Decision Tree Pruning via Multi-Objective Evolutionary Computation

Andrea Brunello, Enrico Marzano, Angelo Montanari, and Guido Sciavicco

***Abstract*—To date, decision trees are among the most used classification models. They owe their popularity to their efficiency during both the learning and the classification phases and, above all, to the high interpretability of the learned classifiers. This latter aspect is of primary importance in those domains in which understanding and validating the decision process is as important as the accuracy degree of the prediction. Pruning is a common technique used to reduce the size of decision trees, thus improving their interpretability and possibly reducing the risk of overfitting. In the present work, we investigate on the integration between evolutionary algorithms and decision tree pruning, presenting a decision tree post-pruning strategy based on the well-known multi-objective evolutionary algorithm NSGA-II. Our approach is compared with the default pruning strategies of the decision tree learners C4.5 (J48 - on which the proposed method is based) and C5.0. We empirically show that evolutionary algorithms can be profitably applied to the classical problem of decision tree pruning, as the proposed strategy is capable of generating a more variegate set of solutions than both J48 and C5.0; moreover, the trees produced by our method tend to be smaller than the best candidates produced by the classical tree learners, while preserving most of their accuracy and sometimes improving it.**

***Index Terms*—Data mining, decision trees, evolutionary computation, pruning methodologies.**

I. Introduction

As it is commonly recognized, decision trees have a predominant position among classification models [1]. This is mainly due to the facts that (1) they can be trained and applied efficiently even on big datasets and (2) they are easily interpretable. Thanks to the latter feature, they turn out to be useful not only for prediction, but also for highlighting relevant patterns or regularities in the data. This is extremely beneficial in those application domains where understanding the classification process is at least as important as the accuracy of the prediction itself.

A typical decision tree is constructed recursively, starting from the root, following the traditional *Top Down Induction of Decision Trees* (TDIDT) approach: at each node the attribute that best partitions the training data, according to a predefined score, is chosen as a test to guide the partitioning of instances into child nodes, and the process continues until a sufficiently high degree of purity (with respect to the target class), or a minimum cardinality constraint (with respect to the number of instances reaching the node), is achieved in the generated partitions. A decision tree induced by the TDIDT approach tends to *overgrow*, and this leads to a loss in interpretability as well as to a risk of *overfitting* training data, that results in capturing unwanted noise. As a direct consequence, such trees typically do not perform well on new, independent instances, since they fit the training data "too perfectly".

In order to simplify the tree structure, thus making the trees more general, *pruning* methods are typically applied. According to the time when such an operation is performed, we may distinguish among: (1) *pre-pruning*, consisting of interrupting the construction of the decision tree according to a stopping criterion such as minimum node cardinality, or when none of the attributes leads to a sufficiently high splitting score, and (2) *post-pruning*, that is, building the entire tree first, and then removing or condensing some parts of it. While pre-pruning has the advantage of not requiring the construction of the whole tree, it usually leads to worse accuracy results than post-pruning [2].

In this paper, we focus on post-pruning approaches. There are two main strategies for evaluating the error rate in this setting. The first one consists of keeping part of the training data as an independent *hold out set* (and, thus, working on three, independent, datasets: training, hold out, and test), and deciding whether to prune a section of the tree or not on the basis of the resulting classification error on it. Examples of such techniques include a variant of CART's *Cost-Complexity Pruning* [3] and the so-called *Reduced-Error Pruning* [4]. It should be noticed that splitting data in three different partitions reduces the amount of labelled instances available for training which, in some cases, are already scarce. The second strategy, on the contrary, focuses on estimating the *apparent* classification error due to pruning on the basis of the training data only, as in, for instance, *Pessimistic Error Pruning* [4] and C4.5's *Error-Based Pruning* [2].

From a computational point of view, it is known that the problem of constructing an optimal binary decision tree is NP-Complete [5]. The result is that all practical implementations of TDIDT algorithm and pruning methodologies are based on heuristics, that typically have a polynomial complexity in the number of instances and features in the training set.

In the following, we pursue an approach to the post-pruning of decision trees based on *Evolutionary Algorithms* (EAs), observing that such a problem can be considered as a search in the space of possible subtrees [6]. In fact, EAs have already been successfully applied to various phases of the decision tree induction process (see, for

Manuscript received September 30, 2017; November 15, 2017.

A. Brunello and A. Montanari are with the University of Udine, Udine, Italy (e-mail: andrea.brunello@uniud.it, angelo.montanari@uniud.it).

E. Marzano is with Gap SRL Company, Udine, Italy (e-mail: e.marzano@gapitalia.it).

G. Sciavicco is with the University of Ferrara, Ferrara, Italy (e-mail: guido.sciavicco@unife.it).

doi: 10.18178/ijmlc.2017.7.6.641

instance, [7]). Despite of that, to the best of our knowledge, the integration between evolutionary algorithms and decision tree pruning has not been investigated in detail yet. Even among recent works evolutionary computation has not been taken into account, see for example [8], [9].

The only proposed EA for classical, that is, not *oblique* [10], decision tree post-pruning is Chen and al.'s single-objective algorithm [11]. In that work, the fitness function is given by a weighted sum of the number of nodes in the tree and the error rate. Oddly enough, the latter is estimated directly on the same test dataset also used to evaluate the accuracy of the final solution. As pointed out in [7], this constitutes a serious methodological mistake, since the test set should only be used to assess the validity of the finally generated tree, and the training set, or an independent pruning set, should have been used in evaluating the fitness of individuals during EA computation.

In this paper, we correct and extend the approach outlined in [11], making use of the well-known, elitist, multi-objective evolutionary algorithm NSGA-II [12]. We design a post-pruning strategy that optimizes *two objectives*: the accuracy of the obtained tree (on the training dataset) and the number of its nodes. We compare our approach with the default post-pruning methodologies of both the algorithms J48/C4.5 [2] (on which our method is built) and C5.0 [13]. In both cases (EA-based pruning and default pruning strategies), a third hold-out set is not necessary: this makes our comparison easier and, of course, it is advantageous for those cases in which training instances are scarce.

The paper is organized as follows. Section II gives a short account of the main methodologies and concepts used through the article. Section III presents the proposed approach in detail. Section IV is devoted to the experimental analysis of the achieved solution. Finally, the obtained results are discussed in Section V. Conclusions and future work are outlined in Section VI.

II. Background

In this section, we present the main methodologies and concepts used in the paper.

A. The Decision Tree Learner J48 (C4.5)

J48 is the *Weka* [1] implementation of C4.5 [2], which, to date, is probably the single most used machine learning algorithm (note that the terms J48 and C4.5 will be used interchangeably in the remainder of the paper). C4.5 is known to provide good classification performances, to be computationally efficient, and to guarantee the interpretability of the generated model.

In short, C4.5 recursively builds a decision tree from a set of training instances by using the *Information Gain* and *Gain Ratio* criteria, both based on the concept of *Shannon Entropy*. Starting from the root, at each node C4.5 chooses the data attribute that most effectively splits the set of samples into subsets, with respect to the class labels. The process continues until the sample reaches a minimum number of instances, or when no attribute proves to be useful in splitting the data. In such cases, the corresponding node becomes a leaf of the tree. After the tree growing phase, two default, independent, methodologies are typically employed to reduce the size of the generated model: *Collapsing* and *Error-Based Pruning* (EBP), which are usually employed together.

Collapsing a tree can be seen as a special case of pruning, in which parts of the tree that do not improve the classification error on the training data are discarded. For example, given a node N that roots a subtree in which all leaves predict the same class C, the entire subtree can be collapsed into the node N that becomes a leaf predicting C.

EBP is based on a different idea. Since the decision tree error rate on the training set is biased by the learning phase, it does not provide a suitable estimate for the classification performance on future cases. Intuitively, EBP consists of systematically evaluating each node N and deciding, using statistical confidence estimates, whether to prune the subtree rooted in N or not. Since pruning always worsens the accuracy of the tree on the training dataset (or leaves it unchanged), EBP evaluates the effect of a pruning by trying to correct the bias, that is, estimating the true error that would be observed on independent data.

In essence, given a node covering n training instances, e of which misclassified, the *observed error rate* $f = e/n$ is calculated; then, the method tries to estimate the *true error rate* p that would be observed over the entire population of instances ever reaching that node, based on several additional hypothesis, among which assuming a *binomial distribution* for the error.

This solution gives rise to a simple method to control the pruning aggressiveness consisting in suitably varying the binomial *confidence intervals* [14], but, at the same time, it has been criticized for the lack of a proper statistical foundation.

Also, it has been observed to have a tendency for under-pruning, especially on large datasets [15].

B. The Decision Tree Learner C5.0

C5.0 (also known as *See5*) is an updated, commercial version of C4.5, reported to be much more efficient than its predecessor in terms of memory usage and computation time [16]. Moreover, the resulting trees tend to be smaller and more accurate than those generated by C4.5 [17]. The learning algorithm follows a similar TDIDT strategy as its predecessor, relying on information gain and gain ratio scores to partition the training instances. The pruning is based on an EBP-like strategy, complemented by an optional *global pruning* step. Other important characteristics of C5.0 include the possibility of generating an ensemble of trees through *boosting*, the integration of an attribute selection strategy, called *winnowing*, the support for *asymmetric costs* for different kinds of error, *soft-thresholds* for numeric attributes, splitting on *value subsets* for discrete attributes, and a *multi-threaded* architecture [13]. A single-threaded version of C5.0 is available under the Gnu GPL (*https://www.rulequest.com/download.html*).

C. Evolutionary Algorithms

Evolutionary Algorithms (EAs) are adaptive meta-heuristic search algorithms, inspired by the process of natural selection, biology, and genetics. Unlike blind random search, they are capable of exploiting historical information to direct the search into the most promising regions of the search space and, in order to achieve that, their basic characteristics are designed to mimic the processes that in natural systems lead to adaptive evolution. In nature, a population of individuals tends to evolve, in order to adapt to the environment; in EAs, each individual represents a

possible solution for the optimization problem, and its degree of "adaptation" to the problem is evaluated through a *fitness* function, which can be single or multi-objective.

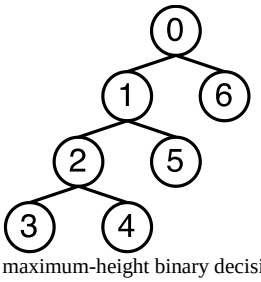

Fig. 1. A maximum-height binary decision tree.

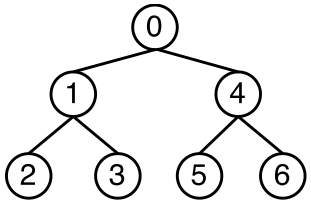

Fig. 2. A balanced and complete binary decision tree.

The elements of the population iteratively evolve toward better solutions, going through a series of generations. At each generation, the individuals which are considered best by the fitness function are given a higher probability of being selected for reproduction; the *selection strategy* mainly distinguishes one particular meta-heuristic from another.

NSGA-II, on which our method is based, uses a Pareto-based multi-objective ($\lambda + \mu$) strategy with a *binary tournament selection* and a *rank crowding better* function [18]. To the selected individuals, operations such as *crossover* and *mutation* are applied, with the goal of generating new offspring, creating a new generation of solutions. The iterations stop when a predefined criteria is satisfied, which can be a bound on the number of iterations, or a minimum fitness increment that must be achieved between subsequent generations.

Multi-objective EAs are designed to solve a set of minimization/maximization problems for a tuple of n functions $f_1(\vec{x}), \ldots, f_n(\vec{x})$, where $\vec{x}$ is a vector of parameters belonging to a given domain. A set *F* of solutions for a multi-objective problem is said to be *non-dominated* (or *Pareto optimal*) if and only if for each $\vec{x} \in F$, there exists no $\vec{y} \in F$ such that (1) $f_i(\vec{y})$ improves $f_i(\vec{x})$ for some i, with $1 \leq i \leq n$, and (2) for all $j, 1 \leq j \leq n, j \neq i, f_j(\vec{x})$ does not improve $f_j(\vec{y})$. The set of non-dominated solutions from *F* is called *Pareto front*.

Multi-objective approaches are particularly suitable for multi-objective optimization, as they search for multiple optimal solutions in parallel. Such algorithms are able to find a set of optimal solutions in the final population in a single run, and once such a set is available, the most satisfactory one can be chosen by applying a preference criterion. In our case, we propose a system that optimizes, together, the accuracy on the training dataset of a pruned tree and the number of its nodes, as well as an *a posteriori* decision method to choose the best pruned tree in the resulting Pareto front.

D. Complexity of the Decision Tree Pruning Problem

Let us now focus our attention on the pruning problem viewed as a search problem. We are interested in establishing suitable lower and upper bounds to the search space and, to this end, we restrict our attention to *binary trees*, which makes it simpler to compute the bounds. More general lower and upper bounds can then be easily derived. Notice that binary decision trees are always *full*, that is, each node has zero or two children, and thus the pruning of a full binary decision tree, for any given internal node, either removes both subtrees or maintains both of them.

The search space consists of all the different pruned trees that can be obtained from a fully-grown tree, that is, from the tree generated by the TDIDT recursive procedure.

Let n be the number of nodes of the given (fully-grown) tree. A lower bound on the cardinality of the search space is given by the number of pruned trees that can be obtained from the highest full binary decision tree that can be generated with n nodes at our disposal. Consider the case with $n = 7$. The height of the highest full binary decision tree with 7 nodes is $h = 3$ (see Fig. 1). The number of distinct pruned trees that can obtained from it is 4: the complete tree, the one obtained by deleting nodes 3 and 4, the one obtained by deleting nodes 2, 3, 4, and 5; and the one consisting of the root 0 only. In general, if the height of the highest full binary decision tree is h, the number of its distinct pruned trees is $h + 1$, i.e., $O(h)$.

An upper bound on the cardinality of the search space is given by the number of pruned trees that can be obtained from the *perfect* binary tree of height h, whose levels are all complete (see Fig. 2). In such a case, the number of pruned trees can be determined by a recursive formula:

$$f(h) = \begin{cases} 1 & \text{if } h = 0; \\ f(h-1)^2 + 1 & \text{otherwise.} \end{cases}$$

An upper bound on the cardinality of the search space is thus $O(f(h)) = \Omega(2^h)$, which is a function that grows very fast.

As an example, we have that:

$$f(0) = 1 \qquad f(4) = 677$$
$$f(1) = 2 \qquad f(5) = 458.330$$
$$f(2) = 5 \qquad f(6) = 210.066.388.901$$
$$f(3) = 26 \qquad f(7) = 4,412788775 * 10^{22}$$

The above formula can be easily generalized to cope with non-perfect, non-binary, and, therefore, non-full, decision trees. Let *N* be the root of the given tree, let $f'(N)$ be the function that computes the number of its pruned trees (we identify the tree with its root), and let $C(N)$ be the set of all children of the node N. The function f', that generalizes f, can be defined as follows:

$$f'(N) = \begin{cases} 1 & \text{if } N \text{ is a leaf;} \\ 1 + \prod_{M \in C(N)} f'(M) & \text{otherwise.} \end{cases}$$

Clearly, both f and f' grow too fast to allow a systematic search of the space of all the pruned trees. To see how the number of solutions may grow in real-world cases, consider, as an example, the decision tree generated by applying Weka's J48 on the UCI's *Soybean* benchmark dataset (683 instances, 36 attributes) [1], disabling pruning and collapsing operations (weka.classifiers.trees.J48 -O -U -M 2). The resulting model has 207 nodes (141 leaves) and, according to the previous formula, it admits 131.165.197.804 possible

pruned trees. For the sake of comparison, the default EBP pruned version has 93 nodes (61 leaves).

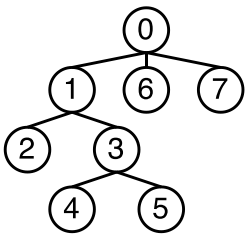

Fig. 1. A fully-grown decision tree with nodes labeled in a pre-order fashion.

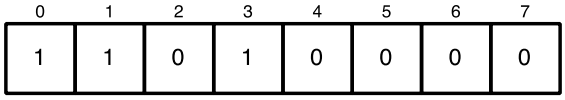

Fig. 2. The gene corresponding to the tree in Fig. 3, representing a solution.

III. EA-Based Pruning

In this section, we present a novel, *wrapper-based* approach to the pruning of decision trees. In wrapper-based pruning, a search algorithm explores the search space of all possible pruned trees that can be obtained from a single, unpruned and un-collapsed decision tree, and potential candidates are evaluated step-by-step. We chose to implement our method with J48, the Weka implementation of the algorithm C4.5 and with the evolutionary search algorithm known as NSGA-II (see Section II).

A. Representation of Solutions and Initial Population

Given a training dataset, a fully-grown, un-collapsed and unpruned J48 decision tree is first built. Each solution to the search problem is conveniently represented as a binary array, whose size is equal to the number of nodes of the original J48 tree given as input. Each cell of the arrays tracks whether the subtree rooted at the specific node of the tree is being kept or not. In order to establish a correspondence between the tree and the array, nodes are numbered according to a pre-order visit of the tree. As an example, the tree represented in Fig. 3 corresponds to the binary array of Fig. 4. The decision tree obtained by removing the subtree rooted at node 1 and, consequently, the subtree rooted at node 3, is represented in Fig. 5. Its corresponding gene representation is shown in Fig. 6. Basically, each gene keeps track of the subtree status with respect to the original, fully-grown decision tree.

The *initial population* has been generated with the following schema to ensure both its correctness and its heterogeneity.

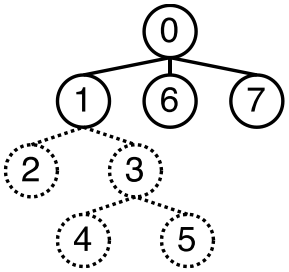

Fig. 3. The pruned decision tree, with nodes labeled in a pre-order fashion.

A binary array corresponding to a particular solution is initially set to represent the fully-grown tree. Then, a "non-pruning probability threshold" t is established (empirical evaluation suggested a random value in the range $[0.85,0.95]$ for the threshold). For each cell of the array, we proceed as follows. The distance d of the corresponding node from the root of the tree is computed (starting from 1), and, then, d random values are generated: if at least one of them is greater than t, the subtree corresponding to the array cell is pruned, that is, the cell is set to 0.

The idea behind such a generation strategy is that pruning at higher levels should be more difficult than pruning at lower levels, as pruning a shallow node removes most of the tree. Obviously, pruning operations must be carried out in such a way that the resulting tree is a valid tree: if the subtree rooted at node N is removed, then all subtrees rooted at descendants of N must be removed as well.

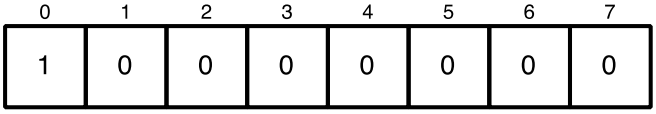

Fig. 6. The gene representing the pruned tree of Fig. 5.

B. Operators

We use the classical EA operators crossover and mutation.

a) *Crossover:* Given two parent solutions, two children solutions are generated via *crossover* by simply performing a pairwise AND and a pairwise OR of the two corresponding binary arrays. Thus, one child will contain the subtrees which are in common between the two parents, whereas the other one will contain the union of the subtrees of the two parents. Correctness of the generated solutions straightforwardly follows from the correctness of the parents.

b) *Mutation:* Given a binary array of length l corresponding to a solution, mutation is carried out as follows. A random number n, $1 \leq n \leq l$ is generated, and n random flips are carried out in the binary array, preserving the correctness of the generated solution: if the subtree rooted at N is removed, then all subtrees rooted at descendants of N must be removed as well, and if the subtree rooted at node N is restored, then all subtrees rooted at ancestors of N must be restored as well.

C. Fitness Functions

We used two fitness functions in order to optimize two objectives. The first objective is to maximize the accuracy of the pruned tree on the training dataset and, to this end, we used the standard evaluation method provided by J48. The second objective is to minimize the size of the pruned tree, and we used a simple function to count the number of nodes, also provided by Weka. The two objectives are clearly antithetical: pruning a tree may possibly reduce the accuracy of it on the training set, but it never increases it.

D. Decision Method

As it happens with any other multi-objective optimization algorithm, the result of our post-pruning method is a set of non-dominated solutions. To evaluate the quality of the proposed method, we compare the solutions it returns with the solutions provided, on the same training datasets, by C4.5 and by C5.0. Thus, an *a posteriori* decision-making method must be designed to select a single solution (or a subset of solutions) in a systematic and controllable way. The proposed strategy is inspired by the *Minimum Description Length* (MDL) principle (see, e.g., [19]), which is based on the

assumption that any regularity in a given dataset can be exploited to compress the data. More specifically, each non-dominated tree can be considered as a *theory*, which can be used to "explain" the training data. Each theory will have a related (possibly empty) set of *exceptions*, which are not "captured" by the model. Thus, we can define the *coding cost* of a candidate solution as the coding cost of the theory (in the present case, the relative size of the tree with respect to the original one) plus the coding cost of the exceptions, that is, the error rate of the tree, calculated over the entire training set. Both values, denoted here by $SIZE$ and ER, respectively, belong to interval $[0,1]$, and they can be arranged into a weighted sum as follows:

$$W * ER + (1 - W) * \text{SIZE},$$

where $W \in [0, 1]$ is a weight which can be modified by the user in order to vary the pruning aggressiveness. The candidate solution with the *lowest combined value* is selected. Intuitively, a large value of W tends to favor larger, but more accurate, models, as far as the training set is concerned, whereas smaller W values should result in more general trees being selected. Intentionally, the weight W plays a similar role as the *confidence factor* in C4.5 and C5.0, and this makes it possible to better compare the results.

TABLE I: THE UCI DATASETS UNDER STUDY

Dataset	#inst	#preds	Preds Type	#nodes unpr
Adult	48842	14	cat, num	31145
Bank	41188	20	num	6820
Breast W	699	9	num	53
Credit Card	30000	23	num	5613
Credit G	1000	20	cat, num	433
Diabetes	768	8	num	69
Eye State	14980	14	num	1579
Labor	57	16	cat, num	16
Sonar	208	60	num	29
Spambase	4601	57	num	361
Voting	435	16	cat	51
Waveform 5k	5000	40	num	555

IV. EXPERIMENTAL RESULTS

In this section, we provide an experimental comparison among three post-pruning approaches, that is, the standard C4.5 and C5.0 EBP pruning and our wrapper-based EA for post-pruning. All experiments have been carried out on an Intel Core i5 processor running at 2.4 GHz, equipped with a main memory of 8 GB.

A. Datasets

We have used 12 standard UCI datasets (available at the website https://archive.ics.uci.edu/ml/datasets.html), which have been selected in order to maximize the variability in terms of number of instances as well as number and types of attributes. As witnessed by the recent literature, UCI datasets are a standard *de facto* in the machine learning community (see, e.g., [11], [14], [15], [20]). The chosen datasets are detailed in Table I, where, for each case, we show the number of nodes in the respective un-collapsed and unpruned J48 decision tree. The kind of attributes (categorical, numeric) is also displayed.

B. Methods

The experiment phase has been designed as follows. Each dataset has been partitioned into a training set (75%) and a test set (25%), according to a stratified approach. Then, on each dataset, the three methods, namely, J48, C5.0, and EA-based, for post-pruning have been applied. As for C4.5 and C5.0, we followed an approach similar to the one adopted in [14]. In particular, we trained a set of 27 decision trees for each dataset by varying the confidence factor over in the range $[0.001, 0.49]$ (this is the widest implemented range). We tested all values from 0.1 to 0.49 (included) by steps of 0.02, other than the very low values 0.005 and 0.001 — the lower the confidence factor, the more aggressive the pruning performed. For the sake of comparison, recall that Weka's default confidence factor is 0.25. Collapsing option has also been activated. Moreover, in the case of C5.0, trees have been generated non-bagged, winnowing had been disabled, while global pruning and soft thresholds remained active. On the other hand, the EA-based experiment has been designed as follows. On each dataset we executed 30 independent runs, each with a different seed value. This led to 30 sets of non-dominated solutions which have been merged into a final, single set (from which all dominated solutions have been eliminated). Finally, for each dataset we empirically assessed the minimum amount of evaluations, in the range $[10000, 100000]$, that are necessary to reach a satisfactory solution (given the population size of 100, 10000 evaluations correspond to 100 evolution steps). Table II summarizes the number of evaluations, and the required computation time, for each independent run over the considered datasets.

TABLE II: EVALUATIONS NUMBER AND COMPUTATION TIME PER DATASET

Dataset	# evaluations	Computation time (sec)
Adult	75000	3200
Bank	100000	720
Breast W	10000	< 1
Credit Card	60000	400
Credit G	100000	46
Diabetes	10000	1
Eye State	100000	188
Labor	10000	< 1
Sonar	10000	< 1
Spambase	100000	50
Voting	10000	1
Waveform 5k	60000	37

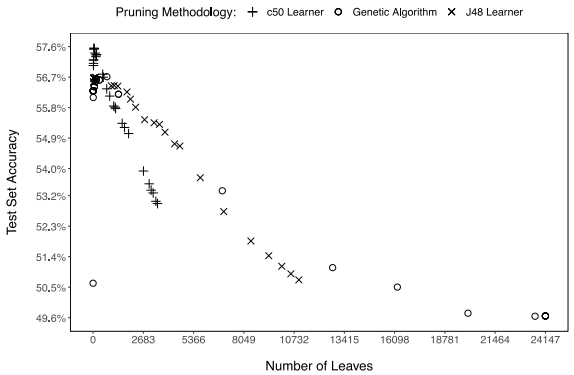

Fig. 7. Results on the Adult dataset.

C. Results

For each dataset, we compared the results of the three post-pruning methods. The results are shown in a number of figures (from Fig. 7 to Fig. 18). Each graph shows the relation that emerges between the predictive accuracy and the size

(number of leaves) of each tree produced by a specific setting of the parameter that governs the pruning aggressiveness (confidence interval for C4.5 and C5.0, weight W for the EA-based wrapper), generating three curves for each dataset. Each point in the graphs represents a model. Even if we did not focus on learning more precise trees than classical methods, in 7 out of 12 cases, the EA-based pruning produced at least one tree with equal or better predictive accuracy than the best tree produced by C4.5 or by C5.0, regardless its size. These are: *Breast W*, *Credit Card*, *Diabetes*, *Labor*, *Spambase*, *Sonar*, and *Voting*.

In the case of *Breast W* (Fig. 9) the EA-based pruning generated a tree with 20 leaves and 97.7% accuracy (best overall accuracy), and the smallest tree generated by classical methods has 5 leaves and an accuracy of 94.2%, while the wrapper has been able to produce a tree with only 2 leaves and an accuracy of 93.7%.

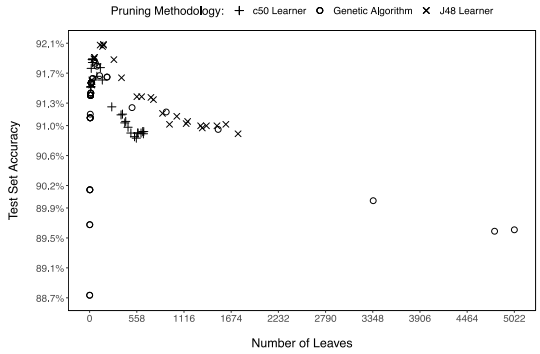

Fig. 8. Results on the Bank dataset.

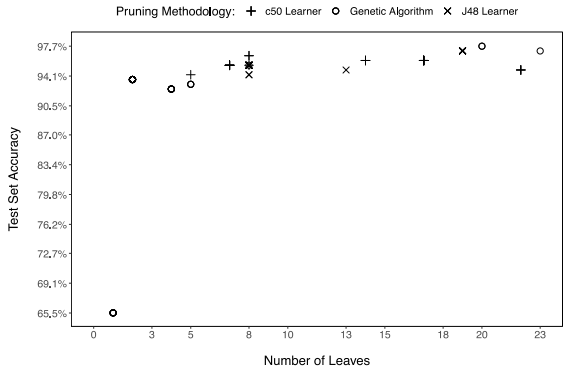

Fig. 9. Results on the Breast W dataset.

In the case of the dataset *Credit Card* (Fig. 10), we produced a tree with 24 leaves and 82.2% accuracy (best overall accuracy), as well as the same smallest result as C5.0 (2 leaves, 81.7% accuracy), and we surpassed the smallest tree generated by J48 (7 leaves and 81.8% accuracy).

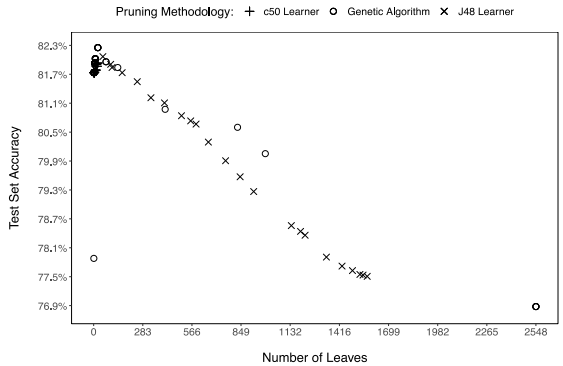

Fig. 10. Results on the Credit Card dataset.

As for *Diabetes* (Fig. 12), the EA-based pruning matches the best result obtained by J48 (3 leaves and 75.5% accuracy), and proposes a smaller but almost as much as accurate tree (2 leaves and 75.0% accuracy), while C5.0 is capable of achieving the same result but with a (slightly) bigger tree (3 leaves, 75.0% accuracy).

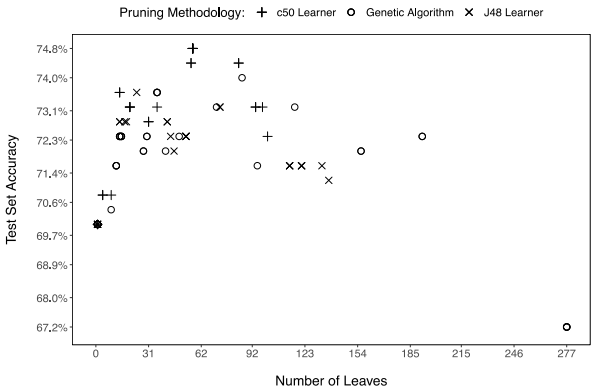

Fig. 11. Results on the Credit G dataset.

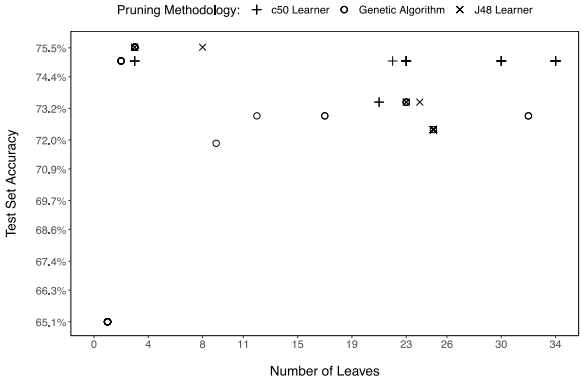

Fig. 12. Results on the Diabetes dataset.

In the case of *Labor* (Fig. 14), J48, C5.0 and the EA-based wrapper are all capable of generating the best tree (2 leaves and 85.7% accuracy), while in the case of *Sonar* (see Fig. 15), the wrapper constantly surpasses J48 in terms of classification performance, and it is also capable of achieving the same accuracy result of C5.0 (78.4%) with a slightly smaller tree (9 vs 11 leaves).

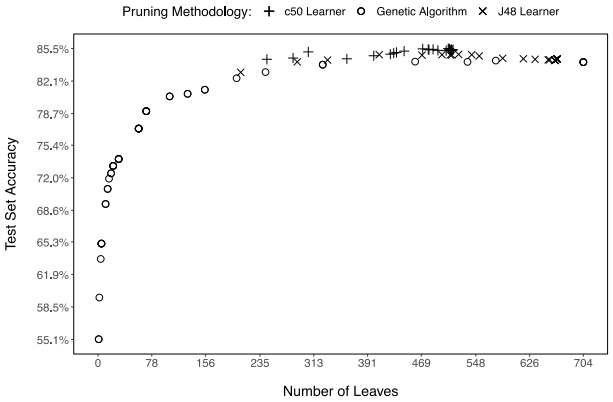

Fig. 13. Results on the eye state dataset.

As for *Spambase* (Fig. 16), we observe that the results obtained by the three approaches tend to be quite similar; however, the wrapper is capable of generating the best tree (141 leaves and 92.7% accuracy) as well as a very small

(smaller than those proposed by J48 and C5.0) yet accurate enough tree (12 leaves, 89.6% accuracy).

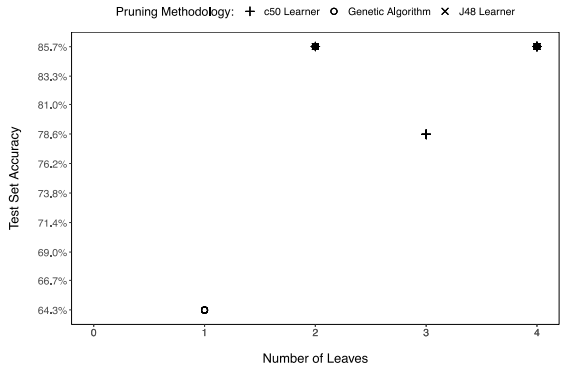

Fig. 14. Results on the Labor dataset.

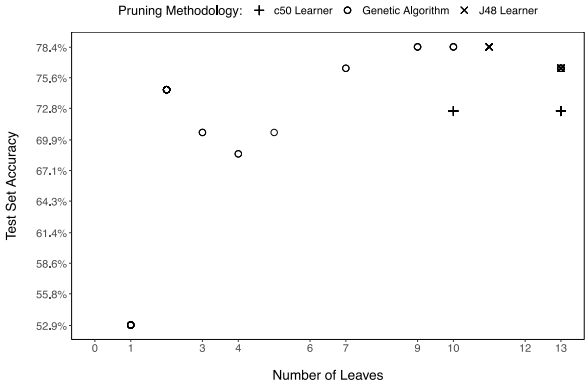

Fig. 15. Results on the Sonar dataset.

Finally, in the case of *Voting* (Fig. 17), the three methods all produce the best (and smallest) tree (2 leaves and 96.3% accuracy).

In the case of the dataset *Adult* (Fig. 7), the best accuracy is obtained by C5.0; however, while the most accurate model generated by J48 has 144 leaves and 56.7% accuracy, its smallest tree has 17 leaves and 56.5% accuracy. In comparison, the wrapper produced a pruned tree with 10 leaves, while still retaining an accuracy of 56.1%. Given the exceptionally big unpruned tree, probably, a better result would be obtained with more EA evaluations.

As for the dataset *Bank* (Fig. 8), even if the wrapper is surpassed by both J48 and C5.0 in terms of accuracy, it is still capable of generating a pruned tree with 7 leaves, while still retaining a test set accuracy of 91.1%, and a tree with only 3 leaves, keeping an accuracy of 90.1%. This contrasts with the smallest tree generated by J48 (25 leaves and 91.5% accuracy) and the one produced by C.50 (13 leaves and 91.2% accuracy.

In the case of *Eye State* (Fig. 13), while the wrapper does not reach the same accuracy as C4.5 and C5.0, it is able of providing a variety of much smaller trees, without loosing too much accuracy.

Finally, in the case of *Credit G* (Fig. 11), the wrapper achieved an overall higher accuracy than J48 (74.0% vs 73.6%), but lower than C5.0 (74.8%), while in the case of

Waveform 5k (Fig. 18) the wrapper produced more accurate trees than J48, and, while the most accurate model is generated by C5.0 (120 leaves and 77.9% accuracy), the wrapper generated two smaller, yet accurate enough, models (one with 76 leaves and 77.7% accuracy and the other with 39 leaves and 75.4% accuracy).

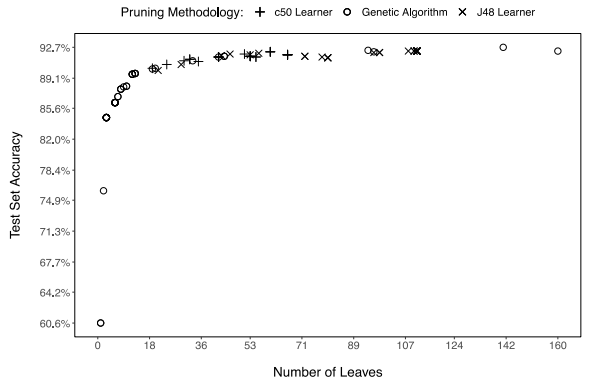

Fig. 16. Results on the Spambase dataset.

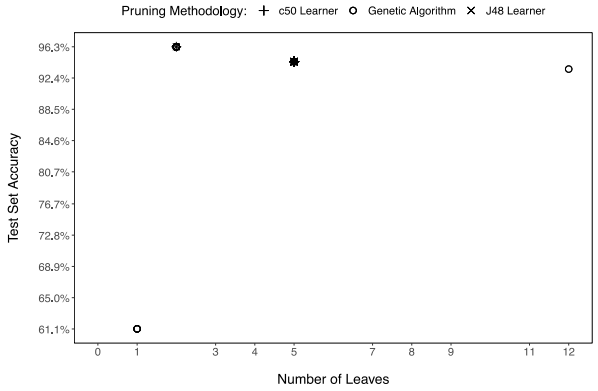

Fig. 17. Results on the Voting dataset.

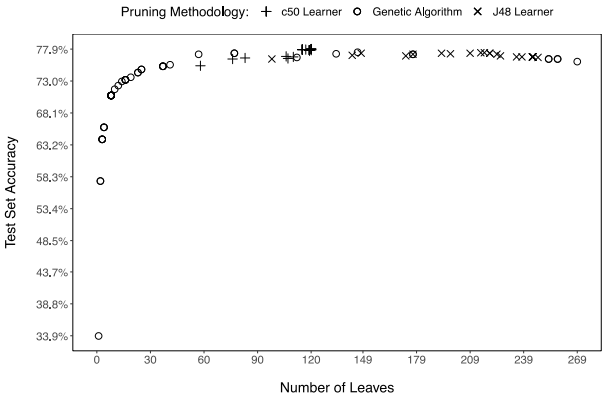

Fig. 18. Results on the Waveform 5k dataset.

V. Discussion

Results reported in Section IV clearly show that the proposed EA approach to pruning decision trees is capable of matching, and sometimes surpassing, the performances of classical C4.5 and C5.0 strategies in terms of the size-to-accuracy ratio of the generated trees. In particular, the proposed method is capable of producing a more variegate set of solutions, often characterized by smaller trees, which, nevertheless, preserve most of the accuracy of those traditionally pruned. The proposed EA-based pruning approach starts from a fully-grown C4.5 decision tree. Given that, as reported in [17], the trees grown by C5.0 tend to behave better than those grown by C4.5, we may speculate that better results could be achieved by applying our method to the former instead of the latter. Moreover, there are several EA-related aspects that could be taken into account to enhance our results. First, the classical selection strategy implemented in NSGA-II has been improved in, for instance,

the algorithm ENORA [21], [22]. Second, independently from the selection strategy, state-of-the-art implementations of EAs do not require explicit and fixed setting of the crossover and mutation rates, which are, instead, considered as characteristics of adaptiveness of each of the solutions. Empirically, adaptation has been observed to better the performance of traditional operators in terms of convergence to the Pareto optimal front and in diversity of the final solutions. Finally, in the last few years, the use of fitness functions is being progressively substituted by convergence directly based on the hypervolume, which seems to behave better than traditional fitness-based methods [23], and it would be interesting to understand its effects on our method as well. Overall, this entire work should be considered as a worthy proof-of-concept of the idea of EA-based post-pruning, which deserves some further investigation.

Although EAs have been extensively applied in the past to various phases of decision tree induction [7], previous to our work, to the best of our knowledge, only Chen et al. [11] approach has used evolutionary algorithms for the purpose of pruning (non-oblique) decision trees. Their method and the present one differ from each other in various nontrivial aspects. First of all, Chen's pruning is applied to a fully-grown ID3 decision tree, which is a predecessor of C4.5 [24]. Concerning the representation of the solutions, our encoding is similar to theirs, except for the fact that we have chosen to denote nodes, that is, subtrees, instead of edges. On the contrary, our crossover operator is very different from the one proposed in [11], where its application partitions each of the two parents into a prefix and suffix substrings (then, to generate the offspring, the prefix of the first parent is combined with the suffix of the second parent, and vice versa). Moreover, our mutation strategy is more aggressive than Chen's, where a single random bit flip is performed: empirically, allowing to perform more than one flip per individual has proven to help avoiding local optima convergence. Additionally, in [11], the single-objective fitness function is evaluated directly on the test dataset. As pointed out in [7], this is a serious methodological mistake, since such data should be kept aside for validation purposes, and not being used for tuning the algorithm. Finally, UCI datasets were also used in [11]; unlike our experiment, though, the authors have used only 4 datasets, making our experimental setting more comprehensive and variegate.

VI. Conclusions and Future Work

Pruning is a technique commonly used to reduce the size of decision trees, with the aim of improving their interpretability and classification performance, while diminishing the risk of overfitting. In this paper, the integration between evolutionary algorithms and decision tree pruning has been studied, by presenting a multi-objective evolutionary approach to the problem of post-pruning decision trees. The method is based on the well-known NSGA-II evolutionary algorithm, and it starts from a fully-grown, unpruned and un-collapsed C4.5 decision tree. An *a posteriori* decision method to choose the best pruned tree in the resulting Pareto front has also been suggested. The proposed solution has been tested against the standard pruning strategies of C4.5 and C5.0 decision tree learners over a selection of 12 UCI datasets, and it has proven to be capable of generating smaller trees than those offered by such competitors, while preserving most of their accuracy, and sometimes improving it. Despite the fact that this can be considered an exploratory study, it clearly demonstrates the feasibility of the overall idea, shading a light on the role that EAs can play even in a classical problem such as decision tree pruning. As for future work, state-of-the-art evolutionary algorithms such as adaptive NSGA-II and ENORA are to be considered for the problem of pruning a fully-grown C5.0 decision tree. Moreover, also the *a posteriori* decision process for the selection of the final solution from the set of non-dominated candidates generated by the evolutionary algorithm deserves some further investigation.

References

[1] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical Machine Learning Tools and Techniques*, 4th ed. Morgan Kaufmann Publishers Inc., 2016.

[2] J. R. Quinlan, *C4.5: Programs for Machine Learning*, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[3] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees*, Monterey, CA: Wadsworth and Brooks, 1984.

[4] J. R. Quinlan, "Simplifying decision trees," *International Journal of Man-Machine Studies*, vol. 27, no. 3, pp. 221-234, 1987.

[5] L. Hyafiland and R. L. Rivest, "Constructing optimal binary decision trees is NP-Complete," *Information Processing Letters*, vol. 5, no. 1, pp. 15–17, 1976.

[6] F. Esposito, D. Malerba, and G. Semeraro, "Decision tree pruning as a search in the state space," in *Proc. the 6th European Conference on Machine Learning (ECML)*, 1993, pp. 165-184.

[7] R. C. Barros and A. A. Freitas, "A survey of evolutionary algorithms for decision-tree induction", in *Proc. the IEEE Transactions on Systems, Man, and Cybernetics*, Part C: Applications and Reviews, 2011, pp. 291-312.

[8] L. Luo, X. Zhang, H. Peng, W. Lv, and Y. Zhang, "A new pruning method for decision tree based on structural risk of leaf node," *Neural Computing and Applications*, vol. 22, pp. 17-26, 2013.

[9] Z. Nie, B. Lin, S. Huang, N. Ramakrishnan, W. Fan, and J. Ye, "Pruning decision trees via max-heap projection," in *Proc. the 2017 SIAM International Conference on Data Mining*, 2017, pp. 10-18.

[10] D. Heath, S. Kasif, and S. Salzberg, "Induction of oblique decision trees," in *Proc. the International Joint Conference on Artificial Intelligence (IJCAI)*, 1993, pp. 1002-1007.

[11] J. Chen, X. Wang, and J. Zhai, "Pruning decision tree using genetic algorithms," in *Proc. the 2009 International Conference on Artificial Intelligence and Computational Intelligence (AICI)*, vol. 3, 2009, pp. 244-248.

[12] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182-197, 2002.

[13] C5.0: An informal tutorial. [Online]. Available: https://www.rulequest.com/see5- unix.html

[14] L. O. Hall, R. Collins, K. W. Bowyer, and R. Banfield, "Error-based pruning of decision trees grown on very large data sets can work!" in *Proc. the 14th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, 2002, pp. 233-238.

[15] F. Esposito, D. Malerba, and G. Semeraro, "A comparative analysis of methods for pruning decision trees," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 5, pp. 476-491, 1997.

[16] L. Rokach and O. Maimon, *Data Mining with Decision Trees: Theory and Applications*, 2nd ed. World Scientific Publishing Company Inc., 2014.

[17] Is C5.0 better than C4.5? [Online]. Available: http://rulequest.com/see5-comparison.html

[18] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*, SpringerVerlag, 2003.

[19] M. Mehta, J. Rissanen, and R. Agrawal, "MDL-based decision tree pruning," in *Proc. the First International Conference on Knowledge Discovery and Data Mining (KDD)*, 1995, pp. 216-221.

[20] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim, "Do we need hundreds of classifiers to solve real world classification

problems?" *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 3133-3181, 2014.
[21] F. Jiménez, A. Gómez-Skarmeta, G. Sánchez, and K. Deb, "An evolutionary algorithm for constrained multi-objective optimization," in *Proc. the Congress on Evolutionary Computation (CEC)*, vol. 2, 2002, pp. 1133-1138.
[22] F. Jiménez, E. Marzano, G. Sánchez, G. Sciavicco, and N. Vitacolonna, "Attribute selection via multi-objective evolutionary computation applied to multi-skill contact center data classification," in *Proc. the IEEE Symposium Series on Computational Intelligence (SSCI)*, 2015, pp. 488-495.
[23] H. Ishibuchi, R. Imada, Y. Setoguchi, and Y. Nojima, "Performance comparison of NSGA-II and NSGA-III on various many-objective test problems," in *Proc. the IEEE Congress on Evolutionary Computation (CEC)*, 2016, pp. 3045–3052.
[24] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81-106, 1986.

Andrea Brunello obtained his master degree in computer science from the University of Udine, Italy, in 2015. Later, he spent a year as a research fellow at the same University. Currently, he is a PhD student at the Department of Computer Science at the University of Udine, Italy. His main research interests are in data integration and modeling, data mining, and machine learning. He is co-author of some scientific seminars, and co-supervisor of theses.

Enrico Marzano holds a master degree in computer science from the University of Udine. Then, he was research fellow at the same university. Currently, he is chief information officer and head of research at Gap srl, a Business Process Outsourcer and IT company. He has been involved in research and development projects with academic staff. He is co-author of some scientific publications and seminars, co-supervisor of thesis and of a Ph.D.

Angelo Montanari is full professor of computer science at the University of Udine, Italy, where he chairs the Data science laboratory. He obtained his PhD in logic and computer science from the University of Amsterdam, The Netherlands, in 1996. His main research interests are in formal methods, AI knowledge representation and reasoning, and databases. He has published over 200 papers in international journals, conferences, and handbook chapters.

Guido Sciavicco holds a bachelor degree, a master degree, and a PhD in computer science from the University of Udine, Italy. Currently, he is associate professor at the University of Ferrara, Italy. He has been working in logics for computer science for more than 10 years and, more recently, he has been doing research in the field of artificial intelligence applied to data science. He is co-author of more than 80 papers in international journals and conferences.