evtree: Evolutionary Learning of Globally Optimal Classification and Regression Trees in R

Thomas Grubinger
Innsbruck Medical University

Achim Zeileis
Universität Innsbruck

Karl-Peter Pfeiffer
Innsbruck Medical University

Abstract

This introduction to the R package **evtree** is a (slightly) modified version of Grubinger, Zeileis, and Pfeiffer (2014), published in the *Journal of Statistical Software*.

Commonly used classification and regression tree methods like the CART algorithm are recursive partitioning methods that build the model in a forward stepwise search. Although this approach is known to be an efficient heuristic, the results of recursive tree methods are only locally optimal, as splits are chosen to maximize homogeneity at the next step only. An alternative way to search over the parameter space of trees is to use global optimization methods like evolutionary algorithms. This paper describes the **evtree** package, which implements an evolutionary algorithm for learning globally optimal classification and regression trees in R. Computationally intensive tasks are fully computed in C++ while the **partykit** package is leveraged for representing the resulting trees in R, providing unified infrastructure for summaries, visualizations, and predictions. `evtree` is compared to the open-source CART implementation `rpart`, conditional inference trees (`ctree`), and the open-source C4.5 implementation `J48`. A benchmark study of predictive accuracy and complexity is carried out in which `evtree` achieved at least similar and most of the time better results compared to `rpart`, `ctree`, and `J48`. Furthermore, the usefulness of `evtree` in practice is illustrated in a textbook customer classification task.

Keywords: machine learning, classification trees, regression trees, evolutionary algorithms, R.

1. Introduction

Classification and regression trees are commonly applied for exploration and modeling of complex data. They are able to handle strongly nonlinear relationships with high order interactions and different variable types. Commonly used classification and regression tree algorithms, including *CART* (Breiman, Friedman, Olshen, and Stone 1984) and *C4.5* (Quinlan 1993), use a greedy heuristic, where split rules are selected in a forward stepwise search for recursively partitioning the data into groups. The split rule at each internal node is selected to maximize the homogeneity of its child nodes, without consideration of nodes further down the tree, hence yielding only locally optimal trees. Nonetheless, the greedy heuristic is computationally efficient and often yields reasonably good results (Murthy and Salzberg 1995). However, for some problems, greedily induced trees can be far from the optimal solution, and a global search over the tree's parameter space can lead to much more compact and accurate models.

The main challenge in growing globally optimal trees is that the search space is typically

huge, rendering full-grid searches computationally infeasible. One possibility to solve this problem is to use stochastic optimization methods like evolutionary algorithms. In practice, however, such stochastic methods are rarely used in decision tree induction. One reason is probably that they are computationally much more demanding than a recursive forward search but another one is likely to be the lack of availability in major software packages. In particular, while there are several packages for R (R Core Team 2014) providing forward-search tree algorithms, there is only little support for globally optimal trees. The former group of packages includes (among others) **rpart** (Therneau and Atkinson 1997), the open-source implementation of the CART algorithm; **party**, containing two tree algorithms with unbiased variable selection and statistical stopping criteria (Hothorn, Hornik, and Zeileis 2006; Zeileis, Hothorn, and Hornik 2008); and **RWeka** (Hornik, Buchta, and Zeileis 2009), the R interface to **Weka** (Witten and Frank 2011) with open-source implementations of tree algorithms such as `J48` and `M5P`, which are the open source implementation of *C4.5* and *M5*, respecitively (Quinlan 1992). A notable exception is the **LogicReg** package (Kooperberg and Ruczinski 2013) for logic regression, an algorithm for globally optimal trees based on binary covariates only and using simulated annealing. Furthermore, the **GA** package Scrucca (2013) provides a collection of general purpose functions, which allows the application of a wide range of genetic algorithm methods. See Hothorn (2014) for an overview of further recursive partitioning packages for R.

To fill this gap, we introduce a new R package **evtree**, available from the Comprehensive R Archive Network at `http://CRAN.R-project.org/package=evtree`, providing evolutionary methods for learning globally optimal classification and regression trees. Generally speaking, evolutionary algorithms are inspired by natural Darwinian evolution employing concepts such as inheritance, mutation, and natural selection. They are population-based, i.e., a whole collection of candidate solutions – trees in this application – is processed simultaneously and iteratively modified by *variation operators* called *mutation* (applied to single solutions) and *crossover* (merging different solutions). Finally, a survivor selection process favors solutions that perform well according to some quality criterion, usually called *fitness function* or *evaluation function*. In this evolutionary process the mean quality of the population increases over time (Bäck 1996; Eiben and Smith 2007). In the case of learning decision trees, this means that the variation operators can be applied to modify the tree structure (e.g., number of splits, splitting variables, and corresponding split points etc.) in order to optimize a fitness function such as the misclassification or error rate penalized by the complexity of the tree. A notable difference to comparable algorithms is the survivor selection mechanism where it is important to avoid premature convergence. In the following, we use a steady state algorithm with deterministic crowding (Mahfoud 1992). Here, each parent solution competes with its most similar offspring for a place in the population. In this way, a fast convergence to similar solutions is avoided and the diversity of candidate solutions is maintained. Furthermore, the applied survivor selection mechanism can be argued to offer computational advantages for the application to classification and regression trees.

Classification and regression tree models are widely used and are especially attractive for many applications, as tree-structured models offer a compact and intuitive representation that can be easily interpreted by practitioners. The goal of `evtree` is to maintain this simple tree structure and offer better performance (in terms of predictive accuracy and/or complexity) than commonly-used recursive partitioning algorithms. However, in cases where the interpretation of the model is not important, other "black-box" methods including *support vector machines*

(SVM; Vapnik 1995) and tree ensemble methods like *random forests* (Breiman 2001) typically offer better predictive performance (Caruana and Niculescu-Mizil 2006; Hastie, Tibshirani, and Friedman 2009).

The remainder of this paper is structured as follows: Section 2 describes the problem of learning globally optimal decision trees and contrasts it to the locally optimal forward-search heuristic that is utilized by recursive partitioning algorithms. Section 3 introduces the `evtree` algorithm before Section 4 addresses implementation details along with an overview of the implemented functions. A benchmark comparison – comprising 14 benchmark datasets, 3 real-world datasets, and 3 simulated scenarios – is carried out in Section 5, showing that the predictive performance of `evtree` is often significantly better compared to the commonly used algorithms `rpart` (from the **rpart** package), `ctree` (from the **party** package) and `J48` (from the **RWeka** interface to **Weka**). Section 6 investigates how the choice of the user-defined hyperparameters influences `evtree`'s classification performance. Finally, Section 7 gives concluding remarks about the implementation and the performance of the new algorithm.

2. Globally and locally optimal decision trees

Classification and regression tree analysis aims at modeling a response variable Y by a vector of P predictor variables $X = (X_1, ..., X_P)$ where for classification trees Y is qualitative and for regression trees Y is quantitative. Tree-based methods first partition the input space X into a set of M rectangular regions R_m $(m = 1, ..., M)$ then fit a (typically simple) model within each region $\{Y|X \in R_m\}$, e.g., the mean, median, or variance etc. Typically, the mode is used for classification trees and the arithmetic mean is employed for regression trees.

To show why forward-search recursive partitioning algorithms typically lead to globally suboptimal solutions, their parameter spaces and optimization problems are presented and contrasted in a unified notation. Although all arguments hold more generally, only binary tree models with some maximum number of terminal nodes $M_{\max}$ are considered. Both restrictions make the notation somewhat simpler while not really restricting the problem: (a) Multiway splits are equivalent to a sequence of binary splits in predictions and number of resulting subsamples; (b) the maximal size of the tree is always limited by the number of observations in the learning sample.

In the following, a binary tree model with M terminal nodes (which consequently has $M - 1$ internal splits) is denoted by

$$\theta \;=\; (v_1, s_1, ..., v_{M-1}, s_{M-1}), \tag{1}$$

where $v_r \in \{1, \ldots, P\}$ are the splitting *variables* and s_r the associated *split* rules for the internal *nodes* $r \in \{1, ..., M-1\}$. Depending on the domain of X_{v_r}, the split rule s_r contains either a cutoff (for ordered and numeric variables) or a non-empty subset of $\{1, \ldots, c\}$ (for a categorical variable with c levels), determining which observations are sent to the first or second subsample. In the former case, there are $u - 1$ possible split rules if X_{v_r} takes u distinct values; and in the latter case, there are $2^{c-1} - 1$ possible splits. Thus, the product of all of these combinations forms all potential elements θ from Θ_M, the space of conceivable trees with M terminal nodes. The overall parameter space is then $\Theta = \bigcup_{M=1}^{M_{\max}} \Theta_M$ (which in practice is often reduced by excluding elements θ resulting in too small subsamples etc.).

Finally, $f(X, \theta)$ denotes the prediction function based on all explanatory variables X and the

chosen tree structure θ from Equation 1. As pointed out above, this is typically constructed using the means or modes in the respective partitions of the learning sample.

2.1. The parameter space of globally optimal decision trees

As done by Breiman *et al.* (1984), let the complexity of a tree be measured by a function of the number of terminal nodes, without further considering the depth or the shape of trees. The goal is then to find that classification and regression tree which optimizes some tradeoff between prediction performance and complexity:

$$\hat{\theta} \;=\; \underset{\theta \in \Theta}{\operatorname{argmin}} \; \operatorname{loss}\{Y, f(X, \theta)\} \;+\; \operatorname{comp}(\theta). \tag{2}$$

where $\operatorname{loss}(\cdot, \cdot)$ is a suitable loss function for the domain of Y; typically, the misclassification (MC) rate and the mean squared error (MSE) are employed for classification and regression, respectively. The function $\operatorname{comp}(\cdot)$ is a function that is monotonically non-decreasing in the number of terminal nodes M of the tree θ, thus penalizing more complex models in the tree selection process. Note that finding $\hat{\theta}$ requires a search over all Θ_M.

The parameter space Θ becomes large for already medium sized problems and a complete search for larger problems is computationally intractable. In fact, Hyafil and Rivest (1976) showed that building optimal binary decision trees, such that the expected number of splits required to classify an unknown sample is minimized, is NP-complete. Zantema (2000) proved that finding a decision tree of minimal size that is decision-equivalent to a given decision tree is also NP-hard. As a consequence the search space is usually limited by heuristics.

2.2. The parameter space of locally optimal decision trees

Instead of searching all combinations in Θ simultaneously, recursive partitioning algorithms only consider one split at a time. At each internal node $r \in \{1, ..., M-1\}$, the split variable v_r and the corresponding split point s_r are selected to locally minimize the loss function. Starting with an empty tree $\theta_0 = (\emptyset)$, the tree is first grown recursively and subsequently *pruned* to satisfy the complexity tradeoff:

$$\tilde{\theta}_r \;=\; \underset{\theta = \theta_{r-1} \cup (v_r, s_r)}{\operatorname{argmin}} \; \operatorname{loss}\{Y, f(X, \theta)\} \qquad (r = 1, \ldots, M_{\max} - 1), \tag{3}$$

$$\tilde{\theta} \;=\; \underset{\tilde{\theta}_r}{\operatorname{argmin}} \; \operatorname{loss}\{Y, f(X, \tilde{\theta}_r)\} \;+\; \operatorname{comp}(\tilde{\theta}_r). \tag{4}$$

For nontrivial problems, forward-search recursive partitioning methods only search a small fraction of the global search space $(v_1, s_1, \ldots, v_{M_{\max}-1}, s_{M_{\max}-1})$. They only search each (v_r, s_r) once, and independently of the subsequent split rules, hence typically leading to a globally suboptimal solution $\tilde{\theta}$.

Note that the notation above uses an exhaustive search for the r-th split, jointly over (v_r, s_r), as is employed in CART or C4.5. So-called *unbiased* recursive partitioning techniques modify this search by first selecting the variable v_r using statistical significance tests and subsequently selecting the optimal split s_r for that particular variable. This approach is used in conditional inference trees (see Hothorn *et al.* 2006, for references to other algorithms) and avoids selecting variables with many potential splits more often than those with fewer potential splits.

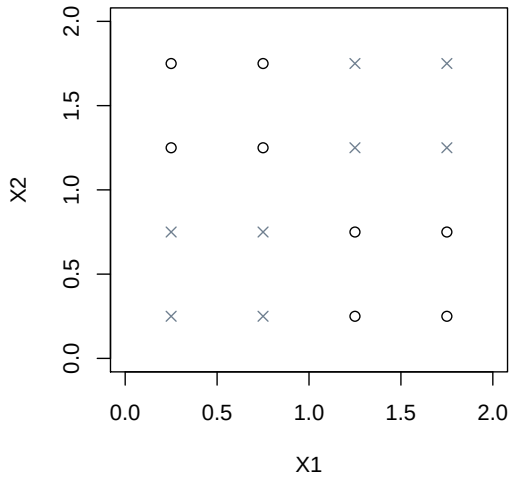


Figure 1: Class distribution of the (X_1, X_2)-plane. The two classes are indicated by black circles and gray crosses.

2.3. An illustration of the limitations of locally optimal decision trees

A very simple example that illustrates the limitation of forward-search recursive partitioning methods is depicted in Figure 1. The example only contains two independent variables and can be solved with three splits that partition the input space into four regions. As expected the recursive partitioning methods `rpart`, `ctree`, and `J48` fail to find any split at all, as the loss function on the resulting subsets cannot be reduced by the first split. For methods that explore Θ in a more global fashion it is straightforward to find an optimal solution to this problem. One solution is the tree constructed by `evtree`:

```
Model formula:
Y ~ X1 + X2

Fitted party:
[1] root
|   [2] X1 < 1.25
|   |   [3] X2 < 1.25: X (n = 4, err = 0.0%)
|   |   [4] X2 >= 1.25: O (n = 4, err = 0.0%)
|   [5] X1 >= 1.25
|   |   [6] X2 < 1.25: O (n = 4, err = 0.0%)
|   |   [7] X2 >= 1.25: X (n = 4, err = 0.0%)

Number of inner nodes:    3
Number of terminal nodes: 4
```

All instances are classified correctly. Each of the terminal nodes 3 and 7 contain four instances of the class `X`. Four instances of class `O` are assigned to each of the terminal nodes 4 and 6.

2.4. Approaches for learning globally optimal decision trees

When compared with the described forward stepwise search, a less greedy approach is to calculate the effects of the split rules deeper down in the tree. In this way optimal trees can be found for simple problems. However, split selection at a given node in Equation 3 has complexity $O(PN)$ (if all P variables are numeric/ordered with N distinct values). Through a global search up to D levels – i.e., corresponding to a full binary tree with $M = 2^D$ terminal nodes – the complexity increases to $O(P^D N^D)$ (Papagelis and Kalles 2001). One conceivable compromise between these two extremes is to look ahead d steps with $1 < d < D$ (see e.g., Esmeir and Markovitch 2007), also yielding a locally optimal tree but less constrained than that from a 1-step-ahead search.

Another class of algorithms is given by stochastic optimization methods that, given an initial tree, seek improved solutions through stochastic changes to the tree structure. Thus, these algorithms try to explore the full parameter space Θ but cannot be guaranteed to find the globally optimal solution but only an approximation thereof. Besides evolutionary algorithms (Koza 1991), *Bayesian CART* (Denison, Mallick, and Smith 1998), and *simulated annealing* (Sutton 1991) were used successfully to solve difficult classification and regression tree problems. Koza (1991) first formulated the concept of using evolutionary algorithms as a stochastic optimization method to build classification and regression trees. Papagelis and Kalles (2001) presented a classification tree algorithm and provided results on several datasets from the UCI machine learning repository (Bache and Lichman 2013). Another method for the construction of classification and regression trees via evolutionary algorithms was introduced by Gray and Fan (2008) and Fan and Gray (2005), respectively. Cantu-Paz and Kamath (2003) used an evolutionary algorithm to induce so-called oblique classification trees. An up-to-date survey of evolutionary algorithms for classification and regression tree induction is provided by Barros, Basgalupp, de Carvalho, and Freitas (2012). A comprehensive survey on the application of genetic programming to classification problems – including classification trees – can be found in (Espejo, Ventura, and Herrera 2010).

3. The `evtree` algorithm

The general framework of evolutionary algorithms emerged from different representatives. Holland (1992) called his method *genetic algorithms*, Rechenberg (1973) invented *evolution strategies*, and Fogel, Owens, and Walsh (1966) introduced *evolutionary programming*. More recently, Koza (1992) introduced a fourth stream and called it *genetic programming*. All four representatives only differ in the technical details, for example the encoding of the individual solutions, but follow the same general outline (Eiben and Smith 2007). Evolutionary algorithms are being increasingly widely applied to a variety of optimization and search problems. Common areas of application include data mining (Freitas 2003; Cano, Herrera, and Lozano 2003), statistics (de Mazancourt and Calcagno 2010), signal and image processing (Man, Tang, Kwong, and Halang 1997), and planning and scheduling (Jensen 2003).

The pseudocode for the general evolutionary algorithm is provided in Table 1. In the context of classification and regression trees, all *individuals* from the population (of some given size) are θs as defined in Equation 1. The details of their evolutionary selection is given below following the general outline displayed in Table 1.

As pointed out in Section 2, some elements $\theta \in \Theta$ are typically excluded in practice to satisfy

1. Initialize the population.
2. Evaluate each individual.
3. While(termination condition is not satisfied) do:
   a. Select parents.
   b. Alter selected individuals via variation operators.
   c. Evaluate new solutions.
   d. Select survivors for the next generation.

Table 1: Pseudocode of the general evolutionary algorithm.

minimal subsample size requirements. In the following, the term *invalid node* refers to such excluded cases, not meeting sample size restrictions.

3.1. Initialization

Each tree of the population is initialized with a valid, randomly generated, split rule in the root node. First, v_1 is selected with uniform probability from $1, ..., P$. Second, a split point s_1 is selected. If X_{v_1} is numeric or ordinal with u unique values, a split point s_1 is selected with uniform probability from the $u - 1$ possible split points of X_{v_1}. If X_{v_1} is nominal and has c categories, each $k = 1, ..., c$ has a probability of 50% to be assigned to the left or the right daughter node. In cases where all k are allocated to the same terminal node, one of the c categories is allocated to the other terminal node, to have the effect of ensuring both terminal nodes are nonempty. If this procedure results in a non-valid split rule, the two steps of random split variable selection and split point selection are repeated. With the definition of $r = 1$ (the root node) and the selection of v_1 and s_1, the initialization is complete and each individual of the population of trees is of type $\theta_1 = (v_1, s_1)$.

3.2. Parent selection

In every iteration, each tree is selected once to be modified by one of the variation operators. In cases where the crossover operator is applied, the second parent is selected randomly from the remaining population. In this way, some trees are selected more than once in each iteration.

3.3. Variation operators

Four types of mutation operators and one crossover operator are utilized by our algorithm. In each modification step, one of the variation operators is randomly selected for each tree. The mutation and crossover operators are described below.

Split

Split selects a random terminal-node and assigns a valid, randomly generated, split rule to it. As a consequence, the selected terminal node becomes an internal node r and two new terminal nodes are generated.

The search for a valid split rule is conducted as in Section 3.1 for a maximum of P iterations. In cases where no valid split rule can be assigned to internal node r, the search for a valid split rule is carried out on another randomly selected terminal node. If, after 10 attempts, no valid split rule can be found, then $\theta_{i+1} = \theta_i$. Otherwise, the set of parameters in iteration $i + 1$ are given by $\theta_{i+1} = \theta_i \cup (v_r, s_r)$.

Prune

Prune chooses a random internal node r, where $r > 1$, which has two terminal nodes as successors and prunes it into a terminal node. The tree's parameters at iteration $i + 1$ are $\theta_{i+1} = \theta_i \setminus (v_r, s_r)$. If θ_i only comprises one internal node, i.e., the root node, then $\theta_{i+1} = \theta_i$ and no pruning occurs.

Major split rule mutation

Major split rule mutation selects a random internal node r and changes the split rule, defined by the corresponding split variable v_r, and the split point s_r. With a probability of 50%, a value from the range $1, ..., P$ is assigned to v_r. Otherwise v_r remains unchanged and only s_r is modified. Again, depending on the domain of X_{v_r}, either a random split point from the range of possible values of X_{v_r} is selected, or a non-empty set of categories is assigned to each of the two terminal nodes. If the split rule at r becomes invalid, the mutation operation is reversed and the procedure, starting with the selection of r, is repeated for a maximum of 3 attempts. Subsequent nodes that become invalid are pruned.

If no pruning occurs, θ_i and θ_{i+1} contain the same set of parameters. Otherwise, the set of parameters $(v_{m_1}, s_{m_1}, ..., v_{m_f}, s_{m_f})$, corresponding to invalid nodes, is removed from θ_i. Thus, $\theta_{i+1} = \theta_i \setminus (v_{m_1}, s_{m_1}, ..., v_{m_f}, s_{m_f})$.

Minor split rule mutation

Minor split rule mutation is similar to the *major split rule mutation* operator. However, it does not alter v_r and only changes the split point s_r by a minor degree, which is defined by one of the following 4 cases:

- X_{v_r} is numerical or ordinal and has at least 20 unique values: The split point s_r is randomly shifted by a non-zero number of unique values of X_{v_r} that is not larger than 10% of the range of unique values.
- X_{v_r} is numerical or ordinal and has less than 20 unique values: The split point is changed to the next larger, or the next lower, unique value of X_{v_r}.
- X_{v_r} is nominal and has at least 20 categories: At least one and at most 10% of the variable's categories are changed.
- X_{v_r} is nominal and has less than 20 categories: One of the categories is randomly modified.

In cases where subsequent nodes become invalid, further split points are searched that preserve the tree's topology. After five non-successful attempts at finding a topology preserving split point, the non-valid nodes are pruned.

Equivalently to the *major split rule mutation* operator the subsequent solution $\theta_{i+1} = \theta_i \setminus (v_{m_1}, s_{m_1}, ..., v_{m_f}, s_{m_f})$.

Crossover

Crossover exchanges, randomly selected, subtrees between two trees. Let θ_i^1 and θ_i^2 be the two trees chosen from the population for crossover. First, two internal nodes r_1 and r_2 are selected randomly from θ_i^1 and θ_i^2, respectively. Let $\text{sub1}(\theta_i^j, r_j)$ denote the subtree of θ_i^j rooted by r_j $(j = 1, 2)$, i.e., the tree containing r_j and its descendant nodes. Then, the complementary part of θ_i^j can be defined as $\text{sub2}(\theta_i^j, r_j) = \theta_i^j \setminus \text{sub1}(\theta_i^j, r_j)$. The crossover operator creates two new trees $\theta_{i+1}^1 = \text{sub2}(\theta_i^1, r_1) \cup \text{sub1}(\theta_i^2, r_2)$ and $\theta_{i+1}^2 = \text{sub2}(\theta_i^2, r_2) \cup \text{sub1}(\theta_i^1, r_1)$. If the crossover creates some invalid nodes in either one of the new trees θ_{i+1}^1 or θ_{i+1}^2, they are omitted.

3.4. Evaluation function

The evaluation function represents the requirements to which the population should adapt. In general, these requirements are formulated by Equation 2. A suitable evaluation function for classification and regression trees maximizes the models' accuracy on the training data, and minimizes the models' complexity. This subsection describes the currently implemented choices of evaluation functions for classification and for regression.

Classification

The quality of a classification tree is most commonly measured as a function of its misclassification rate (MC) and the complexity of a tree by a function of the number of its terminal nodes M. `evtree` uses $2N \cdot \text{MC}(Y, f(X, \theta))$ as the loss function. The number of terminal nodes, weighted by $\log N$ and a user-specified parameter α, measures the complexity of trees.

$$\begin{aligned} \text{loss}(Y, f(X, \theta)) &= 2N \cdot \text{MC}(Y, f(X, \theta)) \\ &= 2 \cdot \sum_{n=1}^{N} I(Y_n \neq f(X_{\cdot n}, \theta)), \\ \text{comp}(\theta) &= \alpha \cdot M \cdot \log N. \end{aligned} \tag{5}$$

With these particular choices, Equation 2 seeks trees $\hat{\theta}$ that minimize the misclassification loss at a BIC-type tradeoff with the number of terminal nodes.

Other, existing and commonly used choices of evaluation functions include the *Bayesian information criterion* (BIC, as in Gray and Fan 2008) and *minimum description length* (MDL, as in Quinlan and Rivest 1989). For both evaluation functions deviance is used for accuracy estimation. Deviance is usually preferred over the misclassification rate in recursive partitioning methods, as it is more sensitive to changes in the node probabilities (Hastie *et al.* 2009, pp. 308–310). However, this is not necessarily an advantage for global tree building methods like evolutionary algorithms.

Regression

For regression trees, accuracy is usually measured by the mean squared error (MSE). Here, it is again coupled with a BIC-type complexity measure:

Using $N \cdot \log \text{MSE}$ as a loss function and $\alpha \cdot 4 \cdot (M+1) \cdot \log N$ as the complexity part, the general formulation of the optimization problem in can be rewritten as:

$$\begin{aligned} \text{loss}(Y, f(X,\theta)) &= N \log \text{MSE}(Y, f(X,\theta)) \\ &= N \log \left\{ \sum_{n=1}^{N} (Y_n - f(X_{\cdot n}, \theta))^2 \right\}, \\ \text{comp}(\theta) &= \alpha \cdot 4 \cdot (M+1) \cdot \log N. \end{aligned} \tag{6}$$

Here, $M+1$ is the effective number of estimated parameters, taking into account the estimates of a mean parameter in each of the terminal nodes and the constant error variance term. With $\alpha = 0.25$ the criteria is, up to a constant, equivalent to the BIC used by Fan and Gray (2005). However, the effective number of parameters estimated for is actually much higher than $M+1$ due to the selection of parameters in the split variable and the selection of the variable itself. It is however unclear how these should be counted (Gray and Fan 2008; Ripley 2008, p. 222). Therefore, a more conservative default value of $\alpha = 1$ is assumed.

3.5. Survivor selection

The population size stays constant during the evolution and only a fixed subset of the candidate solutions can be kept in memory. A common strategy is the $(\mu + \lambda)$ selection, where μ survivors for the next generation are selected from the union of μ parents and λ offsprings. An alternative approach is the (μ, λ) strategy where μ survivors for the next generation are selected from λ offsprings.

Our algorithm uses a deterministic crowding approach, where each parent solution competes with its most similar offspring for a place in the population. In the case of a mutation operator, either the solution before modification, θ_i, or after modification, θ_{i+1}, is kept in memory. In the case of the crossover operator, the initial solutions of θ_i^1 competes with its subsequent solutions θ_{i+1}^1. Correspondingly, one of the two solutions θ_i^2 and θ_{i+1}^2 is rejected. The survivor selection is done deterministically. The tree with lower fitness, according to the evaluation function, is rejected. Note that, due to the definition of the crossover operator, some trees are selected more than once in an iteration. Correspondingly, these trees undergo the survival selection process more than once in an iteration.

As in classification and regression tree analysis the individual solutions are represented by trees. This design offers computational advantages over $(\mu + \lambda)$ and (μ, λ) strategies. In particular, for the application of mutation operators no new trees have to be constructed. The tree after modification is simply accepted or reversed to the previous solution.

There are two important issues in the evolution process of an evolutionary algorithm: population diversity and selective pressure (Michalewicz 1994). These factors are related, as with increasing selective pressure the search is focused more around the currently best solutions. An overly strong selective pressure can cause the algorithm to converge early in local optima. On the other hand, an overly weak selective pressure can make the search ineffective. Using a $(\mu + \lambda)$ strategy, a strong selective pressure can occur in situations as follows. Suppose the

b-th tree of the population is one of the fittest trees in iteration i, and in iteration i one split rule of the b-th tree is changed only by a minor degree. Then very few instances are classified differently and the overall misclassification might not even change. However, as the parent and the offspring represent one of the best solutions in iteration i, they are both selected for the subsequent population. This situation can occur frequently, especially when a fine-tuning operator like the *minor split rule mutation* is used. Then, the diversity of different trees is lost quickly and the algorithm likely terminates in a local optimum. The deterministic crowding selection mechanism clearly avoids these situations, as only the parent or the offspring can be part of the subsequent population.

3.6. Termination

Using the default parameters, the algorithm terminates when the quality of the best 5% of trees stabilizes for 100 iterations, but not before 1000 iterations. If the run does not converge the algorithm terminates after a user-specified number of iterations. In cases where the algorithm does not converge, a warning message is written to the command line. The tree with the highest quality according to the evaluation function is returned.

4. Implementation and application in practice

Package **evtree** provides an efficient implementation of an evolutionary algorithm that builds classification trees in R. CPU- and memory-intensive tasks are fully computed in C++, while the user interfaces and plot functions are written in R. The `.C()` interface (Chambers 2008) was used to pass arguments between the two languages. **evtree** depends on the **partykit** package (Hothorn and Zeileis 2014), which provides an infrastructure for representing, summarizing, and visualizing tree-structured models.

4.1. User interface

The principal function of the **evtree** package is the eponymous function `evtree()` taking arguments

```
evtree(formula, data = list(), weights = NULL, subset = NULL,
  control = evtree.control(...), ...)
```

where `formula`, `data`, `weights`, and `subset` specify the data in the usual way, e.g., via `formula = y ~ x1 + x2`. Additionally, `control` comprises a list of control parameters

```
evtree.control(minbucket = 7L, minsplit = 20L, maxdepth = 9L,
  niterations = 10000L, ntrees = 100L, alpha = 1,
  operatorprob = list(pmutatemajor = 0.2, pmutateminor = 0.2,
    pcrossover = 0.2, psplit = 0.2, pprune = 0.2),
  seed = NULL, ...)
```

where the parameters `minbucket`, `minsplit`, and `maxdepth` constrain the solution to a minimum number of observations in each terminal node, a minimum number of observations in each internal node, and a maximum tree depth. Note that the memory requirements increase by the square of the maximum tree depth. Parameter `alpha` regulates the complexity

parameter α in Equations 5 and 6, respectively. `niterations` and `ntrees` specify the maximum number of iterations and the number of trees in the population, respectively. With the argument `operatorprob`, user-specified probabilities for the variation operators can be defined. For making computations reproducible, argument `seed` is an optional integer seed for the random number generator (at C++ level). If not specified, the random number generator is initialized by `as.integer(runif(1, max = 2^16))` in order to inherit the state of `.Random.seed` (at R level). If set to `-1L`, the seed is initialized by the system time.

The trees computed by `evtree` inherit from class '`party`' supplied by the **partykit** package. The methods inherited in this way include standard `print()`, `summary()`, and `plot()` functions to display trees and a `predict()` function to compute the fitted response or node number etc.

4.2. Case study: Customer targeting

An interesting application for classification tree analysis is target marketing, where limited resources are aimed at a distinct group of potential customers. An example is provided by Lilien and Rangaswamy (2004) in the *Bookbinder's Book Club* marketing case study about a (fictitious) American book club. In this case study, a brochure of the book "The Art History of Florence" was sent to 20,000 customers, 1,806 of which bought the book. The dataset contains a subsample of 1,300 customers with 10 explanatory variables (see Table 2) for building a predictive model of customer choice.

Besides predictive accuracy, model complexity is a crucial issue in this application: Smaller trees are easier to interpret and communicable to marketing experts and management professionals. Hence, we use `evtree` with a maximal depth of two levels of splits only. This is contrasted with `rpart` and `ctree` with and without such a restriction of tree depth to show that the evolutionary search of the global parameter space can be much more effective in balancing predictive accuracy and complexity compared to forward-search recursive partitioning. Results for `J48` on this data set are not reported in detail because the tree depth is very large (even with pruning the depth is 8) and `J48` does not support restriction of the tree depth.

All trees are constrained to have a minimum number of 10 observations per terminal node.

Variable	Description
`choice`	Did the customer buy the advertised book?
`amount`	Total amount of money spent at the book Club.
`art`	Number of art books purchased.
`child`	Number of children's books purchased.
`cook`	Number of cookbooks purchased.
`diy`	Number of do-it-yourself books purchased.
`first`	Number of months since the first purchase.
`freq`	Number of books purchased at the book Club.
`gender`	Factor indicating gender.
`last`	Number of months since the last purchase.
`youth`	Number of youth books purchased.

Table 2: Variables of the Bookbinder's Book Club data.

Additionally, a significance level of 1% is employed in the construction of conditional inference trees, which is more appropriate than the default 5% level for 1,300 observations. To provide uniform visualizations and predictions of the fitted models, '`party`' objects are used to represent all trees. For '`rpart`' trees, **partykit** provides a suitable `as.party()` method while a reimplementation of `ctree()` is provided in **partykit** (as opposed to the original in **party**) that directly leverages the '`party`' infrastructure.

First, the data is loaded and the forward-search trees are grown with and without depth restriction, visualizing the unrestricted trees in Figure 2.

```
R> data("BBBClub", package = "evtree")
R> library("rpart")
R> rp  <- as.party(rpart(choice ~ ., data = BBBClub, minbucket = 10))
R> rp2 <- as.party(rpart(choice ~ ., data = BBBClub, minbucket = 10,
+    maxdepth = 2))
R> ct  <- ctree(choice ~ ., data = BBBClub, minbucket = 10, mincrit = 0.99)
R> ct2 <- ctree(choice ~ ., data = BBBClub, minbucket = 10, mincrit = 0.99,
+    maxdepth = 2)
R> plot(rp)
R> plot(ct)
```

With the objective of building a smaller, but at still accurate tree, `evtree` is constrained to a maximum tree depth of 2, see Figure 3.

```
R> set.seed(1090)
R> ev <- evtree(choice ~ ., data = BBBClub, minbucket = 10, maxdepth = 2)
```

The resulting evolutionary tree is printed below and visualized in Figure 3.

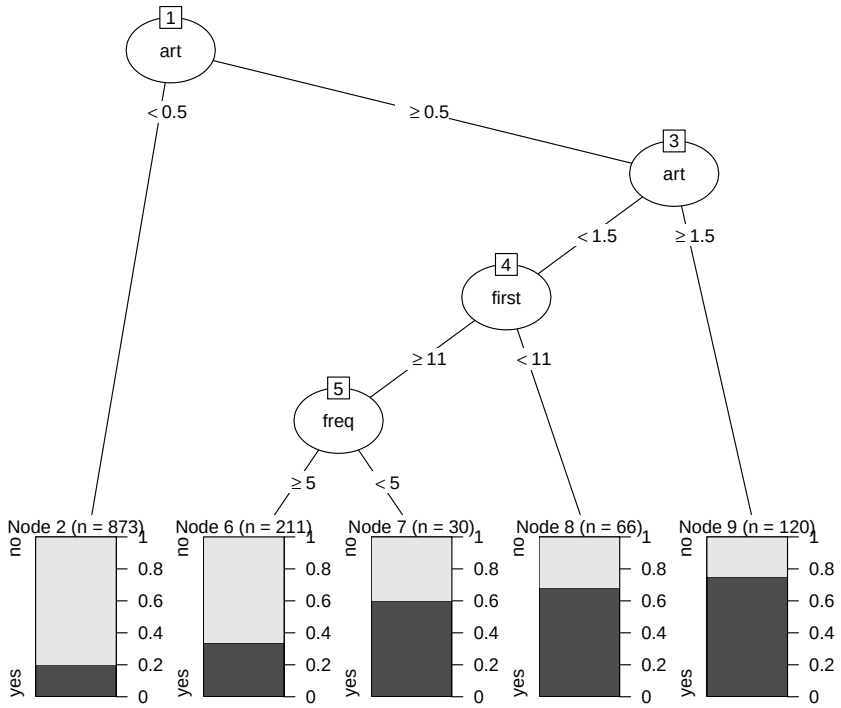
```
R> plot(ev)
R> ev

Model formula:
choice ~ gender + amount + freq + last + first + child + youth +
    cook + diy + art

Fitted party:
[1] root
|   [2] first < 12
|   |   [3] art < 1: no (n = 250, err = 30.8%)
|   |   [4] art >= 1: yes (n = 69, err = 30.4%)
|   [5] first >= 12
|   |   [6] art < 2: no (n = 864, err = 21.8%)
|   |   [7] art >= 2: yes (n = 117, err = 25.6%)

Number of inner nodes:    3
Number of terminal nodes: 4
```

`rpart`



`ctree`

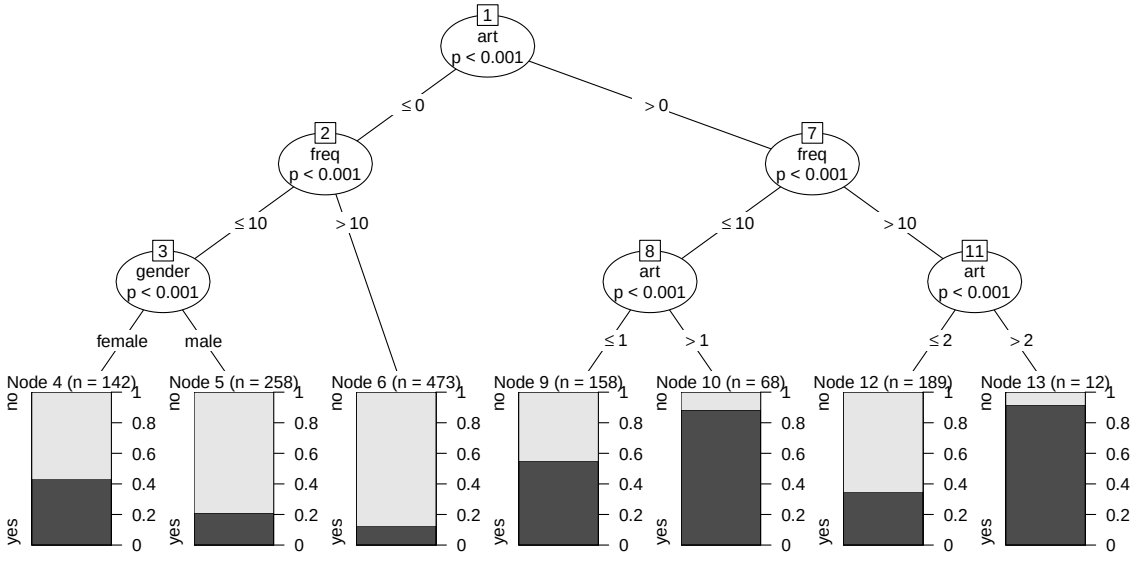


Figure 2: Trees for customer targeting constructed by `rpart` (upper panel) and `ctree` (lower panel). The target variable is the customer's choice of buying the book. The variables used for splitting are the number of art books purchased previously (`art`), the number of months since the first purchase (`first`), the frequency of previous purchases at the Bookbinder's Book Club (`freq`), and the customer's `gender`.

evtree

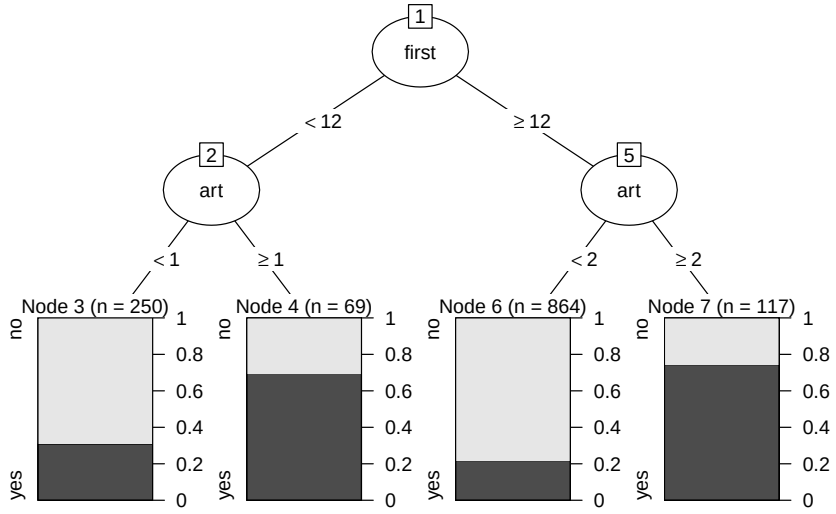


Figure 3: Tree for customer targeting constructed by `evtree`. The target variable is the customer's choice of buying the book. The variables used for splitting are the number of art books purchased previously (`art`), and the number of months since the first purchase (`first`).

Not surprisingly, the explanatory variable `art` – the number of art books purchased previously at the book club – plays a key role in all constructed classification trees along with the number of months since the first purchase (`first`), the frequency of previous purchases (`freq`), and the customer's `gender`. Interestingly, though, the forward-search trees select the arguably most important variable in the first split while the evolutionary tree uses `first` in the first split and `art` in both second splits. Thus, the evolutionary tree uses a different cutoff in `art` for book club members that made their first purchase during the last year as opposed to older customers. While the former are predicted to be buyers if they had previously bought at least one art book, the latter are predicted to purchase the advertised art book only if they had previously bought at least two other art books. Certainly, this classification is easy to understand and communicate (helped by Figure 3) to practitioners.

However, we still need to answer the question how well it performs in contrast to the other trees. Hence, we set up a function `mc()` the computes the misclassification rate as a measure of predictive accuracy and a function `evalfun()` that computes the evaluation function (i.e., penalized by tree complexity) from Equation 5.

```
R> mc <- function(obj) 1 - mean(predict(obj) == BBBClub$choice)
R> evalfun <- function(obj) 2 * nrow(BBBClub) * mc(obj) +
+    width(obj) * log(nrow(BBBClub))
R> trees <- list("evtree" = ev, "rpart" = rp, "ctree" = ct, "rpart2" = rp2,
+    "ctree2" = ct2)
R> round(sapply(trees, function(obj) c("misclassification" = mc(obj),
+    "evaluation function" = evalfun(obj))), digits = 3)
```

```
                     evtree   rpart   ctree  rpart2  ctree2
misclassification     0.243   0.238   0.248   0.262   0.255
evaluation function 660.680 655.851 694.191 701.510 692.680
```

Not surprisingly the evolutionary tree `evtree` outperforms the depth-restricted trees `rpart2` and `ctree2`, both in terms of misclassification and the penalized evaluation function. However, it is interesting to see that `evtree` performs even better than the unrestricted conditional inference tree `ctree` and is comparable in performance to the unrestricted CART tree `rpart`. Hence, the practitioner may choose the evolutionary tree `evtree` as it is the easiest to communicate.

Although the constructed trees are considerably different, the code above shows that the predictive accuracy is rather similar. Moreover, below we see that the structure of the individual predictions on the dataset are rather similar as well:

```
R> ftable(tab <- table(evtree = predict(ev), rpart  = predict(rp),
+    ctree  = predict(ct), observed = BBBClub$choice))
```

```
                   observed  no yes
evtree rpart ctree
no     no    no             799 223
             yes             38  24
       yes   no               0   0
             yes             12  18
yes    no    no               0   0
             yes              0   0
       yes   no              21  19
             yes             30 116
```

```
R> sapply(c("evtree", "rpart", "ctree"), function(nam) {
+    mt <- margin.table(tab, c(match(nam, names(dimnames(tab))), 4))
+    c(abs = as.vector(rowSums(mt))[2],
+      rel = round(100 * prop.table(mt, 1)[2, 2], digits = 3))
+  })
```

```
     evtree   rpart   ctree
abs 186.000 216.000 238.000
rel  72.581  70.833  66.387
```

In this case, `evtree` classifies fewer customers (186) as buyers as `rpart` (216) and `ctree` (238). However, `evtree` achieves the highest proportion of correct classification among the declared buyers: 72.6% compared to 70.8% (`rpart`) and 66.4% (`ctree`).

In summary, this illustrates how `evtree` can be employed to better balance predictive accuracy and complexity by searching a larger space of potential trees. As a final note, it is worth pointing out that in this setup, several runs of `evtree()` with the same parameters typically lead to the same tree. However, this may not always be the case. Due to the stochastic nature of the search algorithm and the vast search space, trees with very different structures

but similar evaluation function values may be found by subsequent runs of `evtree()`. Here, this problem is alleviated by restricting the maximal depth of the tree, yielding a clear solution.

5. Performance comparison

In this section, we compare `evtree` with `rpart`, `ctree`, and `J48` in a more rigorous benchmark comparison.

In the first part of the analysis (Section 5.1) the tree algorithms are compared on 14 benchmark datasets that are publicly available and 3 real-world datasets from the Austrian *Diagnosis Related Group (DRG)* system (Bundesministerium für Gesundheit 2010). As `J48` can only be used for classification, the algorithm is only employed for the 12 classification datasets.

The analysis is based on the evaluation of 250 bootstrap samples for each of the datasets. The misclassification rate on the *out-of-bag* samples is used as a measure of predictive accuracy (Hothorn, Leisch, Zeileis, and Hornik 2005). Furthermore, the complexity is estimated by the number of terminal nodes. The results are summarized by the mean differences of the 250 runs – each corresponding to one of the 250 different bootstrap samples. For the assessment of significant differences in predictive accuracy and complexity, respectively, Dunnett's correction from R package **multcomp** (Hothorn, Bretz, and Westfall 2008) was used for calculating simultaneous 95% confidence intervals on the individual datasets. Confidence intervals that do not encompass the zero-line indicate significant differences at the 5% level.

In the second part (Section 5.2) the algorithms' performances are assessed on an artificial chessboard problem that is simulated with different noise levels. The estimation of predictive accuracy and the number of terminal nodes is based on 250 realizations for each simulation.

`evtree`, `rpart`, and `ctree` models are constrained to a minimum number of 7 observations per terminal node, 20 observations per internal node, and a maximum tree depth of 9. Apart from that, the default settings of the algorithms are used. `J48` is only constrained to a minimum number of 7 observations per terminal node, as other restrictions are available in this implementation.

As missing values are currently not supported by `evtree` (e.g., by surrogate splits), the 16 missing values in the *Breast cancer database* – the only dataset in the study with missing values – were removed before analysis.

5.1. Benchmark and real-world problems

In Table 3 the benchmark and real-world datasets from the Austrian DRG system are described. In the Austrian DRG system, resources are allocated to hospitals by simple rules mainly regarding the patients' diagnoses, procedures, and age. Regression tree analysis is performed to model patient groups with similar resource consumption. A more detailed description of the datasets and the application can be found in Grubinger, Kobel, and Pfeiffer (2010).

The datasets were chosen from different domains and cover a wide range of dataset characteristics and complexities. The sample sizes of the selected datasets range from 214 instances (*Glass identification* data) to 19020 instances (*MAGIC gamma telescope*). The number of attributes varies between 4 (*Servo*) and 180 (*DNA*). The types of attributes vary among datasets, and include datasets which have both categorical and numerical variables or just

Dataset	Instances	Attributes				
		Binary	Nominal	Ordered	Metric	Classes
Glass identification$^{\#}$	214	-	-	-	9	6
Statlog heart*	270	3	3	1	6	2
Ionosphere$^{\#}$	351	2	-	-	32	2
Musk^{+}	476	-	-	-	166	2
Breast cancer database$^{\#}$	685	-	4	5	-	2
Pima Indians diabetes$^{\#}$	768	-	-	-	8	2
Vowel$^{\#}$	990	-	1	-	9	11
Statlog German credit*	1000	2	10	1	7	2
Contraceptive method*	1437	3	-	4	2	3
DNA$^{\#}$	3186	180	-	-	-	3
Spam^{+}	4601	-	-	-	57	2
MAGIC gamma telescope*	19020	-	-	-	10	2
Servo$^{\#}$	167	-	4	-	-	-
Boston housing$^{\#}$	506	1	-	-	12	-
MEL0101$^{\diamond}$	875	1	4	1	108	-
HDG0202$^{\diamond}$	3933	1	7	1	46	-
HDG0502$^{\diamond}$	8251	1	7	1	91	-

Table 3: Description of the evaluated benchmark datasets. The datasets marked with $*$ originate from the UCI machine learning repository (Bache and Lichman 2013) and are made available in the **evtree** package. Datasets marked with # and + are from the R packages **mlbench** (Leisch and Dimitriadou 2012) and **kernlab** (Karatzoglou *et al.* 2004), respectively. The three real-world datasets from the Austrian DRG system are marked with $\diamond$.

one of them. The number of classes for the classification task vary between 2 and 11 classes.

The relative performance of `evtree` and `rpart` is summarized in Figure 4 (upper panels). Performance differences are displayed relative to `evtree`'s performance. For example, on the *Glass* dataset, the average misclassification rate of `rpart` is 2.7% higher than the misclassification rate of `evtree`. It can be observed that on 12 out of 17 datasets `evtree` significantly outperforms `rpart` in terms of predictive accuracy. Only on the *Contraceptive method* dataset does `evtree` perform slightly worse. In terms of complexity, `evtree` models are significantly more complex on 9 and less complex on 7 datasets.

Figure 4 (lower panels) summarizes the relative performance of `evtree` and `ctree`. For 15 out of 17 datasets `evtree` shows a better predictive performance. The algorithms' performances is significantly worse on the *MEL0101* dataset, where the mean squared error of `ctree` is 5.6% lower. However, on this dataset, `ctree` models are on average 86.5% larger than `evtree` models. The relative complexity of `evtree` models is significantly smaller for 15 and larger for 1 dataset.

The relative performance of `evtree` and `J48` is summarized in Figure 5. It can be observed that on 8 out of 11 classification datasets `evtree` significantly outperforms `J48` in terms of predictive accuracy. `evtree`'s performance is significantly worse on the *Vowel* dataset, where the misclassification error of `evtree` is 2.7% higher. As `J48` allows multiway splits, the complexity of the two algorithms is compared by the total number of nodes (internal nodes +

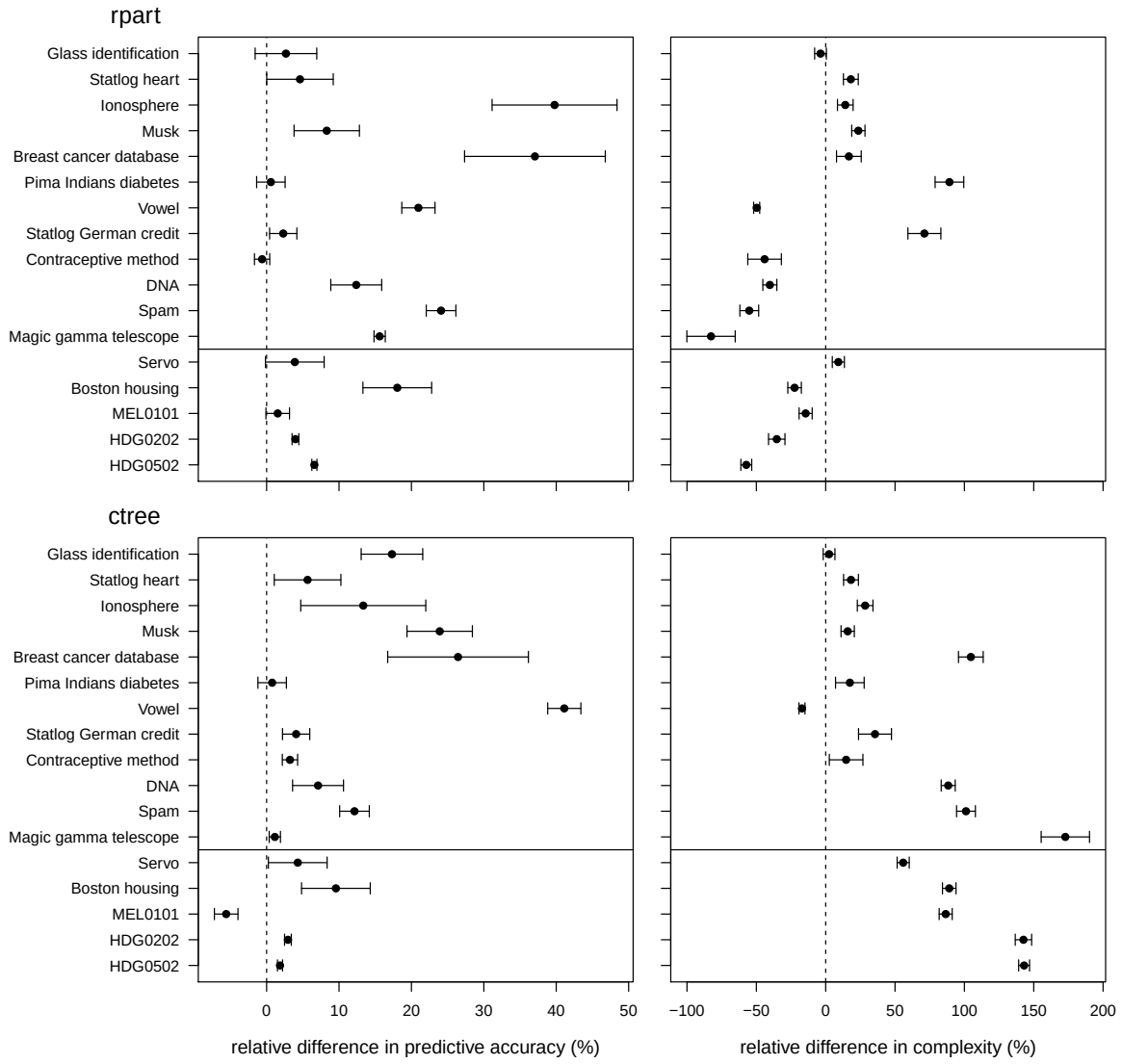


Figure 4: Performance comparison of `evtree` vs. `rpart` (upper panels) and `evtree` vs. `ctree` (lower panels). Prediction error (left panels) is compared by the relative difference of the misclassification rate or the mean squared error. The complexity (right panels) is compared by the relative difference of the number of terminal nodes.

terminal nodes). `evtree` model are significantly less complex on all 11 datesets. Note that we also investigated whether the reduced-error pruning option in `J48` improves the performance of `J48`. However, pruning only slightly reduced the complexity while deteriorating accurcay on the *Vowel*, *Musk*, and *Spam* datasets. Therefore, we only report the unpruned results here.

From these results, it is not obvious which characteristics drive `evtree`'s relative performance. Presumably, for some datasets the forward-search algorithms already yield trees that are close to optimal, thus leaving little room for further improvements. In contrast, for other datasets

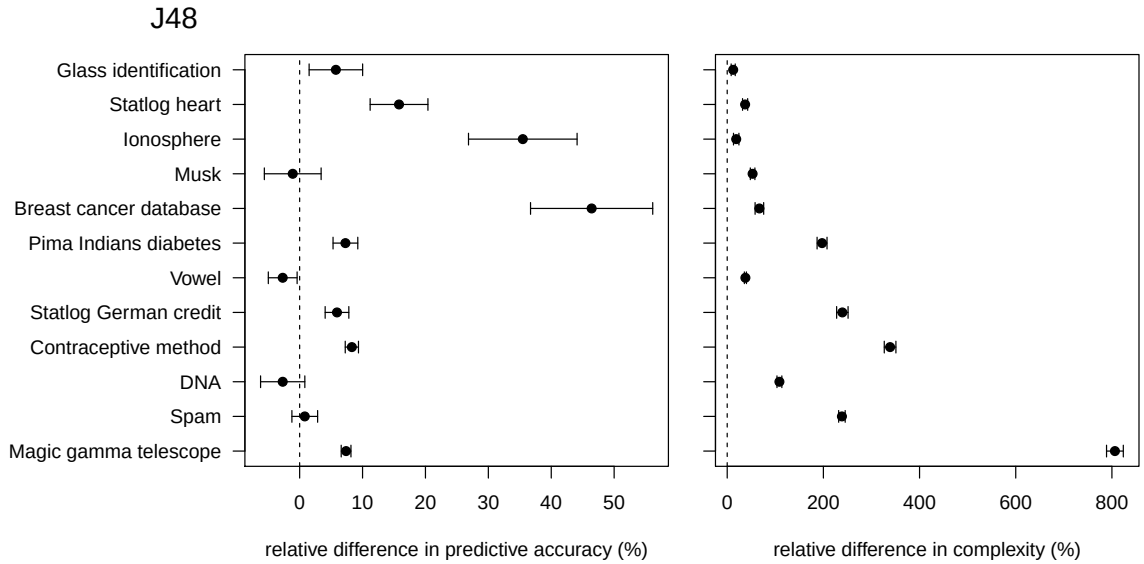


Figure 5: Performance comparison of `evtree` vs. `J48`. Prediction error (left panel) is compared by the relative difference of the misclassification rate. The complexity (right panel) is compared by the relative difference of the total number of nodes.

with more complex interaction patterns (and possibly low main effects) `evtree`'s global-search strategy is probably able to provide better predictive accuracy and/or sparser trees.

Disadvantages of the `evtree` algorithm are computation time and memory requirements. While the smallest of the analyzed datasets, *Glass identification*, only needed approximately 4–6 seconds to fit, larger datasets demanded several minutes. The fit of a model from the largest dataset, *MAGIC gamma telescope*, required approximately 40–50 minutes and a main memory of 400 MB. The required resources were measured on an Intel Core 2 Duo with 2.2 GHz and 2 GB RAM using the 64-bit version of Ubuntu 10.10.

Another important issue to be considered is the random nature of evolutionary algorithms. For larger datasets, frequently, considerably different solutions exist that yield a similar or even the same evaluation function value. Therefore, subsequent runs of `evtree` can result in very different tree structures. This is not a problem if the tree is intended only for predictive purposes, and it is also not a big issue for many decision and prognosis tasks. Typically, in such applications, the resulting model has to be accurate, compact, and meaningful in its interpretation, but the particular tree structure is of secondary importance. Examples of such applications include the presented marketing case study and the Austrian DRG system. In cases where a model is not meaningful in its interpretation, the possibility of constructing different trees can even be beneficial. However, if the primary goal is to interpret relationships in the data, based on the selected splits, the random nature of the algorithm has to be considered.

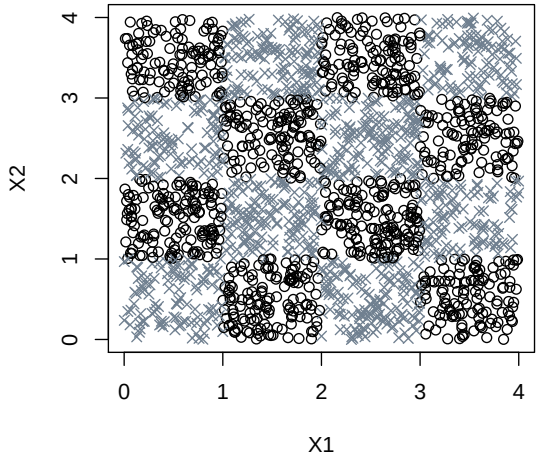


Figure 6: Class distribution of the simulated 4×4 chessboard problem with zero noise, plotted on the (X_1, X_2)-plane. The two classes are indicated by black circles and gray crosses, respectively.

5.2. Artificial problem

In this section we demonstrate the ability of `evtree` to solve an artificial problem that is difficult to solve for most recursive classification tree algorithms (Loh 2009). The data was simulated with 2000 instances for both the training set and the test set. Predictor variables X_1 and X_2 are simulated to be uniformly distributed in the interval $[0, 4]$. The classes are distributed in alternating squares forming a 4×4 chessboard in the (X_1, X_2)-plane. One realization of the simulated data is shown in Figure 6. Furthermore, variables X_3–X_8 are noise variables that are uniformly distributed on the interval [0, 1]. The ideal model for this problem only uses variables X_1 and X_2 and has 16 terminal nodes, whereas each terminal node comprises the observations that are in the region of one square. Two further simulations are done in the same way, but 5% and 10% percent of the class labels are randomly changed to the other class.

Noise	Accuracy				Complexity			
	`evtree`	`rpart`	`ctree`	`J48`	`evtree`	`rpart`	`ctree`	`J48`
0%	93.2(7.4)	69.1(18.3)	49.9(1.1)	50.0(1.1)	14.4(2.2)	16.6(8.2)	1.1(0.3)	1.2(1.1)
5%	89.0(6.8)	65.7(17.4)	50.1(1.6)	50.1(1.1)	14.4(2.2)	14.6(8.0)	1.1(0.7)	1.2(1.1)
10%	84.5(5.6)	62.8(14.1)	50.1(1.3)	50.2(3.6)	14.6(2.0)	14.3(7.3)	1.1(0.4)	1.5(4.8)

Table 4: Mean (and standard deviation) of accuracy and complexity for simulated 4×4 chessboard examples.

The results are summarized in Table 4. It can be seen that, in the absence of noise, `evtree` classifies 93.2% of the instances correctly and requires 14.4 terminal nodes. `rpart` models, on

the other hand, on average classify 69.1% of the data points correctly and have 16.6 terminal nodes. An average `ctree` model has only 1.1 terminal nodes – i.e., typically does not split at all (as expected in an unbiased forward selection) – and consequently a classification accuracy of 49.9%. `J48` trees on average have 1.2 nodes and classify 50.0% of the datapoints correctly. In the presence of 5% and 10% noise, `evtree` classifies 89.0% and 84.5% of the data correctly but still performs clearly better than the other models.

6. Choice of `evtree` parameters

In this section, `evtree` is simulated with different (hyper)parameter choices. In general, the optimal choice of parameters clearly depends on the particular dataset. This section gives some insight into which parameter choices work for which kind of data complexity. Furthermore, the results give insight into the robustness of the default parameter choices.

As in Section 5, `evtree` models are constrained to a minimum number of 7 observations per terminal node, 20 observations per internal node, and a maximum tree depth of 9. The analysis is based on 4 datasets (*Statlog heart*, *Statlog german credit*, *Spam*, and the *4x4 Chessboard problem* with 5% noise). Again, the analysis of each of the 4 datasets is based on 250 bootstrap samples. In Section 6.1, `evtree` is simulated with a diverse set of variation operator probability choices. Section 6.2 provides results for the choice of different population sizes.

6.1. Variation operator probabilities

Table 5 displays the different operator probability settings that are used in Figure 7. The *Mutation* column summarizes the probability of selecting the *minor split rule mutation* or the *major split rule mutation* operator – which both have the same probability. The *Split/Prune* column summarizes the probability of selecting the *prune* or the *split* operator – again both operators have the same probability. For example, the operator probability setting *c0m50sp50* has a 0% probability of selecting the *crossover* operator, a 50% probability for selecting one of the mutation operators (25% for *minor split rule mutation* and 25% for *major split rule mutation*) and a 50% probability for selecting one of the *split* (with 25% probability) or the *prune* operators (with 25% probability). Note that thus the default choice in `evtree.control` corresponds to c20m40sp40. The different operator probability settings are simulated with different numbers of iterations. Simulations with a very low number of iterations (two settings with 200 and 500 iterations are used) should give insight into the efficiency of the variation operator settings. Additionally, the results from a simulation with 10000 iterations is included, which provides an estimate of performance differences for the default setting.

Figure 7 summarizes `evtree`'s performance with different variation operator probabilities. For the two datasets *Statlog heart* and *Statlog German credit*, the results of the different variation operator settings are nearly the same. The simulations with only 200 iterations and 500 iterations give similar results as the simulations with 10000 iterations. Thus, for the two smaller datasets, neither the choice of variation operator probabilities, nor the duration of the search, has a relevant effect on the results.

For the more complex *spam* dataset and the *Chessboard 4x4* problem, an increased number of iterations leads to a (much) better performance. For the simulation with only 200 iterations, large differences between the variation operator settings can be observed. For both datasets

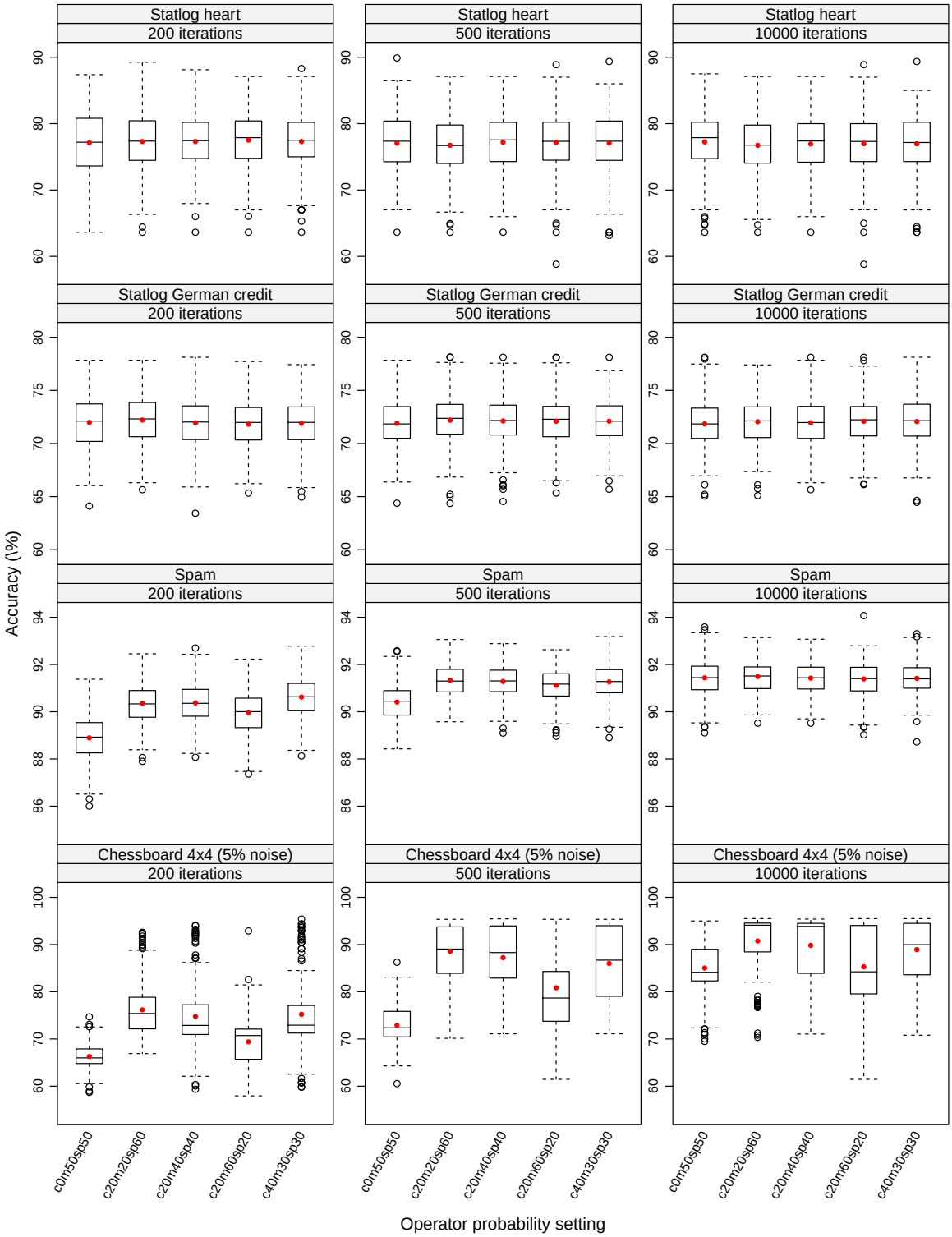


Figure 7: Results of `evtree` with different variation operator probabilities (see Table 5) and number of iterations. In addition to the standard boxplot statistics, the mean differences are indicated by red dots.

Operator probability setting	Crossover [%]	Mutation [%]	Split/Prune[%]
c0m50sp50	0	50	50
c20m20sp60	20	20	60
c20m40sp40	20	40	40
c20m60sp20	20	60	20
c40m30sp30	40	30	30

Table 5: Operator probability settings used for the simulation of the different operator probabilities. The default choice in `evtree.control` corresponds to c20m40sp40. The corresponding results are displayed in Figure 7.

the variation operator setting without crossover performed worst. The performance differences of the different variation operator settings become smaller with an increasing number of iterations. For the *Spam* dataset, a search over 10000 iterations leads to approximately the same performance for all variation operator settings. In case of the *Chessboard 4x4* problem the best setting ($c20m20sp60$; 20% crossover, 20% mutation and 60% split/prune) classifies, on average, 90.8% of the instances correctly. `evtree`'s default setting ($c20m40sp40$; 20% crossover, 40% mutation and 40% split/prune) is the second-best setting and classifies 89.8% of the instances correctly.

6.2. Population size

In this subsection, `evtree` is simulated with population sizes between 25 trees and 500 trees. The results of the simulation are illustrated in Figure 8. For the two smaller datasets *Statlog heart* an *Statlog German credit* the results are similar for all population sizes. In fact, the smallest population sizes (25 trees; *Statlog heart*: 77.1%, *Statlog German credit*: 72.2%) even perform very slightly bettr on average (500 trees; *Statlog heart*: 76.5%, *Statlog German credit*: 71.7%). Compared to the overall variation over samples, these differences are very small though. The average number of terminal nodes for the largest population size is larger than the trees from the smalles population size (*Statlog heart*: 7.3 terminal nodes when simulated with a population of 500 trees, 6.5 terminal nodes when simulated with a population of 25 trees; *Statlog German credit*: 13.4 terminal nodes when simulated with a population of 500 trees, 10.9 terminal nodes when simulated with a population of 25 trees).

However, for the more complex datasets *Spam* and *Chessboard 4x4 (with* 5% *noise)*, the predictive accuracy is monotonically increasing with the search space. The largest performance difference can be observed on the chessboard simulation problem. With a simulation of 25 trees only 81.0% of the instance are classified correctly, while with a simulation of 500 trees, 93.6% of the instances are classified correctly.

7. Conclusions

In this paper, we present the **evtree** package, which implements classification and regression trees that are grown by an evolutionary algorithm. The package uses standard `print()`, `summary()`, and `plot()` functions to display trees and a `predict()` function to predict the class labels of new data from the **partykit** package. As evolutionary learning of trees is

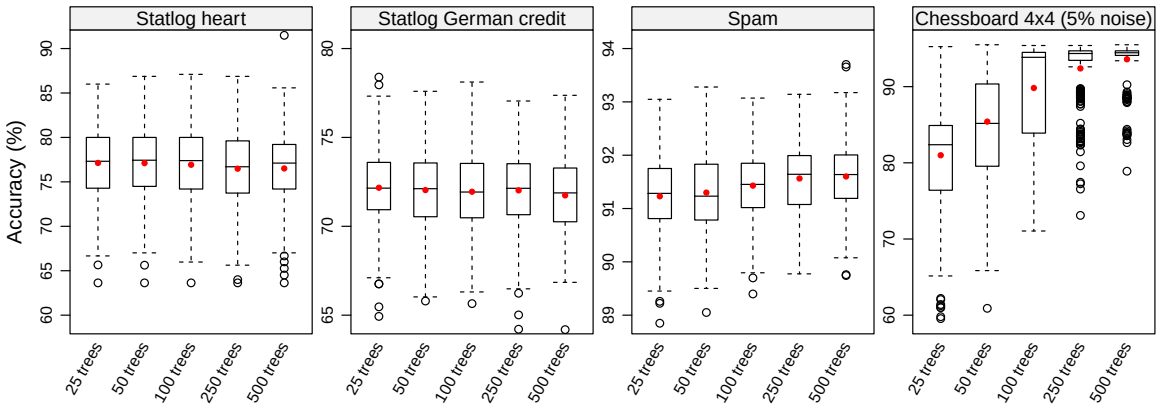


Figure 8: Results of `evtree` with different population sizes. In addition to the standard boxplot statistics, the mean differences are indicated by red dots.

computationally demanding, most calculations are conducted in C++. At the moment our algorithm does not support parallelism. However, we intend to extend **evtree** to parallel computing in future versions.

The comparisons with recursive partitioning methods `rpart`, `ctree`, and `J48` in Sections 4 and 5 shows that `evtree` performs very well in a wide variety of settings, often balancing predictive accuracy and complexity better than the forward-search methods.

In Section 6, we compare different parameter settings for the `evtree` algorithm. It can be observed that the particular choice of variation operator probabilities is fairly robust, provided the population size is sufficiently large. The default settings in the number of iterations and the population size are sufficient for most datasets with medium complexity. However, for very complex datasets an increase in the number of iterations or the population size, may significantly improve `evtree`'s predictive performance.

The goal of `evtree` is not to replace the well-established algorithms like `rpart`, `ctree`, and `J48` but rather to complement the tree toolbox with an alternative method which may perform better given sufficient amounts of time and main memory. By the nature of the algorithm it is able to discover patterns which cannot be modeled by a greedy forward-search algorithm. As `evtree` models can be substantially different to recursively fitted models, it can be beneficial to use both approaches, as this may reveal additional relationships in the data.

References

Bache K, Lichman M (2013). "UCI Machine Learning Repository." URL `http://archive.ics.uci.edu/ml/`.

Bäck T (1996). *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York.

Barros RC, Basgalupp MP, de Carvalho A, Freitas AA (2012). "A Survey of Evolutionary

Algorithms for Decision-Tree Induction." *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, **42**(3), 291–312.

Breiman L (2001). "Random Forests." *Machine Learning*, **45**(1), 5–32.

Breiman L, Friedman JH, Olshen RA, Stone CJ (1984). *Classification and Regression Trees*. Wadsworth, Belmont.

Bundesministerium für Gesundheit (2010). "The Austrian DRG System." URL: http://www.bmg.gv.at/home/EN/Topics/The_Austrian_DRG_system_brochure_ (accessed on 2014-08-09).

Cano JR, Herrera F, Lozano M (2003). "Using Evolutionary Algorithms as Instance Selection for Data Reduction in KDD: An Experimental Study." *IEEE Transactions on Evolutionary Computation*, **7**(6), 561–575.

Cantu-Paz E, Kamath C (2003). "Inducing Oblique Decision Trees with Evolutionary Algorithms." *IEEE Transactions on Evolutionary Computation*, **7**(1), 54–68.

Caruana R, Niculescu-Mizil A (2006). "An Empirical Comparison of Supervised Learning Algorithms." In *Proceedings of the 23rd International Conference on Machine learning*, pp. 161–168. ACM Press, New York.

Chambers JM (2008). *Software for Data Analysis: Programming with R*. Springer-Verlag, New York.

de Mazancourt C, Calcagno V (2010). "**glmulti**: An R Package for Easy Automated Model Selection with (Generalized) Linear Models." *Journal of Statistical Software*, **34**(12), 1–29. URL http://www.jstatsoft.org/v34/i12/.

Denison DGT, Mallick BK, Smith AFM (1998). "A Bayesian CART Algorithm." *Biometrika*, **85**(2), 363–377.

Eiben AE, Smith JE (2007). *Introduction to Evolutionary Computing*. Springer-Verlag, New York.

Esmeir S, Markovitch S (2007). "Anytime Learning of Decision Trees." *Journal of Machine Learning Research*, **8**, 891–933.

Espejo PG, Ventura S, Herrera F (2010). "A Survey on the Application of Genetic Programming to Classification." *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, **40**(2), 121–144.

Fan G, Gray JB (2005). "Regression Tree Analysis Using TARGET." *Journal of Computational and Graphical Statistics*, **14**(1), 206–218.

Fogel LJ, Owens AJ, Walsh MJ (1966). *Artificial Intelligence Through Simulated Evolution*. John Wiley & Sons, New York.

Freitas AA (2003). "A Survey of Evolutionary Algorithms for Data Mining and Knowledge Discovery." In A Ghosh, S Tsutsui (eds.), *Advances in Evolutionary Computing*, pp. 819–845. Springer-Verlag, New York.

Gray JB, Fan G (2008). "Classification Tree Analysis Using TARGET." *Computational Statistics & Data Analysis*, **52**(3), 1362–1372.

Grubinger T, Kobel C, Pfeiffer KP (2010). "Regression Tree Construction by Bootstrap: Model Search for DRG-Systems Applied to Austrian Health-Data." *BMC Medical Informatics and Decision Making*, **10**(9).

Grubinger T, Zeileis A, Pfeiffer KP (2014). "**evtree**: Evolutionary Learning of Globally Optimal Classification and Regression Trees in R." *Journal of Statistical Software*, **61**(1), 1–29. URL http://www.jstatsoft.org/v61/i01/.

Hastie T, Tibshirani R, Friedman J (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2nd edition. Springer-Verlag, New York.

Holland JH (1992). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. MIT Press, Cambridge.

Hornik K, Buchta C, Zeileis A (2009). "Open-Source Machine Learning: R Meets **Weka**." *Computational Statistics*, **24**(2), 225–232.

Hothorn T (2014). "CRAN Task View: Machine Learning." Version 2014-08-08, URL http://CRAN.R-project.org/view=MachineLearning.

Hothorn T, Bretz F, Westfall P (2008). "Simultaneous Inference in General Parametric Models." *Biometrical Journal*, **50**(3), 346–363.

Hothorn T, Hornik K, Zeileis A (2006). "Unbiased Recursive Partitioning: A Conditional Inference Framework." *Journal of Computational and Graphical Statistics*, **15**(3), 651–674.

Hothorn T, Leisch F, Zeileis A, Hornik K (2005). "The Design and Analysis of Benchmark Experiments." *Journal of Computational and Graphical Statistics*, **14**(3), 675–699.

Hothorn T, Zeileis A (2014). "**partykit**: A Modular Toolkit for Recursive Partytioning in R." *Working Paper 2014-10*, Working Papers in Economics and Statistics, Research Platform Empirical and Experimental Economics, Universität Innsbruck. URL http://EconPapers.RePEc.org/RePEc:inn:wpaper:2014-10.

Hyafil L, Rivest RL (1976). "Constructing Optimal Binary Decision Trees Is NP-Complete." *Information Processing Letters*, **5**(1), 15–17.

Jensen MT (2003). "Generating Robust and Flexible Job Shop Schedules Using Genetic Algorithms." *IEEE Transactions on Evolutionary Computation*, **7**(3), 275–288.

Karatzoglou A, Smola A, Hornik K, Zeileis A (2004). "**kernlab** – An S4 Package for Kernel Methods in R." *Journal of Statistical Software*, **11**(9), 1–20. URL http://www.jstatsoft.org/v11/i09/.

Kooperberg C, Ruczinski I (2013). ***LogicReg***: *Logic Regression*. R package version 1.5.5, URL http://CRAN.R-project.org/package=LogicReg.

Koza JR (1991). "Concept Formation and Decision Tree Induction Using the Genetic Programming Paradigm." In *Proceedings of the 1st Workshop on Parallel Problem Solving from Nature*, pp. 124–128. Springer-Verlag, London.

Koza JR (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA.

Leisch F, Dimitriadou E (2012). ***mlbench**: Machine Learning Benchmark Problems*. R package version 2.1-1, URL http://CRAN.R-project.org/package=mlbench.

Lilien GL, Rangaswamy A (2004). *Marketing Engineering: Computer-Assisted Marketing Analysis and Planning*. 2nd edition. Trafford Publishing, Victoria.

Loh WY (2009). "Improving the Precision of Classification Trees." *Annals of Applied Statistics*, **3**(4), 1710–1737.

Mahfoud SW (1992). "Crowding and Preselection Revisited." *Parallel Problem Solving from Nature*, **2**, 27–36.

Man KF, Tang KS, Kwong S, Halang WA (1997). *Genetic Algorithms for Control and Signal Processing*. Springer-Verlag, New York.

Michalewicz Z (1994). *Genetic Algorithms Plus Data Structures Equals Evolution Programs*. Springer-Verlag, New York.

Murthy SK, Salzberg S (1995). "Decision Tree Induction: How Effective Is the Greedy Heuristic?" In UM Fayyad, R Uthurusamy (eds.), *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, pp. 222–227. AAAI Press, San Mateo.

Papagelis A, Kalles D (2001). "Breeding Decision Trees Using Evolutionary Techniques." In CE Brodley, AP Danyluk (eds.), *Proceedings of the Eighteenth International Conference on Machine Learning*, pp. 393–400. Morgan Kaufmann Publishers, San Mateo.

Quinlan JR (1992). "Learning with Continuous Classes." In *Proceedings of the 5th Australian Joint Conference on Artificial Intelligence*, pp. 343–348. World Scientific.

Quinlan JR (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, San Mateo.

Quinlan JR, Rivest RL (1989). "Inferring Decision Trees Using the Minimum Description Length Principle." *Information and Computation*, **80**(3), 227–248.

R Core Team (2014). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

Rechenberg I (1973). *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution*. Frommann-Holzboog Verlag, Stuttgart.

Ripley BD (2008). *Pattern Recognition and Neural Networks*. Cambridge University Press, Cambridge.

Scrucca L (2013). "**GA**: A Package for Genetic Algorithms in R." *Journal of Statistical Software*, **53**(4), 1–37. URL http://www.jstatsoft.org/v53/i04/.

Sutton CD (1991). "Improving Classification Trees with Simulated Annealing." In EM Keramidas, SM Kaufman (eds.), *Computing Science and Statistics: Proceedings of the 23rd Symposium on the Interface*, pp. 333–344. Interface Foundation of North America, Fairfax Station.

Therneau TM, Atkinson EJ (1997). "An Introduction to Recursive Partitioning Using the `rpart` Routines." *Technical Report 61*. URL http://www.mayo.edu/hsr/techrpt/61.pdf.

Vapnik VN (1995). *The Nature of Statistical Learning Theory*. Springer-Verlag, New York.

Witten IH, Frank E (2011). *Data Mining: Practical Machine Learning Tools and Techniques*. 3rd edition. Morgan Kaufmann Publishers, San Francisco.

Zantema H (2000). "Finding Small Equivalent Decision Trees Is Hard." *International Journal of Foundations of Computer Science*, **11**(2), 343–354.

Zeileis A, Hothorn T, Hornik K (2008). "Model-Based Recursive Partitioning." *Journal of Computational and Graphical Statistics*, **17**(2), 492–514.

Affiliation:

Thomas Grubinger, Karl-Peter Pfeiffer
Department for Medical Statistics, Informatics and Health Economics
Innsbruck Medical University
6020 Innsbruck, Austria
E-mail: Thomas.Grubinger@scch.at, Karl-Peter.Pfeiffer@i-med.ac.at

Achim Zeileis
Department of Statistics
Faculty of Economics and Statistics
Universität Innsbruck
6020 Innsbruck, Austria
E-mail: Achim.Zeileis@R-project.org
URL: http://eeecon.uibk.ac.at/~zeileis/