Theoretical Computer Science 805 (2020) 1–18

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

An evolutionary algorithm for automated machine learning focusing on classifier ensembles: An improved algorithm and extended results

João C. Xavier-Júnior[a,*], Alex A. Freitas[d], Teresa B. Ludermir[c], Antonino Feitosa-Neto[b], Cephas A.S. Barreto[b]

[a] *Digital Metropolis Institute - Federal University of Rio Grande do Norte, Natal, Brazil*
[b] *Department of Informatics and Applied Mathematics - Federal University of Rio Grande do Norte, Natal, Brazil*
[c] *Center for Information Technology - Federal University of Pernambuco, Recife, Brazil*
[d] *School of Computing - University of Kent, Canterbury, United Kingdom*

A R T I C L E I N F O

Article history:
Received 6 February 2019
Received in revised form 28 October 2019
Accepted 3 December 2019
Available online 5 December 2019

Keywords:
Automated Machine Learning (Auto-ML)
Classification
Evolutionary algorithms
Estimation of distribution algorithms

A B S T R A C T

A large number of classification algorithms have been proposed in the machine learning literature. These algorithms have different pros and cons, and no algorithm is the best for all datasets. Hence, a challenging problem consists of choosing the best classification algorithm with its best hyper-parameter settings for a given input dataset. In the last few years, Automated Machine Learning (Auto-ML) has emerged as a promising approach for tackling this problem, by doing a heuristic search in a large space of candidate classification algorithms and their hyper-parameter settings. In this work we propose an improved version of our previous Evolutionary Algorithm (EA) – more precisely, an Estimation of Distribution Algorithm – for the Auto-ML task of automatically selecting the best classifier ensemble and its best hyper-parameter settings for an input dataset. The new version of this EA was compared against its previous version, as well as against a random forest algorithm (a strong ensemble algorithm) and a version of the well-known Auto-ML method Auto-WEKA adapted to search in the same space of classifier ensembles as the proposed EA. In general, in experiments with 21 datasets, the new EA version obtained the best results among all methods in terms of four popular predictive accuracy measures: error rate, precision, recall and F-measure.

1. Introduction

Classification is a very popular Machine Learning task where each instance (object being classified) consists of a set of predictive features and a nominal (or discrete) class variable. In essence, the goal of a classification algorithm is to learn a classification model that can be used to predict the class value (label) of a new instance, based on the values of the features of that instance. Several decades of classification research have produced a large number of classification algorithms [20]. In practice, none of these algorithms is the best for all possible datasets, since the predictive performance of an algorithm is

* Corresponding author.
E-mail addresses: jcxavier@imd.ufrn.br (J.C. Xavier-Júnior), a.a.freitas@kent.ac.uk (A.A. Freitas), tbl@cin.ufpe.br (T.B. Ludermir), antonino_feitosa@yahoo.com (A. Feitosa-Neto), cephasax@gmail.com (C.A.S. Barreto).

https://doi.org/10.1016/j.tcs.2019.12.002
0304-3975/© 2019 Elsevier B.V. All rights reserved.

strongly dependent on characteristics of the input dataset [10], as well as on the hyper-parameter settings of the algorithm. This creates the very difficult problem of how to choose the best classification algorithm for the dataset at hand.

A promising and relatively recent approach for tackling this problem is the Automated Machine Learning (Auto-ML) approach [4], [6], [24]. This approach is promising because it uses a search and optimization method to automatically perform a search in a very large space of candidate algorithms and hyper-parameter settings, in order to find the best combination of a classification algorithm and its hyper-parameter settings for the problem at hand. Hence, this approach relieves the user from the task of performing ad-hoc, tedious and very time-consuming experiments with different algorithms and their hyper-parameter settings.

This work follows the Auto-ML approach, focusing on a broad type of classification algorithms called classifier ensembles. An ensemble combines the outputs of many base classifiers (e.g. by majority voting), which tends to improves predictive accuracy by comparison with the use of a single base classifier [1]. Ensembles are usually considered one of the state-of-the-art types of classification algorithms in terms of predictive accuracy. For instance, a relatively recent study [9] compared the predictive accuracy of 179 classification algorithms across 121 datasets, and concluded that overall the best algorithms were versions of random forests, which are ensembles of decision-tree classifiers. Even focusing on ensembles, however, there are still many different types of ensembles, and their predictive accuracies also depend on both the characteristics of the input dataset and their hyper-parameter settings.

In this context, the main contribution of this work is to propose an improved version of our previous Evolutionary Algorithm (EA) [28] for the difficult Auto-ML problem of automatically selecting the best ensemble method and its best hyper-parameter settings for an input dataset. EAs have the advantages of performing a global search in the space of candidate solutions (less likely to get trapped into a local optimum than a greedy search) and being robust to noise [12,13].

More precisely, the type of EA proposed in this work (as well as its previous version) is an Estimation of Distribution Algorithm (EDA) [7]. Unlike most EAs, where new candidate solutions are produced by genetic operators like crossover and mutation, EDAs evolve a probabilistic model of the best solutions and their components, and at each generation (iteration) they use the current probabilistic model to generate new candidate solutions, as discussed in Section 2.3. Hence, EDAs combine methods and concepts from both EAs and probability theory, which arguably gives them a sounder mathematical basis than conventional EAs. In addition, EDAs avoid the need to decide which genetic operators to use, and so avoid the need for time-consuming experiments for optimizing the parameters of genetic operators (like the crossover probability and mutation probability).

The proposed improved version of our EDA was compared against its previous version [28] and against two other methods, namely a random forest (a well-known type of ensemble) method and the well-known Auto-ML method Auto-WEKA [4], which was adapted to search in the same space of classifier ensembles and their configurations (hyper-parameter settings) as the two EDA versions.

This current paper extends the experimental results reported in [28] in three ways. First, the number of datasets used in the experiments was increased from 15 to 21. Second, whilst in our previous work we reported results for a single predictive accuracy measure, the classification error rate, in the current work we report both the error rate and the values of the precision, recall and F-measure. Third, this current work extends the experiments to include the aforementioned random forest algorithm.

The results of these new experiments across 21 datasets showed that, overall, the proposed new version of the EDA obtained better predictive performance than its previous version and the other two methods (Auto-WEKA and random forest) for all the four predictive accuracy measures used in our experiments, namely error rate, precision, recall and F-measure. In addition, the new proposed EDA version obtained statistically significantly better results than Auto-WEKA and random forest in most cases. More precisely, the new EDA was significantly better than Auto-WEKA for three out of the four measures (viz., error rate, recall and F-measure); and the new EDA was also significantly better than random forest for three measures: precision, recall and F-measure.

The remainder of this paper is organized as follows. Section 2 discusses background on classifier ensembles, Automated Machine Learning (Auto-ML) and EDAs. Section 3 discusses related work on the automated selection of ensemble methods and on the use of EAs for Auto-ML. Section 4 presents the proposed new version of our EDA, and describes how it differs from its previous version. Section 5 describes the experimental methodology, and Section 6 presents the computational results. Finally, Section 7 presents our conclusions and a direction for future work.

2. Background

2.1. Classification and classifier ensembles

In the classification task of machine learning, each instance (object) in the input dataset is represented by a set of features (characteristics) and a class attribute. A classification algorithm has access to the class values of instances in the training set, but not in the test set. Hence, the goal is to learn a model from the training set that is able to predict the class value of each instance in the test set (with instances unseen during training), based on the feature values for that instance.

Classifier ensembles learn a classification model consisting of a set of base classifiers. Such ensembles have a two-layer structure. In the first layer, each of the base classifiers receives input data and predicts a class for a new instance. These predictions are sent to a combination module in the second layer, which combines all received predictions into a single predicted class for each instance (e.g. via majority voting). Combining the results of different base classifiers often outperforms a single base classifier [1], [2], since an ensemble's predictions are usually more robust than the predictions performed by a single classifier.

The two main aspects of the design of classifier ensembles are the selection of the type(s) of classifiers to be used as base classifiers and the combination method. Regarding the choice of classifier types, classifier ensembles can be categorized into homogeneous ensembles, which use multiple base classifiers of the same type, or heterogeneous ensembles, which use different types of base classifiers. Regarding the type of method used to combine the predictions of the base classifiers, several methods have been proposed [3], such as: simple majority voting, weighted voting (where the vote for a class is weighted by its estimated probability), using a full classification algorithm – treating the classes predicted by the base classifiers as features for predicting the class at the meta-level, etc.

2.2. Automated Machine Learning (Auto-ML)

Many types of classification algorithms have been proposed and are often used, such as Decision Trees, Neural Networks, Support Vector Machines, among many others [9]. However, in general different types of classification algorithm have different pros and cons, and different biases; therefore no single type of classification algorithm can be considered the best for all datasets or application domains. In practice the predictive accuracy of a classification algorithm strongly depends on two major factors: (a) the characteristics of the input dataset [10]; and (b) the algorithm's hyper-parameter settings. This leads to the challenging optimization problem of how to select the best classification algorithm and its corresponding best hyper-parameter settings for each input dataset provided by a user. An emergent approach to solve this problem involves Automated Machine Learning (Auto-ML) methods, which automatically search for the combination of classification algorithm and hyper-parameter settings that maximizes predictive accuracy in an input dataset.

Recent research on Auto-ML has provided some off-the-shelf Auto-ML tools for machine learning researchers and practitioners; such as Auto-sklearn [6] and Auto-WEKA [4]. Here we briefly review only Auto-WEKA, which is used as a strong baseline method in our experiments reported later.

Auto-WEKA, which can be easily installed within the well-known WEKA tool [5], is a method for automatically selecting the best combination of a machine learning algorithm and its hyper-parameter settings, as proposed in [4]. Auto-WEKA uses a Bayesian optimization search method called SMAC (Sequential Model-based Algorithm Configuration) to automatically search through the joint space of WEKA's learning algorithms and their respective hyper-parameter settings, with the goal of maximizing predictive accuracy. Auto-WEKA has been shown to perform well for a wide variety of data sets [11].

2.3. Estimation of distribution algorithms

In this subsection we assume the reader is broadly familiar with Evolutionary Algorithms (EAs), and focus on discussing the specific type of EA used as the basis of the Auto-ML method for ensembles proposed in this work, namely Estimation of Distribution Algorithms (EDAs), as well as discussing the main differences between EDAs and more conventional EAs. For a general discussion of EAs, the reader is referred to [12,13].

EDAs are a type of EA which explore the space of potential solutions by building and sampling explicit probabilistic models of promising candidate solutions [7]. EDAs have been applied to several types of machine learning tasks, including classification [17] and feature selection [14], [15], [18].

EDAs iteratively generate and evaluate a population of candidate solutions (individuals) to a problem. The initial population is generated at random, using a uniform distribution over all possible candidate solutions. Then, each generated individual has its quality evaluated by a fitness function. Next, individuals are ranked based on their fitness values, and a subset of the best individuals (usually the 50% best) are selected. Then, a probabilistic model is constructed aiming to estimate the probability distribution of the selected individuals (candidate solutions). Once the model is constructed, new individuals are generated by sampling the distribution encoded by this model. The fitness of each new individual is evaluated, and so on. This process is repeated until some termination criterion is met, as usual in EAs in general.

The crucial difference between EDAs and other EAs is how they generate new individuals at each generation (iteration), as follows. In EDAs the selected individuals are used to update a probabilistic model, from which new individuals will be probabilistically sampled in the next generation. By contrast, in other EAs the next generation's individuals are generated by applying solution-alteration operators like crossover and mutation to the selected individuals of the current generation. EDAs explicitly maintain and evolve a probabilistic model of the best solutions evaluated so far, unlike other EAs. Hence, an advantage of EDAs, by comparison with more conventional EAs, is that EDAs directly use sound concepts of probability theory to guide the evolutionary process. Another advantage of EDAs is that they require fewer parameters than most EAs. In particular, most EAs require the user to choose which type of crossover and mutation operators should be used to create new solutions, as well as choosing the corresponding crossover and mutation probability rates. EDAs relieve the user from such concerns, since they do not use any operator to generate new solutions, and simply sample new individuals from the currently available probabilistic model, which is gradually evolved along the search.

The Population-Based Incremental Learning (PBIL) algorithm, proposed in [8] and recently reviewed in [16], is an EDA that evolves a probability vector, where each vector component represents the probability of that component being selected for inclusion in a candidate solution. The vector components' values are usually initialized with a probability of 0.5. Then, at each generation, a population of individuals (candidate solutions) are sampled from the probability vector based on its probability values, and each individual is evaluated using a fitness function, which measures the predictive accuracy of each individual. A predefined number of the best individuals (based on fitness) in the current generation are selected, and the relative frequencies of solution components in those selected individuals are used to update the probability vector, by increasing the probability values for the solution components that occurred most often in the selected individuals. The amount of increase is controlled by a learning rate parameter. More precisely, the probability of each i-th component of the probability vector at the current g-th generation (iteration), denoted by $p[i]$, is updated with the equation $p[i] = (1 - LR) \times p[i]_{g-1} + LR \times RF[i]_g$, where LR is the learning rate and RF is the relative frequency of the i-th component among all individuals selected in the current g-th generation.Hence, the probability vector gradually evolves towards components with probability values closer to 1 or 0, depending on whether or not, respectively, the component has been used in the best candidate solutions evaluated along the generations.

The PBIL algorithm has only 3 parameters, namely: (a) the population size, i.e., the number of individuals sampled from the probability vector at each generation; (b) the Learning Rate, which specifies how large the steps towards good solutions are; (c) the number of best individuals selected for updating the probability vector at each generation.

3. Related work

This section reviews related work on the automated selection of classifier ensembles (Subsection 3.1) and Evolutionary Algorithms (EAs) used for Auto-ML purposes (Subsection 3.2). Note that both these subsections focus on Auto-ML methods. That is, we do not review here conventional methods for learning classifier ensembles without using Auto-ML concepts and methods, since such conventional ensemble learning methods are already extensively discussed in the literature – see e.g. some relevant reviews in [29], [30], [31].

3.1. Automated selection of ensembles methods

Current Auto-ML methods typically use a search space with many types of classification algorithms, without focusing on ensembles as in this work. However, some studies have proposed different approaches for automating the creation of classifier ensembles (base classifiers and their hyper-parameter settings) [21], [22], [23]. In particular, Wistuba et al. [21] proposed an automatic approach to generate ensembles with several layers, called Automatic Frankensteining, where a Bayesian Optimization method is used to select base classifiers and their settings using a bagging strategy.

In fact, most Auto-ML methods are based on Bayesian optimization. For instance, Lacoste et al. [23] proposes an extension of SMBO (Sequential Model-Based Optimization) to optimize the selection of ensemble members based on their performance on randomly selected subsets of the validation data produced by a bootstrap method. In addition, Levesque et al. [22] propose an approach to build fixed-size ensembles, optimizing the configuration of one base classifier of the ensemble at each iteration of the hyper-parameter optimization algorithm, considering the interaction with other models when evaluating performance. In this way, the Bayesian optimization method estimates which prediction model is the best candidate to be added to the ensemble.

It is important to emphasize that all three aforementioned methods for automating the selection and configuration of classifier ensembles use the Bayesian optimization method, whereas our proposed method is based on an Estimation of Distribution Algorithm (EDA) – a type of Evolutionary Algorithm (EA). Unlike the sequential nature of the search performed by the Bayesian optimization method, EAs evolve a population of candidate classifier ensembles, performing a more global, broader search (conceptually a parallel search) in the space of candidate solutions.

3.2. The use of evolutionary algorithms for Auto-ML

Several Evolutionary Algorithms (EAs) have been proposed for Auto-ML, such as [24], [26], [25], [19]. For example, in [25] the authors proposed the use of a genetic algorithm for searching a very large search space of many different multi-label classification algorithms and their hyper-parameters; whilst in [24] and [26] the authors proposed a genetic programming method for automating the selection and configuration of both classification algorithms and data pre-processing methods (classification pipelines).

On the other hand, in a very recent work [19], the authors proposed the use of a genetic programming (GP) method to search the space of possible architectures of hierarchical ensembles and to optimize their hyper-parameters. Broadly speaking, the GP method proposed in [19] addresses the same type of problem addressed in our work (the automatic creation of ensembles), but using GP, a type of EA that is very different from the EDA proposed in this work – for a brief review of the differences between EDAs and other types of EAs, see Section 2.3. In addition, the work in [19] focuses more on transfer learning and meta-learning, which is not the focus of this current work.

Hence, to the best of our knowledge, our recent work in [28] was the first work to propose an EDA for the Auto-ML task of optimizing the selection and configuration of classifier ensembles. As mentioned in the Introduction, this current work extends our previous work by proposing a new version of that EDA (described in detail in the next section), as well as performing extended computational experiments with more datasets and more classification methods.

4. The proposed PBIL-Auto-Ens method for Auto-ML focusing on ensembles

As mentioned earlier, the main contribution of this work is to propose an improved version of our previous Estimation of Distribution Algorithm (EDA) for the Auto-ML problem of automatically selecting the best ensemble method and its best configuration (hyper-parameter settings) for an input dataset. Both the proposed new version and the previous version of our EDA are based on the general PBIL algorithm described in Section 2.3.

The first version of our PBIL for the aforementioned Auto-ML problem was proposed in [28], where it was called PBIL-Auto-Ens (PBIL for Auto-ML focusing on Ensembles). That version is hereafter referred to as PBIL-Auto-Ens-v1 (Version 1), since in this current paper we propose the second, improved version of this method, hereafter referred to as PBIL-Auto-Ens-v2 (Version 2). Next, we first focus on describing in detail PBIL-Auto-Ens-v2, and briefly discuss later the main differences between PBIL-Auto-Ens-v2 and PBIL-Auto-Ens-v1.

The proposed PBIL-Auto-Ens-v2 extends the original PBIL algorithm [8] and its more recent variants [16] in two major ways. First, while a standard PBIL typically has a single probability vector, PBIL-Auto-Ens-v2 has many probability vectors (PVs for short), which are organized into a hierarchical structure. Second, the creation of individuals by sampling solution components from the probability vectors is adapted to follow the hierarchical structure of the set of probability vectors, as described later.

As an overview of the proposed PBIL-Auto-Ens-v2, it consists of the following main steps. First, we initialize a number of probability vectors, whose components contain the probabilities of different components of a candidate solution. In this work, these are essentially the probabilities of selecting each type of ensemble method, the probabilities of selecting each parameter setting for each ensemble method, the probabilities of selecting each base classification algorithm within each ensemble method, and the probabilities of selecting each parameter setting for each base classification algorithm. Second, PBIL-Auto-Ens-v2 creates a population of individuals (candidate solutions) by sampling solution components from those probability vectors, so that each individual consists of a complete specification of an ensemble method, i.e., with all its parameter settings, its base classification algorithm, and the latter's parameter settings. Third, each individual is evaluated according to a fitness function, which measures the predictive accuracy of the ensemble represented by that individual. Fourth, the best (highest fitness) individuals of the current generation are selected, and their solution components are used to update the probability vector. The basic idea is that, if a solution component has been used very often in the selected individuals, the probability of that solution component will be increased in the corresponding probability vector; therefore, that solution component will be more likely to be sampled for creating new individuals in future generations, leading to an improvement of the candidate solutions over time. This iterative process is repeated until a stopping criterion (like a runtime limit) is satisfied. The PBIL-Auto-Ens-v2 method is described in detail in the following sections.

4.1. The hierarchical structure of the set of probability vectors

Let us first describe in detail the hierarchical structure of the set of probability vectors used by PBIL-Auto-Ens-v2, shown in Fig. 1. At the first level, there is a PV for selecting the type of ensemble used. This PV has 7 components, representing the probabilities of selecting each of the following ensemble types: Random Committee (RC), AdaBoost (AD), Bagging (BA), Random Forest (RF), Stacking (ST), Vote (VT), and No Ensemble (NE). The latter gives PBIL-Auto-Ens-v2 the option of producing a candidate solution (individual) using only a single base classifier, without any ensemble method.

The second level is the ensembles' hyper-parameter optimization level. At this level, there is in general one PV for each hyperparameter of each of the ensemble methods at level 1. The exceptions are the nodes at level 2 indicating the selection of a base classifier among the corresponding child nodes, where that selection is implemented by PVs at level 3, as discussed below. For instance, for the No Ensemble option at level 1, its child node at level 2 is a node called Base Classifier, which is not associated with any PV by itself at level 2, and is placed in the hierarchy only as a bridge between the No Ensemble node at level 1 and the list of classifiers candidate for selection at level 3. Analogously, the node W at level 2 (a child of the RC node) indicates a selection between base classifiers in level 3, and this selection is implemented by a PV at level 3.

Note that in Fig. 1 the "..." between the J48 and MLP nodes is a short-hand notation to simplify the figure, referring to all other base classifiers. I.e., the set of base classifiers available for RC (as well as for all other ensemble types) is the same set of 9 base classifiers shown at the right-hand side of level 3 in that figure. Note also that, in order to further simplify Fig. 1, this figure shows only the hyper-parameters for the RC ensemble method, but this level 2 also includes hyper-parameters for all other ensemble methods at level 1.

Table 1 shows the full set of PVs at level 2. The number of PVs for each ensemble type (shown in the second column) is also the number of hyper-parameters for that ensemble type – not counting the hyper-parameter W specifying the base classifier, which is associated with a PV at the level 3, as explained earlier. In the next three columns, the row for each ensemble type is divided into several sub-rows – one row for each PV, i.e., one row for each of the hyper-parameters being optimized. The third column shows the PV name, a string of the form L2-XX-Y, where L2 indicates that the current PV is at level 2, XX is a two-character variable whose value denotes the type of ensemble method (e.g. RC for Random Committee) and Y is a single character variable whose value identifies the hyper-parameter being optimized (e.g. I for number of iterations). The fourth column shows the number of components of each PV, which is the number of candidate discrete values for the corresponding hyper-parameter. The last column shows the corresponding candidate discrete values for that hyper-parameter.

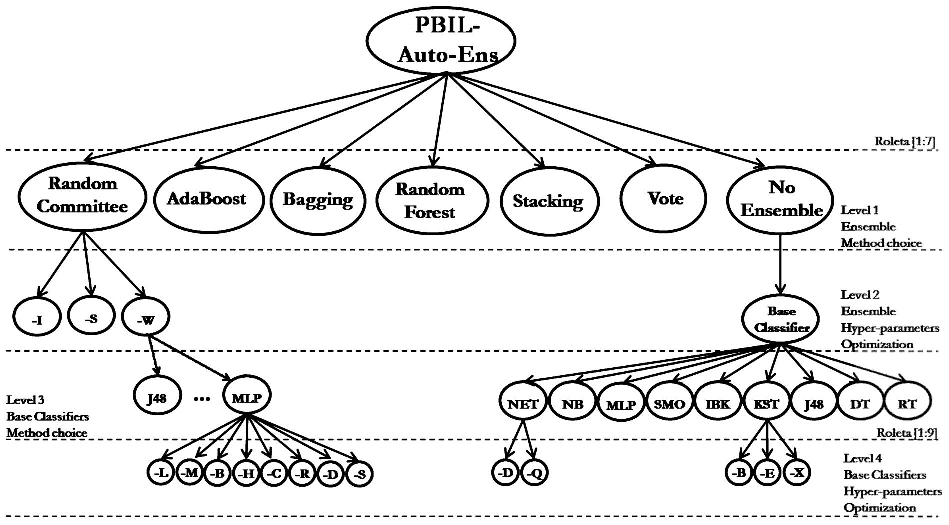

Fig. 1. The general structure of PBIL-Auto-Ens-v2's search space.

Table 1
Probability Vectors (PVs) for ensemble methods' hyper-parameter optimization at level 2 of Fig. 1. The columns of the table refer to the type of the ensemble at level 1, the number of PVs for the ensemble, the name of the PV, the number of components in the PV, and the values of the hyper-parameters, respectively.

Ens. type L1	# PVs Ens.	PV name	# Comp. in PV	Hyper-parameter values
RC	2	L2-RC-I	63	from 2 to 64
		L2-RC-S	255	from 1 to 255
AD	4	L2-AD-Q	2	true / false
		L2-AD-P	51	from 50 to 100
		L2-AD-I	127	from 2 to 128
		L2-AD-S	255	from 1 to 255
BA	4	L2-BA-P	91	from 10 to 100
		L2-BA-I	127	from 2 to 128
		L2-BA-S	255	from 1 to 255
		L2-BA-O	2	true / false
RF	3	L2-RF-I	255	from 2 to 256
		L2-RF-K	32	from 1 to 32
		L2-RF-W	20	from 1 to 20
ST	4	L2-ST-X	10	from 1 to 10
		L2-ST-S	255	from 1 to 255
		L2-ST-B	9	from 1 to 9
		L2-ST-NBC	10	from 1 to 10
VT	4	L2-VT-S	255	from 1 to 255
		L2-VT-R	6	AVG, PROD, MAJ, MIN, MAX, MED
		L2-VT-B	9	from 1 to 9
		L2-VT-NBC	10	from 1 to 10

At the third level, there is a PV for selecting the type of base classifier used. This leads to 6 PVs at this level, as shown in Table 2, where each row describes the characteristics of a PV. In the second column, the PV name is a string of the form L3-XX-BC-sel, where L3 indicates that the current PV is at level 3 in Fig. 1, XX is a two-character variable whose value denotes the type of ensemble method (e.g. RC for Random Committee, and NE for No Ensemble) and BC-sel denotes base classifier selection. Note that all PVs in this table have the substrings L3 and BC-sel in their name, since all refer to the selection of a base classifier at level 3 of Fig. 1. The number of components in the PV (in the third column) is the number of candidate base classifiers for the corresponding type of ensemble. The list of such classifiers is show in the last column. Note that all 6 ensemble types in this table use the same set of 9 candidate base classifiers. Note also that there is no PV for selecting base classifiers when the RF ensemble method is used, since RFs always use DTs as the base classifier.

The fourth level is the base classifiers' hyper-parameter optimization level. At this level, there is again one PV for each hyperparameter of each of the base classifiers at the third level. Table 3 shows the full set of PVs at level 4. This table has

Table 2
Probability Vectors (PVs) for base classifier selection at level 3 of Fig. 1. The columns of the table refer to the type of the ensemble at level 1, the number of PVs for the ensemble, the name of the PV, the number of components in the PV, and the candidate base classifiers (one PV component for each of them), respectively.

Ensem. type at Level 1	PV name	# Comp. in the PV	Candidate base classifiers
Rand. Comm.	L3-RC-BC-sel	9	NET, NB, MLP, SMO, IBK, KST, J48, DT, RT
AdaBoost	L3-AD-BC-sel	9	NET, NB, MLP, SMO, IBK, KST, J48, DT, RT
Bagging	L3-BA-BC-sel	9	NET, NB, MLP, SMO, IBK, KST, J48, DT, RT
Stacking	L3-ST-BC-sel	9	NET, NB, MLP, SMO, IBK, KST, J48, DT, RT
Vote	L3-VT-BC-sel	9	NET, NB, MLP, SMO, IBK, KST, J48, DT, RT
No ensemble	L3-NE-BC-sel	9	NET, NB, MLP, SMO, IBK, KST, J48, DT, RT

a structure analogous to the one of Table 1. Note that at level 4 of Fig. 1 each PV is used to optimize the hyper-parameters of a base classifier regardless of which type of ensemble (at level 2) is using that base classifier.

This approach has the disadvantage of being a relatively coarse-grained approach for hyper-parameter optimization, limiting PBIL's ability to find fine-grained hyper-parameters settings of a base classifier that would be particularly tailored for a specific type of ensemble method (with its specific hyper-parameter settings). For example, intuitively, the optimal hyperparameter settings for J48 would depend on whether it is being used as a base classifier for AdaBoost or for Stacking (as well as on their hyper-parameter settings).

This coarse-grained hyper-parameter optimization tends to occur more strongly in the early generations of PBIL, when a given base classifier would tend to be used as part of several different ensemble types in different individuals (since initially all ensemble types have the same probability of being selected to be used in an individual). As generations pass by, the problem should be to some extent mitigated (although not completely eliminated) as the PV for selecting the ensemble type at level 1 is expected to gradually converge to the best ensemble type. Hence, in late generations a given base classifier should be used in different individuals mainly as part of the same best ensemble type, giving more opportunities for the algorithm to focus on a finer-grained optimization of hyper-parameter settings of that base classifier, i.e. finding hyper-parameter settings that are more tailored to that particular best ensemble type.

To compensate for the above disadvantage, however, this approach of having a single PV for each base classifier's hyper-parameter at level 4 has the important advantage that it drastically reduces the number of PVs that need to be optimized by the PBIL, which should drastically reduce the time for the algorithm to converge to a near-optimal solution. In addition, the much smaller number of PVs associated with this approach can also help to reduce the chances of overfitting, particularly on datasets that are not very large.

Finally, there is also a set of PVs referring to hyper-parameters that are specific to the base classifier SMO, a type of Support Vector Machine. These PVs are specified in Table 4, and they are PVs at the fifth level of the PV hierarchy. This fifth level was not shown in Fig. 1 in order to keep the figure relatively simple.

At the start of the evolution (generation 0), all solution components in each of the PVs are initialized with a uniform probability distribution, so that each component is equally likely to be sampled from each PV. During the evolutionary search, at each generation (iteration) individuals are generated by sampling solution components from the PVs; then the generated individuals are evaluated by the fitness function, and the best 50% of the individuals in the current generation are selected for updating the probabilities of individual solution components in the PVs. These processes are described in the next subsections.

4.2. The procedure for generating individuals by sampling from the probability vectors

We now describe how PBIL-Auto-Ens-v2 generates individuals at each generation, by sampling solution components from the hierarchy of Probability Vectors (PVs) shown in Fig. 1.

An individual represents a full candidate solution, specifying the choice of an ensemble method and its hyper-parameter settings (or No Ensemble if this option was chosen), as well as the choice of a base classifier and its hyper-parameter settings.

To generate an individual, PBIL-Auto-Ens-v2 first samples a value from the PV determining the choice of an ensemble method, at level 1 of Fig. 1. The chosen ensemble method is then used to determine which probability vector will be sampled at level 2, and so on, so that the generation of an individual can be conceptualized as following a path in the

Table 3
Probability Vectors (PVs) for the base classifiers' hyper-parameter optimization at level 4 of Fig. 1. The columns of the table refer to the type of the base classifier at level 3, the number of PVs for that base classifier, the name of the PV, the number of components in the PV, and the values of the hyper-parameters, respectively.

BC L2	# PVs BCs.	PV name	# Comp. in PV	Hyper-parameter values
DT	4	LV4-DT-E	4	acc, rmse, mae, auc
		LV4-DT-I	2	true / false
		LV4-DT-S	2	BestFirst, GreedyStepWise
		LV4-DT-X	1	from 1 to 4
IBK	5	LV4-IBK-E	2	true / false
		LV4-IBK-K	64	from 1 to 64
		LV4-IBK-X	2	true / false
		LV4-IBK-F	2	true / false
		LV4-IBK-I	2	true / false
J48	8	LV4-J48-O	2	true / false
		LV4-J48-U	2	true / false
		LV4-J48-B	2	true / false
		LV4-J48-J	2	true / false
		LV4-J48-A	2	true / false
		LV4-J48-S	2	true / false
		LV4-J48-M	64	from 1 to 64
		LV4-J48-C	95	from 0,05 to 5,0
KST	3	LV4-KST-B	100	from 1 to 100
		LV4-KST-E	2	true / false
		LV4-KST-X	4	a, d, m n
MLP	8	LV4-MLP-L	10	from 0,1 to 1,0
		LV4-MLP-M	10	from 0,1 to 1,0
		LV4-MLP-B	2	true / false
		LV4-MLP-H	4	a, i, o, t
		LV4-MLP-C	2	true / false
		LV4-MLP-R	2	true / false
		LV4-MLP-D	2	true / false
		LV4-MLP-S	255	from 1 to 255
NB	2	LV4-NB-D	2	true / false
		LV4-NB-K	2	true / false
NET	2	LV4-NET-Q	6	K2, HC, LHC, SA, TS, TAN
		LV4-NET-D	2	true / false
RT	5	LV4-RT-M	64	from 1 to 64
		LV4-RT-K	33	from 0 to 32
		LV4-RT-depth	21	from 0 to 20
		LV4-RT-N	6	from 0 to 5
		LV4-RT-U	2	true / false
SMO	1	LV4-SMO-SEL	4	Def., N.P.Kernel, P.Kernel, Puk, RBFKernel

graph representing the hierarchical structure of the set of PVs, sampling from one PV at each level, until a full candidate solution is specified.

An example of how an individual is generated is shown in Fig. 2. In this example, the individual is generated by the following sequence of PV samplings and solution component choices:

- At level 1: sampling from the PV for ensemble method, choosing AdaBoost (AD) ensemble (represented by value 2);
- At level 2: sampling from the PVs for hyper-parameters Q, P, I and S of AD, choosing values True, 51, 127 and 3, respectively;
- At level 3: sampling from the PV for hyper-parameter BC-Sel (Base Classifier Selection) of AD, choosing base classifier IBK (represented by value 5);
- At level 4: sampling from the PVs for hyper-parameters E, K, X, F, I of IBK, choosing values False, 63, False, True, True, respectively.

Hence, an individual contains a variable-length list of pairs of the form (i-th PV, i-th Index), where in each pair, the first element denotes the id (name) of a PV used to generate the individual, and the second element denotes the index of the component of that PV that was sampled; i.e., the index of the method or of the hyper-parameter setting, among the components of the PV.

Table 4
Probability Vectors (PVs) for the SVM base classifier's hyper-parameter optimization.

SMOs L4, L5	PVs SMOs.	PV name	Comp. in PV	Hyper-parameter values
def.	4	LV5-default-C	11	from 0.5 to 1.5
		LV5-default-N	3	from 0 to 2
		LV5-default-M	2	true / false
		LV5-default-K	4	N.P.Kernel, P.Kernel, Puk, RBFKernel
N.P.Kernel	6	LV5-NPKernel-C	11	from 0.5 to 1.5
		LV5-NPKernel-N	3	from 0 to 2
		LV5-NPKernel-M	2	true / false
		LV5-NPKernel-K	4	N.P.Kernel, P.Kernel, Puk, RBFKernel
		LV5-NPKernel-E	49	from 0.2 to 5.0
		LV5-NPKernel-L	2	true / false
PolyKernel	6	LV5-PKernel-C	11	from 0.5 to 1.5
		LV5-PKernel-N	3	from 0 to 2
		LV5-PKernel-M	2	true / false
		LV5-PKernel-K	4	N.P.Kernel, P.Kernel, Puk, RBFKernel
		LV5-PKernel-E	49	from 0.2 to 5.0
		LV5-PKernel-L	2	true / false
Puk	6	LV5-Puk-C	11	from 0.5 to 1.5
		LV5-Puk-N	3	from 0 to 2
		LV5-Puk-M	2	true / false
		LV5-Puk-K	4	N.P.Kernel, P.Kernel, Puk, RBFKernel
		LV5-Puk-S	100	from 0.1 to 10.0
		LV5-Puk-O	10	from 0.1 to 1.0
RBFKernel	5	LV5-RBFKernel-C	11	from 0.5 to 1.5
		LV5-RBFKernel-N	3	from 0 to 2
		LV5-RBFKernel-M	2	true / false
		LV5-RBFKernel-K	4	N.P.Kernel, P.Kernel, Puk, RBFKernel
		LV5-RBFKernel-G	1000	from 0.001 to 1.0

For instance, the individual generated by the above sequence of PV samplings would be represented by the following list of pairs (we show next only the first two pairs and the last pair of this list, to simplify):

(L1-EnsType, 2), (L2-AD-Q, 1), . . ., (L4-IBK-I, 1).

where, in the first pair, L1-EnsType is the PV at level 1 for selecting the ensemble method and the value 2 is the index of the component AdaBoost of that PV; in the second pair, L2-AD-Q is the PV at level 2 for selecting the value of hyper-parameter Q of the AD method and 1 is the index of the value True, which was chosen (sampled) for this hyper-parameter, etc.

4.3. Fitness computation

At each generation, the fitness (quality measure) of each just-created individual is evaluated by applying the classifier ensemble represented by that individual to the training set, using an internal 10-fold cross-validation procedure (a well-known evaluation procedure in machine learning) to estimate the error rate of that ensemble. That is, the fitness of an individual is the mean of the error rates over the 10 folds of the internal cross-validation procedure. We emphasize that fitness is computed using only the training set, without any access to the test set, which is reserved for the final evaluation of the predictive accuracy of the best ensemble returned by the algorithm.

4.4. Selection of the best individuals and updating of the probability vectors

At the end of each generation, the 50% best individuals in terms of fitness of that generation (i.e. the 50% individuals with the smallest estimated error rates on the training set) are selected and their solution components are used to update the corresponding component probabilities in all the PVs that were used to create the selected individuals. This updating consists of increasing the probability for each solution component in proportion to the relative frequency of use of that component among the selected individuals (the 50% best ones), and also in proportion to the learning rate parameter. More precisely, for each i-th solution component, its probability at the end of generation g, denoted by $p[i]_g$, is updated using the formula:

$$p[i]_g = (1 - LR) * p[i]_{g-1} + LR * RelFreq[i]_g,$$

where LR is the learning rate and $RelFreq[i]_g$ is the relative frequency of the i-th solution component among the individuals selected at generation g. Hence, by iteratively selecting the best candidate solutions and increasing the probabilities of each

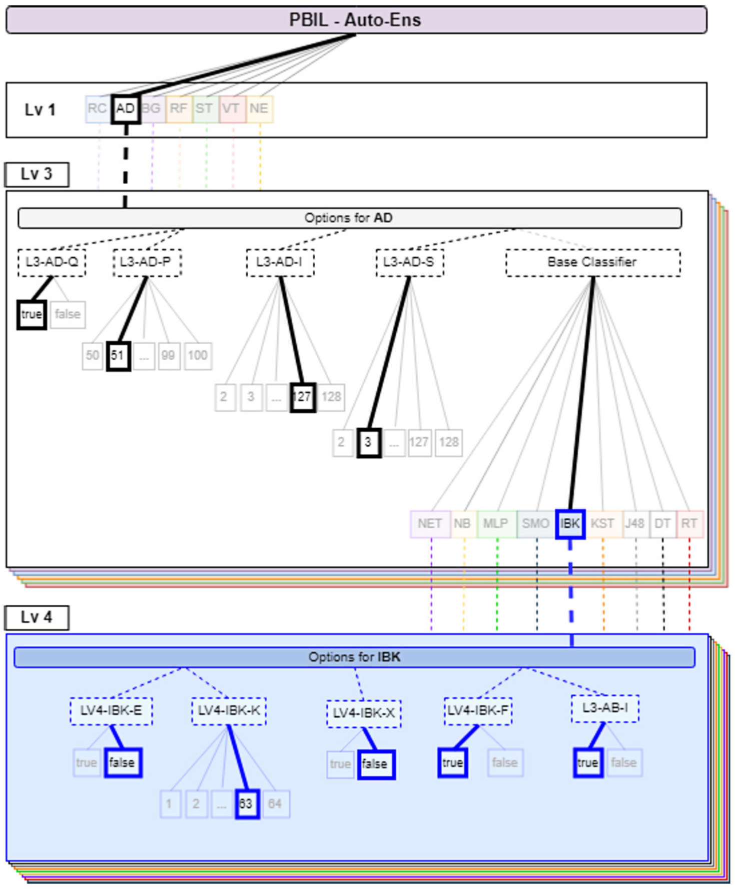

Fig. 2. An example of how an individual is generated.

of their components in the probability vector, gradually the probability vector evolves to contain higher probability values for the best solution components, leading to the creation of better and better classifier ensembles. This evolutionary process terminates when a predefined runtime limit is reached.

Fig. 3 shows, in diagrammatic form, an example of how the probability vectors are updated after selecting the individual shown in Fig. 2. In Fig. 3, the small yellow circles with the symbol "+p" identify the probability vectors' components whose probability values are increased, due to the fact that those components were included in the candidate solution represented by the individual shown in Fig. 2.

PBIL-Auto-Ens-v2 differs from its predecessor (PBIL-Auto-Ens-v1) [28] in two major ways. First, PBIL-Auto-Ens-v2 uses a more elaborated hierarchical structure for the set of PVs. In particular, in PBIL-Auto-Ens-v1, at the first level of the hierarchy there is a mixture of classifier ensemble methods and single base classifiers (which directly compete with each other for selection at the first level), whilst in PBIL-Auto-Ens-v2 classifier ensembles and base classifiers are kept in separate hierarchical levels. Hence, there are more modular competitions between the classifier ensembles and their base classifiers, with separate competitions for selection within each of these two groups of classifiers. Second, in PBIL-Auto-Ens-v1 each individual is represented by a variable-length binary vector, with a sub-optimal procedure for decoding those bits into choices of ensembles, base classifiers, and hyper-parameter settings for both types of methods. By contrast, BIL-Auto-Ens-v2 uses a simpler and more natural individual representation, using PV names and index values to directly encode choices of ensembles, base classifiers, and hyper-parameter settings.

4.5. The space complexity of PBIL-Auto-Ens-v2

PBIL-Auto-Ens-v2's space complexity can be calculated by considering the space taken by the set of probability vectors and the space taken by the current population of individuals, as follows. First, the algorithm stores in memory a set of probability vectors, whose total size, denoted PVSize is given by the total number of components in all probability vectors. Second, the space taken by the current population of individuals (candidate solutions) equals the population size (PopSize) times the (average) size of each individual (IndSize). Hence, the space complexity is $O(PVSize + PopSize \times IndSize)$. To make this formula more concrete, PVSize is given by the summation of all values in the columns "#Comp. in PV" in

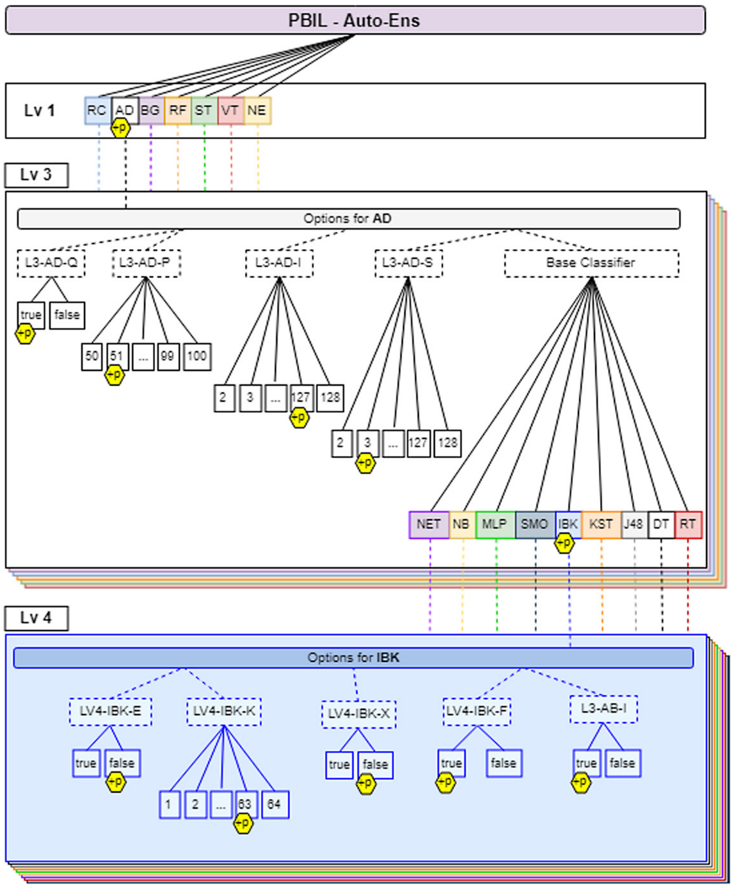

Fig. 3. An example of probability vector updates after selecting the individual shown in Fig. 2. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

Tables 1, 2, 3, 4 plus 9 components for selecting the type of ensemble method at level 1, which is in total 4,261. PopSize is fixed at 50, but IndSize varies across individuals. In practice, IndSize is substantially less than 50 for any given individual (since each individual needs to store only the chosen parameter settings of chosen algorithms, a small subset of the entire set of PVs), so this space complexity is dominated by PVSize.

5. Experimental methodology

5.1. Datasets used in the experiments

The proposed PBIL-Auto-Ens method was evaluated on 21 classification datasets, available for download from the well known UCI machine learning repository. Most of these datasets have also been used in recent Auto-ML studies [4], [6], [21]. Table 5 shows the number of instances, attributes (separately for discrete and continuous attributes) and classes in each of the datasets.

5.2. Methods compared in the experiments

The two versions of PBIL-Auto-Ens were compared against two strong baseline methods: random forests (a well-known ensemble method) and an Auto-ML method, namely Auto-WEKA [4]. All experiments were done using a well-known 5-fold cross-validation (CV) procedure. For the Auto-ML methods, i.e. the two PBIL-Auto-Ens versions and Auto-WEKA, we used a nested version of the CV procedure. More precisely, these Auto-ML methods have been run with an external 5-fold CV, and an internal 10-fold CV. Hence, at the external CV level, the input dataset is randomly divided into 5 folds (each with about 20% of the instances), and then the two versions of PBIL-Auto-Ens and Auto-WEKA are run 5 times, each using a different fold as the test set and the other 4 folds as the training set. In each of those 5 runs, the training set (80% of the full dataset) is randomly divided into 10 folds, each with about 10% of the training set (i.e., about 8% of the full dataset). Then, whenever a candidate solution (ensemble method or base classifier) is generated by PBIL-Auto-Ens or Auto-WEKA, that solution is evaluated by running its configuration using the internal 10-fold CV, so that each of the 10 runs of that candidate solution

Table 5
Main characteristics of the datasets used in the experiments.

Id	Dataset	# Instances	# Disc. Attr.	# Cont. Attr.	# Classes
d1	Abalone	4,177	1	7	28
d2	Adult	32,561	8	6	2
d3	Arrhythmia	452	0	260	13
d4	Automobile	205	11	15	7
d5	Car	1,728	0	6	4
d6	Dermatology	366	1	33	6
d7	Ecoli	336	0	7	8
d8	Flags	194	20	10	8
d9	GermanCredit	1,000	13	7	2
d10	Glass Identification	214	0	10	7
d11	Image Segmentation	2,310	0	19	7
d12	KR-vs-KP	3,196	36	0	2
d13	Madelon	2,600	0	500	2
d14	Nursery	12,960	8	0	5
d15	Secom	1,567	0	590	2
d16	Semeion	1,593	0	256	10
d17	SolarFlare1	323	13	0	8
d18	Sonar	208	0	60	2
d19	Waveform	5,000	0	40	3
d20	Wine	4,898	0	11	11
d21	Yeast	1,484	0	8	10

Table 6
Configurations for both PBIL-Auto-Ens versions.

Population size	Learning rate	% of best individuals selected
50	0.5	50%

uses 9 internal folds (72% of the full dataset) as a learning set (to learn the classification model) and one internal fold (8% of the full dataset) as a validation set to evaluate the predictive accuracy of the learned model. The quality measure of that candidate solution is given by the mean error rate of the learned model over the 10 internal validation sets. Hence, the evaluation of each candidate solution uses only the training set, not the external test set, which is reserved for measuring the predictive accuracy of the best solution returned by the two versions of PBIL-Auto-Ens and Auto-WEKA.

Note that both PBIL-Auto-Ens and Auto-WEKA are non-deterministic search methods, i.e. their results depend on a seed number used to randomly initialize the candidate solutions. For each method, we report its mean result over experiments with 5 random seeds (the same seeds are used by all methods), running an external 5-fold CV for each seed as explained above – i.e., each reported result is the mean over 25 results (with 25 different test sets). Random forest is also non-deterministic, and it was also run with the same 5 random seeds and 5-fold cross-validation – i.e. its results are also the mean over 25 test sets.

We used all default parameter settings of Auto-WEKA, including the classification error rate (the proportion of incorrectly classified instances) as the evaluation function to be optimized during training. To make the comparison between both PBIL-Auto-Ens versions and Auto-WEKA fair, we also used error rate as the fitness function of them. The three Auto-ML methods used the same runtime limit, 60 minutes for each run, where one run means one execution of one iteration of the external 5-fold CV (applying the method to the training set of that iteration) for a single value of the random seed, for each dataset. Hence, in total each Auto-ML method was run for 525 hours, considering 5 external CV iterations times 5 seeds times 21 datasets.

Both PBIL-Auto-Ens versions were run with the following parameter settings (as summarized in Table 6): population size of 50, learning rate of 0.5, and 50% of the best individuals of the current generation selected for updating the probability model. The latter two parameter settings are relatively standard in the PBIL literature, whilst the population size was set based on preliminary experiments. Note that, unlike most PBIL (and EA) implementations, the two PBIL-Auto-Ens versions do not have a parameter for the number of generations, because their stopping criterion is a runtime limit (like in Auto-WEKA). The random forest algorithm was run with its default parameter settings in WEKA.

Both PBIL-Auto-Ens versions have been implemented in Java using the WEKA API. We run the experiments on a desktop PC with Ubuntu 16.04 64 bit operating system driven by an Intel(R) Xeon(R) CPU E5-4610 v4 @ 1.80 GHz, 6 core, and RAM with 6 Gb. The program code of PBIL-Auto-Ens-v2 is freely available at: https://github.com/ml-imd/PBIL-AutoEns-v2.

5.3. Predictive accuracy measures

All methods were evaluated based on 4 predictive accuracy measures, which can be divided into two groups: (a) the classification error rate; and (b) the precision, recall and F-measure. The error rate is simply the ratio of the number of

Table 7
Error Rate on the test set (mean over 25 runs for each dataset).

Id	Dataset	Auto-WEKA	Random Forest	PBIL-Auto-Ens-v1	PBIL-Auto-Ens-v2
d1	Abalone	0.7316	0.7616	**0.7312**	0.7459
d2	Adult	0.1452	0.1504	0.1427	**0.1426**
d3	Arrhythmia	0.2769	0.3221	**0.2743**	0.2751
d4	Automobile	0.1912	**0.1824**	0.1834	0.1842
d5	Car	**0.0028**	0.0576	0.0102	0.0082
d6	Dermatology	0.0294	0.0339	**0.0268**	0.0284
d7	Ecoli	0.1387	0.1542	0.1420	**0.1363**
d8	Flags	0.3352	**0.3155**	0.3455	0.3537
d9	German-Credit	0.2716	0.2612	0.2700	**0.2602**
d10	Glass Identification	0.2383	**0.2142**	0.2392	0.2195
d11	Image Segmentation	**0.0199**	0.0203	0.0239	0.0231
d12	KR-vs-KP	0.0527	0.0091	0.0154	**0.0057**
d13	Madelon	0.3009	0.3541	**0.2869**	0.2991
d14	Nursery	0.0228	0.0121	0.0098	**0.0085**
d15	Secom	0.0779	0.0664	0.0691	**0.0658**
d16	Semeion	0.0989	0.0637	**0.0588**	0.0694
d17	SolarFlare1	0.1182	0.1251	0.1140	**0.1126**
d18	Sonar	0.2153	0.1788	**0.1539**	0.1701
d19	Waveform	**0.1333**	0.1485	0.1476	0.1349
d20	Wine-quality	0.3393	0.3220	**0.3210**	0.3384
d21	Yeast	0.3980	0.3954	0.3952	**0.3950**
Number of wins		3/21	3/21	7/21	**8/21**
Average Rank		2.95	2.86	2.19	**1.95**

misclassified instances over the total number of instances. The error rate does not distinguish between different types of misclassifications, so it tends to give more importance to the correct classification of instances belonging to the most frequent class in the dataset than to the correct classification of instances belonging to the other class(es).

By contrast, the other 3 measures, used as a whole, tend to evaluate the predictive accuracy in a more balanced way across all classes. Precision is the ratio of the number of correctly classified positive instances over the total number of instances classified as positive (regardless of they belonging to the positive or negative class). Recall is the ratio of the number of correctly classified positive instances over the total number of positive instances (regardless of they being correctly or wrongly classified). The F-measure is the harmonic average between precision (prec) and recall (rec), defined as:

$$F - measure = \frac{(2 * precision * recall)}{(precision + recall)} \tag{1}$$

Note that the definitions of precision, recall and F-measure consider one class as the positive class and the other class(es) as the negative class. Hence, to compute average values of these measures across classes, we need to consider one class at a time as the positive class. More precisely, the average values for these measures are computed as follows. For each run of a classification method, the average precision and recall were computed by first measuring the precision and recall per class by considering each class in turn as the positive class, then computing the arithmetic mean of those precision and recall values over all classes, and finally averaging over all runs of the method for that dataset. The average F-measure was computed by applying the above F-measure formula to the average values of precision and recall.

6. Experimental results

This section presents experimental results comparing the predictive performance of the proposed PBIL-Auto-Ens-v2 against three other methods, namely: (a) its previous version (PBIL-Auto-Ens-v1); (b) the Auto-WEKA version adapted to focus on ensembles, using the same search space as both versions of PBIL-Auto-Ens; and (c) a random forest algorithm with default hyper-parameter settings, as a strong baseline ensemble method. The results are discussed in terms of two types of predictive performance measures: the error rate; and the precision, recall and F-measure.

6.1. Results for the mean error rate

Table 7 presents the mean error rates for Auto-WEKA, PBIL-Auto-Ens-v1 and PBIL-Auto-Ens-v2. In this table, the best result for each dataset is shown in boldface. In addition, for each method, its number of wins (i.e., the number of datasets where it obtained the best result) and its average rank are shown at the bottom of the table. The lower the average rank of a method, the better its predictive performance.

As shown at the bottom of Table 7, PBIL-Auto-Ens-v2 obtained the best (smallest) average rank (1.95), with PBIL-Auto-Ens-v1 in the second place (rank 2.19). In addition, PBIL-Auto-Ens-v2 and PBIL-Auto-Ens-v1 achieved the smallest error

Table 8
Average Precision on the test set (mean over 25 runs for each dataset).

Id	Dataset	Auto-WEKA	Random Forest	PBIL-Auto-Ens-v1	PBIL-Auto-Ens-v2
d1	Abalone	0.0999	0.1030	0.1075	**0.1078**
d2	Adult	0.8181	0.8038	**0.8208**	0.8197
d3	Arrhythmia	0.4407	0.3061	**0.4505**	0.4467
d4	Automobile	0.6662	0.6114	0.6578	**0.6861**
d5	Car	**0.9944**	0.8589	0.9913	0.9925
d6	Dermatology	0.9614	0.9684	**0.9730**	0.9700
d7	Ecoli	0.6259	0.5502	0.6227	**0.6296**
d8	Flags	**0.4728**	0.4412	0.4443	0.4101
d9	GermanCredit	0.6711	0.6911	0.6693	**0.6916**
d10	Glass	0.7334	0.6950	0.7587	**0.7636**
d11	Image Seg.	0.9768	**0.9801**	0.9780	0.9785
d12	KR-vs-KP	0.9494	0.9910	0.9937	**0.9944**
d13	Madelon	0.7022	0.6482	**0.7167**	0.7041
d14	Nursery	0.7547	0.7712	0.7772	**0.8021**
d15	Secom	**0.4962**	0.4834	0.4668	0.4773
d16	Semeion	0.9056	0.9367	**0.9435**	0.9339
d17	SolarFlare1	0.1351	0.1373	**0.1752**	0.1488
d18	Sonar	0.7895	0.8292	**0.8555**	0.8373
d19	Waveform	0.8581	0.8524	0.8534	**0.8658**
d20	Wine	0.3141	**0.3170**	0.3137	0.3092
d21	Yeast	**0.5512**	0.4751	0.5223	0.4985
Number of wins		4/21	2/21	7/21	**8/21**
Average Rank		2.81	3.10	2.14	**1.95**

among all methods in 8 and 7 of the 21 datasets, respectively, whilst Auto-WEKA and random forest were the winner in only 3 datasets each.

In order to conduct a statistical analysis of the results, the Friedman test and Nemenyi post-hoc test were used (as recommended in [27]) to determine whether or not there is a statistically significant difference between the predictive accuracies of the methods across the 21 datasets. Both tests are applied at the conventional significance level of 5%. The Friedman test was chosen because it is non-parametric (avoiding the assumption of normality), being based on the average rank of the four methods across all datasets. Its null hypothesis is that there is no difference in the average ranks of the four methods. If this null hypothesis is rejected, we apply the post-hoc Nemenyi test to evaluate if there is a significant difference between each pair of methods. This is necessary because the Friedman test compares the four methods as a whole, without indicating which pairs of methods have significantly difference performance.

The Friedman test produced the p-value $= 0.0208$, therefore the difference between the error rates of the four methods is statistically significant. The pairwise comparisons using the Nemenyi post-hoc test produced only one statistically significant result: PBIL-Auto-Ens-v2 obtained a significantly better average rank of error rates than Auto-WEKA (p-value $= 0.0420$). That is, there is no significant difference between the error rates of other pairs of methods.

6.2. Results for the average precision, recall and F-measure

Tables 8, 9 and 10 present the average values of precision, recall, and F-measure, respectively, for all methods being compared. As shown at the bottom of Table 8, PBIL-Auto-Ens-v2 obtained the best (smallest) average rank (1.95), with PBIL-Auto-Ens-v1 in the second place (rank 2.14). In addition, PBIL-Auto-Ens-v2 and PBIL-Auto-Ens-v1 achieved the highest precision among all methods in 8 and 7 of the 21 datasets, respectively; whilst Auto-WEKA and random forest were the winner in only 4 and 2 datasets, respectively.

In Table 9 (for recall), PBIL-Auto-Ens-v2 obtained the best average rank (1.76), with PBIL-Auto-Ens-v1 in the second place (rank 2.24). In addition, PBIL-Auto-Ens-v2 and PBIL-Auto-Ens-v1 achieved the highest recall among the three methods in 9 and 6 of the 21 datasets, respectively; whilst Auto-WEKA and random forest were the winner in only 4 and 2 datasets, respectively.

Regarding the F-measure (involving a trade-off between precision and recall), as shown in Table 10, PBIL-Auto-Ens-v2 obtained again the best average rank (1.81), with PBIL-Auto-Ens-v1 again in the second place (rank 2.19). In addition, PBIL-Auto-Ens-v2 and PBIL-Auto-Ens-v1 achieved the highest F-measure in 9 and 6 of the 21 datasets, respectively; whilst Auto-WEKA and random forest were the winner in only 4 and 2 datasets, respectively.

We also applied the aforementioned Friedman and post-hoc Nemenyi tests to the methods' results for precision, recall and F-measure, based on the methods' average ranks shown in Tables 8, 9 and 10, respectively. For both statistical tests, we used again the conventional significance level of 5%. For precision, recall and F-Measure, the Friedman test produced the p-values of 0.0114, 0.0023 and 0.0032, respectively. All these results are statistically significant, so we used the Nemenyi post-hoc test to compare the precision, recall and F-measure for each of the six pairs of methods.

Table 9
Average Recall on the test set (mean over 25 runs for each dataset).

Id	Dataset	Auto-WEKA	Random Forest	PBIL-Auto-Ens-v1	PBIL-Auto-Ens-v2
d1	Abalone	0.1029	0.1028	0.1089	**0.1167**
d2	Adult	0.7562	0.7641	**0.7770**	0.7690
d3	Arrhythmia	0.4202	0.3081	0.4198	**0.4206**
d4	Automobile	0.6491	0.6090	0.6449	**0.6731**
d5	Car	**0.9891**	0.8591	0.9886	0.9888
d6	Dermatology	0.9568	0.9565	**0.9717**	0.9690
d7	Ecoli	0.6190	0.5483	0.6206	**0.6207**
d8	Flags	**0.4593**	0.4472	0.4441	0.4237
d9	GermanCredit	0.6348	0.6452	0.6292	**0.6599**
d10	Glass	0.7136	0.6931	0.7299	**0.7385**
d11	Image Seg.	0.9759	**0.9799**	0.9775	0.9791
d12	KR-vs-KP	0.9461	0.9907	0.9935	**0.9942**
d13	Madelon	0.6991	0.6470	**0.7131**	0.7013
d14	Nursery	0.7476	0.7713	0.7796	**0.8025**
d15	Secom	**0.5019**	0.4835	0.4997	0.5009
d16	Semeion	0.9006	0.9369	**0.9410**	0.9305
d17	SolarFlare1	0.1308	0.1370	**0.1372**	0.1319
d18	Sonar	0.7824	0.8185	**0.8423**	0.8262
d19	Waveform	0.8558	0.8520	0.8528	**0.8654**
d20	Wine	0.2284	**0.2659**	0.2400	0.2425
d21	Yeast	**0.5174**	0.4775	0.4881	0.4938
Number of wins		4/21	2/21	6/21	**9/21**
Average Rank		2.86	3.14	2.24	**1.76**

Table 10
Average F-Measure on the test set (mean over 25 runs for each dataset).

Id	Dataset	Auto-WEKA	Random Forest	PBIL-Auto-Ens-v1	PBIL-Auto-Ens-v2
d1	Abalone	0.1012	0.1039	0.1082	**0.1086**
d2	Adult	0.7877	0.7835	**0.7902**	0.7895
d3	Arrhythmia	0.4282	0.3075	0.4324	**0.4360**
d4	Automobile	0.6592	0.6130	0.6511	**0.6772**
d5	Car	**0.9917**	0.8586	0.9899	0.9900
d6	Dermatology	0.9601	0.9623	**0.9724**	0.9695
d7	Ecoli	0.6219	0.5506	0.6213	**0.6247**
d8	Flags	**0.4646**	0.4401	0.4433	0.4160
d9	GermanCredit	0.6523	0.6742	0.6483	**0.6748**
d10	Glass	0.7215	0.6947	0.7474	**0.7506**
d11	Image Seg.	0.9765	**0.9805**	0.9778	0.9789
d12	KR-vs-KP	0.9478	0.9909	0.9938	**0.9943**
d13	Madelon	0.7006	0.6476	**0.7149**	0.7051
d14	Nursery	0.7510	0.7722	0.7784	**0.8021**
d15	Secom	**0.5091**	0.4831	0.4827	0.4876
d16	Semeion	0.9031	0.9368	**0.9423**	0.9322
d17	SolarFlare1	0.1315	0.1377	**0.1513**	0.1370
d18	Sonar	0.7859	0.8237	**0.8488**	0.8317
d19	Waveform	0.8579	0.8522	0.8531	**0.8656**
d20	Wine	0.2641	**0.2795**	0.2767	0.2780
d21	Yeast	**0.5332**	0.4741	0.5036	0.4925
Number of wins		4/21	2/21	6/21	**9/21**
Average Rank		2.90	3.10	2.19	**1.81**

Regarding the precision measure, the pairwise comparisons using the Nemenyi post-hoc test produced only one statistically significant result: PBIL-Auto-Ens-v2 obtained a significantly better result than the random forest algorithm (p-value = 0.0210).

Regarding the recall measure, the Nemenyi post-hoc test produced two statistically significant results: PBIL-Auto-Ens-v2 was significantly better than both Auto-WEKA (p-value = 0.0256) and random forest (p-value = 0.0037).

Regarding the F-measure, the Nemenyi post-hoc test produced again two statistically significant results: PBIL-Auto-Ens-v2 was significantly better than both Auto-WEKA (p-value = 0.0304) and random forest (p-value = 0.0069).

These results can be summarized as follows. PBIL-Auto-Ens-v2 consistently obtained the best results (in terms of both average rank and number of wins) for all the four measures: error rate, Precision, Recall and F-measure. There was no statistically significant difference between the results of PBIL-Auto-Ens-v2 and PBIL-Auto-Ens-v1, for all measures. However,

Table 11
Pearson correlation between Number of Classes and Accuracy Measures.

Pearson	Error	Precision	Recall	F-measure
Auto-WEKA	0.7224	−0.6297	−0.6167	−0.6254
Random Forest	0.7308	−0.6305	−0.6197	−0.6253
PBIL-Auto-Ens-v1	0.7290	−0.6259	−0.6158	−0.6194
PBIL-Auto-Ens-v2	0.7355	−0.6253	−0.6146	−0.6209

PBIL-Auto-Ens-v2's results were statistically significantly better than the results of both Auto-WEKA and random forest for three out of the four measures of predictive performance used in our experiments. By contrast, although PBIL-Auto-Ens-v1 outperforms both Auto-WEKA and random forest for all those four measures, the difference of results is not statistically significant for any measure. Hence, PBIL-Auto-Ens-v2 represents a clear improvement over PBIL-Auto-Ens-v1, in terms of predictive performance.

Finally, it is worth noting that in general the results for all predictive accuracy measures, for all methods, tend to be worse in datasets with a large number of class labels. The most typical example of this scenario are the results for the Abalone dataset, which has by far the largest number of class labels (28). The results of all methods in this dataset are by far worse than their results in other datasets. To investigate in more detail the relationship between the number of class labels and the predictive accuracy results, we have measured the well-known Pearson's linear correlation coefficient (r) between the number of class labels and the values of each accuracy measure (error rate, precision, recall and F-measure) across the 21 datasets, for each of the four methods being compared in our experiments. These results are shown in Table 11.

As observed in this table, there is a strong positive correlation (r greater than 0.7) between the number of class labels and the error rate for all methods – i.e., in general larger numbers of class labels are associated with larger (worse) values of the error rate. Conversely, there is a strong negative correlation (r smaller than -0.6) between the number of class labels and the values of precision, recall and F-measure for all methods – i.e., in general larger numbers of class labels are associated with smaller (worse) values of precision, recall and F-measure. Note also that, for each predictive accuracy measure, there is little variation in the r values across the methods, i.e. all four methods are equally affected by the difficulty of predicting a large number of class labels.

6.3. An analysis of the best solutions returned by PBIL-Auto-Ens-v2

We now analyze the relative frequency with which different types of classifiers are returned by PBIL-Auto-Ens-v2 as the best solution found during its search. We perform this analysis only for PBIL-Auto-Ens-v2 because it obtained overall the highest predictive accuracy among the four methods, as discussed earlier. Although each returned solution consists of the name of a classifier and its configuration (i.e., its hyper-parameter settings), we report only the name of returned classifier, which is higher level information, much easier to interpret than the low-level information associated with hyper-parameter settings.

Fig. 4 presents bar graphs displaying the 25 best solutions (classifiers) returned by PBIL-Auto-Ens-v2 (i.e., the best solution returned by each run of this method) for each of the 21 datasets. For each cell (dataset) in this figure, the horizontal axis shows the acronyms of the classifiers selected for that dataset, and the numbers at the top of each bar represent the number of times that the corresponding classifier was selected, out of the 25 runs. Recall that, although the search space of PBIL-Auto-Ens-v2 includes mainly classifier ensembles (and their configurations), it also includes the option of selecting and configuring only a single base classifier. The latter is a useful option, since in some datasets a single classifier can have a similar or perhaps even somewhat better predictive performance than an ensemble. In addition, a single classifier has the advantage of avoiding the need for the extra computational cost and complexity of an ensemble.

Overall, considering the results across all datasets in Fig. 4, the classification algorithm most frequently selected by PBIL-Auto-Ens-v2 was by far AdaBoost (ADA), selected in 205 cases; followed by Bagging (BAG) and Random Forest (RF), selected in 65 and 60 cases, respectively. Hence, one of these three ensemble algorithms was selected as the best classifier in 330 cases (about 63% of all 525 cases).

Comparing the selection frequencies of ensembles and single classifiers as a whole, PBIL-Auto-Ens-v2 selected a classifier ensemble in 355 cases (about 68% of the cases), whilst it selected a single base classifier in the remaining 170 cases.

Moreover, for 5 datasets, PBIL-Auto-Ens-v2 selected an ensemble algorithm in all of its 25 runs. This was the case for the Abalone, Flags, Glass, KR-vs-KP, and Yeast datasets. By contrast, there was no dataset where PBIL-Auto-Ens-v2 selected a single base classifier (rather than an ensemble) in all 25 runs.

However, for 7 of the 21 datasets, PBIL-Auto-Ens-v2 selected a single base classifier in the majority of its runs. This was the case for the Adult, German Credit, Madelon, Semeion, Solar Flare1, Waveform and Wine datasets.

Among the single base classifiers returned as best solutions across all datasets, the most frequently selected ones were a Suport Vector Machine (SMO), Multilayer Perceptron (MLP), and a K-Nearest Neighbor algorithm (IBK), which were selected in 40, 35 and 30 cases, respectively. Overall, there is less variation in the selection frequency of single classifiers than in the selection frequency of the ensemble algorithms, since the other three single classifiers, namely the J48 decision tree, Naive Bayes (NB) and Decision Table (DCT), were selected in 25, 20 and 20 cases, respectively.

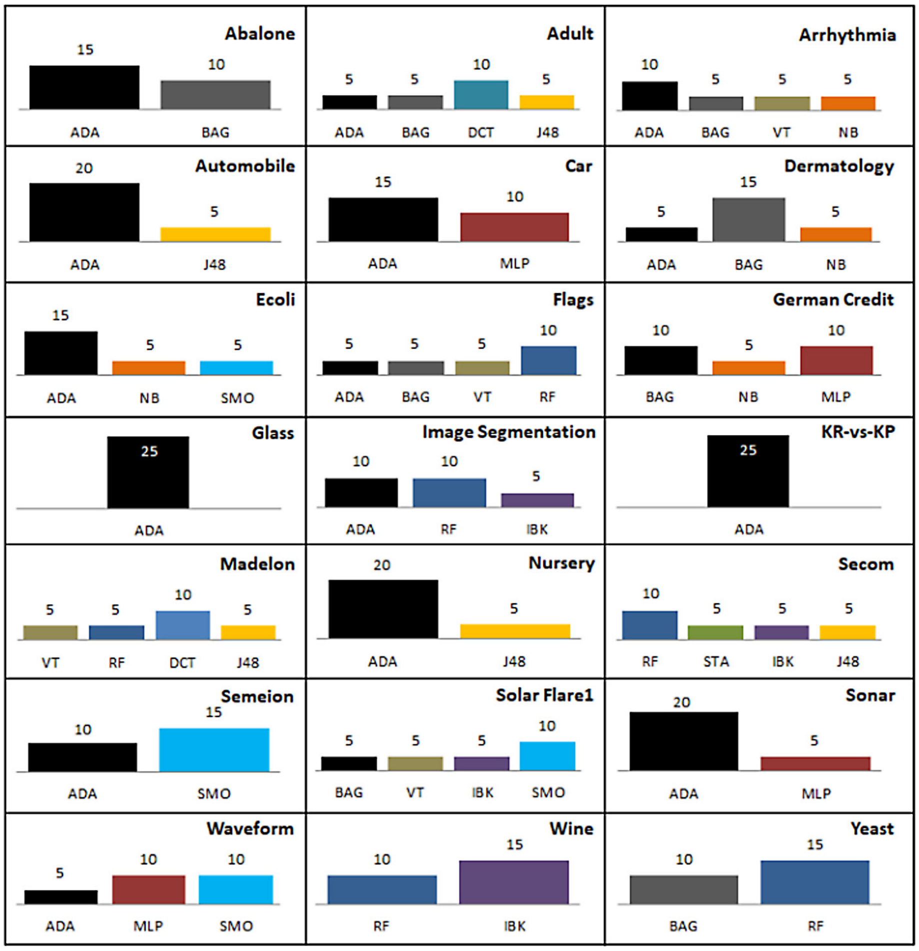

Fig. 4. The best algorithms returned by PBIL-Auto-Ens-v2 for each dataset. The classifiers' acronyms: ADA, BAG, RF, STA, VT, NB, MLP, SMO, IBK, DCT and J48 refer to AdaBoost, Bagging, Random Forest, Stacking, Vote, Naive Bayes, Multilayer Perceptron, Suport Vector Machine, K-Nearest Neighbors, Decision Table, and Decision Tree, respectively.

In summary, despite the overall dominance of three classifier ensembles (ADA, BAG, and RF), PBIL-Auto-Ens-v2 is exhibiting great flexibility in selecting the best classification algorithm for each dataset, which is the core motivation for Auto-ML.

7. Conclusion and future work

This work proposed a new version of our Estimation of Distribution Algorithm (a type of Evolutionary Algorithm), called PBIL-Auto-Ens-v2, for the Automated Machine Learning (Auto-ML) problem of automatically selecting the best classifier-ensemble method and its best hyper-parameter settings for an input dataset. PBIL-Auto-Ens-2 was compared against its previous version (PBIL-Auto-Ens-1) [28] and against two strong baseline methods: a random forest algorithm (a popular type of ensemble) and an adapted version of the well-known Auto-WEKA method [4] as an Auto-ML method. Both PBIL-Auto-Ens versions and the adapted Auto-WEKA version used the same search space of candidate solutions (focusing on classifier ensembles) and the same evaluation function to guide their search. Hence, the differences in their predictive accuracies reflect mainly their different search methods, as discussed earlier.

In experiments using 21 classification datasets, overall, the proposed PBIL-Auto-Ens-2 obtained better predictive performance than its previous version and the other two methods (Auto-WEKA and random forest) for all the four predictive accuracy measures used in our experiments, namely error rate, precision, recall and F-measure. In addition, PBIL-Auto-Ens-2 significantly outperformed Auto-WEKA for three out of the four measures (viz., error rate, recall and F-measure); and PBIL-Auto-Ens-2 also significantly outperformed random forest for three measures (precision, recall and F-measure).

We also analyzed the frequencies with which different types of classification algorithms were chosen as the best algorithm by the overall best Auto-ML method in our experiments (i.e., PBIL-Auto-Ens-v2). As the main result of this analysis, overall the three most frequently selected algorithms were Adaboost, Bagging and Random Forests, with Adaboost being the clear winner overall.

As a direction for future work, it would be interesting to extend the experiments to consider other Auto-ML methods, like the one recently proposed in [19]. Another future work direction would be to carry out experiments with much larger datasets, using more powerful computational resources.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work has been financially supported by the Brazilian Research Council (CNPq) under process number 150748/2017-5 and 407518/2018-5. We thank Mr. Douglas Felipe Alves Lima for helping code the PBIL-Auto-Ens-v2 algorithm.

References

[1] Zhi-Hua Zhou, Ensemble Methods: Foundations and Algorithms, 1st edition, Chapman & Hall/CRC, 2012.
[2] L.I. Kuncheva, Classifier ensembles for changing environments, in: F. Roli, J. Kittler, T. Windeatt (Eds.), Multiple Classifier Systems, in: Lecture Notes in Computer Science, vol. 3077, Springer, 2004, pp. 1–15.
[3] L.I. Kuncheva, Combining Pattern Classifiers: Methods and Algorithms, Wiley-Interscience, 2004.
[4] C. Thornton, F. Hutter, H.H. Hoos, K. Leyton-Brown, Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms, in: Proc. 19th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, ACM Press, 2013, pp. 847–855.
[5] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I. Witten, The WEKA data mining software: an update, ACM SIGKDD Explor. Newsl. 11 (1) (2009) 10–18.
[6] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, F. Hutter, Efficient and robust automated machine learning, Adv. Neural Inf. Process. Syst. 28 (2015) 2962–2970.
[7] P. Larrañaga, J.A. Lozano, Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation, Kluwer, Boston, MA, 2002.
[8] S. Baluja, R. Caruana, Removing the genetics from the standard genetic algorithm, in: Proc. 12th Int. Conf. on Machine Learning, California, July, 1995, 1995, pp. 38–46.
[9] M. Fernández-Delgado, E. Cernadas, S. Barro, D. Amorim, Do we need hundreds of classifiers to solve real world classification problems?, J. Mach. Learn. Res. 15 (1) (2014) 3133–3181.
[10] P. Brazdil, C. Giraud-Carrier, C. Soares, R. Vilalta, Metalearning: Applications to Data Mining, Springer, 2009.
[11] L. Kotthoff, C. Thornton, H.H. Hoos, F. Hutter, K. Leyton-Brown, Auto-WEKA 2.0: automatic model selection and hyperparameter optimization in WEKA, J. Mach. Learn. Res. 18 (1) (2017) 826–830.
[12] A.A. Freitas, Data Mining and Knowledge Discovery with Evolutionary Algorithms, Springer, 2002.
[13] A.E. Eiben, J.E. Smith, Introduction to Evolutionary Computing, 2nd edition, Springer, 2015.
[14] I. Inza, P. Larrañaga, B. Sierra, Feature subset selection by estimation of distribution algorithms, in: Estimation of Distribution Algorithms, Springer, 2002, pp. 269–293.
[15] K. Shelke, S. Jayaraman, S. Ghosh, J. Valadi, Hybrid feature selection and peptide binding affinity prediction using an EDA based algorithm, in: Proc. IEEE Congress on Evolutionary Computation (CEC), 2013, pp. 2384–2389.
[16] M. Zangari, R. Santana, A. Mendibury, A.T.R. Pozo, Not all PBILs are the same: unveiling the different learning mechanisms of PBIL variants, Appl. Soft Comput. 53 (April 2017) 88–96.
[17] X. Yang, H. Dong, H. Zhang, Naive Bayes based on estimation of distribution algorithms for classification, in: International Conference on Information Science and Engineering, 2009, pp. 908–911.
[18] Y. Saeys, S. Degroeve, D. Aeyels, P. Rouzé, Y. Van de Peer, Feature selection for splice site prediction: a new method using EDA-based feature ranking, BMC Bioinform. 5 (64) (2004), 11 pages.
[19] P. Kordík, Jan Černý, T. Frýda, Discovering predictive ensembles for transfer learning and meta-learning, Mach. Learn. 107 (1) (2018) 177–207.
[20] S.B. Kotsiantis, Supervised machine learning: a review of classification techniques, in: Emerging Artificial Intelligence Applications in Computer Engineering, IOS Press, 2007, pp. 3–24.
[21] M. Wistuba, N. Schilling, L. Schmidt-Thieme, Automatic frankensteining: creating complex ensembles autonomously, in: Proc. SIAM Int. Conf. on Data Mining, SIAM, 2017, pp. 741–749.
[22] J. Lévesque, C. Gagné, R. Sabourin, Bayesian hyper-parameter optimization for ensemble learning, in: Proc. 32nd Conference on Uncertainty in Artificial Intelligence (UAI), Jersey City, New Jersey, USA, 2016, 2016, pp. 437–446.
[23] A. Lacoste, H. Larochelle, F. Laviolette, M. Marchand, Sequential model-based ensemble optimization, Computing Research Repository (CoRR) (2014).
[24] R. Olson, R. Urbanowicz, P. Andrews, N. Lavender, L. Kidd, J.H. Moore, Automating biomedical data science through tree-based pipeline optimization, in: European Conference on the Applications of Evolutionary Computation, Springer, 2016, pp. 123–137.
[25] A.G.C. de Sá, G.L. Pappa, A.A. Freitas, Automated selection and configuration of multi-label classification algorithms with grammar-based genetic programming, in: Proc. of the 15th International Conf. on Parallel Problem Solving from Nature (PPSN-2018), to be Held in Coimbra, Portugal, Sep. 2018, 2018, https://doi.org/10.1007/978-3-319-99259-4_25, in press.
[26] A.G.C. de Sá, W.J.G.S. Pinto, L.O.V.B. Oliveira, G.L. Pappa, RECIPE: a grammar-based framework for automatically evolving classification pipelines, in: Proc. of the 20th European Conference on Genetic Programming (EuroGP'17), in: LNCS, vol. 10196, Springer, 2017, pp. 246–261.
[27] Janez Demsar, Statistical comparisons of classifiers over multiple data sets, J. Mach. Learn. Res. 7 (2006) 1–30.
[28] J.C. Xavier-Júnior, A.A. Freitas, A. Feitosa-Neto, T.B. Ludermir, A novel evolutionary algorithm for automated machine learning focusing on classifier ensembles, in: 7th Brazilian Conference on Intelligent Systems (BRACIS 2018), 2018, pp. 462–467.
[29] L. Rokach, Ensemble-based classifiers, Artif. Intell. Rev. 33 (1–2) (2010) 1–39.
[30] S.B. Kotsiantis, Bagging and boosting variants for handling classification problems: a survey, Knowl. Eng. Rev. 29 (1) (2014) 78–100.
[31] O. Sagi, L. Rokach, Ensemble learning: a survey, Wiley Interdiscip. Rev. Data Min. Knowl. Discov. 8 (4) (2018) e1249.