# Evolving Meta-Ensemble of Classifiers for Handling Incomplete and Unbalanced Datasets in the Cyber security Domain

G. Folino and F. S. Pisani

*ICAR-CNR Istituto di Calcolo e Reti ad Alte Prestazioni*
*Via P. Bucci, 87036 Rende (CS)–Italy*

**Abstract**

Cyber security classification algorithms usually operate with datasets presenting many missing features and strongly unbalanced classes. In order to cope with these issues, we designed a distributed Genetic Programming (GP) framework, named CAGE-MetaCombiner, which adopts a meta-ensemble model to operate efficiently with missing data. Each ensemble evolves a function for combining the classifiers, which does not need of any extra phase of training on the original data. Therefore, in the case of changes in the data, the function can be recomputed in an incremental way, with a moderate computational effort; this aspect together with the advantages of running on parallel/distributed architectures make the algorithm suitable to operate with the real time constraints typical of a cyber security problem. In addition, an important cyber security problem that concerns the classification of the users or the employers of an e-payment system is illustrated, in order to show the relevance of the case in which entire sources of data or groups of features are missing. Finally, the capacity of approach in handling groups of missing features and unbalanced datasets is validated on many artificial datasets and on two real datasets and it is compared with some similar approaches.

## 1. Introduction

In the last few years, as a consequence of our interconnected society, the interest in cyber security problems has really been increasing and cyber crime seriously threatens national governments and the economy of many

industries[1]. Indeed, computer and network technologies have intrinsic security vulnerabilities, i.e., protocol, operating system weaknesses, etc. Therefore, potential threats and the related vulnerabilities need to be identified and addressed to minimize the risks. In addition, computer network activities, human actions, etc. generate large amounts of data and this aspect must be seriously taken into account.

Data mining techniques could be used to fight efficiently, to alleviate the effect or to prevent the action of cybercriminals, especially in the presence of large datasets. In particular, classification can be used efficiently for many cyber security applications, i.e. classification of the user behavior, risk and attack analysis, intrusion detection systems, etc. However, in this particular domain, datasets often have different number of features and each attribute could have different importance and cost. Furthermore, the entire system must also work if some features are missing and/or the classes are unbalanced. Therefore, a single classification algorithm performing well for all the datasets would be really unlikely, especially in the presence of changes and with constraints of real time and scalability.

In the ensemble learning paradigm [2][3], multiple classification models are trained by a predictive algorithm, and then their predictions are combined to classify new tuples. This paradigm presents a number of advantages with regard to using a single model, i.e., it reduces the variance of the error, the bias, and the dependence on a single dataset and works well in the case of unbalanced classes; furthermore, the ensemble can be build in an incremental way and can be easily implemented on a distributed environment. If we consider a stream of data, the ensemble needs to be re-trained to take into account changes in the data. This process could be computationally expensive, especially if it is necessary to retrain the models or to regenerate new models on the new data.

Therefore, in order to classify large datasets in the field of cyber security, usually having the above-cited issues of unbalanced classes and missing features, a new framework, named CAGE-MetaCombiner, is proposed. The framework extends a well-known implementation of distributed GP (CellulAr GEnetic programming (CAGE) environment) and adopts a meta-ensemble model in order to cope with missing data, while the GP system, which evolves the combiner function of the ensemble, permits to handle unbalanced classes thanks to a weighted fitness function. In practice, an ensemble is built for each group of likely missing features, as explained in the following, and the different ensembles perform a weighted vote in order to decide the correct

class. Each ensemble evolves a function for combining the classifiers, which can be trained only on a portion of the training set and does not need any extra phase of training on the original data. In fact, in the case of changes in the data, the function can be recomputed in an incremental way, with a moderate computational effort. In addition, all the phases of the algorithm are distributed and can exploit the advantages of running on parallel/distributed architectures to cope with real time constraints.

The rest of the paper is structured as follows: in Section 2 presents some related works; in Section 3, a real scenario in the field of cyber security is illustrated; Section 4 is devoted to some background information concerning the problem of missing data and incomplete datasets and the ensemble of classifiers; in Section 5, the framework and its software architecture is illustrated; Section 6 shows a number of experiments conducted to verify the effectiveness of the approach and to compare it with other similar approaches; finally, Section 7 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.

In the following, we analyze two groups of approaches. The first group comprises GP-based ensembles used to evolve the combination function. Most of the analyzed approaches employ a high number of resources to generate the function and therefore, their usage is not particularly recommended for large real datasets. The approaches of the second group adopt the ensemble paradigm to cope with incomplete and/or unbalanced data, but differently from our work, require to use the training set also in the phase of generating the combiner function, with a considerable overhead in this phase. The only exception is the last analyzed paper, which does not use the training set in this phase; however, it builds a number of random subsets of the features of the original dataset and therefore, its application is problematic when the number of features is high.

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 a homogenous ensemble and show the effect of a cut-off level on the total number of classifiers used in the generated model. In [11], the authors develop a GP-based framework to evolve the fusion function of the ensemble both for heterogeneous 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. These works can also operate on incomplete datasets, but differently from our approach, use an oversampling technique. In addition, they do not take into account the problems concerning the unbalanced datasets, while our technique permits to efficiently handle them by considering different weights derived from the performance of the classifiers on the training sets.

In [12], Brameier and Banzhaf use linear genetic programming to evolve teams of ensembles. A team consists of a predefined number of heterogeneous classifiers. The aim of a genetic algorithm is to find the best team, i.e. the team obtaining the highest 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, the evolution process can freely select the best combination of the base classifiers.

Chen et al. [14] use multiple ensembles to classify incomplete datasets. Their strategy consists in partitioning the incomplete datasets in multiple complete sets and in training the different classifiers 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. This approach is orthogonal to our and therefore, it could be included in our system.

Another approach to cope with incomplete datasets can be found in [13]. The authors build all the possible LCP (Local Complete Pattern), i.e., a

partition of the original datasets into complete datasets, without any missing features; a different classifier is built on each LCP, and then they are combined to predict the class label, basing on a voting matrix. The experiments compared the proposed approach with two techniques to cope with missing data, i.e., deletion and imputation, on small datasets and show how the approach outperforms the other two techniques. However, the phase of building the LCP could be really expensive.

Learn++.MF [15] is an ensemble-based algorithm with base classifiers trained on a random subset of the features of the original dataset. The approach generates a large number of classifiers, each trained on a different feature subset. In practice, the instances with missing attributes are classified by the models generated on the subsets of the remaining features. Then, the algorithm uses a majority voting strategy in order to assign the correct class under the condition that at least one classifier must cover the instance. When the number of attributes is high, it is unfeasible to build classifiers with all the possible sets of features; therefore, the subset of the features is iteratively updated to favor the selection of those features that were previously undersampled. However, this limits the real applicability of the approach to datasets with a low number of attributes.

## 3. A real scenario: classification of user profiles in e-payment systems.

The inspiration of the approach taken in this paper comes from a project on cyber security for e-payment systems, in which one of the main tasks consists in dividing the users of an e-payments systems into homogenous groups on the basis of their weakness or vulnerabilities from the cyber security point of view. In this way, the provider of an e-payment system can conduct a different information and prevention campaign for each class of users, with obvious advantages in terms of time and cost savings.

This technique is usually named segmentation, i.e. to the process of classifying customers into homogenous groups (segments), so that each group of customers shares enough characteristics in common to make it viable for a company to design specific offerings or products for it. It is based on a preliminary investigation in order to individuate the variables (segmentation variables) necessary to distinguish one class of customers from others. Typically, the goal is to increase the purchases and/or to improve customer satisfaction.

Different techniques can be employed to perform this task; in order to cope with large datasets, the most used are based on data mining approaches, mainly clustering and classification; anyway, many other techniques can be employed (see [16] for a survey of these techniques).

Another issue to be considered in order to construct the different profiles is the information collection process used to gather raw information about the user, which can be conducted through direct user intervention, or implicitly, through software that monitors user activity. Finally, profiles maintaining the same information over time are considered static, in contrast to dynamic profiles that can be modified or improved over time [17].

In the general case of computer user profiling, the entire audit source can include information from a variety of sources, such as command line calls issued by users, system calls monitoring for unusual application use/events, database/file accesses, and the organization policy management rules and compliance logs. The type of analysis used is primarily the modeling of statistical features, such as the frequency of events, the duration of events, the co-occurrence of multiple events combined through logical operators, and the sequence or transition of events. An interesting approach to computer user modeling is the process of learning about ordinary computer users by observing the way they use the computer. In this case, a computer user behavior is represented as the sequence of commands she/he types during her/his work. This sequence is transformed into a distribution of relevant subsequences of commands in order to find out a profile that defines its behavior. The ABCD (Agent behavior Classification based on Distributions of relevant events) algorithm discussed in [18] is an interesting approach using this technique.

To summarize, in our scenario, first the classes, in which the users will be divided, are individuated on the basis of their expertise in computer science and in the domain of the e-payments systems. This is done because most of the vulnerabilities are associated with the behavior and the practices correlated with the knowledge of the computer and/or of the e-payment system. Contrarily to the normal belief, a vulnerability study confirmed that software developers are the most vulnerable to attacks [19]. Indeed, an excess of confidence and the consequent download and installation of a number of applications can cause vulnerabilities; in the same way, misconfigurations of the system due to inconsistent application of security associated with a lack of competency could abilitate other kinds of vulnerabilities.

Given these considerations, Figure 1 illustrates the scenario used in the

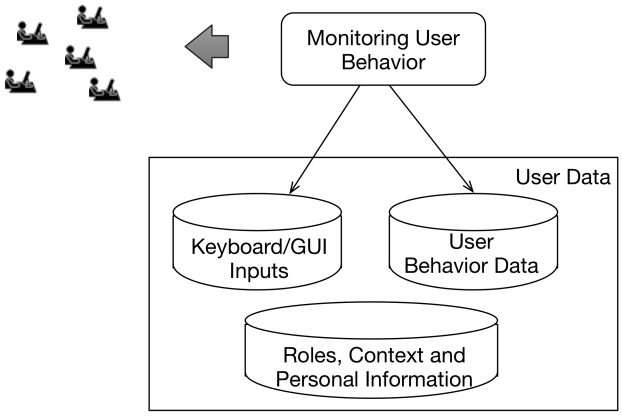


Figure 1: A scenario for collecting user data from different sources of information and/or monitoring tools.

above-cited project. The information concerning the user is supplied by using different sources of information or monitoring tools (i.e. generally automatic software analyzing the action and the behavior of the users). Going more into detail, user datasets can include demographic and education information, e.g., name, age, country, education level, computer knowledge, task knowledge, etc. and may also includes information concerning the contest in which the users operate and the roles they have in the systems. In addition to these data, which usually do not change if we consider a reasonable amount of time, the monitoring tool collects operational and behavioral data (e.g. IP addresses from which users connect to the system, operating system and browser used, the duration of the session, etc.), for which changes over time should be also considered. Finally, we also collect user input (i.e., commands entered using the keyboard or via GUI, using the mouse, etc.) This information should be captured in a dynamic way, by logging user actions. Unfortunately, all these kinds of data are not present for each user for clear reasons of privacy and for a number of different motivations (i.e., we have users with different roles and therefore, it is possible to monitor only some types of user, some users do not want to give authorization to disclose some data, etc.). Therefore, for different users, some sources are missing and this problem must be faced efficiently in order to obtain an accurate classification.

However, the data and the results used in this project (used also for testing the algorithms described in this paper) cannot be disclosed for reasons

of privacy. Therefore, in this paper a real dataset, with analogous characteristics, concerning the creation of user profiles from sequences of UNIX commands is used. It is the command-line data collected by Greenberg [20] by using UNIX csh command interpreter. This data are classified into four target groups, which represent a total of 168 male and female users, on the basis of their experience in programming. The four classes used are: non-programmer, novice programmers, experienced programmers and computer scientist.

More detail and the formalization of this approach, in which groups of features (typically coming from the same source of data) are missing, will be given in the next section.

## 4. Background

In this section, we give some background information useful to understand our approach, i.e. the main methods to cope with missing data and incomplete datasets and a general schema for combining an ensemble of classifiers and 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.

### *4.1. Incomplete datasets and missing data*

Typically, there are some main patterns in missing data: missing completely at random (MCAR), to describe data, in which the complete cases are a random sample of the originally dataset, i.e., the probability of a feature being missing is independent of the value of that or any other feature in the dataset; missing at random (MAR) describe data that are missing for reasons related to completely observed variables in the data set. Finally, the MNAR case considers the probability that an entry will be missing depends on both observed and unobserved values in the data set. Therefore, even if MCAR is more easy to handle,we try to cope with the MAR case, as it is a more realistic model and it is suitable to many real-world applications, i..e, the scenario described in the previous section.

Data mining and in particular classification algorithms must handle the problem of missing values on useful features. The presence of missing features complicates the classification process, as the effective prediction may depend heavily on the way missing values are treated. The performance is strictly related to rates of missing data: a low rate (typically less than 5%) is

generally considered manageable, while higher rate can be very problematic to handle.

In the scenario of the previous subsection, we can consider the missing values are present in both the training and the testing data as the same sources of data are not available for all the users. However, in our approach, without any loss of generality, we suppose that the training dataset is complete. Even in the case of the presence of a moderate number of tuples presenting missing data, it can be reported to the previous case, simply by deleting all the incomplete tuples. However, handling missing data by eliminating cases with missing data will bias results, if the remaining cases are not representative of the entire sample. Therefore, different techniques can be used to handle these missing features (see [21] for a detailed list of them). In addition to the above-mentioned strategy to remove any tuple with missing values, we remember the option of using a classification algorithm, which can deal with missing values in the training phase and the strategy of imputing all missing values before training the classification algorithm, i.e., replace the missing value with a meaningful estimate.

Indeed, in our particular problem, we are more interested in handling groups of missing features, and consequently, we focus on constructing a classifier on the incomplete dataset directly.

More formally, in our scenario, we have $D_1, D_2, \ldots, D_k$ datasets; typically each dataset comes from a different source of data, but can be used to predict the same class. Therefore, the corresponding $i_{th}$ tuple of the different datasets can be used to predict the class of the same user. However, a particular tuple of a dataset can be missing, i.e., all the features belonging to the same source of data of that tuple are missing.

However, without any loss of generality, even a problem of missing features of an incomplete dataset can be reported to the previous one, by grouping tuples with the same missing features.

If we consider a dataset

$D = \{(x_{1i}, x_{2i}, ..., x_{di}), i = 1..N\}$

the dataset is incomplete if at least one entry in $[1, d]$ is missing.

For instance, consider the incomplete dataset represented in Table 1 consisting of 6 tuples and 5 features.

This dataset can be partitioned in complete datasets by grouping features having the same missing features. A possible partition could be the following; $D_1 = \{1, 4\}$ considering the features $x_1$, $x_2$ and $x_5$, $D_2 = \{2, 5\}$ considering the features $x_1$, $x_3$, $x_4$ and $x_5$ and $D_3 = \{3, 6\}$ considering all

Table 1: An incomplete dataset of 6 tuples and 5 features.

| N | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ |
|---|---|---|---|---|---|
| 1 | | | ? | ? | |
| 2 | | ? | | | |
| 3 | | | | | |
| 4 | | | ? | ? | |
| 5 | | ? | | | |
| 6 | | | | | |

the features. Then, each complete dataset obtained can be used as training set for a classifier algorithm and the different models obtained can be used to classify each tuple, as we will show in the description of our framework in subsection 5.1. The problem of decomposing the dataset could become complex whether the missing features follows a random pattern and cannot be easily grouped in order to decompose the original datasets in a few complete datasets. In this case, the technique illustrated in this paper is not adequate. However, we want to remark that our pattern of missing data covers a large number real applications in which a group of features come from the same source of data and potentially can be all missing. For instance, consider the real world dataset representing the oceanographic and surface meteorological data, which support the prediction of El Niño cycles[1,2]. The data are mainly supplied by moored buoys. However, there are many missing values in the data for different reasons. First of all, a few buoys are able to measure currents, rainfall, and solar radiation; so in many cases, this entire group of values is missing. In addition, some buoys were commissioned earlier than others, so all the data supplied in a given period of time by these buoys are missing. Finally, for different motivations (bad weather, problems of communication, etc.) some buoys could not transmit their data for a given period of time. Therefore, this is a typical example in which our approach could be useful to handle the missing data.

[1]http://www.pmel.noaa.gov/tao

[2]http://archive.ics.uci.edu/ml/machine-learning-databases/el_nino-mld/el_nino.data.html

*4.2. Ensemble of classifiers and non-trainable functions*

In this subsection, 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.

The ensemble permits the combination of 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 Figure 2, in which different algorithms are used on the same dataset in order to build the different classifiers/models.

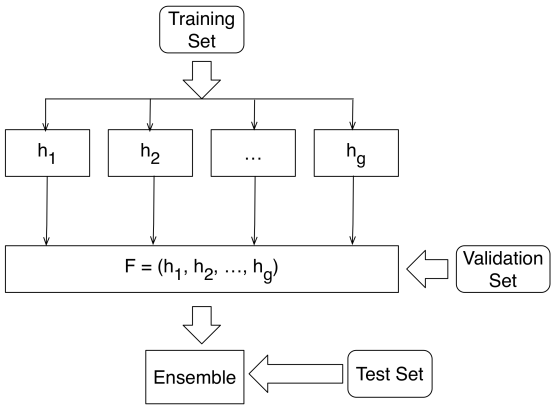


Figure 2: The approach used in this work to combine the learners composing the ensemble.

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 of predicting 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 widely used boosting algorithm, introduced by Schapire [22] and Freund [23], 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" [23], the method adaptively changes the distribution of the training set according to 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, such cyber security, have real-time requirements, which do not permit re-training the base models. On the contrary, ensemble strategies following the schema shown in Figure 2 do not need any further phase of training, whether the functions used can be combined without using the original training set or not. The majority vote is a classical example of this kind of combiner function. Some types of combiner have 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 [24] 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 belongs 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 $\omega_j$. The larger the support, the more likely the class label $\omega_j$. 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 $\mu$.

## 5. A distributed tool for evolving combining functions

In this section, we illustrate the software architecture and detail the pseudo-code of the meta-ensemble approach; then, we show how the distributed GP framework used to evolve the combining function of the ensemble works, including the nodes, the terminals and the fitness function employed.

### *5.1. The software architecture of the meta-ensemble approach.*

The overall software architecture of the meta-ensemble approach is illustrated in Figure 3. Note that CAGE-MetaCombiner is able to work on incomplete datasets (named $D_1, D_2, \ldots, D_k$ in the figure). It is worth noticing that, as described in the background section, it is equivalent whether each dataset comes from a different source of data, or they are obtained from a partition of an incomplete dataset by removing groups of missing features. The only strong assumption is that each corresponding tuple of the different datasets is used to predict the same class, i.e., the class of the user of the scenario shown in section 3. The corresponding tuple can be missing in one or more datasets, but if it is missing in all the datasets, it will be discarded and counted as a wrong prediction in the evaluation phase.

In practice, an ensemble is built for each dataset by using a distributed GP tool, CAGE (better described in the next subsection), to generate the combiner function (see Figure 5 for an example). The learning models (classifiers) composing the ensemble are taken from the well-known WEKA tool (see subsection 6.1 for more details on the algorithms used). The different ensembles perform a weighted vote in order to decide the correct class. It is worth remembering that each ensemble evolves a function for combining the classifiers, which does not need any extra phase of training on the original data. The final classification is obtained computing the error using the same formulae as the Adaboost.M2 algorithm used by the boosting algorithm, by

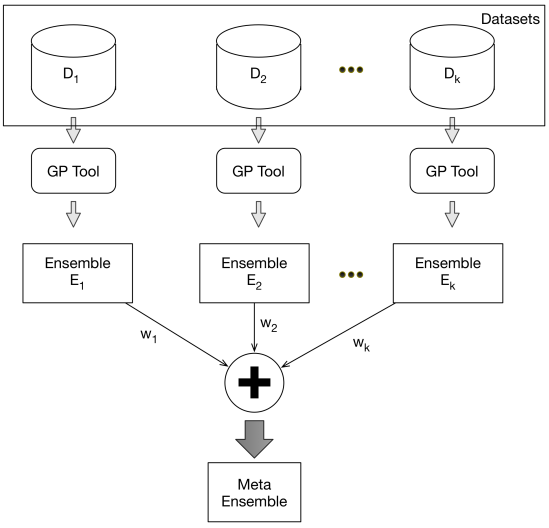


Figure 3: The software architecture of the meta-ensemble architecture.

computing the error of the entire ensemble instead of a single classifier as in the original boosting algorithm.

The entire process is better detailed in the pseudocode in Figure 4. We consider $l$ base classifiers and $k$ incomplete datasets. Each incomplete dataset is partitioned into train, validation and test set. A number of classification algorithms were trained on the training sets and only the best $l$ (a predefined threshold) are maintained and take part in the ensemble. Then, a decision profile matrix is built for each classifier in order to optimize the subsequent phase, in which the GP tool evolves the combiner function of the ensemble, by using the validation set. A weight is associated with each ensemble on the basis of the error of the ensemble on the validation set.

Afterwards, for each tuple $x$, for each possible class $j$ and for each ensemble $i$, the errors are computed using a weighted mean: $\mu_j(x) = \frac{\sum w_i * E_{ij}(x)}{\sum w_i}$ and the final classification is obtained by using the formula $class(x) = argmax_j(\mu_j(x))$

if a tuple $x_i$ is missing, the corresponding ensemble $E_i$ is discarded.

To summarize, if we consider a dataset partitioned into 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 with each clas-

Let $\alpha$ be the total number of base classification algorithms.
Let $l$ be the number of base classification algorithms effectively used.
Given a set of $k$ incomplete datasets $D_1, D_2, \ldots, D_k$, having respectively $m_1, m_2, \ldots, m_k$ tuples,
where the dataset $D_i = \{X_{i_1}, X_{i_2}, \ldots, X_{i_{mi}}\}$ and $X_{i_k}$ is a tuple $\{A_1, A_2, \ldots, A_d, C\}$
where $A_i$ is an attribute and the class C can have $c$ possible values.
Note that each tuple $X_{i_k}$ can potentially be missing.
**For each** $D_i$
    Consider the dataset $D_i$ partitioned into train, validation and test set: $Dtrain_i$, $Dvalid_i$ and $Dtest_i$
    Train $\alpha$ different classification algorithms on $Dtrain_i$
    Maintain the $l$ classifiers obtaining the highest accuracy on the training set.
    Build $l$ decision profile matrices, one for each of the classifiers, $DP_1, DP_2, \ldots, DP_l$
    using the respective validation set, one for each classifier of dimension $|Dvalid_i| \times c$
    Run the distributed GP tool on the validation set $Dvalid_i$ in order to obtain
    the combiner function of the ensemble.
    Obtain an Ensemble $E_i$, a combiner function $F_i$ and a weight $W_i$,
    computed on the basis of the error of the given ensemble on the validation set.
**end for each**
Build the decision profile matrix $DP$ of the entire ensemble $E$,
where each element $H_{i,j}(x)$ is the support that classifier $h_i$ gives to the hypothesis
that the tuple $x$ comes from class $j$.
Compute the weighted mean for each class $j$: $\mu_j(x) = \frac{\sum w_i * H_{ij}(x)}{\sum w_i}$ on the test set.
Compute the class by using the formula $class(x) = argmax_j(\mu_j(x))$.
Note that if a tuple $X_{i_k}$ is missing, the corresponding ensemble $E_i$ does not participate
to the evaluation procedure for that tuple.

Figure 4: The pseudo-code of the algorithm.

sifier 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 of 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 whether 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.

### *5.2. 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, the 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 more suitable classifiers/models to the specific datasets considered.

*5.3. Functions, terminals and fitness evaluation*

In this subsection, we describe the model that our GP system uses 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 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. The choice of this set of functions is due to the fact that most of the papers adopting non-trainable functions use these combiners and obtain good experimental results [24]. The only function we do not include in this set was the product, which obviously does not perform well in the multi-class case.

More formal 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.

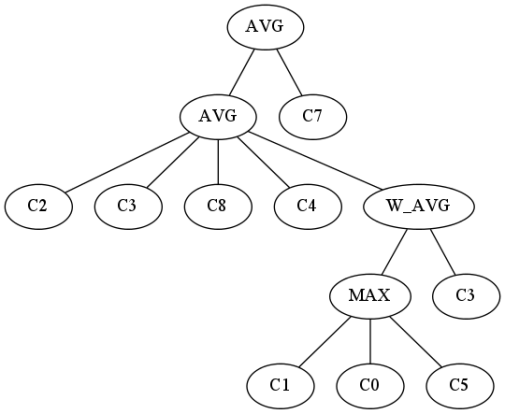


Figure 5: An example of the combiner function generated from the GP tool.

In order to better clarify, how the tree is built, in Figure 5, 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. However, in the particular case of unbalanced datasets, a weighted fitness is adopted. In practice, if a tuple belonging to a minority class is misclassified, the fitness function is penalized by a weight equal to the ratio between the total number of tuples and the total number of tuples belonging to that class (to avoid really high weights, if the weight exceeds the threshold value of 10, it is fixed to this threshold). For the tuple belonging to the majority class, the penalty weight is fixed to 1, as in the case of balanced datasets.

## 6. Experimental Results

In this section, the experiments conducted to analyze the capacity of our approach on coping with unbalanced datasets and on handling miss-

ing features are described together with the main parameters and the main characteristics of the datasets used. In addition to a number of well-known benchmark datasets, two real and hard datasets were used to validate the approach: Unix dataset and KDD 99. The first was used to test the performance of the algorithm for the case of missing features, while the latter presents a distribution of the strongly unbalanced classes and therefore it is useful to test the capacity of coping with unbalanced datasets. Finally, our algorithm is compared with two correlated works: one for the case of missing data and the other for the case of unbalanced classes.

### *6.1. Datasets and Parameter Settings*

All the experiments were performed on a Linux cluster with 16 Itanium2 1.4GHz nodes, each with 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 of 0.7 and of mutation of 0.1, a maximum depth of 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. All the results were obtained by averaging 30 runs.

In Table 2, the size, the number of features and classes and the percentage of the minority class of the datasets used in the experiments are shown. In Table 3, are illustrated the characteristics of three additional datasets, with a significant number of features and used for the comparison of the capacity of our framework in handling missing data with the work in [15].

These datasets present different characteristics in terms of number of attributes and classes and were used to assess the capacity of our framework to cope with unbalanced datasets; in fact, most of them have a distribution of the tuples belonging to one or more really unbalanced classes, as is evident from the percentage of the minority class. The Covtype, M-Feat, OCR and DNA datasets come from the UCI KDD Archive [3], the Pendigit and the Satimage are taken from the UCI Machine Learning Repository[4], the Phoneme dataset is from the ELENA project[5].

[3] http://kdd.ics.uci.edu/.

[4] http://www.ics.uci.edu/ mlearn/MLRepository.html

[5] ftp.dice.ucl.ac.be in the directory pub/neural/ELENA/databases.

Table 2: 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 | 3,190 | 61 | 3 | 0.2404 |
| Phoneme | 5,404 | 5 | 2 | 0.2938 |
| Pendigits | 10,992 | 16 | 10 | 0.0959 |
| KDDCup | 494,020 | 41 | 5 | 1.052E-4 |

Table 3: Description of the additional datasets used for the missing data experiments.

| Dataset | Number of examples | Number of features | Number of Class | Partitions |
|---|---|---|---|---|
| OCR | 5,620 | 62 | 10 | 3 |
| M-Feat | 2,000 | 216 | 10 | 4 |
| Covtype | 581,012 | 54 | 7 | 3 |

Each dataset is partitioned into three subsamples: 70% of original dataset is used to train the classifiers, which will compose the ensemble, the remaining 30% is equally partitioned into two parts: validation and test set. The validation part is used by the evolutionary algorithm to build the combination function of the ensemble, while the error rate, i.e. the ratio of the number of misclassified tuples to the total number of tuples, of the best tree is calculated on the test partition. The learning algorithms are implemented in the WEKA tool and the different models are built by using standard parameters.

More in detail, the algorithms used as classifiers in the experiments are based on the WEKA implementation[6] and are listed in the following: J48 (decision trees), K random tree, Function Tree (based on logistic regression) JRIP rule learner (Ripper rule learning algorithm), ConjunctiveRule, NBTree (Naive Bayes tree), Naive Bayes, DTNB (decision table with naive bayes), 1R classifier, logistic model trees, logistic regression, decision stumps and 1BK (k-nearest neighbor algorithm).

### *6.2. 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 due to the fact we want to analyze (Table 4) the effect of the size of the combiner function on the accuracy, i.e. the ratio of the number of correctly classified tuples to the

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

total number of tuples, varying the value of the parsimony factor. Generally, when using GP-based algorithms, there are different methods to limit the uncontrolled growth of the average size of an individual in the population (bloat problem); the simplest way is to limit their maximum depth and to punish individuals of excessive size. In the depth analysis presented in [25], the effect of many other complex methods are experimentally tested, but none of them results predominant over the other to justify the complexity introduced. Therefore, we adopted the widely used method of parsimony, consisting in simply adding to the fitness function a penalty given by the product of a constant parameter (the parsimony factor) and of the overall number of nodes and leaves of the genetic programming tree. Typically, the higher the parsimony, the simpler the tree, but the accuracy could diminish. The parsimony factor is varied using the values of 0 (no parsimony), 0.01 and 0.1.

Table 4: 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 |
| Pendigits | 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 |

For each dataset, we compare experiments by using different values of the parsimony factor and we highlight values having statistically significant differences using the Friedman test. The critical value of the Friedman test [26] is obtained from a chi-square distribution with two degree of freedom and the number of cases considered is 30 for each set. A significancy level of 5% is used. In all the tables in this Section, we use the following rules: the parsimony value is underlined when the Friedman test, comparing experiments with same value of parsimony but different values of the missing data, presents statistically significant differences. The accuracy/error values are marked in bold when the Friedman test, comparing experiments with the same missing percentage but with different parsimony factors, presents

statistically significant differences.

In Table 4, as only the behavior of the algorithm when the parsimony factor is changed is considered, values in bold represent significantly different results in terms of parsimony. In two of the four datasets, the differences in terms of error rates are significant statistically; however, the differences are not remarkable. On the contrary, the size of the trees and the distinct classifiers selected by the algorithm are greatly affected by the parsimony factor. For this reason, we choose a parsimony factor of 0.1 for the other experiments conducted on the following.

Table 5: Error rate for different strategies for the 4 datasets used in the experiments.

| | Satimage | Phoneme | Pendigits | Dna |
|---|---|---|---|---|
| CAGE-MetaCombiner | 9.09 | 10.91 | 0.67 | 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 5, CAGE-MetaCombiner 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-MetaCombiner obtain better or comparable accuracy for all the datasets; however, we would like to remark that the number of classifiers used is considerably smaller than the number of 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 in the ensemble. The maximum value of cut-off used in the paper (0.8) and shown in the table permits reduction of the number of classifiers to about 25% of the original size, while our approach (see Table 4) 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.

### *6.3. Experiments on the capacity of handling missing values*

Two sets of experiments were performed in order to evaluate the ability of the CAGE-MetaCombiner approach in handling missing features. The first set analyzes the behavior of the algorithm when we vary the percentage of

tuples with missing data and the parsimony factor affecting the size of the solutions. The second set compares our approach with a related work present in the literature: the Learn++.MF framework [15], described in the related work section.

For these experiments, in addition to the OCR, M-Feat and Pendigits datasets, used by the Learn++.MF framework, we also used the Satimage, Dna and Covtype datasets, which present a significant number of features. It is worth remembering that we are interested in handling cases in which entire groups of features are missing and not in coping with random patterns of missing features. Therefore, as the features of the above-mentioned datasets are logically divided into groups, we partitioned the datasets OCR, M-Feat, Satimage, Pendigits, Dna and Covtype, respectively in 3, 4, 3, 2, 3 and 3 partitions, trying to not separate correlated (or coming from the same source) attributes. Then, to simulate the missing data, for each partition a tuple can be removed according to a probability threshold, i.e., this parameter controls the percentage of tuples, which have missing attributes. For instance, if this parameter is set to 10%, the entire partition of the features belonging to this tuple has a probability of 0.1 to be missing. If all the partitions of a tuple are missing, this tuple will be considered as an error of classification. We choose the values for the threshold in the range 0-40%, with an interval of 10% with 0% means no missing data.

We would like to remark that the Covtype dataset is a real large dataset, representing the prediction of forest cover type from cartographic variables determined by the U.S. Forest Service and the U.S. Geological Survey. The task of classifying this data set is not easy, especially in the presence of missing attributes, as it contains 44 binary attributes out of 54 totals, representing qualitative independent variables such as wilderness areas and soil type.

Table 6 shows the error of classification of the CAGE-MetaCombiner algorithm by varying the parsimony factor using the values of 0 (no parsimony), 0.1 and 0.4, using the above-defined percentages of missing features. Therefore, we want to evaluate the effect of the two parameters: the parsimony factor, to reduce the complexity of the overall meta-ensemble, and the percentage of missing values, to evaluate the capacity of the algorithm to handle missing data. We conducted a Friedman test in the same way specified in the previous subsection to highlight statistically significant differences. Varying the parsimony, for each value of missing data, there is no significant difference in most of the cases. On the contrary, by varying the percentage of missing

Table 6: The error rate of CAGE-MetaCombiner using three parsimony values and different percentages of missing data.

| | | Missing Percentage | | | | | Classifiers | | |
|---|---|---|---|---|---|---|---|---|---|
| Dataset | Pars. | 0% | 10% | 20% | 30% | 40% | Distinct | Total | Functions |
| OCR | 0 | **5.43** ± 0.26 | **6.37** ± 0.21 | **7.90** ± 0.35 | **10.27** ± 0.37 | **14.16** ± 0.57 | 8.77 ± 0.12 | 84.80 ± 4.36 | 33.97 ± 0.21 |
| | 0.1 | **5.36** ± 0.27 | **6.19** ± 0.28 | **7.65** ± 0.40 | **10.31** ± 0.39 | **13.99** ± 0.62 | 7.50 ± 0.22 | 21.20 ± 0.93 | 9.10 ± 0.64 |
| | 0.4 | **5.30** ± 0.22 | **6.23** ± 0.20 | **7.54** ± 0.39 | **9.95** ± 0.35 | **13.85** ± 0.41 | 6.33 ± 0.19 | 11.63 ± 0.24 | 5.23 ± 0.41 |
| Pendigits | 0 | **4.96** ± 0.28 | **6.81** ± 0.34 | **10.21** ± 0.36 | **15.09** ± 0.29 | **21.31** ± 0.35 | 8.95 ± 0.05 | 104.75 ± 13.75 | 41.95 ± 3.25 |
| | 0.1 | **4.97** ± 0.27 | **6.88** ± 0.19 | **10.12** ± 0.42 | **14.89** ± 0.19 | **21.35** ± 0.55 | 7.75 ± 0.35 | 25.55 ± 1.05 | 11.30 ± 0.30 |
| | 0.4 | **5.20** ± 0.34 | **7.07** ± 0.40 | **10.49** ± 0.48 | **15.21** ± 0.50 | **21.24** ± 0.35 | 5.70 ± 0.60 | 11.05 ± 2.95 | 4.95 ± 1.25 |
| Dna | 0 | **7.33** ± 1.40 | **9.29** ± 1.45 | **11.91** ± 1.26 | **15.02** ± 0.94 | **18.98** ± 0.73 | 8.77 ± 0.19 | 85.33 ± 6.36 | 33.67 ± 2.72 |
| | 0.1 | **7.86** ± 1.52 | **9.93** ± 1.16 | **12.55** ± 1.32 | **16.11** ± 1.30 | **20.10** ± 0.57 | 6.29 ± 1.23 | 18.17 ± 5.69 | 8.46 ± 1.94 |
| | 0.4 | **7.31** ± 0.77 | **9.49** ± 0.65 | **12.05** ± 0.90 | **15.56** ± 0.83 | **19.21** ± 1.33 | 5.77 ± 0.38 | 9.47 ± 1.16 | 4.13 ± 0.80 |
| M-Feat | 0 | **4.85** ± 0.22 | 4.95 ± 0.24 | **5.35** ± 0.30 | **6.11** ± 0.44 | **7.70** ± 0.51 | 8.10 ± 0.12 | 53.80 ± 14.24 | 21.23 ± 5.58 |
| | 0.1 | **4.93** ± 0.22 | 5.04 ± 0.18 | **5.32** ± 0.25 | **6.17** ± 0.41 | **7.66** ± 0.75 | 5.36 ± 0.29 | 9.94 ± 0.88 | 4.53 ± 0.85 |
| | 0.4 | **4.78** ± 0.14 | 4.97 ± 0.20 | **5.20** ± 0.17 | **5.97** ± 0.28 | **7.64** ± 0.58 | 3.20 ± 1.27 | 4.28 ± 2.00 | 1.77 ± 1.16 |
| Satimage | 0 | **12.62** ± 0.23 | 12.84 ± 0.26 | **13.54** ± 0.28 | **15.05** ± 0.34 | **17.13** ± 0.40 | 8.80 ± 0.14 | 74.60 ± 13.45 | 30.53 ± 5.33 |
| | 0.1 | **12.63** ± 0.17 | 12.86 ± 0.12 | **13.59** ± 0.20 | **15.02** ± 0.24 | **17.79** ± 0.31 | 7.07 ± 0.69 | 18.23 ± 1.24 | 8.47 ± 0.59 |
| | 0.4 | **12.62** ± 0.20 | 12.86 ± 0.21 | **13.64** ± 0.20 | **14.84** ± 0.41 | **17.53** ± 0.56 | 5.30 ± 0.57 | 8.73 ± 0.73 | 3.40 ± 0.42 |
| Covtype | 0 | **21.95** ± 1.09 | **23.08** ± 0.89 | **24.54** ± 0.72 | **26.43** ± 0.54 | 28.95 ± 0.43 | 8.90 ± 0.08 | 123.87 ± 26.44 | 50.43 ± 9.73 |
| | 0.1 | **21.81** ± 0.83 | **22.69** ± 0.50 | **24.22** ± 0.40 | **26.25** ± 0.32 | 28.80 ± 0.27 | 8.33 ± 0.12 | 57.00 ± 4.65 | 26.07 ± 2.04 |
| | 0.4 | **21.71** ± 0.54 | **22.90** ± 0.45 | **24.38** ± 0.41 | **26.32** ± 0.33 | 28.89 ± 0.27 | 7.20 ± 0.36 | 32.27 ± 11.01 | 14.80 ± 4.81 |

data, the differences for the error rate are statistically significant with the exception of the Covtype dataset for the cases of 30% and 40% of missing data.

For all the datasets, and for a percentage of missing features up to 20%, the degradation in accuracy is moderate (it is always less than 3%), while using 30% or 40% as values of the threshold, the error has a remarkable increase (in some cases, it arrives at 7%). More specifically, the error for the M-Feat and for the Satimage dataset does not deteriorate much, even though a threshold of 30% and 40% is used.

As for the covtype dataset, the number of distinct classifiers selected by the algorithm does not vary much (from 7 to 9), while the average size of the tree is substantially different; therefore, with a parsimony factor set to 0.4, the algorithm performs quite well and the size of the overall ensemble is compact. Increasing the missing percentage threshold from 0% to 40%, the degradation of the accuracy is limited. These results confirm the effectiveness of CAGE-MetaCombiner in handling missing data.

The comparison between Learn++.MF and CAGE-MetaCombiner is evaluated on the three datasets, which are also used in [15] and the results are shown in Table 7. For CAGE-MetaCombiner, the value of 0.4 is used for the parsimony factor. For Learn++.MF, the number of missing features is chosen to be equal to the size of the partition used by our algorithm.

Owing to the strategy of computing the base classifiers used by Learn++.MF algorithm, the ability to handle missing data is determined by how many fea-

tures are missing. So the missing percentage values, reported in the original paper, arrive at 30%, and in almost all cases both the algorithms are able to classify each dataset with a reasonable accuracy. As for the M-Feat dataset, the 'n/a' value in the table means that no classifiers are usable for instances with more of 30% of missing attributes for the Learn++.MF algorithm. This limits the applicability of the algorithm for large percentages of missing data. Anyway, for all the datasets, CAGE-MetaCombiner has better results then Learn++.MF when missing data increase, and this is more evident when the datasets have many features.

Table 7: Comparison between CAGE-MetaCombiner and Learn++.MF. *nof* represents the number of missing features for each record of the dataset.

| | | Percentage of Missing | | | | | |
|---|---|---|---|---|---|---|---|
| | | 10% | | 20% | | 30% | |
| | nof | CageMC | Learn++.MF | CageMC | Learn++.MF | CageMC | Learn++.MF |
| OCR | 20 | 6.23 ± 0.20 | 3.50 ± 0.10 | 7.54 ± 0.39 | 8.20 ± 0.4 | 9.95 ± 0.35 | 13.50 ± 1.10 |
| Pendigits[7] | 8 | 7.07 ± 0.40 | 10 | 10.49 ± 0.48 | 13 | 15.21 ± 0.50 | 17 |
| M-Feat | 50 | 4.97 ± 0.20 | 6.62 ± 0.08 | 5.20 ± 0.17 | 7.44 ± 2.87 | 5.97 ± 0.28 | n/a |

### *6.4. Analysis on two real-world datasets: KDD 99 and Unix dataset.*

In this section, we aim to test our framework on two-well known real world datasets: KDD 99 and Unix dataset. The first is known to have really unbalanced classes, while the latter represents the case study described in Section 3 and is used to test the case of missing data.

One aspect of the framework to be analyzed is its capability of coping with datasets with unbalanced classes. A typical example is the cyber security problem of preventing intrusion in a system, trying to minimize the false alarms. Typically, given a set of pre-processed features of a normal connection or of a possible attack, a classification algorithm is used to verify if it is an attack or not. However, the problem is complex because a low number of attacks compared to the normal connection is present in the data and that makes it hard to apply general-purpose classification algorithms.

Therefore, we performed the same experiments as the previous subsection, using one of the most used real dataset for the task of classification of intru-

[7]The value reported concerning the accuracy Learn++.MF for the Pendigit dataset have been taken from a graph of the original paper, in which the values of the standard deviation were not present.

sions: KDD Cup 1999[8]. 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 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 8: 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 8, 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 not statistically significant.

However, we are more interested in 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 [27], which describes a boosting approach, named Greedy-Boost, to build an ensemble of classifiers 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 adjusts only the weights and not the models).

In Table 9, CAGE-MetaCombiner 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.

The second real dataset tested is the Unix Users Data created by Greenberg [20]. It contains the commands used in a Unix shell by 168 users with different level of skills. Each user is assigned to one of four profiles: non-

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

Table 9: 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-MetaCombiner | Greedy-Boost | CAGE-MetaCombiner |
| 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 |

programmer, novice programmers, experienced programmers and computer scientist.

This dataset is preprocessed in the same way used in [18]. For each user we consider the first 100 and 500 commands used; then the commands subsequences of fixed length (from 3 to 6) are extracted from the list. Each user represents a record in the processed dataset and all the distinct subsequences are used as record attribute and the attribute value is the number of times the subsequence is typed by the user (0 means that the subsequence is never typed).

In Table 10, we show the Cage-MetaCombiner performance on the Unix dataset using different subsequence lengths and different percentage of missing data. The classification rate for the 100 commands experiment is slightly better than for the case with 500 commands probably because the increase in the number of commands could lead to an increment in the number of features and consequently the training phase becomes harder than the previous case. Owing to the high number of features, the results are not much affected by the missing rate, that is the algorithm has good performance with all the percentages of missing data. In all the cases, the results show that the algorithm presents comparable performances by using different values of parsimony.

In Table 11 the results of comparison between Cage-MetaCombiner and EvABCD are reported [18]. The EvABCD algorithm is a technique for classification of the behavior profiles of users. As Cage-MetaCombiner, it learns different behaviors from training data. For a different profile, it builds one or more prototypes computing the frequency of all the sequences of commands with a defined length. Moreover, it updates the models in an incremental way. Our algorithm performs better than EvABCD in most cases. By using Cage-MetaCombiner, the results for a different number of commands extracted do not influence the accuracy much, while the EvABCD algorithm

Table 10: Classification accuracy (Percent) for Cage-MetaCombiner with the Unix dataset using three parsimony values, different subsequence lengths and with different percentages of missing data.

| Commands | Sequence | Percentage of Missing | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Parsimony | 0% | 10% | 20% | 30% | 40% | 80% | Distinct Classifiers | Total Classifiers | Functions |
| 100 | 3 | 0 | 83.96 ± 1.90 | 83.84 ± 3.68 | 83.70 ± 2.43 | 82.67 ± 2.91 | 82.64 ± 2.51 | 81.32 ± 2.21 | 7.75 ± 0.45 | 52.97 ± 9.42 | 21.62 ± 3.27 |
| | | 0.1 | 85.73 ± 1.22 | 84.00 ± 1.33 | 83.60 ± 1.95 | 83.57 ± 2.07 | 83.20 ± 2.82 | 81.69 ± 3.00 | 4.19 ± 0.31 | 6.73 ± 0.83 | 3.51 ± 0.32 |
| | | 0.4 | 84.16 ± 0.35 | 83.96 ± 1.20 | 83.90 ± 1.05 | 83.86 ± 1.64 | 83.49 ± 1.64 | 83.00 ± 2.92 | 3.41 ± 0.28 | 4.66 ± 0.39 | 2.49 ± 0.25 |
| | 4 | 0 | 83.81 ± 3.97 | 83.70 ± 1.89 | **82.81 ± 1.72** | 82.54 ± 2.24 | **81.98 ± 3.65** | 81.82 ± 2.42 | 7.71 ± 0.28 | 44.79 ± 3.04 | 18.41 ± 1.17 |
| | | 0.1 | 83.86 ± 0.81 | 83.73 ± 1.24 | **83.20 ± 1.74** | 83.10 ± 1.73 | **82.89 ± 1.51** | 82.11 ± 3.01 | 4.50 ± 0.31 | 7.61 ± 1.51 | 3.75 ± 0.63 |
| | | 0.4 | 84.76 ± 0.53 | 84.19 ± 0.54 | **83.86 ± 0.93** | 83.76 ± 1.34 | **83.13 ± 1.70** | 81.65 ± 2.70 | 3.57 ± 0.33 | 4.68 ± 0.38 | 2.49 ± 0.18 |
| | 5 | 0 | 83.37 ± 1.31 | **82.97 ± 1.54** | **82.67 ± 2.09** | 82.00 ± 2.41 | 81.92 ± 3.25 | 81.45 ± 2.34 | 7.52 ± 0.34 | 43.31 ± 8.23 | 17.79 ± 3.07 |
| | | 0.1 | 83.96 ± 0.74 | **83.93 ± 0.81** | **83.73 ± 1.46** | 83.33 ± 1.62 | 83.21 ± 2.02 | 82.51 ± 2.66 | 4.57 ± 0.40 | 7.40 ± 1.17 | 3.67 ± 0.51 |
| | | 0.4 | 85.14 ± 3.96 | **83.83 ± 1.15** | **84.29 ± 1.19** | 84.09 ± 1.64 | 83.33 ± 3.63 | 82.84 ± 2.23 | 3.77 ± 0.27 | 4.83 ± 0.58 | 2.64 ± 0.31 |
| | 6 | 0 | 84.74 ± 3.96 | 84.09 ± 1.33 | 83.96 ± 1.89 | 83.86 ± 1.18 | **82.05 ± 3.08** | 81.19 ± 2.24 | 8.01 ± 0.24 | 51.23 ± 7.28 | 20.91 ± 2.51 |
| | | 0.1 | 85.06 ± 0.53 | 84.23 ± 0.99 | 84.26 ± 1.29 | 83.60 ± 1.32 | **83.09 ± 2.01** | 82.38 ± 2.43 | 4.54 ± 0.44 | 7.61 ± 1.44 | 3.83 ± 0.63 |
| | | 0.4 | 84.96 ± 0.74 | 84.19 ± 0.54 | 83.93 ± 0.55 | 83.85 ± 1.31 | **83.64 ± 1.72** | 82.21 ± 2.32 | 3.63 ± 0.13 | 4.77 ± 0.39 | 2.56 ± 0.16 |
| 500 | 3 | 0 | **86.70 ± 7.05** | **85.78 ± 5.83** | **85.58 ± 5.37** | **85.08 ± 5.07** | **85.02 ± 5.27** | 82.44 ± 3.05 | 7.87 ± 0.15 | 50.06 ± 5.56 | 20.38 ± 2.12 |
| | | 0.1 | **84.39 ± 7.36** | **84.08 ± 7.59** | **84.26 ± 6.93** | **83.80 ± 6.84** | **83.27 ± 5.38** | 82.01 ± 5.00 | 4.46 ± 0.65 | 7.55 ± 1.66 | 3.93 ± 0.80 |
| | | 0.4 | **90.56 ± 2.21** | **89.58 ± 5.58** | **88.86 ± 5.40** | **87.92 ± 5.38** | **87.46 ± 2.42** | 82.64 ± 3.51 | 3.03 ± 0.32 | 3.77 ± 0.57 | 2.12 ± 0.30 |
| | 4 | 0 | 83.63 ± 1.22 | **82.78 ± 4.03** | 82.31 ± 3.44 | **82.25 ± 3.23** | 82.10 ± 2.27 | 82.08 ± 2.28 | 7.83 ± 0.14 | 57.41 ± 7.51 | 22.93 ± 3.28 |
| | | 0.1 | 83.86 ± 0.85 | **83.57 ± 2.15** | 82.93 ± 1.59 | **82.10 ± 2.12** | 81.91 ± 2.70 | 81.04 ± 3.28 | 4.21 ± 0.27 | 6.97 ± 0.84 | 3.71 ± 0.46 |
| | | 0.4 | 84.16 ± 0.80 | **83.96 ± 0.65** | 83.80 ± 1.26 | **83.43 ± 1.69** | 83.50 ± 1.74 | 82.84 ± 2.33 | 3.27 ± 0.15 | 4.46 ± 0.45 | 2.42 ± 0.22 |
| | 5 | 0 | 83.48 ± 4.32 | 82.97 ± 2.83 | 82.88 ± 3.22 | **82.59 ± 2.86** | 82.54 ± 3.56 | 82.38 ± 2.60 | 7.96 ± 0.28 | 51.84 ± 11.70 | 21.03 ± 4.44 |
| | | 0.1 | 83.76 ± 1.04 | 83.73 ± 1.22 | 83.63 ± 1.51 | **83.30 ± 1.86** | 83.34 ± 2.08 | 82.11 ± 2.53 | 4.15 ± 0.34 | 6.84 ± 0.66 | 3.35 ± 0.43 |
| | | 0.4 | 84.26 ± 0.11 | 84.23 ± 0.67 | 83.93 ± 1.10 | **83.86 ± 1.43** | 83.13 ± 1.18 | 82.44 ± 1.91 | 3.43 ± 0.28 | 5.02 ± 0.46 | 2.67 ± 0.21 |
| | 6 | 0 | 82.87 ± 1.47 | **82.42 ± 4.03** | **82.42 ± 4.21** | **82.08 ± 3.79** | **81.98 ± 2.13** | 80.86 ± 2.53 | 8.08 ± 0.24 | 54.59 ± 4.36 | 21.65 ± 1.69 |
| | | 0.1 | 83.76 ± 1.01 | **83.20 ± 1.41** | **83.17 ± 2.12** | **83.14 ± 2.36** | **83.10 ± 2.11** | 82.44 ± 2.68 | 4.47 ± 0.35 | 7.49 ± 0.96 | 3.80 ± 0.41 |
| | | 0.4 | 84.96 ± 0.74 | **84.36 ± 0.47** | **84.26 ± 1.90** | **84.13 ± 1.24** | **84.06 ± 1.01** | 82.48 ± 3.04 | 3.29 ± 0.15 | 4.29 ± 0.22 | 2.41 ± 0.09 |

improves its accuracy when using 500 commands.

Table 11: Comparison of the Cage-MetaCombiner vs the EvABCD algorithm for the Unix dataset (classification accuracy).

| Commands | Sequence | Cage-Combiner | EvABCD |
|---|---|---|---|
| 100 | 3 | 85.73 | 64.90 |
| | 4 | 83.86 | 64.50 |
| | 5 | 83.96 | 67.90 |
| | 6 | 85.06 | 64.30 |
| 500 | 3 | 84.39 | 59.50 |
| | 4 | 83.86 | 59.20 |
| | 5 | 83.76 | 66.70 |
| | 6 | 83.76 | 70.80 |

## 7. Conclusions and Future work

A meta-ensemble-based GP framework for classifying datasets in the cyber security domain and a real scenario concerning the segmentation of the users of an e-payment system, which illustrates the real applicability of the approach, are presented. The GP system is used to evolve the combiner function of the ensemble and permits to handle unbalanced classes thanks to a weighted fitness function, while the ensembles are specialized to handle

the different groups of likely missing features. Therefore, the main advantages of the framework are its capacity in handling groups of missing features and unbalanced datasets and the possibility of operating in an incremental way without the need for re-training on the original data. Experimental results conducted on artificial datasets and on two real datasets, demonstrate the proposed system improves or is comparable to state-of-the-art ensemble-based approaches, by using a smaller number of models. In particular, our approach behaves well in the classification of the minority classes. Finally, the accuracy does not degrade significantly when the percentage of missing data increases.

Future works aims to test the framework on real cyber security datasets and to extend the approach to work with streams of data.

## 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, Tech. rep. (2012).

[2] L. Breiman, Bagging predictors, Machine Learning 24 (2) (1996) 123–140.

[3] Y. Freund, R. Shapire, Experiments with a new boosting algorithm, in: Machine Learning, Proceedings of the Thirteenth International Conference (ICML '96), Morgan Kaufmann, 1996, pp. 148–156.

[4] G. Folino, C. Pizzuti, G. Spezzano, A scalable cellular implementation of parallel genetic programming, IEEE Transactions on Evolutionary Computation 7 (1) (2003) 37–53.

[5] D. F. de Oliveira, A. M. P. Canuto, M. C. P. de Souto, Use of multi-objective genetic algorithms to investigate the diversity/accuracy dilemma in heterogeneous ensembles, in: International Joint Conference on Neural Networks, IEEE, 2009, pp. 2339–2346.

[6] G. Folino, C. Pizzuti, G. Spezzano, Training Distributed GP Ensemble With a Selective Algorithm Based on Clustering and Pruning for Pattern Classification, IEEE Trans. Evolutionary Computation 12 (4) (2008) 458–468.

[7] C. D. Stefano, G. Folino, F. Fontanella, A. S. di Freca, Using bayesian networks for selecting classifiers in GP ensembles, Information Sciences 258 (2014) 200–216.

[8] J. Sylvester, N. V. Chawla, Evolutionary ensembles: Combining learning agents using genetic algorithms, in: AAAI Workshop on Multiagent Learning, 2005, pp. 46–51.

[9] N. Chawla, J. Sylvester, Exploiting diversity in ensembles: Improving the performance on unbalanced datasets, in: Multiple Classifier Systems, 7th International Workshop, Springer, 2007, pp. 397–406.

[10] J. Sylvester, N. V. Chawla, Evolutionary ensemble creation and thinning, in: Proceedings of the International Joint Conference on Neural Networks, IJCNN 2006, IEEE, 2006, pp. 5148–5155.

[11] N. Acosta-Mendoza, A. Morales-Reyes, H. J. Escalante, A. Gago-Alonso, Learning to assemble classifiers via genetic programming, IJPRAI 28 (7).

[12] M. Brameier, W. Banzhaf, Evolving teams of predictors with linear genetic programming, Genetic Programming and Evolvable Machines 2 (4) (2001) 381–407.

[13] Y. Wang, Y. Gao, R. Shen, F. Yang, Selective ensemble approach for classification of datasets with incomplete values, in: Foundations of Intelligent Systems, Springer, 2012, pp. 281–286.

[14] H. Chen, Y. Du, K. Jiang, Classification of incomplete data using classifier ensembles, in: Systems and Informatics (ICSAI), 2012 International Conference on, 2012, pp. 2229–2232.

[15] R. Polikar, J. DePasquale, H. S. Mohammed, G. Brown, L. I. Kuncheva, Learn++. mf: A random subspace approach for the missing feature problem, Pattern Recognition 43 (11) (2010) 3817–3832.

[16] D. Godoy, A. Amandi, User profiling in personal information agents: A survey, Knowl. Eng. Rev. 20 (4) (2005) 329–361.

[17] S. Gauch, M. Speretta, A. Chandramouli, A. Micarelli, User profiles for personalized information access, in: P. Brusilovsky, A. Kobsa, W. Nejdl (Eds.), The Adaptive Web, Vol. 4321 of Lecture Notes in Computer Science, Springer, 2007, pp. 54–89.

[18] J. A. Iglesias, P. P. Angelov, A. Ledezma, A. Sanchis, Creating evolving user behavior profiles automatically, IEEE Trans. Knowl. Data Eng. 24 (5) (2012) 854–867.

[19] T. D. M. Ovelgonne, V.S. Subrahmanian, A. Prakash, Global cyber-vulnerability report, in: Springer 2015, to appear.

[20] S. Greenberg, Using unix: Collected traces of 168 users, in: Research Report 88/333/45., Department of Computer Science, University of Calgary, Calgary, Canada., 1988.

[21] R. J. A. Little, D. B. Rubin, Statistical Analysis with Missing Data, John Wiley & Sons, Inc., New York, NY, USA, 1986.

[22] R. E. Schapire, The strength of weak learnability, Machine Learning 5 (2) (1990) 197–227.

[23] R. E. Schapire, Boosting a weak learning by majority, Information and Computation 121 (2) (1995) 256–285.

[24] L. Kuncheva, Combining Pattern Classifiers: Methods and Algorithms, Wiley-Interscience, 2004.

[25] S. Luke, L. Panait, A comparison of bloat control methods for genetic programming, Evol. Comput. 14 (3) (2006) 309–344.

[26] J. Demsar, Statistical comparisons of classifiers over multiple data sets, Journal of Machine Learning Research 7 (2006) 1–30.

[27] E. Bahri, N. Harbi, H. N. Huu, Approach based ensemble methods for better and faster intrusion detection, in: 4th Int. Conf. on Computational Intelligence in Security for Information Systems, Springer, June 8-10, 2011, pp. 17–24.