# An Evolutionary Algorithm for Learning Interpretable Ensembles of Classifiers

Henry E.L. Cagnini[1] [0000-0003-3889-2959]
Alex A. Freitas[2] [0000-0001-9825-4700]
Rodrigo C. Barros[1] [0000-0002-0782-9482]

[1] School of Technology, PUCRS
{henry.cagnini@edu., rodrigo.barros@}pucrs.br
[2] Computing School, University of Kent
a.a.freitas@kent.ac.uk

**Abstract.** Ensembles of classifiers are a very popular type of method for performing classification, due to their usually high predictive accuracy. However, ensembles have two drawbacks. First, ensembles are usually considered a 'black box', non-interpretable type of classification model, mainly because typically there are a very large number of classifiers in the ensemble (and often each classifier in the ensemble is a black-box classifier by itself). This lack of interpretability is an important limitation in application domains where a model's predictions should be carefully interpreted by users, like medicine, law, etc. Second, ensemble methods typically involve many hyper-parameters, and it is difficult for users to select the best settings for those hyper-parameters. In this work we propose an Evolutionary Algorithm (an Estimation of Distribution Algorithm) that addresses both these drawbacks. This algorithm optimizes the hyper-parameter settings of a small ensemble of 5 interpretable classifiers, which allows users to interpret each classifier. In our experiments, the ensembles learned by the proposed Evolutionary Algorithm achieved the same level of predictive accuracy as a well-known Random Forest ensemble, but with the benefit of learning interpretable models (unlike Random Forests).

**Keywords:** classification · evolutionary algorithms · ensemble learning · machine learning · supervised learning

## 1 Introduction

The classification task of machine learning consists of training predictive models for decision-making purposes [31]. Traditionally, classification research has focused mainly on the learned model's predictive accuracy, but model interpretability by users is currently a very active and important topic [6], especially in areas such as medicine, credit scoring, bioinformatics, and churn prediction [12]. Model interpretability is particularly critical in scenarios where models can lead to life-or-death decisions (such as in medicine), or influence decisions that can put several lives at risk, such as the use of recommendation algorithms in a nuclear

power plant [12]. Model interpretability is also often required by law, a major example being the European Union's *Data Protection Regulation*, which includes a "right to explanation" [13].

One field of machine learning that could benefit from interpretability is classification with ensemble learning. Ensembles are sets of classifiers which, when combined, usually perform better than a single strong model, such as when comparing a Random Forest ensemble [4] with a single decision tree learned by C4.5 [29, 2]. Ensembles often have hundreds or thousands of models, which greatly hinder their interpretability by human users. Moreover, ensembles often consist of black box base models (e.g. neural networks or support vector machines) that prevent any direct interpretation of their reasoning. Tackling these problems involves learning a small ensemble, consisting of a few directly interpretable models, so that users can interpret each of the models in the ensemble. This is the main problem addressed in this work.

The second problem addressed in this work is that selecting the best setting (or configuration) of hyper-parameters for each base learner in an ensemble is a difficult task *per se* [32, 10], which involves testing a very large number of candidate hyper-parameter settings in order to find the best setting for the dataset at hand. Auto-ML (Automated Machine Learning) has recently gained attention due to its capacity of relieving the end user from a manual optimization of algorithms' hyper-parameters, which can be repetitive, tiresome, and often requires advanced domain-specific knowledge [33, 10, 28].

One way to perform Auto-ML is to employ a population-based algorithm, which explores several regions in the solution space in parallel, and adapts its search depending on the quality of solutions found in those regions. Hence, evolutionary algorithms seem to be a natural choice for the Auto-ML task of optimizing the settings of ensembles' hyper-parameters [23, 20, 33, 14], due to performing a global search in the solution space. Among several types of evolutionary algorithms, we propose an Estimation of Distribution Algorithm (EDA) to evolve an ensemble of interpretable classifiers.

The main difference between EDAs [24] and Genetic Algorithms (GA) [17] is that while GAs implicitly propagate characteristics of good solutions throughout evolution (by carrying on high-quality individuals from one generation to another), EDAs do this explicitly, by encoding those characteristics in a probabilistic graphical model (GM) [16, 27].

The rest of this paper is organized as follows. Section 2 describes our proposed method. Sections 3 and 4 present the experimental setup and experimental results, respectively. Section 5 discusses related work. Section 6 presents the conclusions and future research directions.

## 2 The Proposed Estimation of Distribution Algorithm (EDA) for Evolving Ensembles

EDAs evolve a probabilistic graphic model of candidate solutions, so that candidate solutions (individuals) are sampled from that model and evaluated at

each generation. In general, an EDA consists of three stages performed at each generation: (a) sampling of new individuals (candidate solutions) from the probabilistic graphic model; (b) evaluation of the new individuals' performance; and (c) updating of the probabilistic graphic model, based on the best individuals selected from the current generation. Importantly, EDAs avoid the need for specifying genetic operators like crossover and mutation (and their corresponding probabilities of application). That is, instead of generating new individuals by applying genetic operators to selected individuals, they generate new individuals by sampling from the current probabilistic graphic model, which captures the main characteristics of the best individuals selected (based on fitness) along the evolutionary process.

Among several EDA types, we chose PBIL: Probabilistic Incremental Learning [3]. The main characteristic of PBIL is that it assumes independence between variables in the probabilistic graphical model. Although this has the disadvantage of ignoring interactions among variables, it has an important advantage in the context of our task of evolving an ensemble of classifiers: it makes PBIL much more computationally efficient by comparison with other EDA types that consider complex variable interactions – whilst still allowing PBIL to learn ensembles with good predictive accuracy, as shown later.

Another aspect of PBIL is the use of a learning rate $\alpha$ hyper-parameter for updating probabilities in the graphical model, making this process smoother. Take for example two initial probabilities for a binary variable $V$, $P(V = 0) = 0.5$ and $P(V = 1) = 0.5$, and a learning rate of 0.3. Assume only two individuals are selected to update the graphic model's probabilities, and both have $V = 0$. In this extreme case, an EDA without learning rate would update $V$ so that it would be $P(V = 0) = \frac{2}{2} = 1$ and $P(V = 1) = \frac{0}{2} = 0$ in the next generation. However, using a learning rate, the new probabilities for $V$ are $P(V = 0) = (1 - 0.3) \times 0.5 + 0.3 \times \frac{2}{2} = 0.65$ and $P(V = 1) = (1 - 0.3) \times 0.5 + 0.3 \times \frac{0}{2} = 0.35$. Section 2.3 discusses in more detail how probabilities are updated.

PBIL keeps track of the best individual found so far in a variable $\varphi$. At the end of a PBIL run, the returned solution can be the best individual stored in $\varphi$ or the best individual in the last generation (these two approaches are compared later).

### 2.1 Individuals (Candidate Solutions)

Each individual is an ensemble, composed of five base models (each learned by a different type of base learner) and an aggregation policy. Regarding base learners, we chose the ones that can generate readily interpretable models [12, 18, 26]. The recent literature on classification focuses mainly on producing classifiers with ever-increasing predictive performance, with little attention devoted to interpretability [13]. For instance, deep learning classifiers, which have received great attention lately due to obtaining high predictive accuracy in image tasks, are very difficult to interpret [13], with interested researchers shifting the focus from interpreting the models themselves to interpreting their predictions [22].

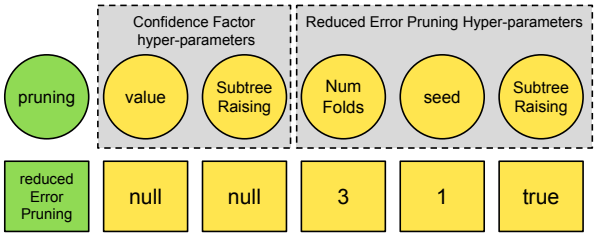



Fig. 1: An example individual in PBIL. Note that, although PBIL assumes probabilistic independence between variables, some values are dependent on others.

The five base learners employed are two decision-tree induction algorithms (C4.5 [29] and CART [5]); two rule induction algorithms (RIPPER [7] and PART [11]); and a Decision Table algorithm [19]. We use these algorithms' implementations in the well-known Weka Toolkit [15]. For the rest of this paper, we will refer to them by their Weka names: J48 for C4.5, SimpleCart for CART, JRip for RIPPER, PART, and Decision Table.

An individual is encoded as an array, where each position denotes a variable, and each value denotes the assigned value for that variable. Some variables may not have any value, because they are not used by an individual. Figure 1 depicts a portion of an individual's array, regarding some variables of its J48 classifier. J48 has three options for tree pruning: *reduced error pruning*, *confidence factor*, and *unpruned*. For this example individual, *reduced error pruning* is used. For this reason, there is no need to set hyper-parameters of the *confidence factor* strategy, which are then set to *null*.

**Aggregators** An aggregator is a method responsible for finding a consensus among votes from base models. Consider a three-dimensional probability matrix P, of dimensions $(B, N, C)$ – respectively the number of base classifiers in the ensemble, number of instances, and number of classes. The objective of an aggregator is to transform this three-dimensional matrix into a unidimensional array of length $N$, where each position has the predicted class for each instance.

We use two types of aggregators: majority voting and weighted aggregators. The probabilistic majority voting aggregator uses the fusion function described in [21, p. 150]:

$$\rho_c^{(j)} = \sum_{i=1}^{B} P_c^{(i,j)} \tag{1}$$

$$h(X^{(j)}) = \arg\max_{c \in C} \left( \frac{\rho_c^{(j)}}{\sum_{k=1}^{C} \rho_k^{(j)}} \right) \tag{2}$$

where $\rho_c^{(j)}$ is the sum of the probabilities that the $j$-th instance has the $c$-th class, over all $B$ classifiers, and $C$ is the number of classes. The weighted aggregator is similar to majority voting, except that individual probabilities from classifiers are weighted according to the fitness of each classifier:

$$\rho_c^{(j)} = \sum_{i=1}^{B} \psi^{(i)} P_c^{(i,j)} \tag{3}$$

$$h(X^{(j)}) = \arg\max_{c \in C} \left( \frac{\rho_c^{(j)}}{\sum_{k=1}^{C} \rho_k^{(j)}} \right) \tag{4}$$

where $\psi^{(i)}$ is the fitness value of individual $S^{(i)}$.

### 2.2 Fitness evaluation

At the start of the evolutionary process, PBIL receives a training set. This set is splitted into five subsets, which are used to compute each individual's fitness by performing an internal 5-fold stratified cross validation (SCV). By keeping the subsets constant throughout all evolutionary process, we allow direct comparisons between individuals from different generations. The fitness function is the Area Under the Receiving Operator Characteristic (ROC) curve (AUC) [8] – a popular predictive accuracy measure.

AUC values are within $[0, 1]$, with the value 0.5 representing the predictive accuracy of random predictions in the case of binary-class problems. In this work, regardless of the number of classes in the dataset, we calculate one AUC for each class, and then average the AUC among all classes. Hence, the fitness of an individual is actually a mean of means: first, the mean AUC among all classes, for a given fold; then, the mean AUC among all five internal folds. Figure 2 depicts the fitness calculation procedure.

```
1:  function COMPUTE_FITNESS(X, y, C)
2:      train ← (generate_train_subsets(y))
3:      val ← (generate_validation_subsets(y))
4:      Ψ ← (0|i = 1, 2, ..., |S|)
5:      for i = 1, 2, ..., |S| do
6:          for k = 1, ..., 5 do
7:              S^(i) ← build_model(X^(train^(k)), y^(train^(k)))
8:              P^(i) ← predict(S^(i), X^(val^(k)))
9:              ψ ← 1/5C Σ_{c=1}^{C} AUC(P_c^(i), (y^(j) = c|j ∈ val^(k)))
10:             Ψ^(i) ← Ψ^(i) + ψ
11:             )
12:     return Ψ
```

Fig. 2: Pseudo-code used for calculating fitness.

### 2.3 PBIL's Probabilistic Graphical Model

At each generation, new individuals are sampled from the probabilistic graphical model (GM), and the best individuals will update GM's probabilities. Recall that PBIL assumes that the variables in the GM are independent, although we know (as shown in Figure 1) that there are some dependencies. However, this does not prevent PBIL from finding good-performing solutions; analogously to the overall good performance of the Naïve Bayes classifier, which also assumes that attributes are independent [30].

The sampling procedure is based on hierarchical relationships among the variables representing hyper-parameters in PBIL's GM, as shown in Figure 3, where the top-level variables are hyper-parameters of base learners that will activate or deactivate the sampling of other variables/hyper-parameters at a lower level. When sampling a new individual, higher-level variables are sampled first, and their descendants are sampled next. Using J48 as example, the variables for this algorithm are *useLaplace, minNumObj, useMDLcorrection, collapseTree, doNotMakeSplitPointActualValue, binarySplits*, and *pruning*. Since none of these variables have any descendent variable, with the exception of *pruning*, the sampling proceeds to choose which type of pruning will be used by J48, and depending on the chosen option, it samples the variables descendent to that option. Unused variables are set to *null*. Once all pertinent variables are sampled, their values are fed to the base classifier constructor, which will in turn generate the model. Figure 3 depicts the variables in PBIL's GM.

**Initial values** There are two types of variables in PBIL's GM: 48 discrete and 2 continuous variables. Discrete variables were first introduced in the original PBIL work [3]. We use the EDA ability of biasing probabilities to increase by 10% the probability to sample values that are the base learner's default in Weka. For all other values, we set uniform probabilities. For instance, for J48's *numFolds*, the default value 3 folds has probability 20%, while each other value in $\{2, 4, 5, 6, 7, 8, 9, 10\}$ has probability 10%. Exceptionally for variable *evaluationMeasure* of Decision Table, value *auc* has a 50% probability of being sampled. We do this to increase the chances that a base learner is using the same metric used as fitness function, which in this work is the AUC.

For continuous variables, we use unidimensional Gaussian distributions. The mean is the default Weka value for the hyper-parameter, and the standard deviation was chosen in a way that borderline values have at least 10% chance of being sampled. Values outside valid range are clipped to the closest valid value. The range of valid values was inferred by inspecting Weka's source code. The list of of variables and its values is present in the source-code of our method [3].

**Updating PBIL's GM** The updating of the variables' probabilities is dependent on their type. If a variable is discrete, the update follows the scheme known as PBIL-iUMDA [33, 34], shown in equation 5:

[3] Available at `https://github.com/henryzord/PBIL`

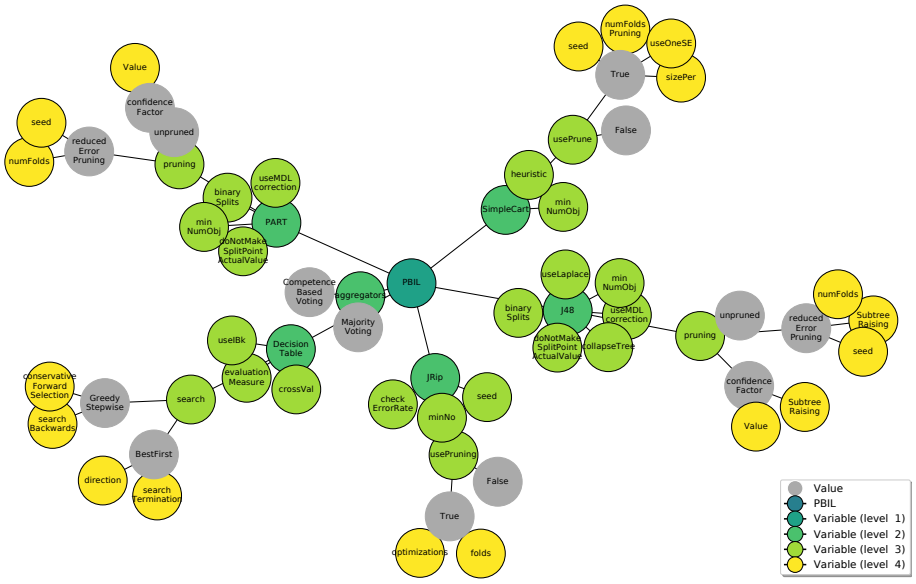



Fig. 3: Graphical Model used by PBIL. Edges denote an implicit correlation, since no probabilistic correlation is designed. Borderless nodes denote values that were sampled from the variable in the above level (i.e. not variables).

$$p_{g+1}(V_j = v) = (1 - \alpha) \times p_g(V_j = v) + \alpha \times p_{\Phi,g}(V_j = v) \tag{5}$$

where $p_g(V_j = v)$ is the probability that variable $V_j$ assumes discrete value $v$ in the $g$-th generation (estimated by the proportion of observed occurrences of value $v$ for variable $V_j$ among all individuals in that generation), $\alpha$ is the learning rate, and $p_{\Phi,g}(V_j = v)$ is the proportion of occurrences of value $v$ for $V_j$ in the set of individuals $\Phi$ which were selected (based on fitness) at the $g$-th generation. This process is iterated over all values of a discrete variable. Note that when computing $p_g(V_j = v)$ and $p_{\Phi,g}(V_j = v)$, if some individuals do not have any value set for variable $V_j$, their *null* values are discarded and do not contribute at all to the updating of probabilities for $V_j$'s values.

Equation 5 was adapted to deal with continuous variables, which encode the mean and the standard deviation of a normal distribution, as follows. The mean of the normal distribution is updated by

$$\mu_{g+1}(V_j) = \mu_g(V_j) + \alpha \times (\mu_g(V_j) - \mu_{\Phi,g}(V_j)) \tag{6}$$

where $\mu_g(V_j)$ is the mean of the normal distribution of the $j$-th variable $V_j$ in the $g$-th generation, $\alpha$ is the learning rate, and $\mu_{\Phi,g}(V_j)$ is the observed mean for the variable $V_j$ in the set of individuals $\Phi$ selected at the $g$-th generation, again considering only individuals where the variable $V_j$ was used.

The standard deviation is decreased as follows:

$$\sigma_{g+1}(V_j) = \sigma_g(V_j) - \frac{\sigma_1(V_j)}{G} \tag{7}$$

where $\sigma_g(V_j)$ is the standard deviation of the $j$-th variable $V_j$ in the $g$-th generation, $\sigma_1(V_j)$ the initial standard deviation for variable $V_j$, and $G$ is the number of generations.

### 2.4 Early Stop and Termination

If the fitness of the best individual does not improve more than $\epsilon$ in $\varepsilon$ generations, we assume that PBIL is overfitting the training data, and terminate the run of the algorithm. In our experiments, we use $\epsilon = 5 \times 10^{-4}$ and $\varepsilon = 10$.

At the end of the evolutionary process, we report the best individual from the last generation as the final solution.

### 2.5 Complexity Analysis

Assume $T(train)$ to be the time to train an ensemble, and $T(fitness)$ to be the time to assert the fitness of said ensemble. At every generation $S$ new ensembles are generated. This process is repeated at most $G$ times (assuming that the early stop mechanism of the previous section is not triggered). This procedure has complexity $GS \times (T(train) + T(fitness))$.

Sampling and updating the graphical model are procedures directly dependent on the number of variables $|V|$. Variables need first to be initialized with default values, for later sampling and update. Variables are sampled $S$ times every generation, and are updated based on the number of fittest individuals, $|\Phi|$. For each variable, we iterate over all of its values, but we assume the number of its values not to be significant – discrete variables have between 2 and 10 values, with 4 as average; continuous variables count as 2 values, i.e. mean and standard deviation of normal distributions. From this analysis we have $|V| \times (1 + G(S + |\Phi|))$. Thus, the overall complexity of training the proposed PBIL is

$$O(GS \times (T(train) + T(fitness)) + |V| \times G(S + |\Phi|)) \tag{8}$$

## 3 Experimental Setup

### 3.1 PBIL's hyper-parameter optimization

In order to find the best configuration to run PBIL, we perform a grid-search for optimizing five of its hyper-parameters, using eight datasets, hereafter called *parameter-optimization* datasets, described in Table 1. We measure PBIL's AUC on each dataset using a well-known 10-fold cross validation procedure. We emphasize that these datasets were used only for PBIL's hyper-parameter optimization, i.e., they were not used to compare PBIL with baseline algorithms, thus avoiding over-optimistic measures of predictive performance.

We optimize 5 hyper-parameters, with the following range of values: population size : {75, 150}; number of generations: {100, 200}; learning rate: {0.3, 0.5, 0.7}; selection share {0.3, 0.5}; and whether the type of solution returned is the best individual from the last generation or the best individual produced across all generations. Thus, from the 48 combinations of hyper-parameter values, we found the combination that provided the best average AUC across all datasets to be: population size $|S| = 75$, number of generations $G = 200$, learning rate $\alpha = 0.7$, share (proportion) of selected individuals $|\Phi|/|S| = 0.5$, and type of solution returned = best individual from last generation. We use this configuration for conducting further experiments.

### 3.2 Baseline Algorithms

We compare PBIL with other two algorithms: a baseline ensemble and Random Forest. The baseline ensemble consists of the five base classifiers from PBIL (namely J48, CART, JRip, PART, and Decision Table) with their default hyper-parameter configuration (according to Weka), and a simple majority voting scheme as aggregation policy. The intention of using this baseline algorithm is to check if there is a difference between simply using an ensemble of classifiers, with the simplest voting aggregation policy (i.e. majority voting), and optimizing their hyper-parameter configuration with PBIL.

Random Forest [4] is a well-known ensemble algorithm, and in general it is among the best classification methods regarding predictive performance [9]. A random forest ensemble is solely composed of decision trees. Each decision tree is learned from a different subset of N instances, randomly sampled with replacement from the training set. For each internal node in each tree, a subset of $M$ attributes is randomly sampled without replacement, and the attribute that minimizes the local class impurity is selected as splitting criterion. This process is repeated recursively until no further split improves the impurity metric, when nodes are then turned into leaves.

Random forests usually require a large number of trees in the ensemble to achieve good predictive performance. Also, despite using decision trees, the ensemble as a whole is not directly interpretable, since there are a very large number of trees. Even if the number of trees were small, interpreting each tree would still be problematic due to the large degree of randomness involved in learning each tree. That randomness is necessary to provide diversity to the ensemble, which improves its predictive accuracy, but it hinders interpretability. There are indirect approaches to interpret random forests, using variable importance measures to rank the variables based on their importance in the model, but such measures are out of the scope of this paper.

We also performed a grid-search for optimizing 3 hyper-parameters of Random Forest, with their following ranges of values: number of trees in the forest: {100, 200, 300, 400, 500}; whether to randomly break ties between equally attractive attributes at each tree node, or to simply use the attribute with the smallest index; and maximum tree depth: {0 (no limit), 1, 2, 3, 4}. Hence, Random Forests and PBIL had about the same number of configurations tested by

the grid search (50 and 48), and both had their configurations optimized in the same 8 parameter-optimization datasets shown in Table 1, to be fair. We found the best combination to be: number of iterations = 300, do not break ties randomly, and max tree depth = 4. We leave the other two hyper-parameters, the number of instances in the bag for learning each decision tree, and the number of sampled attributes at each tree node, at their default Weka values, respectively $N$ and $\log_2(M-1)+1$.

### 3.3 Datasets

We use a different set of 9 datasets, described in Table 1, for comparing the predictive performance of the tested algorithms. All datasets, including the ones used for hyper-parameter optimization, were collected from KEEL[4] [1] and the UCI Machine Learning repository[5] [25].

Table 1: Datasets used in this work.

| Dataset | Instances | Attributes | | | Classes |
|---|---|---|---|---|---|
| | | Total | Categorical | Numeric | |
| Hyper-parameter optimization datasets | | | | | |
| australian | 690 | 14 | 6 | 8 | 2 |
| bupa | 345 | 6 | 0 | 6 | 2 |
| contraceptive | 1473 | 9 | 0 | 9 | 3 |
| flare | 1066 | 11 | 11 | 0 | 6 |
| german | 1000 | 20 | 13 | 7 | 2 |
| pima | 768 | 8 | 0 | 8 | 2 |
| vehicle | 846 | 18 | 0 | 18 | 4 |
| wisconsin | 699 | 9 | 0 | 9 | 2 |
| Predictive performance assessment datasets | | | | | |
| balance-scale | 625 | 4 | 0 | 4 | 3 |
| blood-transfusion | 748 | 4 | 0 | 4 | 2 |
| credit-approval | 690 | 15 | 9 | 6 | 2 |
| diabetic | 1151 | 19 | 3 | 16 | 2 |
| hcv-egypt | 1385 | 28 | 9 | 19 | 4 |
| seismic-bumps | 2584 | 18 | 4 | 14 | 2 |
| sonar | 208 | 60 | 0 | 60 | 2 |
| turkiye | 5820 | 32 | 32 | 0 | 13 |
| waveform | 5000 | 40 | 0 | 40 | 3 |

## 4 Experimental Results

For each algorithm and each dataset, we run 10 times a 10-fold cross validation procedure, and report the mean unweighted area under the ROC curve among the 10 executions. The results are shown in Table 2.

[4] Available at `https://sci2s.ugr.es/keel/datasets.php`
[5] Available at `https://archive.ics.uci.edu/ml/datasets`

Table 2: Area under the ROC curve and standard deviations for compared algorithms. Best result for each dataset is shown in bold font.

| Dataset | Random Forest | baseline ensemble | the proposed PBIL |
|---|---|---|---|
| balance-scale | 0.8441 ± 0.05 | **0.8766** ± 0.01 | 0.8560 ± 0.00 |
| blood-transfusion | **0.7354** ± 0.03 | 0.7335 ± 0.00 | 0.6742 ± 0.00 |
| credit-approval | **0.9358** ± 0.02 | 0.9270 ± 0.00 | 0.9267 ± 0.00 |
| diabetic | 0.7307 ± 0.04 | 0.7370 ± 0.01 | **0.7674** ± 0.00 |
| hcv-egypt | 0.5073 ± 0.05 | 0.4850 ± 0.01 | **0.5167** ± 0.00 |
| seismic-bumps | **0.7823** ± 0.07 | 0.7715 ± 0.01 | 0.7553 ± 0.00 |
| sonar | 0.9214 ± 0.08 | 0.8612 ± 0.01 | **0.9356** ± 0.00 |
| turkiye | **0.8549** ± 0.01 | 0.8542 ± 0.00 | 0.8213 ± 0.00 |
| waveform | 0.9562 ± 0.00 | 0.9502 ± 0.00 | **0.9670** ± 0.00 |

Regarding predictive performance, PBIL and Random forests obtained overall the best results, each with the highest AUC value in 4 datasets. The baseline method obtained the highest value in only one dataset. The largest difference in performance was observed in the blood-transfusion dataset, where the baseline and the Random Forest obtained an AUC value about 6% higher than the AUC of PBIL. In the other datasets, the differences of AUC values among the three methods was relatively small, about 3% or less in general. We believe this is due to the skewed nature of the class distribution in the blood-transfusion dataset.

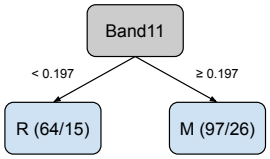
In addition, the ensembles learned by PBIL and the baseline method have the advantage of consisting of only 5 interpretable base classifiers; so they are directly interpretable by users, unlike Random Forests (as discussed earlier).

Figures 4 and 5 show an ensemble learned by PBIL from the sonar dataset, as an example of such ensembles' interpretability. The models learned by J48 and SimpleCART are both small (with 3 and 13 nodes) and consistently identify Band11 as the most relevant variable in their root nodes. The rule lists learned by JRip and PART are also small, with 5 and 8 rules (most being short rules). The decision table is not so short, with 25 rows, but the fact that all rows refer to the same selected attributes and in the same order (unlike decision trees and rule sets) improves interpretability by users [12].

## 5 Related Work

Several evolutionary algorithms have been recently proposed for evolving ensembles of classifiers. In [33], another PBIL version was proposed to select the best combination of ensemble method (e.g. bagging, boosting, etc), base learners (e.g. neural networks, SVMs, decision trees, etc.) and their hyper-parameter settings for a given dataset. However, that work focused only on predictive accuracy, so that their learned ensembles are in general non-interpretable (due to being very large and often consisting of non-interpretable classifiers), unlike the ensembles learned in this current work.

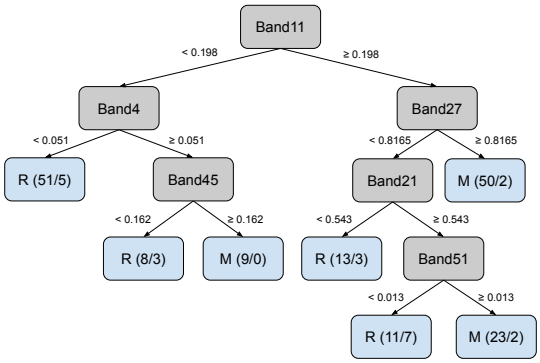
In [20], a Genetic Programming algorithm is used to optimize configurations of ensemble methods (bagging, boosting, etc) and their base learners (logistic regressors, neural networks, etc). In addition, [23] proposes a co-evolutionary




(a) J48

| conditions | predicted class |
|---|---|
| Band11 ≤ 0.168 AND Band49 ≤ 0.04 | R (42.0/1.0) |
| Band37 ≥ 0.46 AND Band17 ≥ 0.42 | R (20.0/2.0) |
| Band9 ≤ 0.097 AND Band31 ≥ 0.353 | R (15.0/2.0) |
| Band51 ≤ 0.012 AND Band23 ≤ 0.681 AND Band41 ≤ 0.271 | R (8.0/0.0) |
| otherwise | M (102.0/7.0) |

(b) JRip




(c) SimpleCART

| conditions | predicted class |
|---|---|
| Band11 ≤ 0.198 AND Band52 ≤ 0.0205 AND Band5 ≤ 0.0695 AND Band10 ≤ 0.1665 AND Band7 > 0.0415 | R (40.0) |
| Band47 > 0.063 AND Band37 ≤ 0.48 AND Band18 ≤ 0.914 AND Band49 > 0.0285 | M (64.0) |
| Band54 ≤ 0.0225 AND Band45 > 0.2745 AND Band2 ≤ 0.044 | M (9.0) |
| Band54 ≤ 0.0225 AND Band8 > 0.0655 AND Band27 ≤ 0.846 AND Band28 > 0.3585 AND Band4 ≤ 0.109 AND Band3 ≤ 0.0655 | R (25.0) |
| Band8 > 0.0625 AND Band12 > 0.154 AND Band54 > 0.0105 | M (17.0) |
| Band8 ≤ 0.104 | R (14.0) |
| Band17 > 0.4445 | R (11.0/3.0) |
| otherwise | M (7.0) |

(d) PART

Fig. 4: Four of the five base classifiers from the best individual of PBIL for a given run of the sonar dataset. *type* is the class variable (with class labels Rock (R) and Metal (M)), and broadly speaking the features represent the echo returned from hitting rock and metal objects with different frequencies of audio waves.

algorithm for finding the best combination of hyper-parameters for a set of base classifiers, which might also include the best combination of data pre-processing methods for a given dataset. AUTO-CVE concurrently evolves two populations: a population of base models (using Genetic Programming) and a population of ensembles (using a Genetic Algorithm). In both [20] and [23], again the focus was on predictive accuracy, and those works tend to produce very large ensembles of

| band11 | band16 | band19 | band36 | band45 | band48 | band52 | band56 | type |
|---|---|---|---|---|---|---|---|---|
| (0.198-∞) | all | all | (-∞-0.4425] | (-∞-0.3855] | (-∞-0.0755] | (0.0095-∞) | all | m |
| (-∞-0.198] | all | all | (-∞-0.4425] | (-∞-0.3855] | (-∞-0.0755] | (0.0095-∞) | all | r |
| (0.198-∞) | all | all | (-∞-0.4425] | (-∞-0.3855] | (0.0755-∞) | (-∞-0.0095] | all | m |
| (-∞-0.198] | all | all | (-∞-0.4425] | (-∞-0.3855] | (0.0755-∞) | (-∞-0.0095] | all | r |
| (0.198-∞) | all | all | (-∞-0.4425] | (0.3855-∞) | (-∞-0.0755] | (-∞-0.0095] | all | r |
| (0.198-∞) | all | all | (0.4425-∞) | (-∞-0.3855] | (-∞-0.0755] | (-∞-0.0095] | all | m |
| (-∞-0.198] | all | all | (0.4425-∞) | (-∞-0.3855] | (-∞-0.0755] | (-∞-0.0095] | all | r |
| (0.198-∞) | all | all | (-∞-0.4425] | (-∞-0.3855] | (-∞-0.0755] | (-∞-0.0095] | all | r |
| (-∞-0.198] | all | all | (-∞-0.4425] | (-∞-0.3855] | (-∞-0.0755] | (-∞-0.0095] | all | r |
| (-∞-0.198] | all | all | (0.4425-∞) | (0.3855-∞) | (0.0755-∞) | (0.0095-∞) | all | r |
| (0.198-∞) | all | all | (0.4425-∞) | (0.3855-∞) | (0.0755-∞) | (0.0095-∞) | all | m |
| (-∞-0.198] | all | all | (-∞-0.4425] | (0.3855-∞) | (0.0755-∞) | (0.0095-∞) | all | m |
| (0.198-∞) | all | all | (-∞-0.4425] | (0.3855-∞) | (0.0755-∞) | (0.0095-∞) | all | m |
| (-∞-0.198] | all | all | (0.4425-∞) | (-∞-0.3855] | (0.0755-∞) | (0.0095-∞) | all | r |
| (0.198-∞) | all | all | (0.4425-∞) | (-∞-0.3855] | (0.0755-∞) | (0.0095-∞) | all | r |
| (0.198-∞) | all | all | (0.4425-∞) | (0.3855-∞) | (-∞-0.0755] | (0.0095-∞) | all | r |
| (0.198-∞) | all | all | (0.4425-∞) | (0.3855-∞) | (0.0755-∞) | (-∞-0.0095] | all | m |
| (-∞-0.198] | all | all | (0.4425-∞) | (0.3855-∞) | (0.0755-∞) | (-∞-0.0095] | all | r |
| (-∞-0.198] | all | all | (-∞-0.4425] | (-∞-0.3855] | (0.0755-∞) | (0.0095-∞) | all | m |
| (0.198-∞) | all | all | (-∞-0.4425] | (-∞-0.3855] | (0.0755-∞) | (0.0095-∞) | all | m |
| (0.198-∞) | all | all | (-∞-0.4425] | (0.3855-∞) | (0.0755-∞) | (-∞-0.0095] | all | r |
| (0.198-∞) | all | all | (0.4425-∞) | (-∞-0.3855] | (-∞-0.0755] | (0.0095-∞) | all | r |
| (-∞-0.198] | all | all | (0.4425-∞) | (-∞-0.3855] | (-∞-0.0755] | (0.0095-∞) | all | r |
| (0.198-∞) | all | all | (0.4425-∞) | (-∞-0.3855] | (0.0755-∞) | (-∞-0.0095] | all | r |
| (-∞-0.198] | all | all | (0.4425-∞) | (-∞-0.3855] | (0.0755-∞) | (-∞-0.0095] | all | r |

Fig. 5: The decision table learned by the best individual from PBIL for a given run of the sonar dataset. This classifier is part of the ensemble composed of classifiers from Figure 4.

non-interpretable base classifiers. By contrast, in the current work the learned ensembles are small (with only 5 base classifiers) and consist of interpretable classifiers by design.

## 6 Conclusion and Future Work

We presented a new evolutionary algorithm (a version of PBIL) for optimizing the configuration of a small ensemble of interpretable classifiers, aiming at maximizing predictive performance on the dataset at hand whilst generating interpretable models by design. The proposed PBIL and Random Forest achieved the best predictive accuracy overall – each was the best in 4 of 9 datasets. The baseline ensemble was the best in one dataset.

Both the proposed PBIL and the baseline ensemble produce interpretable models consisting of only 5 interpretable classifiers, unlike random forest ensembles, which are not directly interpretable as discussed earlier. Note that the

baseline ensemble proposed here is not a standard ensemble in the literature, because the literature focuses on large, non-interpretable ensembles. Hence, the results for the baseline ensemble reported here can also be seen as a contribution to the literature, in the sense of being further evidence (in addition to the PBIL's results) that small ensembles of interpretable classifiers can be competitive against large, non-interpretable ensembles.

Future work will involve designing a more advanced version of PBIL encoding dependencies among variables in the graphical model and doing other experiments with more datasets.

## Acknowledgment

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001.

## References

1. Alcalá-Fdez, J., et al.: Keel data-mining software tool: data set repository, integration of algorithms and experimental analysis framework. Journal of Multiple-Valued Logic & Soft Computing **17**, 255–287 (2011)
2. Ali, J., et al.: Random forests and decision trees. International Journal of Computer Science Issues (IJCSI) **9**(5), 272 (2012)
3. Baluja, S., Caruana, R.: Removing the genetics from the standard genetic algorithm. In: Proceedings of the Twelfth International Conference on Machine Learning. pp. 38–46. Elsevier, Tahoe City, USA (1995)
4. Breiman, L.: Random forests. Machine learning **45**(1), 5–32 (2001)
5. Breiman, L., et al.: Classification and regression trees. Wadsworth International Group, Belmont, California (1984)
6. Carvalho, D.V., Pereira, E.M., Cardoso, J.S.: Machine Learning Interpretability: A Survey on Methods and Metrics. Electronics **8**(8), 832 (2019)
7. Cohen, W.W.: Fast Effective Rule Induction. In: Twelfth International Conference on Machine Learning. pp. 115–123 (1995)
8. Fawcett, T.: An introduction to ROC analysis. Pattern recognition letters **27**(8), 861–874 (2006)
9. Fernández-Delgado, M., et al.: Do we need hundreds of classifiers to solve real world classification problems? The Journal of Machine Learning Research **15**(1), 3133–3181 (2014)
10. Feurer, M., et al.: Efficient and robust automated machine learning. In: Advances in Neural Information Processing Systems. pp. 2962–2970 (2015)
11. Frank, E., Witten, I.H.: Generating Accurate Rule Sets Without Global Optimization. In: Fifteenth International Conference on Machine Learning. pp. 144–151 (1998)
12. Freitas, A.A.: Comprehensible Classification Models: A Position Paper. ACM SIGKDD Explorations Newsletter **15**(1), 1–10 (2014)
13. Fürnkranz, J., Kliegr, T., Paulheim, H.: On cognitive preferences and the plausibility of rule-based models (2018), arXiv preprint arXiv:1803.01316
14. Galea, M., Shen, Q., Levine, J.: Evolutionary Approaches to Fuzzy Modelling for Classification. The Knowledge Engineering Review **19**(1), 27–59 (2004)

15. Hall, M., et al.: The WEKA data mining software: an update. ACM SIGKDD explorations newsletter **11**(1), 10–18 (2009)
16. Hauschild, M., Pelikan, M.: An Introduction and Survey of Estimation of Distribution Algorithms. Swarm and Evolutionary Computation **1**(3), 111–128 (2011)
17. Holland, J.H.: Adaptation in Natural and Artificial Systems: an Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. MIT press (1992)
18. Huysmans, J., et al.: An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models. Decision Support Systems **51**(1), 141–154 (2011)
19. Kohavi, R.: The Power of Decision Tables. In: 8th European Conference on Machine Learning. pp. 174–189 (1995)
20. Kordik, P., Cerny, J., Fryda, T.: Discovering Predictive Ensembles for Transfer Learning and Meta-learning. Machine Learning **107**, 177–207 (2018)
21. Kuncheva, L.I.: Combining pattern classifiers: methods and algorithms. John Wiley & Sons (2004)
22. Lapuschkin, S., et al.: Unmasking Clever Hans predictors and assessing what machines really learn. Nature communications **10**(1), 1096 (2019)
23. Larcher, C., Barbosa, H.: Auto-CVE: a coevolutionary approach to evolve ensembles in automated machine learning. In: Proceedings of The Genetic and Evolutionary Computation Conference. pp. 392–400 (July 2019). https://doi.org/10.1145/3321707.3321844
24. Larrañaga, P., Lozano, J.A.: Estimation of Distribution Algorithms: A new Tool for Evolutionary Computation. Springer (2001)
25. Lichman, M.: UCI machine learning repository (2013), `http://archive.ics.uci.edu/ml`
26. Luštrek, M., et al.: What makes classification trees comprehensible? Expert Systems with Applications **62**, 333–346 (2016)
27. Murphy, K.P.: Machine learning: a probabilistic perspective. MIT Press (2012)
28. Olson, R., et al.: Automating biomedical data science through tree-based pipeline optimization. In: European Conference on the Applications of Evolutionary Computation. pp. 123–137. Springer (2016)
29. Quinlan, J.R.: C4.5: programs for machine learning. Morgan Kaufmann Publishers, San Mateo, CA (1993)
30. Rish, I., et al.: An empirical study of the naive Bayes classifier. In: Proceedings of theWorkshop on empirical methods in artificial intelligence, IJCAI 2001. vol. 3, pp. 41–46. Seattle, USA (2001)
31. Tan, P.N., Steinbach, M., Kumar, V.: Introduction to Data Mining. Pearson (2006)
32. Thornton, C., et al.: Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In: International Conference on Knowledge Discovery and Data Mining. pp. 847–855. ACM (2013)
33. Xavier-Júnior, J.a.C., et al.: A Novel Evolutionary Algorithm for Automated Machine Learning Focusing on Classifier Ensembles. In: Brazilian Conference on Intelligent Systems. pp. 1–6. IEEE, São Paulo, Brazil (2018)
34. Zangari, M., et al.: Not all PBILs are the same: Unveiling the different learning mechanisms of PBIL variants. Applied Soft Computing **53**, 88–96 (2017)