Increasing Boosting Effectiveness with Estimation of Distribution Algorithms

Henry E.L. Cagnini
Escola Politécnica
Pontifícia Universidade Católica
do Rio Grande do Sul
Porto Alegre-RS, Brazil
Email: henry.cagnini@acad.pucrs.br

Márcio Porto Basgalupp
Instituto de Ciência e Tecnologia
Universidade Federal de São Paulo
São José dos Campos-SP, Brazil
Email: basgalupp@unifesp.br

Rodrigo C. Barros
Escola Politécnica
Pontifícia Universidade Católica
do Rio Grande do Sul
Porto Alegre-RS, Brazil
Email: rodrigo.barros@pucrs.br

***Abstract*—Ensemble learning is the machine learning paradigm that aims at integrating several base learners into a single system under the assumption that the collective consensus outperforms a single strong learner, be it regarding effectiveness, efficiency, or any other problem-specific metric. Ensemble learning comprises three main phases: generation, selection, and integration, and there are several possible (deterministic or stochastic) strategies for executing one or more of those phases. In this paper, our focus is on improving the predictive accuracy of the well-known AdaBoost algorithm. By using its former voting weights as starting point in a global search carried by an Estimation of Distribution Algorithm, we are capable of improving AdaBoost up to $\approx 11\%$ regarding predictive accuracy in a thorough experimental analysis with multiple public datasets.**

I. Introduction

Ensemble learning – also known as mixture of experts, consensus aggregation, combination of multiple classifiers, classifier fusion, classifier ensembles, or multiple classifier system [1], [2] – is the paradigm that seeks to integrate several base learners into a single system, which in one or more aspects will perform better than inducing a single strong classifier or regressor. Base learners within ensembles cast votes (for classification) or values (for regression) to predict unknown instances, which will be later integrated into a single prediction.

It has already been demonstrated that integrating a set of surrogate models that approximate a given function is computationally cheaper than to induce a single, stronger-than-all regressor [3]. An ensemble of base classifiers that are marginally better than random guessing, when integrated, can be as accurate as a single strong classifier [4], [5]. The way that an ensemble is induced may also yield base learners that are specialized in certain regions of the input space [4].

Ensembles of base learners have been employed in a great variety of application domains, including intrusion detection [2], [6], wind speed forecasting [7], [8], power grid transformers fault prediction [9], [10], among others.

Ensemble learning comprises three stages, depicted in Figure 1: generation, selection, and integration. While selection is an optional procedure, generation and integration are mandatory. The rationale behind the generation phase is to produce a pool of base learners that have strong individual qualities (e.g., high accuracy when predicting a given class/input region).

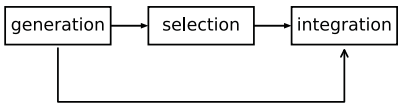

Fig. 1. Ensemble learning induction pipeline.

From this initial pool, a set of base learners that have good cooperation among themselves are selected, so that using several models produce a better outcome than simply using the most accurate base learner. However, several authors argue that performing this selection step is irrelevant for ensemble learning. Lacy et al. [11] affirm that simply selecting the first Φ fittest models from a pool of base learners is more effective than building an ensemble based on diversity measures. They also sustain that their finding is consistent with other studies that simply select the most accurate learners instead of employing diversity measures [12], and that there is little correlation between measures of ensemble diversity and accuracy [13], [14].

Finally, in the integration phase, predictions from each base learner are aggregated in order to provide final consensus for unseen instances. It is possible to use simple solutions such as standard majority voting (each model is equally important when defining the final outcome), or to adjust the voting weights. For instance, if one of the models is exceptionally good at detecting the positive class, then its positive votes should probably have greater importance (larger weights); if it is no better than average in detecting the negative class, then its negative votes should probably have their impact diminished. The ensemble voting scheme for the task of data classification is usually given by:

$$h_B(X^{(i)}) = \underset{c \in C}{\operatorname{argmax}} \left(\sum_{b \in B} w_{b,c} \times [h_b(X^{(i)}) = c] \right) \quad (1)$$

where ${h_B}^{(i)}$ is the ensemble prediction for instance $X^{(i)}$ (i.e., the most voted class by the ensemble); $w_{b,c}$ is the voting

978-1-5090-6017-7/18/$31.00 ©2018 IEEE

weight for classifier b regarding class c; $[h_b(X^{(i)}) = c]$ is the identity function that outputs 1 if classifier b predicts class c for instance $X^{(i)}$ and 0 otherwise.

Since ensemble learning consists of several steps, it is common to have multiple variables involved during its development. There are, for example, different methodologies regarding the composition of ensembles: base learners can be from (1) different paradigms, e.g., mixing models such as decision trees and neural networks; (2) same paradigm, e.g., all models are neural networks; and (3) present differences within the same paradigm, e.g., mixing neural networks with different activation functions and/or architectures.

Among the types of base learners that can be integrated into an ensemble, decision trees are one of the most popular models due to their robustness to noise, speed regarding both training and prediction, and ability to deal with redundant attributes [15]–[18]. There are exponentially many decision trees that can be built from the same dataset, with different levels of predictive quality and compactness. Indeed, inducing decision trees is a combinatorial problem, with complexity NP-hard for generating an optimal decision-tree, and NP-complete for generating a minimal binary decision-tree [17], [18].

There is also no consensus on whether it is better to use greedy, local optimization strategies (e.g., boosting [19], bagging [20], and stacking [21]), or population-based, global optimization approaches (e.g., evolutionary algorithms) to execute those steps. Among the later, Estimation of Distribution Algorithms stand out due to their innate characteristics. The main difference between EDAs (first proposed by Larrañaga and Lozano [22]) and genetic algorithms (GAs) [23], is that GAs perform an implicit propagation of characteristics throughout evolution (i.e., by carrying-on high-quality individuals from one generation to another), whereas EDAs perform it explicitly, by encoding those characteristics in a probabilistic graphical model [24]. This feature is important, since EDAs can start from a previous high-quality point in the fitness landscape. EDAs were successfully employed for decision-tree induction [18], amino-acid alphabet reduction for protein structure prediction [25], data clustering [26]–[28], military antenna design [29], just to name a few.

Seeking to use the best practices from classic ensemble-learning algorithms, while also performing a robust global optimization, in this work we propose to integrate several decision trees for classification from a prior AdaBoost execution into one ensemble. After running AdaBoost for B iterations, we use its base classifiers as the initial population of the proposed, namely EEL – Estimation of Distribution Algorithms for Ensemble Learning. We then perform the population-based global optimization procedure of EDAs to find the best set of voting weights for each base classifier, using former AdaBoost weights as starting point in the fitness landscape. We test the proposed strategy in 15 UCI datasets, improving AdaBoost performance up to $\approx 11\%$.

The rest of this paper is organized as follows. Section II briefly explains AdaBoost and presents the proposed EEL algorithm in details. Section III depicts the experimental setup used in our tests, while Section IV discusses the obtained results. We present related work in Section V, and end the paper with our conclusions and future work directions in Section VI.

II. Proposed Method: EEL

EEL (Estimation of Distribution Algorithms for Ensemble Learning) is composed by two steps: generation of base classifiers, and subsequent integration. It uses AdaBoost, which is a popular boosting algorithm proposed by Freund and Schapire [4], for performing the generation of a pool of decision trees. After this initial step, it further improves its voting weights by making use of an Estimation of Distribution Algorithm. We use all trees from the generation phase, bypassing the optional selection step in ensemble learning.

AdaBoost relies on the technique of iteratively improving the performance of a weak learner [30]. For doing so it uses two types of weights: a set of **instance weights** and a set of **voting weights**. An instance weight denotes the importance of correctly classifying that instance in a given iteration of the algorithm. A voting weight, in turn, represents the strength of the claim of a base classifier in the final ensemble prediction.

AdaBoost starts by receiving a set of predictive attributes $\mathbf{X}$ with Y class labels, $y \in Y = \{-1, +1\}$. In the first iteration, it gives the same importance (i.e., weight) to every instance in the training set, $D_1(i) = 1/N$, $i = 1, \ldots, N$ (where N is the number of instances). It then induces a classifier model on the training data (say, a decision tree). Next it assigns a voting weight to the classifier based on its error rate, $\epsilon_b = \mathrm{P}_{i \sim D_b}[h_b(x_i) \neq y_i]$, and increases the importance of incorrectly predicted instances. The generated decision tree is then stored in the ensemble as the b-th classifier. For the $b+1$-th classifier, it will sample from a new distribution D_{b+1} of training data based on the updated instance weights. This process is repeated for B iterations, resulting in an ensemble of B classifiers.

The voting weight of the b-th classifier is given by:

$$w_b = \frac{1}{2} \ln \left(\frac{1 - \epsilon_b}{\epsilon_b} \right) \tag{2}$$

and then the weight of the i-th instance is adjusted by:

$$D_{b+1}(i) = \frac{D_b(i) \exp(-w_b y_i h_b(X^{(i)}))}{Z_b} \tag{3}$$

where $Z_g = \sum_{i=1}^{N} D_{b+1}(i)$, and therefore can only be obtained after calculating all the N new instance weights. After the B iterations are completed, the prediction of unknown instances is given by:

$$h_b(X) = \mathrm{sign}\left(\sum_{b=1}^{B} w_b h_b(X^{(i)}) \right) \tag{4}$$

As it is expected, classifiers with greater voting weights will have a greater impact on the prediction of unseen instances. However, AdaBoost does not take into account that some classifiers may perform differently among classes (i.e., that

a classifier performs better at detecting the positive class than the negative class, or vice-versa). It is possible to induce a set of voting weights, with one weight *per* classifier *per* class, to overcome that limitation. For that we employ an EDA, since its capability of starting a search procedure from a previous high-quality region in the fitness landscape meets our needs.

We start the EDA optimization step by initializing a probabilistic graphical model (GM). The GM is a matrix of size $C \times B$, where B is the number of decision trees and C the number of classes. Each cell in this matrix is a gaussian with mean $\bar{x}$ and standard deviation σ, as depicted in Figure 2.

	b_1	b_2	b_3	b_4	b_5
c_1	0.96	0.97	1.18	1.07	1.06
c_2	0.96	0.97	1.18	1.07	1.06
c_3	0.96	0.97	1.18	1.07	1.06

Fig. 2. Initial probabilistic graphical model used in the EDA optimization step, with 5 base classifiers and 3 classes. Each cell comprises the mean of a gaussian distribution for that voting weight. Initial values come from former AdaBoost weights. Note that there is no restriction for a single decision tree to sum its votes to 1. Also note that since we use one weight *per* classifier *per* class, we broadcast weights in the first generation of EEL.

EEL makes use of an univariate EDA, which means its GM assumes that there is no correlation between the voting weights of two base classifiers. Due to the nature of the problem, in which we have several classifiers casting votes that are further integrated, we are aware that this is a naïve assumption, but in practice univariate EDAs can provide sufficiently high-quality solutions, while also being computationally efficient [24].

From this initial GM we sample S solutions. A solution is a matrix $B \times C$ representing the whole ensemble voting weights. Note that each classifier within the ensemble never changes its predictions with regard to the training set; the aim of our method is to change the whole ensemble prediction by adjusting the voting weights of each classifier, $w_{b,c} \forall b \in [1, B], c \in [1, C]$.

A. Fitness Computation

We calculate the fitness of individuals (i.e., ensembles with assigned voting weights) as follows. Each individual outputs a $C \times N$ matrix, where N is the number of instances in the training set and C the number of classes. The highest score in each column denotes the prediction for that instance. We sum the scores of the **incorrectly** predicted instances and use that as the individual's fitness. Hence, the objective of the EDA is to decrease the median fitness throughout evolution, thus decreasing the level of certainty in incorrect predictions. Figure 3 depicts the fitness calculation for a single individual.

	$X^{(1)}$	$X^{(2)}$	$X^{(3)}$	$X^{(4)}$
c_1	**25.65**	14.71	22.98	24.61
c_2	2.34	**15.12**	17.95	22.18
c_3	0.55	5.48	13.67	**28.53**
Y	c_2	c_1	c_1	c_1

Fig. 3. Ensemble scores for instances in a given training set, along with the truth labels (Y). Each row denotes a class, and each column an instance. Grey cells indicate the prediction of the ensemble for that instance, with bold values denoting **incorrect** predictions. Fitness for the depicted individual is given by $25.65 + 15.12 + 28.53 = 69.3$.

B. Updating the Probabilistic Graphical Model

After computing the fitness of the entire population, we separate the individuals into two subsets. The first subset comprises all individuals that surpass the median fitness of the current population (elite), whereas the second comprises individuals with fitness below or equal the median. The former will update the probabilistic graphical model (GM), and will be preserved for the following generation, as depicted in Figure 4. The rest of the population will be sampled from the updated GM.

The GM comprises Gaussian distributions that are updated by computing the mean cell value among the elite. The standard deviation is stored into another variable, which is initially set to σ (hyper-parameter), and is iteratively decreased by a factor of τ (another hyper-parameter) at every generation, as recommended by [24].

EEL repeats the process of sampling, calculating fitness, and updating the GM until a sufficient number of generations have been achieved, or the median fitness has not improved over δ for Δ generations. Once the evolutionary procedure halts, we use the fittest individual from the last generation as the ensemble to be used in practice for unseen data.

C. Hyper-parameters and Pseudocode

The proposed algorithm has the following parameters:

- B = the number of decision trees in the ensemble;
- G = maximum number of generations;
- δ = minimum decrease in median fitness allowed;
- Δ = maximum number of generations to be executed when there are decrements smaller than δ in the median fitness;
- $\bar{x}$ = initial mean for Gaussians within GM;
- σ = initial standard deviation for Gaussians within GM;

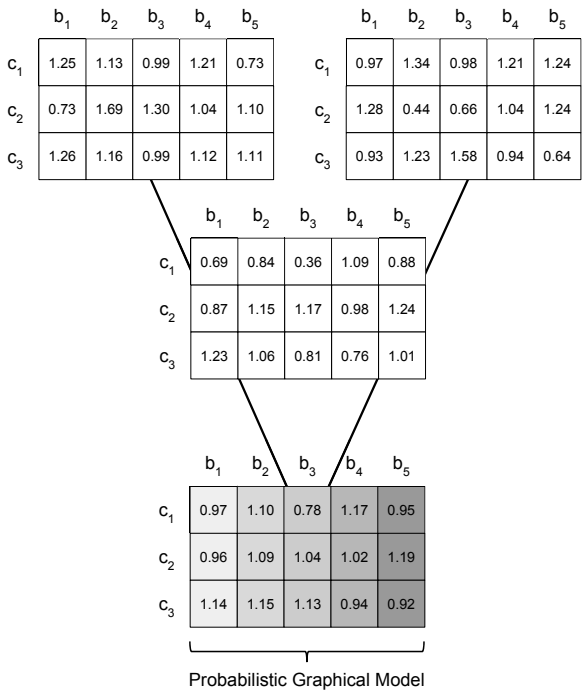

Fig. 4. Each cell of the GM is a Gaussian distribution whose mean is updated according to the mean values of the elite.

- τ = decay rate for σ;
- $\mathbf{X}$ = predictive attributes of the training set;
- Y = class labels of the training set.

The pseudocode for the training and prediction phases are shown in Figures 5 and 6, respectively.

1: **function** TRAIN(B, G, δ, Δ, $\bar{x}$, σ, $\mathbf{X}$, Y)
2: run AdaBoost with $\mathbf{X}$ and Y
3: use AdaBoost base classifiers for initializing the EDA
4: initialize the GM using $\bar{x}$ and σ
5: $g \leftarrow 0$
6: **while** $g < G$ and GM has not yet converged **do**
7: sample population S from GM
8: assess the population fitness, as shown in Figure 3
9: **if** the median fitness improves $< \delta$ for Δ generations **then**
10: Early stop
11: use the individuals with fitness $>$ median to update GM
12: resample individuals replacing those with fitness $\leq$ median
13: $\sigma \leftarrow \sigma - \tau$
14: $g \leftarrow g + 1$
15: **return** best individual from last generation

Fig. 5. EEL pseudocode for the training phase.

D. Complexity Analysis

The complexity of inducing a decision tree is $O(\lambda MN \log N)$, with λ being the number of nodes, M the number of attributes and N the number of instances in the training set. AdaBoost runs this process for B iterations, thus $O(B\lambda MN \log N)$.

Generating the matrix of predictions (of size $B \times N$) has complexity $O(BHN)$, with B being the number of base

1: **function** PREDICT($\mathbf{X}$)
2: $H \leftarrow 0_N$
3: **for** $i \in [1, N]$ **do**
4: $\eta \leftarrow 0_C$
5: **for** $b \in [1, B]$ **do**
6: $\eta_{h_b(\mathbf{X}^{(i)})} \leftarrow \eta_{h_b(\mathbf{X}^{(i)})} + w_{b,c}$
7: $H_i \leftarrow \text{argmax}(\eta)$
8: **return** H

Fig. 6. EEL pseudocode for the prediction phase.

classifiers, H the height of the current decision tree and N the number of training instances. This matrix does not change overtime, and will later be used to perform the predictions.

For each solution in the EDA, sampling new weights from the probabilistic graphical model has complexity of $O(BC)$, with B being the number of base classifiers and C the number of weights *per* classifier. This is done for $S - \Phi$ individuals (i.e., those that have to be replaced for the following generation). The whole ensemble prediction procedure, including the sum of votes of incorrect instances, has complexity of $O(BN)$, since it requires only looking up the prediction table for each classifier and for each training instance (as shown in line 6 of Figure 6). Updating the probabilistic graphical model based on the Φ fittest individuals has complexity of $O(\Phi BC)$.

Since the EDA evolves S solutions and repeats its process for G generations, the complexity of the EDA optimization procedure is $O((BHN)+G\times((S-\Phi)(BC+BN)+\Phi BC))$ (with Φ being zero in the first generation). With the former AdaBoost complexity and after some simplification we have:

$$O((BN(H+\lambda M \log N))+(G((S-\Phi)(B(N+C(1+\Phi))))) \quad (5)$$

E. Source code

We make the source code for our algorithm publicly available at http://github.com/henryzord/eel. The source code is written in Python 2.7.13. A list of third-party packages is listed within the link. Note that our approach makes use of scikit-learn's AdaBoost implementation [31], [32][1] with default parameters, except for the number of decision trees (set to B) and boosting strategy (set to 'SAMME').

III. Experimental Setup

Once all aspects of EEL are covered, it is necessary to assert its quality in providing better solutions than its baseline, AdaBoost. In this section we present the experimentation protocol used in our tests.

A. Baseline Algorithms

We compare our proposed algorithm with the original implementation of AdaBoost, as well as other two possible variations. Comparing to AdaBoost is intuitive, since our algorithm further optimizes its ensemble learning procedure

[1]http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

TABLE I
ACCURACY IN THE TEST SET FOR EACH ONE OF THE ALGORITHMS, IN THE EXPERIMENTED DATASETS. BOLD VALUES DENOTE THE BEST ALGORITHM FOR THAT DATASET, WHEREAS A BOLD AND UNDERLINED RESULT INDICATES A TIE IN FIRST PLACE.

Dataset	AdaBoost	AdaBoost-ones	AdaBoost-normal	EEL (ours)
diabetes	0.755 ± 0.000	0.727 ± 0.000	0.716 ± 0.013	$\mathbf{0.756 \pm 0.004}$
ecoli	0.705 ± 0.000	0.720 ± 0.000	0.661 ± 0.089	$\mathbf{0.818 \pm 0.003}$
glass	0.579 ± 0.000	0.561 ± 0.000	0.487 ± 0.034	$\mathbf{0.598 \pm 0.004}$
hayes roth	0.594 ± 0.000	$\mathbf{0.600 \pm 0.000}$	0.598 ± 0.003	0.598 ± 0.003
ionosphere	0.917 ± 0.000	$\mathbf{0.926 \pm 0.000}$	0.920 ± 0.004	0.917 ± 0.006
iris	0.933 ± 0.000	$\mathbf{0.940 \pm 0.000}$	0.933 ± 0.004	0.933 ± 0.000
KDD synth control	0.745 ± 0.000	0.733 ± 0.000	0.717 ± 0.016	$\mathbf{0.784 \pm 0.003}$
liver disorders	0.725 ± 0.000	0.635 ± 0.000	0.654 ± 0.019	$\mathbf{0.730 \pm 0.006}$
segment	0.818 ± 0.000	0.777 ± 0.000	0.691 ± 0.065	$\mathbf{0.866 \pm 0.000}$
semeion	0.964 ± 0.000	$\mathbf{0.967 \pm 0.000}$	0.954 ± 0.007	0.966 ± 0.001
sonar	$\mathbf{0.815 \pm 0.002}$	0.797 ± 0.007	0.797 ± 0.014	0.809 ± 0.017
vehicle	0.621 ± 0.000	0.603 ± 0.000	0.589 ± 0.012	$\mathbf{0.664 \pm 0.009}$
wine	0.944 ± 0.000	$\mathbf{0.961 \pm 0.000}$	0.950 ± 0.004	0.953 ± 0.006
winequality red	0.548 ± 0.000	0.477 ± 0.000	0.441 ± 0.018	$\mathbf{0.566 \pm 0.003}$
winequality white	0.473 ± 0.000	0.415 ± 0.000	0.408 ± 0.032	$\mathbf{0.494 \pm 0.001}$
Wins	1	5	0	9

by expanding the set of voting weights available for prediction – that is, using multiple weights per classifier, as opposed to AdaBoost single-weight strategy. The other two baseline algorithms are simpler modifications of AdaBoost. In the first version, AdaBoost-ones, we set all voting weights to one (i.e., every classifier has the same importance). This strategy may reduce AdaBoost's tendency to overfit, since it originally gives more importance to classifiers that can properly predict the class of outliers. The second modification, AdaBoost-normal, samples the voting weights from a normal distribution with $\bar{x} = 1$ and $\sigma = 0.25$. This version serves as a "null hypothesis" (or random classifier) regarding the weight optimization strategy: if this variation performs reasonably well in most datasets, then it would indicate that voting weights may have no significant impact in instance prediction and thus be safely discarded. All algorithms are tested with $B = 50$ decision trees in their ensemble. The hyper-parameters for EEL are described in Table II, and no efforts were made to optimize those values.

TABLE II
HYPER-PARAMETERS USED BY EEL.

Parameter	Description	Value
B	Number of decision trees	50
G	Number of generations	50
S	Number of individuals	100
δ	Fitness threshold	0.01
Δ	Generation threshold	5
$\bar{x}$	Mean of Gaussian	AdaBoost weights
σ	Standard deviation of Gaussian	0.25
τ	Standard deviation decrease	0.005

B. Datasets

We evaluate EEL and the baselines in 15 UCI Datasets [33] presented in Table III. All datasets present only numeric attributes and no missing data. The restriction on numeric attributes is due to the implementation of decision trees we are using, which is from Python's scikit-learn package, though it is not a conceptual limitation of our method *per se*. We perform a three-fold cross validation, with $2/3$ being used as training set and $1/3$ as test set, for each test. We execute all algorithms 10 times for evaluating their stability on each fold.

TABLE III
UCI DATASETS USED IN OUR EXPERIMENTS.

Dataset	Instances	Attributes	Min class	Max class	classes
diabetes	768	8	268	500	2
ecoli	336	7	2	143	8
glass	214	9	9	76	6
hayes roth	160	4	31	65	3
ionosphere	351	33	126	225	2
iris	150	4	50	50	3
KDD synth control	600	60	100	100	6
liver disorders	345	6	145	200	2
segment	2310	18	330	330	7
semeion	1593	265	158	1435	2
sonar	208	60	97	111	2
vehicle	846	18	199	218	4
wine	178	13	48	71	3
winequality red	1599	11	10	681	6
winequality white	4898	11	5	2198	7

IV. EXPERIMENTAL RESULTS

We begin the experimental analysis by comparing AdaBoost with its variations, AdaBoost-ones and AdaBoost-normal. AdaBoost-normal outperforms the default AdaBoost in 3 datasets, while presenting the same performance in 1 dataset and being outperformed in the remaining 11, as presented in Table I. As expected, AdaBoost-normal is one of the weakest algorithms, and we are capable of rejecting the hypothesis that random assignments of weights for the base classifiers do not significantly affect the classification outcome.

The second variation, AdaBoost-ones, outperforms the original version in 6 datasets, losing in the remaining 9. These results indicate that there are cases in which AdaBoost indeed suffers from overfitting, which is probably the scenario in which the hard examples are actually noisy data, though in the majority of the cases it is beneficial to learn how to classify hard examples.

Finally, we evaluate how well the proposed algorithm, EEL, performs when compared to the original AdaBoost version. It outperforms AdaBoost in 12 out of 15 datasets, being equally good in other 2, which means there is only one case in which EEL harms the original performance of AdaBoost (*sonar* dataset). EEL is also the algorithm with overall best results, with 9 undisputed wins. Hence, a careful optimization procedure on AdaBoost voting weights is indeed beneficial for the vast majority of the cases. By using its former weights as starting point in our evolutionary EDA search procedure, it is possible to improve predictive performance up to 11%, as it was the case of the *ecoli* dataset.

To assess the statistical significance of our results, we conducted a pairwise Wilcoxon test [34]. The null hypothesis states that the medians of the differences between the pair of algorithms do not differ. We run this test with a significance level of 5%, which is a standard value considering the fact that Wilcoxon's is quite conservative for indicating significant differences. The results are presented in Table IV.

TABLE IV
RESULTS OF THE WILCOXON PAIRWISE TEST REGARDING PREDICTIVE ACCURACY. UNDERLINED VALUES INDICATE A REFUSAL OF THE NULL HYPOTHESIS. CHECKS ✓ INDICATE A SUPERIORITY OF THE ROW ALGORITHM OVER THE COLUMN ONE.

	AdaBoost	AdaBoost-Ones	AdaBoost-Normal	EEL
AdaBoost		✓	✓	
AdaBoost-ones			✓	
AdaBoost-normal				
EEL	✓	✓	✓	

In Figure 7 we depict the mean of the Gaussian distributions throughout EEL's process, in a given run on the *ecoli* dataset. In the first generation (i.e., lowermost line), weights come from a previous AdaBoost execution. Recall that AdaBoost uses one weight *per* classifier (that is, it yields $B = 50$ weights at the end of its execution), whereas EEL uses one weight *per* classifier *per* class (i.e., it must start with $B \times C = 50 \times 7$ weights, with B as the number of base classifiers and C the number of classes). Due to that fact, we broadcast AdaBoost weights into the first Gaussians of the GM (initial population), as explained in Figure 2. With that in mind, note that EEL adapts to the classifiers strengths and weaknesses in certain classes, as demonstrated by the smaller line trends in Figure 7.

By looking at the standard deviation of EEL in Table I, one can verify that the proposed algorithm is very stable. This may indicate that using the scores of incorrect predictions as fitness function provides a smooth fitness landscape, with few local minima. To better visualize that fact, we present the known fitness landscape in Figure 8. The horizontal axis is the mean of the first $W/2$ weights, and the vertical axis the mean of the last $W/2$ weights. By doing this we increase the chances for more than one individual occupying the same spot in the landscape. For this we calculate the mean fitness for all individuals occupying the same spot. Please note that this is an oversimplification of the landscape to fit in a 2D projection. With this in mind, one can verify that it appears to exist a region in the left-down portion of the landscape with better combination of weights. Indeed, we verified in our experiments that EEL starts with a population in the top-right corner that migrates to the bottom-left as the evolution progresses.

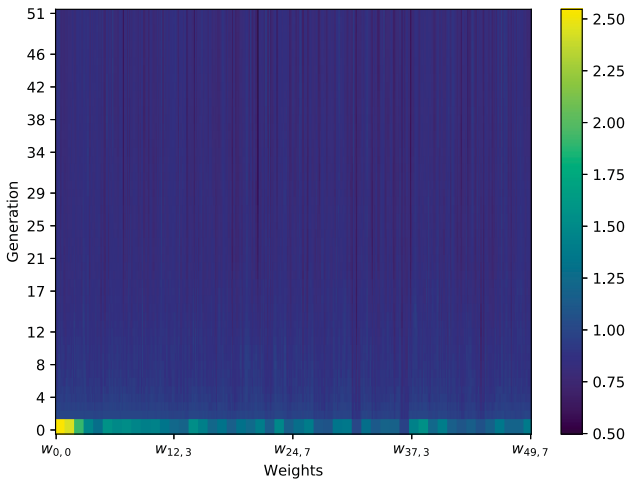

Fig. 7. Probabilistic graphical model evolution on the *ecoli* dataset. Vertical axis denotes the generations whereas the horizontal axis denotes the $50 \times 7 = 350$ weights for that particular dataset. Brighter colors indicate more importance (heavier weights) in a given region. Note that EEL penalizes for incorrect predictions per class, as opposed to AdaBoost that penalizes the entire classifier.

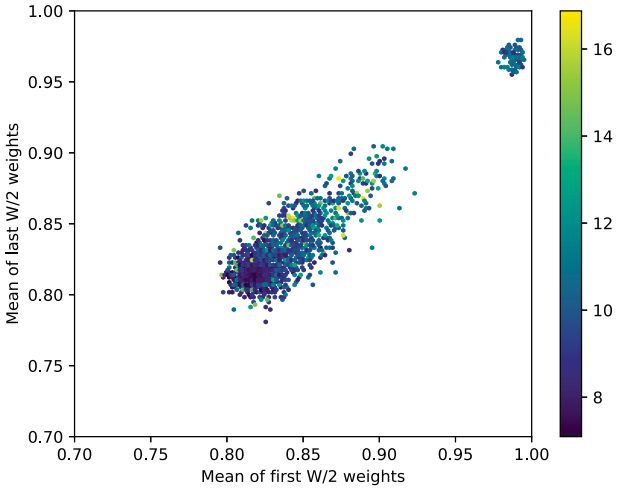

Fig. 8. Fitness landscape of EEL. Darker spots indicate valleys (better regions), whereas brighter spots indicate peaks.

Regarding the training set, the evolutionary algorithm is capable of improving accuracy, precision, recall, and F1 score throughout evolution, while decreasing the score of incorrect predictions, as shown in Figure 9. This once again suggests that the choice for the fitness function is suitable for this task.

TABLE V
SUMMARY OF RELATED WORK.

	PSO + EDA [35]	BAIS [36]	NEVE; NEVE++ [37], [38]	EEL
Application	microarray data	general	data streams	general
Type	classification	classification	classification	classification
Base Learner	Neural Networks	Neural Networks	Neural Networks	Decision Trees
Fitness	error rate (selection)	p-disagreement and accuracy (generation); accuracy (selection)	error rate (generation); error rate (integration)	error rate (integration)

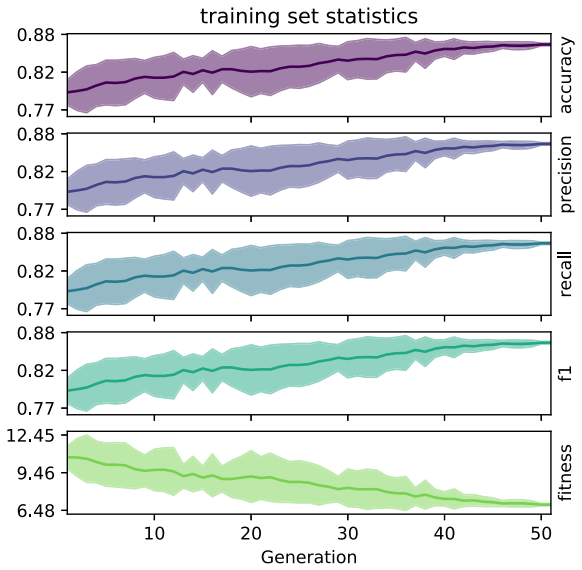

Fig. 9. Training accuracy, precision, recall, and F1 measure throughout the evolutionary process. Bolder lines indicate the mean values and shaded colors denote the standard deviations.

V. Related Work

Estimation of Distribution Algorithms for ensemble learning are not very common. To the best of our knowledge, there are four related studies in that theme. Interestingly, all of these papers employ neural networks as base learners [35]–[38], as opposed to our use of decision trees.

Chen and Zhao [35] propose a hybrid solution for ensemble learning. As mentioned before, ensemble learning comprises three steps (generation, selection and integration), which may be explored in different manners: using traditional methods for one or more steps (such as AdaBoost for generation and integration), or breaking the pipeline and employing evolutionary algorithms in one or more of these steps. Chen and Zhao [35] explore the second strategy by using a Particle Swarm Optimization (PSO) algorithm for generating a pool of neural networks, and then selecting a subset of highly cooperative and accurate classifiers. Their solution is employed for microarray data classification, an application domain where data usually have several attributes but few instances.

Castro and Von Zuben [36] also separete the ensemble learning pipeline, employing the same evolutionary algorithm (namely an Artificial Immune System (AIS) with an embedded EDA) in two steps, i.e., the AIS first generates a pool of base learners (by modifying neural network activation functions) to only then select the most apt to compose the final ensemble. The generation AIS uses two objectives for generating neural networks: (1) pairwise disagreement measure, which measures how similar the predictions of two classifiers are and gives higher scores to different predictions, and (2) training accuracy. The selection AIS uses validation accuracy as fitness.

Finally, Escovedo et al. [37], [38] propose a quantum-inspired evolutionary algorithm (QIEA) for evolving ensembles of neural networks for adaptive learning. The authors employ QIEA for both optimizing neural network weights and biases, and later defining a voting weight for each neural network in the ensemble by using the error rate as fitness.

In Table V we present an overview of the related work. Note that none of the related work attempt to improve a particular ensemble learner such as AdaBoost, which is our goal in this paper.

VI. Conclusion and Future Work

Designing ensembles systems is not a trivial task: there are several ways of either generating, selecting, or integrating base learners. It seems reasonable to employ global optimization strategies to implement one or more of those steps, such as evolutionary algorithms. For instance, Estimation of Distribution Algorithms (EDAs) are notably interesting for this task due to their explicit search operators, as opposed to genetic algorithms. This not only makes the evolutionary process transparent, but also allows EDAs start its search procedure from a previous high-quality spot in the fitness landscape.

With that in mind, in this work we propose EEL: Estimation of Distribution Algorithms for Ensemble Learning. EEL resumes the optimization pipeline started by AdaBoost. It uses AdaBoost previous base classifiers and finds a new set of weights for those classifiers by conducting a population-based, global search procedure with the aim of minimizing the error rate of ensemble predictions.

By using multiple weights, one weight *per* classifier *per* class, as opposed to AdaBoost, which uses a single weight *per* classifier, we are capable of enhancing AdaBoost in 12 cases, and producing the best results in 14 out of 15 datasets.

As future work, we would like to investigate whether a Pareto approach for decreasing error rate and maximizing

accuracy can yield even better results than simply using a single-objective EDA. We also intend to consider a multivariate probabilistic graphical model, which may improve the quality of the produced ensembles. In addition, we would like to investigate whether Random Forests benefits from a voting weight optimization procedure, since it assigns the same importance to all of its base classifiers.

ACKNOWLEDGMENT

The authors would like to thank CAPES, CNPq, FAPERGS, and FAPESP for funding this research.

REFERENCES

[1] R. Saleh, H. Farsi, and S. H. Zahiri, "Ensemble classification of polsar data using multi-objective heuristic combination rule," in *Swarm Intelligence and Evolutionary Computation (CSIEC), 2016 1st Conference on*. IEEE, 2016, pp. 88–92.

[2] M. Milliken, Y. Bi, L. Galway, and G. Hawe, "Multi-objective optimization of base classifiers in stackingc by nsga-ii for intrusion detection," in *Computational Intelligence (SSCI), 2016 IEEE Symposium Series on*. IEEE, 2016, pp. 1–8.

[3] M. Krithikaa and R. Mallipeddi, "Differential evolution with an ensemble of low-quality surrogates for expensive optimization problems," in *Evolutionary Computation (CEC), 2016 IEEE Congress on*. IEEE, 2016, pp. 78–85.

[4] Y. Freund and R. E. Schapire, "A desicion-theoretic generalization of on-line learning and an application to boosting," in *European conference on computational learning theory*. Springer, 1995, pp. 23–37.

[5] K. Jackowski, B. Krawczyk, and M. Woźniak, "Improved adaptive splitting and selection: the hybrid training method of a classifier based on a feature space partitioning," *International journal of neural systems*, vol. 24, no. 03, p. 1430007, 2014.

[6] G. Folino, F. S. Pisani, and P. Sabatino, "An incremental ensemble evolved by using genetic programming to efficiently detect drifts in cyber security datasets," in *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*. ACM, 2016, pp. 1103–1110.

[7] W. Zhang, Z. Qu, K. Zhang, W. Mao, Y. Ma, and X. Fan, "A combined model based on ceemdan and modified flower pollination algorithm for wind speed forecasting," *Energy Conversion and Management*, vol. 136, pp. 439–451, 2017.

[8] W. L. Woon and O. Kramer, "Enhanced svr ensembles for wind power prediction," in *Neural Networks (IJCNN), 2016 International Joint Conference on*. IEEE, 2016, pp. 2743–2748.

[9] A. Peimankar, S. J. Weddell, T. Jalal, and A. C. Lapthorn, "Evolutionary multi-objective fault diagnosis of power transformers," *Swarm and Evolutionary Computation*, 2017.

[10] ——, "Ensemble classifier selection using multi-objective PSO for fault diagnosis of power transformers," in *Evolutionary Computation (CEC), 2016 IEEE Congress on*. IEEE, 2016, pp. 3622–3629.

[11] S. E. Lacy, M. A. Lones, and S. L. Smith, "Forming classifier ensembles with multimodal evolutionary algorithms," in *Evolutionary Computation (CEC), 2015 IEEE Congress on*. IEEE, 2015, pp. 723–729.

[12] Y. Freund, R. E. Schapire *et al.*, "Experiments with a new boosting algorithm," in *Icml*, vol. 96. Bari, Italy, 1996, pp. 148–156.

[13] D. W. Opitz, "Feature selection for ensembles," *AAAI/IAAI*, vol. 379, p. 384, 1999.

[14] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[15] R. C. Barros, M. P. Basgalupp, A. de Carvalho, and A. A. Freitas, "A survey of evolutionary algorithms for decision-tree induction," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 42, no. 3, pp. 291–312, 2012.

[16] R. C. Barros, "On the automatic design of decision-tree induction algorithms," Ph.D. dissertation, Universidade de São Paulo, 2013.

[17] R. C. Barros, A. C. de Carvalho, and A. A. Freitas, *Automatic Design of Decision-Tree Induction Algorithms*, ser. SpringerBriefs in Computer Science. Springer, 2015, vol. 1.

[18] H. E. Cagnini, R. C. Barros, and M. P. Basgalupp, "Estimation of distribution algorithms for decision-tree induction," in *Evolutionary Computation (CEC), 2017 IEEE Congress on*. IEEE, 2017, pp. 2022–2029.

[19] R. E. Schapire, "A brief introduction to boosting," in *Ijcai*, vol. 99, 1999, pp. 1401–1406.

[20] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

[21] D. H. Wolpert, "Stacked generalization," *Neural networks*, vol. 5, no. 2, pp. 241–259, 1992.

[22] P. Larrañaga and J. A. Lozano, *Estimation of distribution algorithms: A new tool for evolutionary computation*. Springer Science & Business Media, 2001, vol. 2.

[23] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[24] M. Hauschild and M. Pelikan, "An introduction and survey of estimation of distribution algorithms," *Swarm and Evolutionary Computation*, vol. 1, no. 3, pp. 111–128, 2011.

[25] J. Bacardit, M. Stout, J. D. Hirst, K. Sastry, X. Llorà, and N. Krasnogor, "Automated alphabet reduction method with evolutionary algorithms for protein structure prediction," in *Proceedings of the 9th annual conference on Genetic and evolutionary computation*. ACM, 2007, pp. 346–353.

[26] H. E. Cagnini, R. C. Barros, C. V. Quevedo, and M. P. Basgalupp, "Medoid-based data clustering with estimation of distribution algorithms," in *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. ACM, 2016, pp. 112–115.

[27] H. E. Cagnini and R. C. Barros, "Pascal: An eda for parameterless shape-independent clustering," in *Evolutionary Computation (CEC), 2016 IEEE Congress on*. IEEE, 2016, pp. 3433–3440.

[28] R. Santana, C. Bielza, and P. Larrañaga, "Affinity propagation enhanced by estimation of distribution algorithms," in *Annual conference on Genetic and evolutionary computation*, 2011, pp. 331–338.

[29] M. Pelikan and A. Hartmann, "Searching for ground states of ising spin glasses with hierarchical boa and cluster exact approximation," *Scalable Optimization via Probabilistic Modeling*, pp. 333–349, 2006.

[30] B. Krawczyk, M. Galar, Ł. Jeleń, and F. Herrera, "Evolutionary undersampling boosting for imbalanced classification of breast cancer malignancy," *Applied Soft Computing*, vol. 38, pp. 714–726, 2016.

[31] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[32] T. Hastie, S. Rosset, J. Zhu, and H. Zou, "Multi-class adaboost," *Statistics and its Interface*, vol. 2, no. 3, pp. 349–360, 2009.

[33] M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[34] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics bulletin*, vol. 1, no. 6, pp. 80–83, 1945.

[35] Y. Chen and Y. Zhao, "A novel ensemble of classifiers for microarray data classification," *Applied Soft Computing*, vol. 8, no. 4, pp. 1664–1669, 2008.

[36] P. A. D. Castro and F. J. Von Zuben, "Learning ensembles of neural networks by means of a bayesian artificial immune system," *IEEE Transactions on Neural Networks*, vol. 22, no. 2, pp. 304–316, 2011.

[37] T. Escovedo, A. V. A. da Cruz, M. Vellasco, and A. S. Koshiyama, "Neve: A neuro-evolutionary ensemble for adaptive learning," in *IFIP International Conference on Artificial Intelligence Applications and Innovations*. Springer, 2013, pp. 636–645.

[38] T. Escovedo, A. A. da Cruz, A. Koshiyama, R. Melo, and M. Vellasco, "Neve++: A neuro-evolutionary unlimited ensemble for adaptive learning," in *Neural Networks (IJCNN), 2014 International Joint Conference on*. IEEE, 2014, pp. 3331–3338.