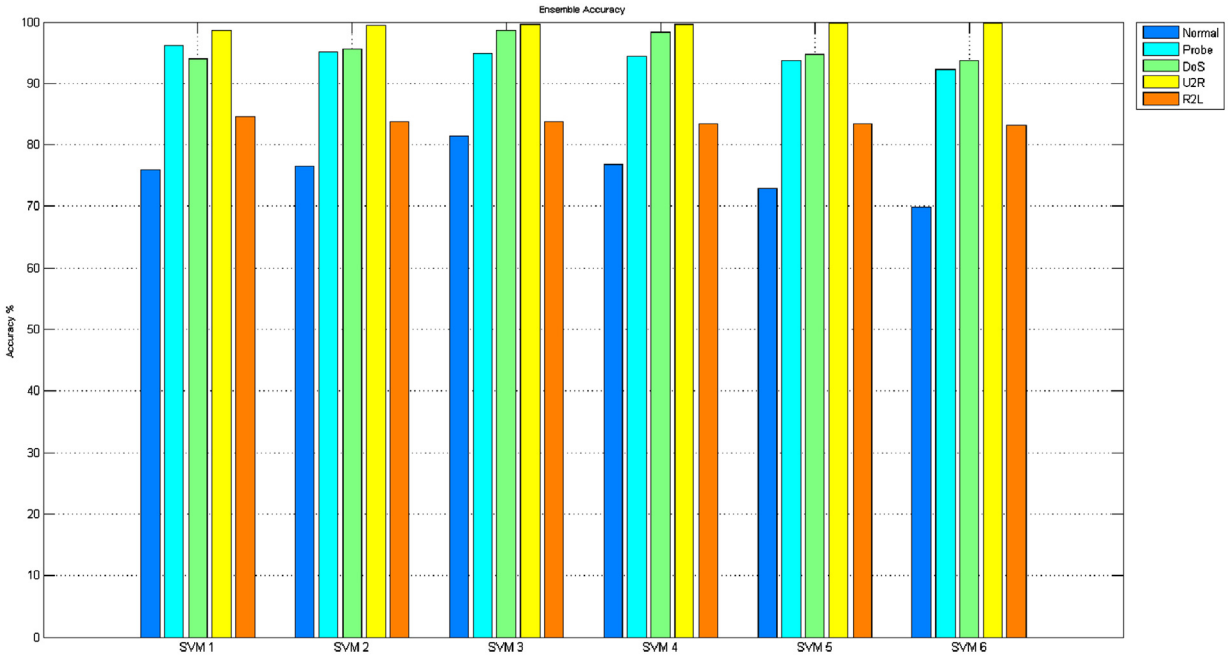

Fig. 6. Accuracy of SVM experts.

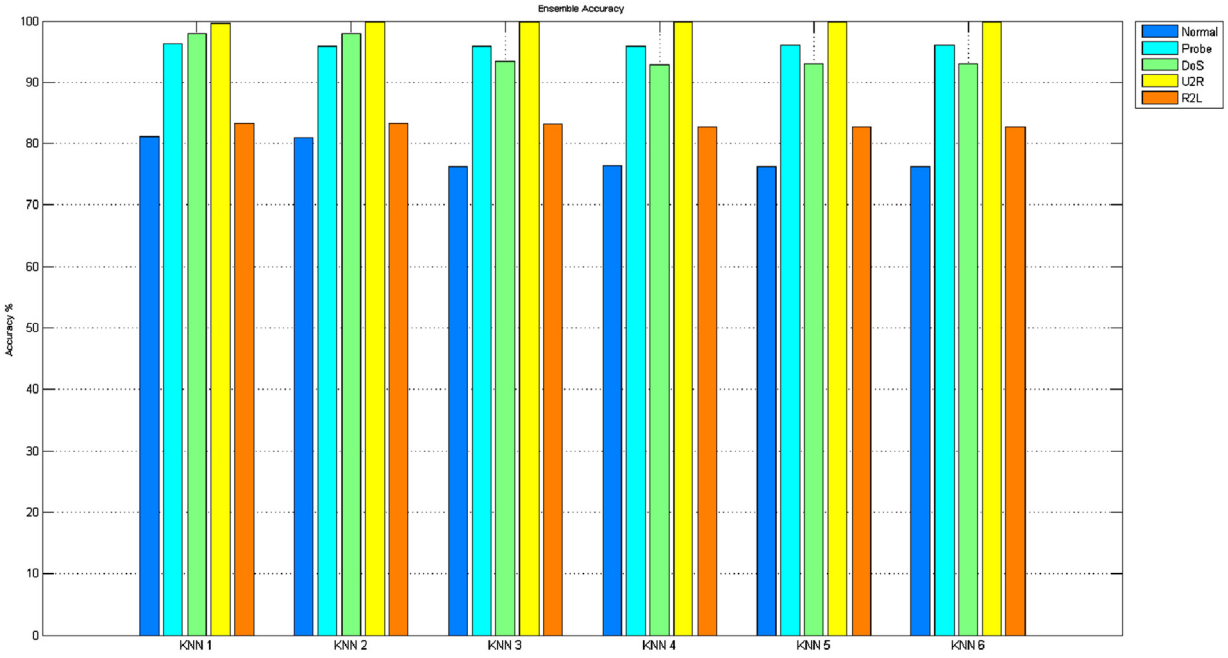

Fig. 7. Accuracy of k-NN experts.

is W_{init} and the lower bound for the total final weight is W_{fin}, then $W_{init}u^m \geq W_{fin}$ must hold, where m is the number of mistakes.

$$m \leq \frac{\log(W_{init}/W_{fin})}{\log(1/u)} \tag{11}$$

WMA is defined by the following pseudocode:

Algorithm 3. WMA

```
procedure WMA
  set W ← W_init
  for i ∈ {1 . . . length(data)} do
    set Q_1 ← Sum(result(i, ) == 1)
    set Q_0 ← Sum(result(i, ) == 0)
    if Q_0 > Q_1 then
      set Q ← 0
    else if Q_0 < Q_1 then
      set Q ← 1
    else
      if rand() < 0.5 then
        set Q ← 0
      else
        set Q ← 1
    if Q == data(i) then
      set W(result(i, ) ! =0) ← βW(result(i, ) ! =0)
      set W(result(i, ) == 0) ← βW(result(i, ) == 0)
  return W
```

4. Experimental results

This study was conducted with Matlab-2012b 64bit, installed on windows 7 professional 64bit with an Intel Core i7 Processor (8M Cache, up to 3.90 GHz), and 16 GB of RAM.

Training, validation and testing samples were drawn from five datasets introduced in Table 1. In order to compare efficiency of proposed algorithms, we establish two characteristics as a comparison basis:

- **Elapsed time**: The time it takes an algorithm to complete a given task and
- **Classification accuracy**: Proportion of the test data correctly classified by an algorithm.

A classification method cannot be considered superior to other methods if it requires a great deal of time to yield relatively small improvements. Since the proposed methods for generating weights with PSO and improving PSO with LUS are novel *in the context of intrusion detection*, we need to measure the time for these methods to accomplish the given task to characterize their efficiency precisely. Measured times are compared to the time taken by WMV method as defined in [12].

More relevant characteristic of the proposed method is the classification accuracy. We define a classifier's accuracy as the number of correctly classified observations C_i of class i divided by the total number of observations s. As depicted in Fig. 2, each expert consists of five binary classifiers. Each binary classifier B_i is used to classify instances of class i from the set of observations $O = O_1, O_2, \ldots, O_s$. Therefore, we determine accuracy A_i of each binary classifier according to Eq. (12).

$$A_i = \frac{C_i}{S} \tag{12}$$

We consider the accuracy of each binary classifier separately. To compare ensemble approaches more easily, we decided to use the average score from each binary classifier. Our objective is to evaluate the efficiency of each classifier – base and ensemble alike. To do so we define an expert's accuracy E_i using Eq. (13).

$$E_i = \frac{\sum_{i=1}^{i=n} A_i}{n} \tag{13}$$

where $n = 5$ is the number of classes. Calculated value is used only as a point of comparison between different experts.

Experimental results, for each data set, are represented separately for SVM, k-NN and ensemble experts in Tables 4–6, respectively.

Table 7 contains averaged results from all 5 datasets, presented for each expert in system.

Times required to complete classification, for each ensemble method, with data drawn from 5 datasets, are represented in Table 8.

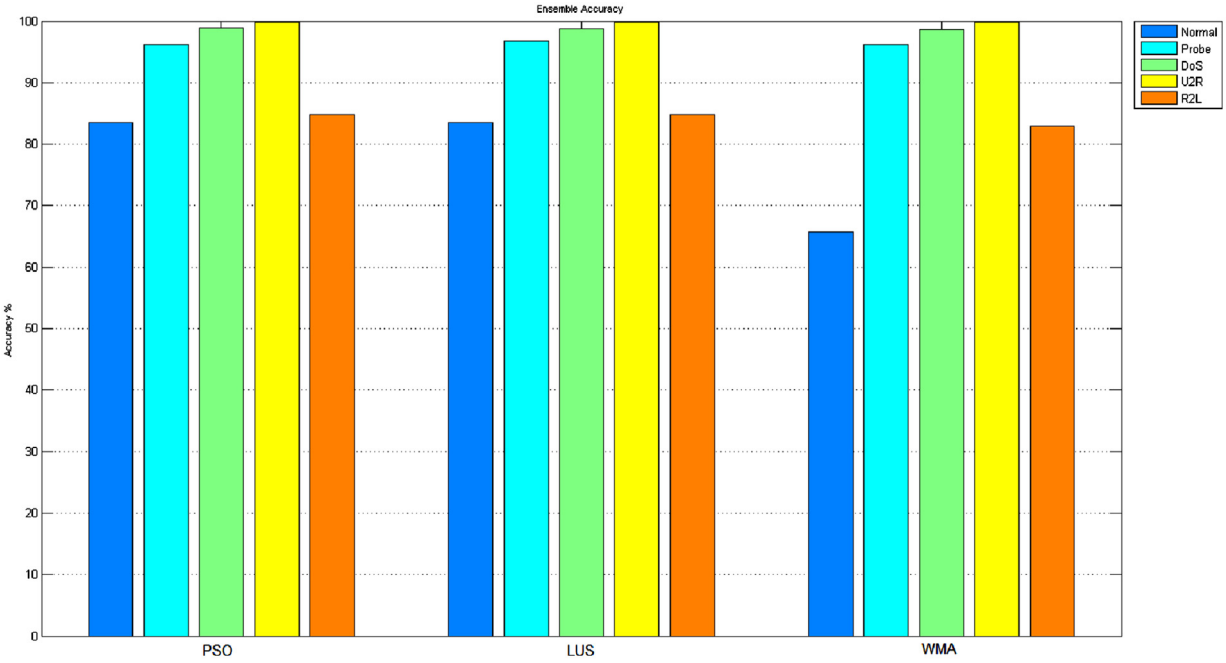

Fig. 8. Accuracy of ensemble experts.

Table 8
Elapsed time for ensemble experts.

Dataset	PSO based	LUS based	WMA based
Dataset 1	7.162 s	3569.219 s	0.384 s
Dataset 2	7.201 s	4234.299 s	0.532 s
Dataset 3	7.089 s	2882.458 s	0.459 s
Dataset 4	7.022 s	3285.989 s	0.403 s
Dataset 5	6.962 s	3997.482 s	0.361 s

Figs. 6–8 are graphical representations of accuracies of SVM, *k*-NN and ensemble experts, respectively. In these figures, we depict accuracy of each expert for dataset 1.

4.1. Discussion

For each base and ensemble expert, we have calculated average accuracy of expert's binary classifiers, and results are presented in Table 9.

Note that accuracy for base experts is relatively high. The accuracy of base experts varied in correspondence to selected value of *RBF* or *k* parameters and obtained expert's accuracies are in the range from 87.44% to 91.67%. Well-selected training data, good choice of *RBF* and *k* parameters and wide range of variation in selected parameters are some of the factors that produced high accuracies of base classifiers.

Somewhat poorer results were obtained when instances of class Normal were classified, with the accuracy of base classifiers going as low as 68.95%. A similar trend, albeit less apparent, can be observed with instances of class R2L. This occurs because the test data set contain observations that were not used in training the classifiers.

When considering WMA method, we observe strikingly low average accuracy. The lower average accuracy of WMA is due to the relatively low accuracy of base classifiers for instances of classes Normal and R2L. WMA method was proven unable to overcome difficulties presented by variations between testing and training data for these classes and performs poorer than base classifier with the lowest accuracy. For other classes, the performance of WMA is significantly better, with accuracy being slightly less than those of the PSO and LUS methods.

The weights generated by WMA are constrained to lie between 0 and 1. Unlike WMA, weights generated by PSO do not have such restrictions and can take on a positive or negative value. This enables us to discredit experts with low accuracy more easily.

Table 9
Experimental results for ensemble experts

Experts	Dataset 1	Dataset 2	Dataset 3	Dataset 4	Dataset 5
SVM 1	89.8869%	90.1934%	89.823%	89.6926%	90.0246%
SVM 2	90.1368%	90.7954%	90.6923%	89.7236%	90.5683%
SVM 3	91.6428%	91.4503%	90.9751%	90.4369%	91.3956%
SVM 4	90.5564%	90.7343%	90.3576%	90.0711%	90.716%
SVM 5	88.9027%	89.8203%	88.3654%	88.3207%	89.9006%
SVM 6	87.7342%	88.4548%	87.4432%	87.456%	88.5415%
k-NN 1	91.6665%	91.6246%	91.5078%	91.6136%	91.5625%
k-NN 2	91.5525%	91.5032%	91.4886%	91.0636%	89.6415%
k-NN 3	89.7318%	91.3354%	91.4011%	90.6029%	89.5795%
k-NN 4	89.5457%	90.81%	89.3396%	89.2949%	89.429%
k-NN 5	89.5777%	90.1916%	89.3551%	89.3487%	89.5129%
k-NN 6	89.5886%	90.4424%	89.4144%	89.4007%	89.5366%
PSO based	92.5987%	92.6817%	92.7638%	92.7565%	92.8979%
LUS based	92.752%	92.7337%	92.7876%	92.7885%	92.9016%
WMA based	88.6564%	88.7212%	88.6719%	88.4457%	88.681%

Simultaneously we can reinforce opinions given by more reliable experts; which provides us with better chances to reach a correct final decision.

An improvement of PSO parameters is obtained by performing LUS meta-optimization; however expected improvement of accuracy is insignificant, especially when considering the exceptionally long time it takes LUS to complete.

5. Conclusion

Through experimental results, we have demonstrated that classification accuracy can be improved by combining opinions from multiple experts into one using an ensemble approach. We have used weighted majority voting (WMV) to combine results from different experts. We combined the expert opinions using the POS, meta-optimized PSO, and the WMA approaches. The three approaches were empirically compared using the KDD99 datasets. We showed empirically that the new method gives better accuracy than WMA.

The best results we obtained were with PSO. On average we have accuracy improvement of 0.756 % compared to the accuracy of the best base expert. If we consider the size of whole testing data, i.e. 311,029 observations, then we can expect better classification results for some 2351 observations. We can also expect a relatively short time for PSO based ensemble to complete its task. The success of PSO-based ensemble can be attributed to the sets of generated weights, which were further optimized to produce results with best possible accuracy.

Combining base experts into an ensemble expert with the methodology proposed in [12] has not proved to be very successful for intrusion detection systems. Very short running time of the approach has little value given its poor accuracy. The poor accuracy of WMA is chiefly due to differences between training and testing sets. This poor accuracy may be corrected by defining different parameters for WMA. Even then, we could not hope to match the PSO results since weights generated by WMA would be strictly positive values.

The accuracy gains contributed by LUS come at the cost of much longer runtimes. Although we have achieved an average improvement of 0.0529% compared to PSO, it took, on average, 500 times longer to achieve this improvement. Consequently, LUS has not proven to be an effective method to combine base classifiers into an ensemble for intrusion detection system.

So far our work was based on binary classification methods, which can distinguish between two states. In case of conflict between binary classifiers final decision is reached by comparing their accuracy. An alternative solution is possible by developing multi-class classification methods to serve as the base experts. A new method would be required to combine such classifiers into an ensemble. Such a method could be the subject of future work. In conclusion, we can state that weights generated with meta-heuristic algorithms can yield improved accuracy for intrusion detection systems. As is frequently the case with randomized algorithms, we have observed some variance in classification accuracy for different datasets. This suggests that further improvement could be achieved by implementing different optimization algorithms also based on meta-heuristics. Therefore, the goal of our future work would be to implement and compare different optimization algorithms for generating weights.

References

[1] X.S. Gan, J.S. Duanmu, J.F. Wang, W. Cong, Anomaly intrusion detection based on PLS feature extraction and core vector machine, Knowl. Based Syst. 40 (2013) 1–6.
[2] S. Peddabachigari, A. Abraham, C. Grosan, J. Thomas, Modeling intrusion detection system using hybrid intelligent systems, J. Netw. Comput. Appl. 30 (1) (2007) 114–132.
[3] Y. Mehmood, M. Ishtiaq, M. Tariq, M.A. Jaffar, Classifier ensemble optimization for gender classification using genetic algorithm, in: 2010 International Conference on Information and Emerging Technologies (ICIET), IEEE, 2010, pp. 1–5.
[4] Y. Chen, M.-L. Wong, H. Li, Applying Ant Colony Optimization to configuring stacking ensembles for data mining, Expert Syst. Appl. 41 (6) (2014) 2688–2702.
[5] J. Kennedy, R.C. Eberhart, Swarm Intelligence, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.
[6] I. Syarif, E. Zaluska, A. Prugel-Bennett, G. Wills, Application of bagging, boosting and stacking to intrusion detection, in: Machine Learning and Data Mining in Pattern Recognition, Springer, 2012, pp. 593–602.
[7] E. Bahri, N. Harbi, H.N. Huu, Approach based ensemble methods for better and faster intrusion detection, in: Computational Intelligence in Security for Information Systems, Springer, 2011, pp. 17–24.
[8] V. Bukhtoyarov, V. Zhukov, Ensemble-distributed approach in classification problem solution for intrusion detection systems, in: Intelligent Data Engineering and Automated Learning-IDEAL 2014, Springer, 2014, pp. 255–265.
[9] A.J. Malik, W. Shahzad, F.A. Khan, Binary PSO and random forests algorithm for probe attacks detection in a network, in: 2011 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2011, pp. 662–668.
[10] Z. Cordeiro Jr., G.L. Pappa, A PSO algorithm for improving multi-view classification, in: 2011 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2011, pp. 925–932.
[11] A. Kausar, M. Ishtiaq, M.A. Jaffar, A.M. Mirza, Optimization of ensemble based decision using PSO, in: Proceedings of the World Congress on Engineering, WCE, vol. 10, 2010.
[12] N. Littlestone, M. Warmuth, The weighted majority algorithm, Inf. Comput. 108 (2) (1994) 212–261, http://dx.doi.org/10.1006/inco.1994.1009 http://www.sciencedirect.com/science/article/pii/S0890540184710091,.
[13] http://kdd.ics.uci.edu/databases/kddcup99.
[14] S.-W. Lin, K.-C. Ying, C.-Y. Lee, Z.-J. Lee, An intelligent algorithm with feature selection and decision rules applied to anomaly intrusion detection, Appl. Soft Comput. 12 (10) (2012) 3285–3290.
[15] N. Araujo, R. de Oliveira, E.-W. Ferreira, A. Shinoda, B. Bhargava, Identifying important characteristics in the KDD99 intrusion detection dataset by feature selection using a hybrid approach, in: 2010 IEEE 17th International Conference on Telecommunications (ICT), IEEE, 2010, pp. 552–558.
[16] F. Kuang, W. Xu, S. Zhang, A novel hybrid KPCA and SVM with GA model for intrusion detection, Appl. Soft Comput. 18 (2014) 178–184.
[17] S.J. Horng, M.Y. Su, Y.H. Chen, T.W. Kao, R.J. Chen, J.L. Lai, C.D. Perkasa, A novel intrusion detection system based on hierarchical clustering and support vector machines, Expert Syst. Appl. 38 (1) (2011) 306–313.
[18] W. Wang, X. Zhang, S. Gombault, Constructing attribute weights from computer audit data for effective intrusion detection, J. Syst. Softw. 82 (12) (2009) 1974–1981.
[19] K.E. Parsopoulos, M.N. Vrahatis, Particle Swarm Optimization and Intelligence: Advances and Applications, 1st ed., Information Science Reference, 2010.
[20] R.C. Eberhart, Y. Shi, A modified particle swarm optimizer, in: Proceedings of the 1998 IEEE International Conference on Evolutionary Computation, 1998, pp. 69–73.
[21] M.E.H. Pedersen, A.J. Chipperfield, Local Unimodal Sampling, HL0801, Hvass Laboratories, 2008.
[22] R.C. Eberhart, J. Kennedy, A new optimizer using particle swarm theory, in: Proceedings of the Sixth International Symposium on Micro Machine and Human Science, vol. 1, New York, NY, 1995, pp. 39–43.