Combining Ensemble of Classifiers by Using Genetic Programming for Cyber Security Applications

Gianluigi Folino(✉) and Francesco Sergio Pisani

Institute of High Performance Computing and Networking (ICAR-CNR), Rende, Italy
{folino,fspisani}@icar.cnr.it

Abstract. Classification is a relevant task in the cyber security domain, but it must be able to cope with unbalanced and/or incomplete datasets and must also react in real-time to changes in the data. Ensemble of classifiers are a useful tool for classification in hard domains as they combine different classifiers that together provide complementary information. However, most of the ensemble-based algorithms require an extensive training phase and need to be re-trained in case of changes in the data.

This work proposes a Genetic Programming-based framework to generate a function for combining an ensemble, having some interesting properties: the models composing the ensemble are trained only on a portion of the training set, and then, they can be combined and used without any extra phase of training; furthermore, in case of changes in the data, the function can be recomputed in an incrementally way, with a moderate computational effort.

Experiments conducted on unbalanced datasets and on a well-known cyber-security dataset assess the goodness of the approach.

1 Introduction

In the last few years, the interest in cyber security problems is really increasing, as cyber crime seriously threatens national governments and the economy of many industries [1]. In this domain, computer and network technologies have intrinsic security weaknesses, i.e., protocol, operating system weaknesses, etc. In addition, computer network activities, human actions, etc. generate large amounts of data. Potential threats to the network need to be identified, and the related vulnerabilities need to be addressed to minimize the risk of the threat.

Therefore, data mining techniques could be used to efficiently fight, alleviate the effect or to prevent the action of the cybercriminals. In particular, classification can be efficiently used for many cyber security application, i.e. in intrusion detection systems, in the analysis of the user behavior, risk and attack analysis, etc. However, in this particular domain, datasets often have different number of features and each attribute could have different importance and costs. Furthermore, the entire system must also works if some datasets are not present. Therefore, it would be really unlikely a single classification algorithm will perform well

A.M. Mora and G. Squillero (Eds.): EvoApplications 2015, LNCS 9028, pp. 54–66, 2015.
DOI: 10.1007/978-3-319-16549-3_5

for all the datasets, especially in presence of changes and with constraints of real time and scalability.

Ensemble [2,3] is a learning paradigm where multiple component learners are trained for the same task by a learning algorithm, and the predictions of the component learners are combined for dealing with new unseen instances. Among the advantages in using ensemble of classifiers, we would like to remind that they help to reduce the variance of the error, the bias, and the dependence from a single dataset; furthermore, they can be build in an incremental way and they are apt to distributed implementations. They are also particularly suitable for distributed intrusion detection, because they permit to build a network profile by combining different classifiers that together provide complementary information. However, the phase of building of the ensemble could be computationally expensive as when new data arrives, it is necessary to restart the training phase.

To this aim, we propose a more flexible approach, and design a distributed Genetic Programming (GP) framework, based on the distributed CellulAr GEnetic programming (CAGE) environment [4], named CAGE-Combiner, to evolve a function for combining the classifiers composing the ensemble, having some attractive characteristics. First, the models composing the ensemble can be trained only on a portion of the training set, and then they can be combined and used without any extra phase of training; furthermore, in case of changes in the data, the function can be recomputed in an incrementally way, with a moderate computational effort. In addition, all the phases of the algorithm are distributed and can exploits the advantages of running on parallel/distributed architectures to cope with real time constraints.

The rest of the paper is structured as follows. Section 2 presents some related works. In Sect. 3, the strategy to combine the ensemble and the distributed GP framework are illustrated. Section 4 shows some experiments conducted to verify the effectiveness of the approach and to compare it with other similar approaches. Finally, Sect. 5 concludes the work.

2 Related Works

Evolutionary algorithms have been used mainly to evolve and select the base classifiers composing the ensemble [5,6] or adopting some time-expensive algorithms to combine the ensemble [7]; however a limited number of papers concerns the evolution of the combining function of the ensemble by using GP, which we illustrate in the following.

Chawla et al. [8] propose an evolutionary algorithm to combine the ensemble, based on a weighted linear combination of classifiers predictions, using many well-known data mining algorithm as base classifiers, i.e. J48, NBTree, JRip, etc. In [9], the authors extend their work in order to cope with unbalanced datasets. In practice, they increase the total number of base classifiers and adopt an oversampling technique. In [10], the authors consider also the case of homogenous ensemble and show the impact of a cut-off level on the total number of classifiers used in the generated model. Our approach also uses heterogeneous

classifiers, but we combine functions of different types, also considering weights derived by the performance of the classifiers on the training set; we take also into account the effect of unbalanced datasets and, in addition, our method is apt to operate with incomplete datasets, without using oversampling techniques.

Yan Wang et al. [11] use multiple ensembles to classify incomplete datasets. Their strategy consists in partitioning the incomplete datasets in multiple complete sets and to train the different classifier on each sample. Then, the predictions of all the classifiers could be combined according to the ratio between the number of features in this subsample and the total features of the original dataset. A similar approach could be included in our system.

In [12], the authors develop a GP-based framework to evolve the fusion function of the ensemble both for heterogenous and homogeneous ensemble. The approach is compared with other ensemble-based algorithms and the generalization properties of the approach are analyzed together with the frequency and the type of the classifiers presents in the solutions. The main aim of the paper is to improve the accuracy of the generated ensemble, while distributed implementations and the problems concerning incomplete and unbalanced datasets are not explored. In addition, differently form our approach, the authors do not consider weights depending from the performance of the classifiers on the datasets.

In [13], Brameier and Banzhaf use linear genetic programming to evolve teams of ensemble. A team consists of a predefined number of heterogeneous classifiers. The aim of genetic algorithm is to find the best team, i.e. the team having the best accuracy on the given datasets. The prediction of the team is the combination of individual predictions and it is based on the average or the majority voting strategy, also considering predefined weights. The errors of the individual members of the team are incorporated into the fitness function, so that the evolution process can find the team with the best combination of classifiers. Differently from our approach, the recombination of the team members is not completely free, but only a maximum pre-defined percentage of the models can be changed. In our approach, GP generates tree-based models and the number of base classifiers in the tree is not predefined; therefore, it will be the evolution process to select the best combination of the base classifiers.

3 Combining Ensemble of Classifiers

In this section, we show a general schema for combining an ensemble of classifiers and introduce the concept of "non-trainable functions", that can be used in order to combine an ensemble of classifiers without the need of a further phase of training. Then, we illustrated the distributed GP framework used to evolve the combining function of the ensemble.

3.1 Background: Ensemble of Classifiers and Non-trainable Functions

Ensemble permits to combine multiple (heterogenous or homogenous) models in order to classify new unseen instances. In practice, after a number of classifiers

are built usually using part of the dataset, the predictions of the different classifiers are combined and a common decision is taken. Different schemas can be considered to generate the classifiers and to combine the ensemble, i.e. the same learning algorithm can be trained on different datasets or/and different algorithms can be trained on the same dataset. In this work, we follow the general approach shown in Fig. 1, in which different algorithms are used on the same dataset in order to build the different classifiers/models.

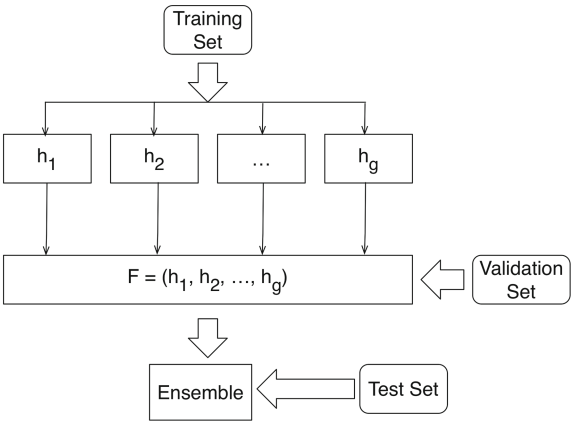

Fig. 1. A general schema for combining ensemble of classifiers

Let $S = \{(x_i, y_i)|i = 1, \ldots, N\}$ be a training set where x_i, called example or tuple or instance, is an attribute vector with m attributes and y_i is the class label associated with x_i. A predictor (classifier), given a new example, has the task to predict the class label for it.

Ensemble techniques build g predictors, each on a different training set, then combine them together to classify the test set. As an alternative, the g predictors could be built using different algorithms on the same/different training set.

The largely used boosting algorithm, introduced by Schapire [14] and Freund [15], follows a different schema; in order to boost the performance of any "weak" learning algorithm, i.e. an algorithm that "generates classifiers which need only be a little bit better than random guessing" [15], the method adaptively changes the distribution of the training set depending on how difficult each example is to classify.

This approach was successfully applied to a large number and types of datasets; however, it has the drawback of needing to repeat the training phase for a number of rounds and that could be really time-consuming for large datasets. The applications and the datasets in hard domains, as cyber security, have real-time requirements, which do not permit to re-train again the base models. On the contrary, ensemble strategies following the schema shown in Fig. 1 do not need any further phase of training, whether the functions used can be combined without using the original training set. The majority vote is a classical example of

this kind of combiner function. Some types of combiner has no extra parameters that need to be trained and consequently, the ensemble is ready for operation as soon as the base classifiers are trained. These are named non-trainable combiners [16] and could be used as functions in a genetic programming tree.

Before describing the GP framework used, here, we introduce some definitions useful to understand how the algorithm works.

Let $x \in R^N$ be a feature vector and $\Omega = \{\omega_1, \omega_2 ..., \omega_c\}$ be the set of the possible class labels. Each classifier h_i in the ensemble outputs c degrees of support, i.e., for each class, it will give the probability that the tuple belong to that class. Without loss of generality, we can assume that all the c degrees are in the interval $[0, 1]$, that is, $h_i : R^N \rightarrow [0, 1]^c$. Denote by $H_{i,j}(x)$ the support that classifier h_i gives to the hypothesis that x comes from class ω_j. The larger the support, the more likely the class label ω_j. A non-trainable combiner calculates the support for a class combining the support values of all the classifiers. For each tuple x of the training set, and considering g classifiers and c classes, a Decision Profile matrix DP can be build as follow:

$$DP(x) = \begin{bmatrix} H_{1,1}(x) \ ... \ H_{1,j}(x) \ ... \ H_{1,c}(x) \\ H_{i,1}(x) \ ... \ H_{i,j}(x) \ ... \ H_{i,c}(x) \\ H_{g,1}(x) \ ... \ H_{g,j}(x) \ ... \ H_{g,c}(x) \end{bmatrix}$$

where the element $H_{i,j}(x)$ is the support for j-th class of i-th classifier.

The functions used in our approach simply combine the values of a single column to compute the support for $j - th$ class and can be defined as follow:

$$\mu_j(x) = F[H_{1,j}(x), H_{2,j}(x), ..., H_{g,j}(x)]$$

For instance, the most simple function we can consider is the average, which can be computed as: $\mu_j(x) = \frac{1}{g} \sum_{i=1}^{g} H_{i,j}(x)$.

The class label of x is the class with maximum support μ.

3.2 Functions, Terminals and Fitness Evaluation

In this subsection, we describe the model that our GP system use in order to combine the predictions of multiple base classifiers.

Differently from classical models in which the GP tool is used to evolve the models, in our approach, the classifiers (with an associated weight previously computed on the training set) are the leaves of the tree, while the combiner functions are placed on the nodes. In particular, the functions chosen to better combine the classifiers composing the ensemble are non-trainable functions and are listed in the following: average, weighted average, multiplication, maximum and median. They can be applied to a different number of classifiers, i.e. each function is replicated with a different arity, typically from 2 to 5. More details are supplied in the following.

The **average** function, used with an arity of 2, 3 and 5, is defined as: $\mu_j(x) = \frac{1}{g} \sum_{i=1}^{g} H_{i,j}(x)$.

The **multiplication** function (arity 2, 3 and 5) is defined as: $\mu_j(x) = \prod_{i=1}^{g} H_{i,j}(x)$.

The **maximum** function returns the maximum support for 2, 3 and 5 classifiers and can be computed as: $\mu_j(x) = \max_i \{H_{i,j}(x)\}$.

The **median** function (arity 3 and 5) can be computed as: $\mu_j(x) = median_i \{H_{i,j}(x)\}$.

Finally, the **weighted** version of the **average** function uses the weights computed during the training phase to give a different importance to the models on the basis of the performance on the training set, and can be computed as: $\mu_j(x) = \frac{1}{\sum_{i=1}^{g} w_{i,j}} \sum_{i=1}^{g} w_{i,j} * H_{i,j}(x)$. For this function the values of 2, 3 and 5 are chosen for the arity.

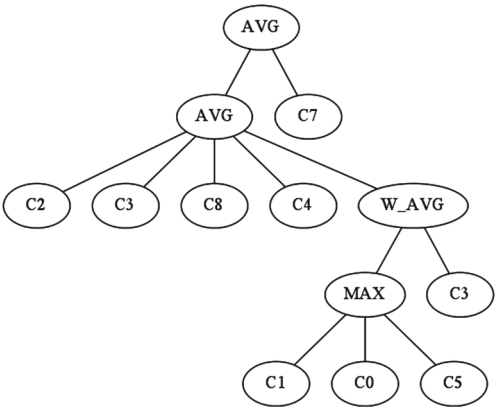

Fig. 2. An example of GP tree generated from the tool.

In order to better clarify, how the tree is built, in Fig. 2, an example of tree generated from the tool is illustrated. As for the fitness function, it is simply computed as the error of the ensemble on the validation set, i.e. the ratio between the tuples not correctly classified and the total number of tuples.

3.3 A Distributed Tool to Evolve Combiner Functions

The tool used to evolve the combining function is a distributed/parallel GP implementation, named CellulAr GEnetic programming (CAGE) [4], running both on distributed-memory parallel computers and on distributed environments. The tool is based on the fine-grained cellular model. The overall population of the GP algorithm is partitioned into subpopulations of the same size. Each subpopulation can be assigned to one processor and a standard (panmictic) GP algorithm is executed on it. Occasionally, migration process between subpopulations is carried out after a fixed number of generations. For example, the n best individuals from one subpopulation are copied into the other subpopulations, thus allowing the exchange of genetic information between populations.

The model is hybrid and modifies the island model by substituting the standard GP algorithm with a cellular GP (cGP) algorithm. In the cellular model each individual has a spatial location, a small neighborhood and interacts only within its neighborhood. The main difference in a cellular GP, with respect to a panmictic algorithm, is its decentralized selection mechanism and the genetic operators (crossover, mutation) adopted.

This tool is used to evolve the combiner functions and obtain an overall combiner function, which the ensemble will adopt to classify new tuples. Implicitly, the function selects the classifiers/models more apt to the particular datasets considered.

To summarize, if we consider a dataset partitioned in training, validation and test set, the approach works using the following steps.

1. The base classifiers are trained on the training set; then, a weight, proportional to the error on the training set, is associated to each classifier together with the support for each class, i.e. the decision support matrix is built. This phase could be computationally expensive, but it is performed in parallel, as the different algorithms are independent from each other.
2. The combiner function is evolved by using the distributed GP tool, CAGE, on the validation set. No extra computation on the data is necessary, as validation is only used to verify the correct class is assigned and consequently to compute the fitness function.
3. The final function is used to combine the base classifiers and classify new data (test set). This phase can be performed in parallel, by partitioning the test set among different nodes and applying the function to each partition.

4 Experimental Section

In this section, in order to assess the goodness of the proposed framework, using the parameters and the datasets described in the next subsection, we analyzed the size of the model and the accuracy obtained by CAGE-Combiner, with different configurations and compared our approach with different state-of-the-art combination strategies (Subsect. 4.2). Then, the performance of our approach was analyzed on a really unbalanced and hard intrusion detection dataset (Subsect. 4.3).

4.1 Datasets and Parameter Settings

All the experiments were performed on a Linux cluster with 16 Itanium2 1.4GHz nodes, each having 2 GBytes of main memory and connected by a Myrinet high performance network. No tuning phase has been conducted for the GP algorithm, but the same parameters used in the original paper [4] were used, listed in the following: a probability of crossover equal to 0.7 and of mutation equal to 0.1, a maximum depth equal to 7, and a population of 132 individuals per node. The algorithm was run on 4 nodes, using 1000 generations and the original training set was partitioned among the 4 nodes. The parsimony factor is varied using the values

of 0, 0.01 and 0.1 in order to generate classifiers of different size and to study the effect of the size of classifiers on the classification error and on the generalization of the algorithm. All the results were obtained by averaging 30 runs.

In Table 1, the size, the number of features and of classes and the percentage of the minority class of the datasets used in the experiments is shown. The datasets present different characteristics in terms of number of attributes and classes; in addition, most of them have a distribution of the tuples belonging to one or more classes really unbalanced, as it is evident from the percentage of the minority class.

Table 1. Description of datasets ordered by decreasing percentage of minority class.

Dataset	Number of examples	Number of features	Number of class	Minority class
Satimage	6,435	36	6	0.0972
DNA/Splice	3,190	61	3	0.2404
Phoneme	5,404	5	2	0.2938
Pendigit	10,992	16	10	0.0959
KDDCup	494,020	41	5	1.052E-4

The different classifiers composing the ensemble are trained on the same training set. In practice, each dataset is partitioned in three subsamples: the 70 % of original dataset is used to train the base classifiers, the remaining 30 % is equally partitioned in two parts: validation and test set. The validation part is used by the evolutionary algorithm to build the combination function, while the error rate of the best tree is calculated on the test partition. The learning algorithms are implemented in WEKA platform and the models are built using standard parameters.

The algorithms used as base classifiers in the experiments are based on the WEKA implementation[1] and are listed in the following: J48 (decision trees), JRIP rule learner (Ripper rule learning algorithm), NBTree (Naive Bayes tree), Naive Bayes, 1R classifier, logistic model trees, logistic regression, decision stumps and 1BK (k-nearest neighbor algorithm).

In Table 2, it is shown the error rate of each base classifier respectively on the training, validation and test set and this helps to understand the improvement obtained in terms of accuracy using an ensemble, how shown in the next subsection.

4.2 Comparing with Other Evolutionary Strategies and Meta-Ensemble Techniques

As stated in the previous subsection, the GP framework is executed without any tuning of the parameters. The only exception is that we want to analyze (Table 3)

[1] http://www.cs.waikato.ac.nz/ml/weka.

Table 2. Error rate of the base classifiers used to build the ensemble

Dataset	Type	J48	Jrip	NBTree	Naive Bayes	OneR	LMT	Logistic	Stump Stump	Ibk
Satimage	Training	2.60	7.90	17.30	19.90	39.10	8.80	12.00	55.80	0.00
	Validation	15.00	13.20	19.10	20.40	43.40	13.20	14.50	58.30	9.30
	Test	15.30	14.20	20.60	22.90	41.30	13.60	15.80	56.60	10.10
Phoneme	Training	8.40	11.30	10.80	23.10	18.20	9.00	24.60	24.30	0.00
	Validation	14.40	14.90	13.90	22.90	25.60	12.80	25.40	24.80	9.40
	Test	14.80	16.60	14.30	25.20	23.20	15.20	26.00	25.40	11.30
Pendigit	Training	0.70	1.20	0.20	13.50	59.40	0.30	3.70	79.70	0.00
	Validation	4.10	4.50	4.90	14.90	63.50	1.70	4.40	79.20	1.00
	Test	3.90	3.30	5.20	14.90	60.50	1.90	4.80	79.80	0.50
Dna	Training	4.00	4.20	0.00	3.30	0.00	0.00	0.00	37.00	0.00
	Validation	5.50	4.80	8.30	5.00	71.40	3.50	11.30	40.90	27.80
	Test	4.50	4.30	5.50	4.30	73.70	3.80	11.00	37.80	24.10

Table 3. The error rate for different values of parsimony (0, 0.1 and 0.01), along with the average number of classifiers and functions used in the best tree.

Dataset	Parsimony	Error train	Error test	Distinct classifiers	Total classifiers	Functions
Satimage	0	7.77 ± 0.60	9.08 ± 0.56	8.64 ± 0.79	78.44 ± 42.03	30.56 ± 15.01
	0.01	7.46 ± 0.62	9.25 ± 0.58	7.26 ± 1.34	25.70 ± 11.49	11.10 ± 4.16
	0.1	7.48 ± 0.41	9.09 ± 0.51	6.57 ± 1.54	14.46 ± 4.61	6.76 ± 2.45
Phoneme	0	**8.27** ± 0.43	**11.63** ± 1.30	8.70 ± 0.55	99.95 ± 74.63	38.85 ± 30.36
	0.01	7.62 ± 0.65	11.14 ± 0.44	6.61 ± 1.41	26.15 ± 18.37	11.96 ± 6.98
	0.1	7.80 ± 0.46	10.91 ± 0.51	5.53 ± 1.33	13.73 ± 5.47	7.00 ± 2.75
Pendigit	0	0.66 ± 0.22	0.74 ± 0.22	8.86 ± 0.33	71.30 ± 37.16	27.95 ± 14.82
	0.01	0.60 ± 0.12	0.68 ± 0.12	6.13 ± 1.50	14.48 ± 7.84	6.10 ± 3.30
	0.1	0.64 ± 0.10	0.67 ± 0.12	6.13 ± 1.08	10.40 ± 3.20	5.16 ± 2.35
Dna	0	**2.46** ± 0.85	**3.71** ± 1.05	8.48 ± 0.89	88.31 ± 92.59	33.86 ± 36.10
	0.01	1.86 ± 0.15	3.48 ± 0.28	6.53 ± 0.92	11.70 ± 2.53	*4.50* ± 1.25
	0.1	1.82 ± 0.13	3.53 ± 0.22	6.26 ± 0.81	9.20 ± 1.75	*4.30* ± 1.29

the effect of the size of the combiner function on the accuracy, varying the value of the parsimony factor. In order to balance the accuracy against the size of tree, in GP algorithms, the fitness is augmented with an optional parameter, the *parsimony*, which measures the complexity of the individuals. Higher is the parsimony, simpler is the tree, but accuracy diminishes.

In order to statistically validate the comparison results, we performed the two-tailed t-test($\sigma = 0.05$) over the 30 runs. The values in bold highlight, for each value of parsimony, the results that, according to the t-test, are significantly differently from the other values. As for the last three columns, most of the values present statistically significant differences, so, for this case, the values are

not significantly different are represented in italic. From the table, it is evident that most of the differences in accuracy is not significantly different with the exception of the DNA and Phoneme datasets for the case of using parsimony vs. not using parsimony, while there is no difference between the case of 0.1 and 0.01. On the contrary, the size of the trees and the distinct classifiers selected by the algorithm are affected by the parsimony factor. For this reason, we choose a parsimony factor of 0.1 for the other experiments conducted.

Table 4. Error rate for different strategies for the 4 datasets used in the experiments.

	Satimage	Phoneme	Pendigit	DNA
CAGE-Combiner	9.09	10.92	0.68	3.53
EVEN	8.91	11.68	0.68	4.20
EVEN (cut-off = 0.8)	8.69	11.06	0.66	4.34
Majority Vote	10.52	15.85	0.98	4.20
Weighted Vote	10.40	15.04	0.93	4.32
Best classifier	10.60	12.59	0.89	4.82
Stacking NB	10.75	14.93	0.81	4.55
Stacking LR	9.72	11.12	0.82	5.03

In Table 4, CAGE-Combiner is compared with the EVEN algorithm, described in the related work section [10] and also with the meta-algorithms used in the same paper. Note that EVEN uses a population size of 120 (the number of classifiers) for 1000 generations. The results show that CAGE-Combiner obtain better or comparable accuracy for all the datasets; however, we would like to remark that the number of classifiers used is sensibly minor than the 120 used by the EVEN algorithm. However, in the latter, a cut-off threshold is introduced and only those classifiers whose weights are above this threshold are allowed to participate to the ensemble. The maximum value of cut-off used in the paper (0.8) and shown in the table permits to reduce the number of classifiers to about 25 % of the original size, while our approach (see Table 3) using the parsimony value of 0.1, obtains a better reduction of the number of classifiers (about 10 %), without any relevant reduction in the accuracy.

4.3 A Dataset in the Cyber Security Domain: KDD Cup 99

To evaluate the system proposed on a real-world dataset in the field of cyber security, we performed the same experiments as the previous subsection, using one of the most used dataset for the task of classification of intrusions: KDD Cup 1999[2]. This dataset contains 494,020 records, representing normal connections and 24 different attack types. Each attack is clustered into four main categories, so each connection belongs to the following classes: normal (normal, i.e., no

[2] http://www.sigkdd.org/kdd-cup-1999-computer-network-intrusion-detection.

attack), DoS (Denial of Service connections), R2L (Remote to User, remote attacks addressed to gain local access), U2R (User to Root, exploits used to gain root access) or Probe (probing attack to discover known vulnerabilities).

Table 5. The error rate for different values of parsimony (0, 0.1 and 0.01), along with the average number of classifiers and functions used in the best tree: KDD Cup 99.

Pars.	Error	Distinct Cls	Total Cls	Functions	DoS	Normal	Probe	R2L	U2R
0	0.0106 ± 0.0015	7.60 ± 0.71	65.23 ± 50.46	26.53 ± 19.03	0.0000	0.0003	0.0114	0.0506	0.2000
0.01	0.0105 ± 0.0012	6.20 ± 1.01	13.30 ± 6.76	5.80 ± 2.65	0.0000	0.0003	0.0109	0.0510	0.2333
0.1	0.0121 ± 0.0016	5.37 ± 0.80	9.03 ± 3.01	3.80 ± 1.45	0.0000	0.0003	0.0106	0.0490	0.2667

In Table 5, it is evident that the size of the trees and the distinct classifiers selected by the algorithm strongly depends on the parsimony factor, while for the accuracy the differences are minimal and the best results are obtained with the parsimony value of 0.01.

However, we are more interested to the behavior of our approach for the unbalanced datasets and in particular for the minority classes of the KDD Cup dataset, i.e., Probe, R2L and U2R.

To this aim, we consider the work in [17], which describes a boosting approach, named Greedy-Boost, to build an ensemble of classifier based on a linear combination of models, specifically designed to operate for the intrusion detection domain. The main idea is to extend the boosting process maintaining the models that behave better on the examples badly predicted in the previous round of the boosting algorithm (while the classical algorithm adjust only the weights and not the models).

In Table 6, CAGE-Combiner is compared with the Greedy-Boost algorithm on the KDDCup 99 datasets and the precision and the recall values are reported for all the classes. It is evident that our approach performs better both for the precision and the recall measure, especially in the case of the minority classes R2L and U2R.

Table 6. Precision and Recall for different strategies for the KDD Cup dataset. In the first column, it is reported the class distribution for the test set.

		Precision		Recall	
	Class distribution	Greedy-Boost	CAGE-Combiner	Greedy-Boost	CAGE-Combiner
DoS	0.7960	100.0	100.0	100.0	100.0
Normal	0.1936	99.1	99.9	100.0	100.0
Probe	0.0079	99.0	99.6	97.1	98.9
R2L	0.0023	93.2	98.5	71.9	94.9
U2R	4.85E-5	88.5	93.1	44.2	76.7

5 Conclusions and Future Work

A distributed framework for classifying unbalanced dataset, based on the ensemble model, is presented. The system evolves a combiner function, which does not need additional phases of training, after the heterogeneous classifiers composing the ensemble are trained. Preliminary experiments showed that the proposed system improves or is comparable to the performance of state-of-the-art approaches for combining ensemble, by using a smaller number of models. In future, we intend to investigate the ability of the algorithm to handle incomplete datasets and changes in data and to test the scalability of the algorithms on distributed machines mainly for large real-world datasets in the cyber security domain.

Acknowledgment. This work has been partially supported by MIUR-PON under project PON03PE_00032_2 within the framework of the Technological District on Cyber Security.

References

1. CERT Australia: Cyber crime and security survey report. Technical report (2012)
2. Breiman, L.: Bagging predictors. Mach. Learn. **24**, 123–140 (1996)
3. Freund, Y., Shapire, R.: Experiments with a new boosting algorithm. In: Machine Learning: Proceedings of the Thirteenth International Conference (ICML 1996), pp. 148–156. Morgan Kaufmann (1996)
4. Folino, G., Pizzuti, C., Spezzano, G.: A scalable cellular implementation of parallel genetic programming. IEEE Trans. Evol. Comput. **7**, 37–53 (2003)
5. de Oliveira, D.F., Canuto, A.M.P., de Souto, M.C.P.: Use of multi-objective genetic algorithms to investigate the diversity/accuracy dilemma in heterogeneous ensembles. In: International Joint Conference on Neural Networks, pp. 2339–2346. IEEE (2009)
6. Folino, G., Pizzuti, C., Spezzano, G.: Training distributed GP ensemble with a selective algorithm based on clustering and pruning for pattern classification. IEEE Trans. Evol. Comput. **12**, 458–468 (2008)
7. Stefano, C.D., Folino, G., Fontanella, F., di Freca, A.S.: Using bayesian networks for selecting classifiers in GP ensembles. Inf. Sci. **258**, 200–216 (2014)
8. Sylvester, J., Chawla, N.V.: Evolutionary ensembles: combining learning agents using genetic algorithms. In: AAAI Workshop on Multiagent Learning, pp. 46–51 (2005)
9. Chawla, N.V., Sylvester, J.: Exploiting diversity in ensembles: improving the performance on unbalanced datasets. In: Haindl, M., Kittler, J., Roli, F. (eds.) MCS 2007. LNCS, vol. 4472, pp. 397–406. Springer, Heidelberg (2007)
10. Sylvester, J., Chawla, N.V.: Evolutionary ensemble creation and thinning. In: Proceedings of the International Joint Conference on Neural Networks, IJCNN 2006, pp. 5148–5155. IEEE (2006)
11. Wang, Y., Gao, Y., Shen, R., Yang, F.: Selective ensemble approach for classification of datasets with incomplete values. In: Wang, Y., Li, T. (eds.) ISKE2011. AISC, vol. 122, pp. 281–286. Springer, Heidelberg (2011)
12. Acosta-Mendoza, N., Morales-Reyes, A., Escalante, H.J., Gago-Alonso, A.: Learning to assemble classifiers via genetic programming. IJPRAI **28** (2014)

13. Brameier, M., Banzhaf, W.: Evolving teams of predictors with linear genetic programming. Genet. Program Evolvable Mach. **2**, 381–407 (2001)
14. Schapire, R.E.: The strength of weak learnability. Mach. Learn. **5**, 197–227 (1990)
15. Schapire, R.E.: Boosting a weak learning by majority. Inf. Comput. **121**, 256–285 (1995)
16. Kuncheva, L.: Combining Pattern Classifiers: Methods and Algorithms. Wiley-Interscience, Chichester (2004)
17. Bahri, E., Harbi, N., Huu, H.N.: Approach based ensemble methods for better and faster intrusion detection. In: Herrero, A., Corchado, E. (eds.) CISIS 2011. LNCS, vol. 6694, pp. 17–24. Springer, Heidelberg (2011)