Journal of Machine Learning Research 1 (2000) 1-48 Submitted 4/00; Published 10/00

Complete Search for Feature Selection in Decision Trees

Salvatore Ruggieri RUGGIERI@DI.UNIPI.IT
Department of Computer Science
University of Pisa
Largo B. Pontecorvo 3, 56127, Pisa, Italy

Editor:

Abstract

We propose a wrapper model for optimal feature subset selection specifically designed for decision tree classifiers. Our starting point is an exact enumeration procedure of the subsets of features that lead to *all and only the* distinct decision trees. The procedure stores in the worst case a number of trees linear in the number of features. By exploiting a further pruning of the search space, we design a complete procedure for finding acceptable feature subsets, which depart by at most δ in misclassification error rate from the an optimal feature subset. The approach is also adapted to the design of a computational optimization of the sequential backward elimination heuristics, extending its applicability to large dimensional datasets. The two procedures are implemented in a multi-core data parallel C++ system. We investigate experimentally the properties of the two procedures on a collection of 17 benchmark datasets, showing that oversearching increases both overfitting and instability.

Keywords: Feature Selection, Decision Trees, Wrapper models, Complete Search

1. Introduction

Feature selection is essential for optimizing accuracy of classifiers, for reducing the data collection effort, for enhancing model interpretability, and for speeding up prediction time (Guyon et al., 2006b). Wrapper models for feature selection have shown superior performance in many contexts (Doak, 1992; Bolón-Canedo et al., 2013). They explore the lattice of feature subsets. For a given feature subset S, a classifier is built over the features and an optimality condition is tested. In this paper, we will consider decision tree classifiers $DT(S)$ built on a training set, and minimization of the misclassification error $err(DT(S))$ evaluated on a search set. Although advanced learning approaches achieve better performances (Caruana and Niculescu-Mizil, 2006; Delgado et al., 2014), decision trees are worth to be investigated, because they are building-blocks of the advanced models, e.g., random forests, or because they represent a good trade-off between accuracy and interpretability.

Complete search of the lattice of feature subsets is know to be NP hard (Amaldi and Kann, 1998). For this reason, heuristics searches are typically adopted in practice. Nevertheless, complete strategies have not to be exhaustive in order to find an optimal subset. In particular, feature subsets that lead to duplicate decision trees can be pruned from the search space. A naïve approach that stores all distinct trees found during the search is, however, unfeasible, since there may be an exponential number of such trees. *Our first contribution* is a non-trivial enumeration algorithm **DTdistinct** of all distinct decision trees built using subsets of the available features. The procedure requires the storage of a linear

number of decision trees in the worst case. The starting point is a recursive procedure for the visit of the lattice of all subsets of features. The key idea is that a subset of features is denoted by the union $R \cup S$ of two sets, where elements in R must *necessarily* be used as split attributes, and elements in S may be used or not. Pruning of the search space is driven by the observation that if a feature $a \in S$ is not used as split attribute by a decision tree built on $R \cup S$, then the feature subset $R \cup S \setminus \{a\}$ leads to the same decision tree. Duplicate decision trees that still pass such a (necessary but not sufficient) pruning condition can be identified through a test on whether or not they use all features in R. An intruiguing result consists in the order of visit of the search space, for which a negligible fraction of trees actually built turn out to be duplicates.

Enumeration of distinct decision trees can be used for searching trees or feature subsets that optimize some criterion. We will consider here optimal feature subsets, i.e., those with the minimum misclassification error on the search set. We introduce the notion of δ-acceptable feature subset, which lead to a decision tree with a misclassification error that departs by at most δ from the minimum error. *Our second contribution* is a complete search procedure $\mathbf{DTaccept}_\delta$ of δ-acceptable and optimal (for $\delta = 0$) feature subsets. The search builds on the enumeration of distinct decision trees. It relies on a key pruning condition that is a conservative extension of the condition above. If for a feature $a \in S$, we have that $err(DT(R \cup S)) \leq \delta + err(DT(R \cup S \setminus \{a\}))$, then $R \cup S \setminus \{a\}$ can be pruned from the search with the guarrantee of missing decision trees whose error is at most δ from the (best) error of visited trees. Hence, visited feature subsets include acceptable ones.

Coupled with the tremendous computational optimization and multi-core parallelization of decision tree induction algorithms, our approach makes it possible to increase the limit of practical applicability of theoretically hard complete searches. We show experimentally that optimal feature subsets can be found in reasonable time for up to 60 features. Beyond such a limit, well-known heuristics approaches may still require a large amount of time. We consider here the classic sequential backward elimination (**SBE**), for which we devise an implementation **DTsbe**, specifically for decision trees, that exploits some of the pruning and computational optimization ideas. *Our third contribution* consists of a white-box version of **SBE** which extends its applicability to large dimensional datasets, and which exhibits, for medium and low dimensional datasets, a computational speedup of up to 100× compared to a black-box approach.

Both $\mathbf{DTaccept}_\delta$ and **DTsbe** are implemented in a multi-core data parallel C++ system, which is made publicly available. We report experiments on 17 benchmark datasets of small-to-large dimensionality. Results confirm previous studies that oversearching increases overfitting. In addition, they also highlight that oversearching increases instability, namely variability of the subset of selected features due to perturbation of the training set.

This paper is organized as follows. First, we recall related work in Section 2. The visit of the lattice of feature subsets is based on a generalization of binary counting enumeration of subsets devised in Section 3. Next, Section 4 introduces a procedure for the enumeration of distinct decision trees as a pruning of the feature subset lattice. Complete search of optimal and acceptable feature subset is then presented in Section 5. Optimization of the sequential backward elimination heuristics is discussed in Section 6. Experimental results are presented in Section 7, with additional tables reported in Appendix A. Finally, we summarize the contribution of the paper in the conclusions.

2. Related Work

Blum and Langley (1997); Dash and Liu (1997); Guyon and Elisseeff (2003); Liu and Yu (2005); Bolón-Canedo et al. (2013) provide a categorization of approaches of feature subset selection along the orthogonal axes of the evaluation criteria, the search strategies, and the machine learning tasks. Common evaluation criteria include filter models, embedded, and wrappers approaches. *Filters* are pre-processing algorithms that select a subset of features by looking at the data distribution, independently from the induction algorithm (Cover, 1977). *Embedded* approaches perform feature selection in the process of training and are specific to the learning algorithm (Lal et al., 2006). *Wrappers* approaches optimize induction algorithm performances as part of feature selection (Kohavi and John, 1997). In particular, training data is split into a building set and a search set, and the space of feature subsets is explored. For each feature subset considered, the building set is used to train a classifier, which is then evaluated on the search set. Search space exploration strategies include (Doak, 1992): *hill-climbing* search (forward selection, backward elimination, bidirectional selection, beam search, genetic search), *random* search (random start hill-climbing, simulated annealing, Las Vegas), and *complete* search. The aim of complete search is to find an optimal feature subset according to an evaluation metric. Typical objectives include minimizing the size of the feature subset provided that the classifier built from it has a minimal accuracy (*dimensionality reduction*), or minimizing the misclassification error of the classifier (*performance maximization*). Finally, feature subset selection has been considered both for classification and clustering tasks. Machine learning models and algorithms can be either treated as *black-boxes* or, instead, feature selection methods can be specific of the model and/or algorithm at hand (*white-box*). White-box approaches are less general, but can exploit assumptions on the model or algorithm to direct and speed up the search.

This paper falls in the category of *complete search using a white-box wrapper model, tailored to decision tree classifiers, for performance maximization.* A feature subset is optimal if it leads to a decision tree with minimal error on the search set. Only complete space exploration can provide the guarantee of finding optimal subsets, whilst heuristics approaches can lead to results arbitrarily worse than the optimal (Murthy, 1998). Complete search is know to be NP hard (Amaldi and Kann, 1998). However, complete strategies do not need to be exhaustive in order to find an optimal subset. For instance, filter models can rely on monotonic evaluation metrics to support Branch & Bound search (Liu et al., 1998). Regarding wrapper approaches, evaluation metrics such as misclassification error lack of the monotonicity property that would allow for pruning the search space in a complete search. Approximate Monotonicity with Branch & Bound (AMB&B) (Foroutan and Sklansky, 1987) tries and tackles this limitation, but it provides no formal guarantee that an optimal feature subset is found. Another form of search space pruning in wrapper approaches for decision trees has been pointed out by Caruana and Freitag (1994), who examine five hillclimbing procedures. They adopt a caching approach to prevent re-building duplicate decision trees. The basic property they observe is reported in a generalized form in this paper as Remark 6. While caching improves on the efficiency of a limited search, in the case of a complete search, it requires an exponential number of decision trees to be stored in cache, while our approach requires a linear number of them. We will also observe that Remark 6 may still leave duplicate trees in the search space, i.e., it is a necessary but not sufficient condition

for enumerating distinct decision trees, while we will provide an exact enumeration and, in addition, a further pruning of trees that cannot lead to optimal/acceptable feature subsets.

This paper significantly extends the preliminary results appeared in Ruggieri (2017) in several directions. First, it improves on the enumeration procedure of distinct decision trees. The new ordering of visits of the search space has a clear theoretical justification, and an overhead (duplicated trees built) close to zero for all experimental datasets. Second, the paper introduces the notion of δ-acceptable feature subsets, which depart from optimal feature subsets by at most δ, and a novel algorithm that further prunes the enumeration of distinct decision trees to find out δ-acceptable feature subsets. Third, the experimental section (and an appendix with additional tables) now includes a comprehensive set of results on a larger collection of benchmark datasets. In particular, the issue of stability of heuristics and complete searches is now discussed, in addition to their accuracy. Fourth, the implementation of all proposed algorithms is now multi-core parallel, and it is publicly available. It reaches run-time efficiency improvements of up to 7× on an 8-core computer.

3. Enumerating Subsets

Let $S = \{a_1, \ldots, a_n\}$ be a set of n elements, with $n \geq 0$. The powerset of S is the set of its subsets: $Pow(S) = \{S' \mid S' \subseteq S\}$. There are 2^n subsets of S, and, for $0 \leq k \leq n$, there are $\binom{n}{k}$ subsets of size k. Figure 1 (left) shows the lattice (w.r.t. set inclusion) of subsets for $n = 3$. The order of visit of the lattice, or, equivalently, the order of enumeration of elements in $Pow(S)$, can be of primary importance for problems that explore the lattice as search space. Well-known algorithms for subset generation produce lexicographic ordering, Grey code ordering, and binary counting ordering (Skiena, 2008). Binary counting maps each subset into a binary number with n bits by setting the i^{th} bit to 1 iff a_i belongs to the subset, and generating subsets by counting from 0 to $2^n - 1$. Subsets for $n = 3$ are generated as $\{\}$, $\{a_3\}$, $\{a_2\}$, $\{a_2, a_3\}$, $\{a_1\}$, $\{a_1, a_3\}$, $\{a_1, a_2\}$, $\{a_1, a_2, a_3\}$. In this section, we introduce a recursive algorithm for a generalization of reverse binary counting (namely, counting from $2^n - 1$ down to 0) that will be the building block for solving the problems of generating distinct decision trees. Let us start by introducing the notation $R \bowtie P = \cup_{S' \in P}\{R \cup S'\}$ to denote sets obtained by the union of R with elements of P. In particular:

$$R \bowtie Pow(S) = \cup_{S' \subseteq S}\{R \cup S'\}$$

consists of the subsets of $R \cup S$ which *necessarily* include R. This generalization of powersets will be crucial later on when we have to distinguish predictive attributes that *must* be used in a decision tree from those that *may* be used. A key observation of binary counting is that subsets can be partitioned between those including the value a_1 and those not including it. For example, $Pow(\{a_1, a_2, a_3\}) = (\{a_1\} \bowtie Pow(\{a_2, a_3\})) \cup (\emptyset \bowtie Pow(\{a_2, a_3\}))$. We can iterate the observation for the leftmost occurrence of a_2 and obtain:

$$Pow(\{a_1, a_2, a_3\}) = (\{a_1, a_2\} \bowtie Pow(\{a_3\})) \cup (\{a_1\} \bowtie Pow(\{a_3\})) \cup \ (\emptyset \bowtie Pow(\{a_2, a_3\})).$$

By iterating again for the leftmost occurrence of a_3, we conclude:

$$\begin{aligned} Pow(\{a_1, a_2, a_3\}) \ = \ & (\{a_1, a_2, a_3\} \bowtie Pow(\emptyset)) \cup (\{a_1, a_2\} \bowtie Pow(\emptyset)) \cup \\ & (\{a_1\} \bowtie Pow(\{a_3\})) \cup (\emptyset \bowtie Pow(\{a_2, a_3\})) \end{aligned}$$

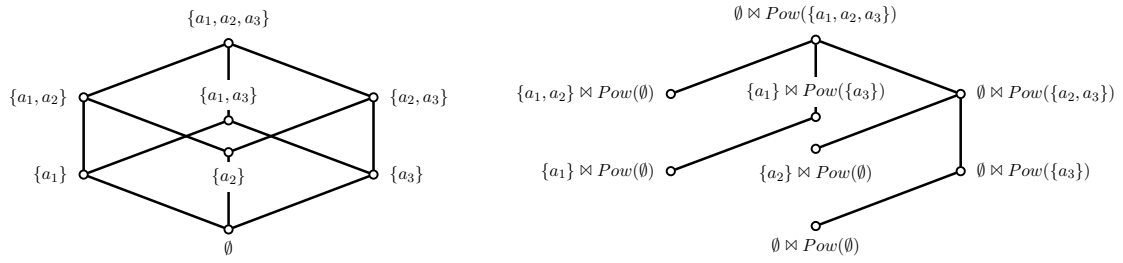


Figure 1: Lattice of subsets and reverse binary counting.

Since $R \bowtie Pow(\emptyset) = \{R\}$, the leftmost set in the above union is $\{\{a_1, a_2, a_3\}\}$. In general, the following recurrence relation holds.

Lemma 1 *Let $S = \{a_1, \ldots, a_n\}$. We have:*

$$R \bowtie Pow(S) = \{R \cup S\} \cup \bigcup_{i=n,\ldots,1} (R \cup \{a_1, \ldots, a_{i-1}\}) \bowtie Pow(\{a_{i+1}, \ldots, a_n\})$$

Proof The proof is by induction on n. The base case $n = 0$ is trivial: $R \bowtie Pow(\emptyset) = \{R\}$ by definition. Consider now $n > 0$. Since $Pow(S) = (\{a_1\} \bowtie Pow(S \setminus \{a_1\})) \cup Pow(S \setminus \{a_1\})$, we have: $R \bowtie Pow(S) = R \bowtie ((\{a_1\} \bowtie Pow(\{a_2, \ldots, a_n\})) \cup (\emptyset \bowtie Pow(\{a_2, \ldots, a_n\})))$. Since the $\bowtie$ operator satisfies:

$$R_1 \bowtie (R_2 \bowtie P) = (R_1 \cup R_2) \bowtie P \quad \text{and} \quad R \bowtie (P_1 \cup P_2) = (R \bowtie P_1) \cup (R \bowtie P_2)$$

we have: $R \bowtie Pow(S) = ((R \cup \{a_1\}) \bowtie Pow(\{a_2, \ldots, a_n\})) \cup (R \bowtie Pow(\{a_2, \ldots, a_n\}))$. By induction hypothesis on the leftmost occurrence of $\bowtie$:

$$\begin{aligned} R \bowtie Pow(S) &= \{R \cup \{a_1\} \cup \{a_2, \ldots, a_n\}\} \cup \\ &\quad \bigcup_{i=n,\ldots,2} (R \cup \{a_1\} \cup \{a_2, \ldots, a_{i-1}\}) \bowtie Pow(\{a_{i+1}, \ldots, a_n\}) \cup \\ &\quad R \bowtie Pow(\{a_2, \ldots, a_n\}) \\ &= \{R \cup S\} \cup \bigcup_{i=n,\ldots,1} (R \cup \{a_1, \ldots, a_{i-1}\}) \bowtie Pow(\{a_{i+1}, \ldots, a_n\}) \end{aligned}$$

■

This result can be readily translated into a procedure **subset(**R**,** S**)** for the enumeration of elements in $R \bowtie Pow(S)$. In particular, since $\emptyset \bowtie Pow(S) = Pow(S)$, **subset(**$\emptyset$**,** S**)** generates all subsets of S. The procedure is shown as Algorithm 1. The search space of the procedure is the tree of the recursive calls of the procedure. The search space for $n = 3$ is reported in Figure 1 (right). According to line 1 of Algorithm 1, the subset outputted at a node labelled as $R \bowtie Pow(S)$ is $R \cup S$. Hence, the output for $n = 3$ is the reverse counting ordering: $\{a_1, a_2, a_3\}$, $\{a_1, a_2\}$, $\{a_1, a_3\}$, $\{a_1\}$, $\{a_2, a_3\}$, $\{a_2\}$, $\{a_3\}$, $\{\}$. Two key properties of Algorithm 1 will be relevant for the rest of the paper.

Algorithm 1 subset(R, S) enumerates $R \bowtie Pow(S)$

```
1: output R ∪ S
2: R' ← R ∪ S
3: S' ← ∅
4: for a_i ∈ S do
5:    R' ← R' \ {a_i}
6:    subset(R', S')
7:    S' ← S' ∪ {a_i}
8: end for
```

Remark 2 *A set $R' \cup S'$ generated at a non-root node of the search tree of Algorithm 1 is obtained by removing an element from the set $R \cup S$ generated at its father node. In particular, $R' \cup S' = R \cup S \setminus \{a\}$ for some $a \in S$.*

The invariant $|R' \cup S'| = |R \cup S|$ readily holds for the loop at lines 4–8 of Algorithm 1. Before the recursive call at line 6, an element of S is removed from R', hence the set $R' \cup S'$ outputted at a child node has one element less than the set $R \cup S$ outputted at its father node.

Remark 3 *The selection order of $a_i \in S$ at line 4 of Algorithm 1 is irrelevant.*

The procedure does not rely on any specific order of selecting members of S, which is a form of *don't care non-determinism* in the visit of the lattice. Any choice generates all elements in $R \bowtie Pow(S)$. In case of an apriori positional order of attributes, namely line 4 is "**for** $a_i \in S$ **order by** i **desc do**", Algorithm 1 produces precisely the reversed binary counting order. However, if the selection order varies from one recursive call to another, then the output is still an enumeration of subsets.

4. Generating All Distinct Decision Trees

We build on the subset generation procedure to devise an algorithm for the enumeration of all distinct decision trees built on subsets of the predictive features.

4.1 On Top-Down Decision Tree Induction

Let us first introduce some notation and assumptions. Let $F = \{a_1, \ldots, a_N\}$ be the set of predictive features, and $S \subseteq F$ a subset of them. We write $T = DT(S)$ to denote the decision tree built from features in S on a fixed training set. Throughout the paper, we make the following assumption on the node split criterion in top-down decision tree induction with univariate split conditions.

Assumption 4 *Let $T = DT(S)$. A split attribute at a decision node of T is chosen as $argmax_{a \in S} f(a, C)$, where $f()$ is a quality measure and C are the cases of the training set reaching the node.*

Our results will hold for any quality measure $f()$ as far as the split attribute is chosen as the one that maximizes $f()$. Examples of quality measures used in this way include

Information Gain (IG), Gain Ratio[1] (GR), and the Gini index, which are adoppted in the C4.5 (Quinlan, 1993) and in the CART systems (Breiman et al., 1984). A second assumption regards the stopping criterion in top-down decision tree construction. Let $stop(S, C)$ be the boolean result of the stopping criterion at a node with cases C and predictive features S.

Assumption 5 *If $stop(S, C) = true$ then $stop(S', C) = true$ for every $S' \subseteq S$.*

The assumption states that either: (1) the stopping criterion does not depend on S; or, if it does, then (2) stopping is monotonic with regard to the set of predictive features. (1) is a fairly general assumption, since typical stopping criteria are based on the size of cases C at a node and/or on the purity of the class attribute in C. We will later on consider one the stopping criteria of C4.5 which halts tree construction if the number of cases of the training set reaching the current node is lower than a minimum threshold m (formally, $stop(S, C)$ is true iff $|C| < m$). (2) applies to criteria which require minimum quality of features for splitting a node. E.g., the C4.5 additional criterion of stopping if IG of all features is below a minimum threshold satisfies the assumption. The following remark, which is part of the decision tree folklore (see e.g., Caruana and Freitag (1994)), states a useful consequence of Assumptions 4 and 5. Let us denote by $features(T)$ the set of split attributes used in the decision tree T. Removing any feature not used in T from the initial set of features does not affect the result of tree building.

Lemma 6 *Let $T = DT(S)$. For every S' such that $S \supseteq S' \supseteq features(T)$, $DT(S') = T$.*

Proof If a decision tree T built from S uses only features from $U = features(T) \subseteq S$, then at any decision node of T it must be $argmax_{a \in S} f(a, C) = argmax_{a \in U} f(a, C)$. Hence, removing any unused attribute in $S \setminus U$ will not change the result of maximizing the quality measure and then, by Assumption 4, the split attribute at a decision node. Moreover, by Assumption 5, a leaf node in T will remain a leaf node for any subset of S. ■

4.2 Enumerating Distinct Decision Trees

Consider a subset of features $R \cup S$, where R must be necessarily used by a decision tree and $S = \{a_1, \ldots, a_n\}$ may be used or not. Let $T = DT(R \cup S)$, and $U = features(T) \supseteq R$ be the features used in split nodes of T. Therefore, $S \cap U = \{a_1, \ldots, a_k\}$ is the set of features in S actually selected as split features, and $S \setminus U = \{a_{k+1}, \ldots, a_n\}$ is the set of features never selected as split features. By Lemma 6, the decision tree T is equal to the one built starting from features $R \cup \{a_1, \ldots, a_k\}$ plus any subset of $\{a_{k+1}, \ldots, a_n\}$. In symbols, all the decision trees for feature subsets in $(R \cup \{a_1, \ldots, a_k\}) \bowtie Pow(\{a_{k+1}, \ldots, a_n\})$ do coincide with T. We will use this observation to remove from the recurrence relation of Lemma 1 some sets

1. Gain Ratio normalizes Information Gain over the Split Information (SI) of an attribute, i.e., GR = IG/SI. This definition does not work well for attributes which are (almost) constants over the cases C, i.e., when SI ≈ 0. (Quinlan, 1986) proposed the heuristics of restricting the evaluation of GR only to attributes with above *average* IG. The heuristics is implemented in the C4.5 system (Quinlan, 1993). It clearly breaks Assumption 4, making the selection of the split attribute dependent on the set S. An heuristics that satisfies Assumption 4 consists of restricting the evaluation of GR only for attributes with IG higher than a *minimum* threshold.

Algorithm 2 DTdistinct(R, S) enumerates distinct decision trees using feature subsets in $R \bowtie Pow(S)$.

```
1: build tree T = DT(R ∪ S)
2: U ← features(T)
3: if R ⊆ U then
4:    output T
5: end if
6: R' ← R ∪ (S ∩ U)
7: S' ← S \ U
8: for a_i ∈ S ∩ U order by rk_frontier(T) do
9:    R' ← R' \ {a_i}
10:   DTdistinct(R', S')
11:   S' ← S' ∪ {a_i}
12: end for
```

in $R \bowtie Pow(S)$ which lead to duplicate decision trees. Formally, when searching for feature subsets that lead to distinct decision trees, the recurrence relation can be modified as:

$$R \bowtie Pow(S) = \{R \cup S\} \cup \bigcup_{i=k,\ldots,1} (R \cup \{a_1, \ldots, a_{i-1}\}) \bowtie Pow(\{a_{i+1}, \ldots, a_n\})$$

since the missing union:

$$\bigcup_{i=n,\ldots,k+1} (R \cup \{a_1, \ldots, a_{i-1}\}) \bowtie Pow(\{a_{i+1}, \ldots, a_n\}) \tag{1}$$

is included in $(R \cup \{a_1, \ldots, a_k\}) \bowtie Pow(\{a_{k+1}, \ldots, a_n\})$, and then it contains sets of features V such that $DT(V) = DT(R \cup S)$. In particular, this implies the following property, which will be useful later on:

$$\begin{gathered} err(DT(R \cup \{a_1, \ldots, a_k\})) = err(DT(V)) \\ \text{for all } V \in \textstyle\bigcup_{i=n,\ldots,k+1} (R \cup \{a_1, \ldots, a_{i-1}\}) \bowtie Pow(\{a_{i+1}, \ldots, a_n\}). \end{gathered} \tag{2}$$

The simplified recurrence relation prunes from the the search space feature subsets that lead to duplicated decision trees. However, we will show in Example 1 that such a pruning alone is not sufficient to generate distinct decision trees only, i.e., duplicates may still exist.

Algorithm 2 provides an enumeration of *all and only the distinct decision trees*. It builds on the subset generation procedure. Line 1 constructs a tree T from features $R \cup S$. Features in the set $S \setminus U$ of unused features in T are not iterated over in the loop at lines 8–12, since those iterations would yield the same tree as T. This is formally justified by the modified recurrence relation above. The tree T is outputted at line 4 only if $R \subseteq U$, namely features *required* to be used (i.e., R) are *actually* used in decision node splits. We will shows that such a test characterizes a uniqueness condition for all feature subsets that lead to a same decision tree. Hence, it prevents outputting more than once a decision tree that can be obtained from multiple paths of the search tree.

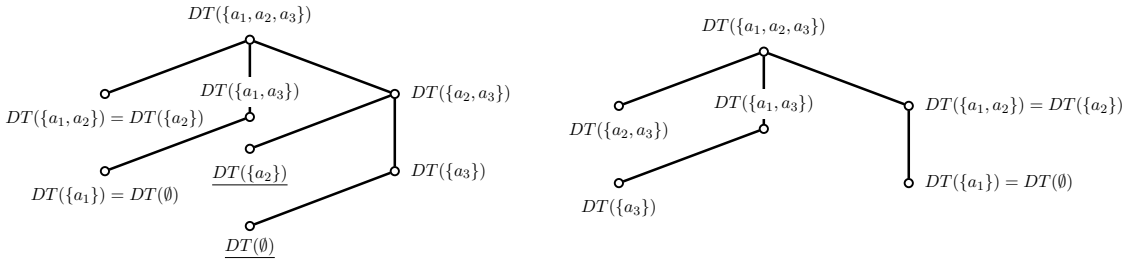


Figure 2: Search spaces of Algorithm 2 for different selection orders.

Example 1 *Let $F = \{a_1, a_2, a_3\}$. Assume that a_1 has no discriminatory power unless data has been split by a_3. More formally, $DT(S) = DT(S \setminus \{a_1\})$ if $a_3 \notin S$. The visit of feature subsets of Figure 1 (right) gives rise to the trees built by* ***DTdistinct(*** $\emptyset$***,*** F***)*** *as shown in Figure 2 (left). For instance, the subset $\{a_1, a_2\}$ visited at the node labelled $\{a_1, a_2\} \bowtie \emptyset$ in Figure 1 (right), produces the decision tree $DT(\{a_1, a_2\})$. By assumption, such a tree is equal to $DT(\{a_2\})$, which is a duplicate tree produced in another node – underlined in Figure 2 (left) – corresponding to the feature set visited at the node labelled $\{a_2\} \bowtie \emptyset$. Another example regarding $DT(\{a_1\}) = DT(\emptyset)$ is shown in Figure 2 (left), together with its underlined duplicated tree. Unique trees for two or more duplicates are characterized by the fact that features appearing to the left of $\bowtie$ must necessarily be used as split features by the constructed decision tree. In the two previous example cases, the nodes underlined output their decision trees, while the other duplicates do not pass the test at line 3 of Algorithm 2.*

The following non-trivial result holds.

Theorem 7 ***DTdistinct(*** R***,*** S***)*** *outputs the distinct decision trees built on sets of features in $R \bowtie Pow(S)$.*

Proof The search space of **DTdistinct** is a pruning of the search space of **subset**. Every tree built at a node and outputted is then constructed from a subset in $R \bowtie Pow(S)$. By Remark 3, the order of selection of $a_i \in S \cap U$ at line 8 is irrelevant, since any order will lead to the same space $R \bowtie Pow(S)$.

Let us first show that decision trees in output are all distinct. The key observation here is that, by line 4, all features in R are used as split features in the outputted decision tree. The proof proceed by induction on the size of S. If $|S| = 0$, then there is at most one decision tree in output, hence the conclusion. Assume now $|S| > 0$, and let $S = \{a_1, \ldots, a_n\}$. By Lemma 1, any two recursive calls at line 10 have parameters $(R \cup \{a_1, \ldots, a_{i-1}\}, \{a_{i+1}, \ldots, a_n\})$ and $(R \cup \{a_1, \ldots, a_{j-1}\}, \{a_{j+1}, \ldots, a_n\})$, for some $i < j$. Observe that a_i is missing as a predictive attribute in the trees in output from the first call, while by inductive hypothesis it must be a split attribute in the trees in output by the second call. Hence, the trees in output from recursive calls are all distinct among them. Moreover, they are all different from $T = DT(R \cup S)$, because recursive calls do not include some feature $a_i \in S \cap U$ that is by definition used in T.

Let us now show that trees pruned at line 8 or at line 4 are already outputted elsewhere, which implies that every distinct decision tree is outputted at least once. First,

by Lemma 6, the trees that would have been outputted in the pruned iterations at line 8 (i.e., for $a_i \in S \setminus U$) are equal to the tree of $T = DT(R \cup S)$. Second, if the tree T is not outputted at line 4, because $R \not\subseteq U$, we have that it is outputted at another node of the search tree. The proof is by induction on $|R|$. For $|R| = 0$ it is trivial, because the the premise $R \not\subseteq U$ does not hold. Let $R = \{a_1, \ldots, a_n\}$, with $n > 0$, and let $a_1, \ldots, a_n$ be in the order they have been added by recursive calls. Fix $R' = \{a_1, \ldots, a_{i-1}\}$ such that $a_i \notin U$ and $R' \subseteq U$. There is a sibling node or a sibling of an ancestor node in the search tree corresponding to a call with parameters R' and $S' \supseteq \{a_{i+1}, \ldots, a_n\} \cup S$. By inductive hypothesis on $|R'| < |R|$, the distinct decision trees with features in $R' \bowtie Pow(S')$ are all outputted, including T because T has split features in $R \cup S \setminus \{a_i\}$ which belongs to $R' \bowtie Pow(S')$. ■

The proof of Theorem 7 does not assume any specific order at line 8 of Algorithm 2. Any order would produce the enumeration of distinct decision trees – this is a consequence of Remark 3. However, the order may impact on the size of the search space.

Example 2 *Reconsider Example 1. The order of selection of a_i's in the visit of Figure 2 (left) is by descending i's. This ordering does not take into account the fact that a_3 has more discriminatory power than a_1, i.e., its presence gives rise to more distinct decision trees. Consider instead having a_3 removed in the rightmost child of the root, e.g., the selection order a_1, a_2, and a_3. The search space of* **DTdistinct($\emptyset$, F)** *is reported in Figure 2 (right). Notice that no duplicate decision tree is built here, and that the size of the search space is smaller than in the previous example. In fact, the node labelled as $DT(\{a_1, a_2\}) = DT(\{a_2\})$ corresponds to the exploration of $\emptyset \bowtie \{a_1, a_2\}$. The a_1 attribute is unused and hence, it is pruned at line 8 of Algorithm 2. The sub-space to be searched consists then of only the subsets of $\{a_1\}$, not all subsets of $\{a_1, a_2\}$. The child node finds $DT(\{a_1\}) = DT(\emptyset)$ and then it stops recursion since there are no used attributes to iterate over at line 8.*

Algorithm 2 adopts a specific order intended to be effective in pruning the search space. In particular, our objective is to minimize the number of duplicated trees built. In fact, even though duplicates are not outputted, building them has a computational cost that should be minimized. Duplicated decision trees are detected through the test $R \subseteq U$ at line 4 of Algorithm 2. Thus, we want to minimize the number of recursive calls **DTdistinct(R', S')** where attributes in R' have lower chances of being used. Since required attributes are removed from R' one at a time at line 9 (and added to the set of possibly used attributes S' at line 11), this means ranking attributes in $S \cap U$ by increasing chances of being used in decision trees of recursive calls. How do we estimate such chances?

Example 3 *Consider the sample decision tree at Figure 3(a). It is built on the set of features $F = \{a_1, a_2, a_3, a_4\}$. Which attributes have the highest chance of being used if included in a subset of F? Whenever included in the subset, a_1 will be certainly used at the root node. In fact, it already maximizes the quality measure on F, hence by Assumption 4 it will also maximizes the quality measure over any subset of F. Assume now a_1 is selected for inclusion. Both a_2 and a_3 will be certainly selected as split attributes, for the same reason as above. Which one should be preferred first? Let us look at the sub-trees rooted at a_2 and*

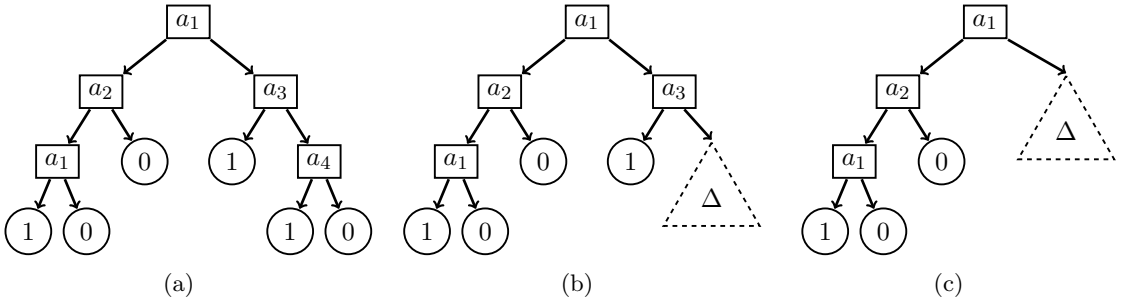


Figure 3: A sample decision tree, and two subtrees replaced with an oracle Δ. Internal nodes are labelled with split attributes, and leaves are labelled with class value.

a_3. If a_2 is selected, then the sub-tree rooted at a_2 uses no further attribute. In fact, a_1 cannot be counted as a further attribute because it is known to be already used. Conversely, if a_3 is selected, the sub-tree rooted at a_3 ensures that attribute a_4 can be used. Therefore, having a_3 gives more chances of using further attributes in decision tree building. Therefore, a_3 should be selected for inclusion before a_2. Finally, sub-trees rooted at a_2 and a_4 use no further attributes, and we break the tie by selecting a_2 before a_4. In summary, the rank of attributes in F with increasing chances of being used is a_4, a_2, a_3, a_1.

Let us formalize the intuition of this example. We define an *a-frontier node* of a decision tree T w.r.t. a set R' of features, a decision node that use $a \notin R'$ for the first time in a path from the root to a node. A frontier node is any a-frontier node, for a feature a. Based on the previous example, we should count the number of attributes that are used in sub-trees rooted at frontier nodes.

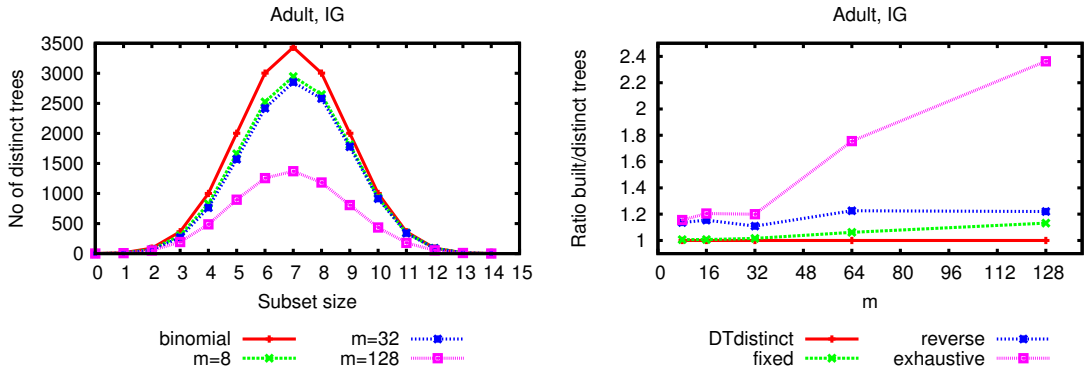
Definition 8 *We define $frontier(T, R', a)$ as the number of distinct features not in R' that are used in sub-trees of T rooted at a-frontier nodes of T w.r.t. R'.*

Notice that attributes in R' are excluded from the counting in *frontier*(). The idea is that R' will include attributes that must already appear somewhere in a decision tree (in between the root and frontier nodes), and thus their presence in the sub-tree rooted at a does not imply further usability of such attributes. We are now in the position to introduce the ranking $rk_{frontier}$ based on ascending *frontier*().

Definition 9 *Let $T = DT(R \cup S)$ and $U = features(T)$. $rk_{frontier}(T)$ is the order $r_1, \ldots, r_k$ of elements in $S \cap U$ such that, for $i = k, \ldots, 1$:*

$$r_i = argmax_{a \in (S \cap U) \setminus \{r_{i+1}, \ldots, r_k\}} \; frontier(T, R \cup \{r_{i+1}, \ldots, r_k\}, a).$$

The definition of the ranking iterates from the last to the first position, and at each step selects the feature which maximizes the *frontier*() measure. Iteration is necessary due to the fact that the frontier nodes depend on the features selected at the previous step. Intuitively, the ordering try and keep as much as possible sub-trees with large sets of not yet ranked attributes. This is in line with the objective of ranking features based on increasing chances of being used in decision trees of recursive calls.



(a) Distribution of distinct decision trees.

(b) Ratio duplicate/distinct decision trees.

Figure 4: Distinct decision trees and overhead of **DTdistinct** on the Adult dataset.

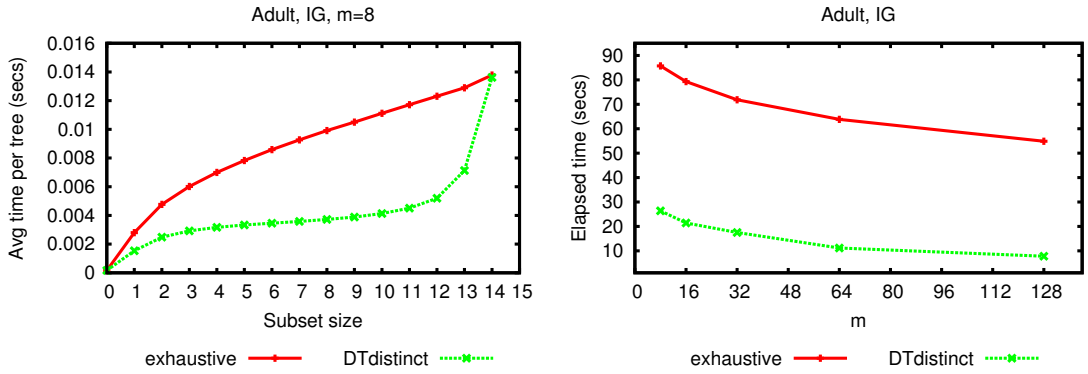
Example 4 *Reconsider Example 3 and the decision tree $T = DT(F)$ in Figure 3(a). Assume that $R = \emptyset$ and $S = F = \{a_1, a_2, a_3, a_4\}$. The set of used features is $U = F$. The rank of attributes starts by defining $r_4 = argmax_{a \in \{a_1,a_2,a_3,a_4\}}\ frontier(T, \emptyset, a)$ which is trivially the split attribute a_1 at root node of T – the only frontier node of T w.r.t. $\emptyset$. Next, $r_3 = argmax_{a \in \{a_2,a_3,a_4\}}\ frontier(T, \{a_1\}, a)$ is defined by looking at the frontier nodes, which are those using a_2 and a_3. As discussed in Example 3, $frontier(T, \{a_1\}, a_2) = 1$ and $frontier(T, \{a_1\}, a_3) = 2$. Thus, we have $r_3 = a_3$. At the third step, $r_2 = argmax_{a \in \{a_2,a_4\}}\ frontier(T, \{a_1, a_2\}, a)$ is defined by looking at the frontier nodes using a_2 and a_4. Both have a frontier of 1, so we fix $r_2 = a_2$. Finally, r_1 must be necessarily be a_4. Summarizing, the ordering provided by $rk_{frontier}(T)$ is a_4, a_2, a_3, a_1 as stated in Example 3.*

Let us now point out some properties of **DTdistinct**.

Property 1: linear space complexity. Consider the set F of all features, with N elements. **DTdistinct($\emptyset$, F)** is computationally linear in space (per number of trees built) in N. In fact, there are at most N nested calls, since the size of the second parameter decreases at each call. Summarizing, at most N decision trees are built and stored in the nested calls, i.e., space complexity is linear in space per number of trees built. An exhaustive search would instead keep in memory the distinct decision trees built in order to check whether a new decision trees is a duplicate. Similarly will do approaches based on complete search with some form of caching of duplicates (Caruana and Freitag, 1994). Those approaches would require exponential space, as shown in the next example.

Example 5 *Let us consider the well-known Adult dataset[2] (Lichman, 2013), consisting of 48,842 cases, and with $N = 14$ predictive features and a binary class. Figure 4(a) shows, for the IG split criterion, the distributions of the number of distinct decision trees w.r.t. the size of feature subset. The distributions are plotted for various values of the stopping parameter m (formally, $stop(S, C)$ is true iff $|C| < m$). For low m values, the distribution approaches the binomial, hence, the number of distinct decision trees approaches 2^N.*

2. See Section 7 for the experimental settings.



(a) Average elapsed time for tree building.

(b) Total elapsed time.

Figure 5: **DTdistinct** elapsed times on the Adult dataset.

Property 2: reduced overhead. Algorithm 2 may construct duplicated decision trees at line 1, which, however, are not outputted due to the test at line 3. It is legitimate to ask ourselves how many duplicates are constructed. Or, in other words, how effective is the selection orderd at line 8 based on $rk_{frontier}()$. Formally, we measure such an overhead as the ratio of all decision trees constructed at line 1 over the number of distinct decision trees. An ideal ratio of 1 means that no duplicate decision tree is constructed at all.

Example 6 (Ctd.) *Figure 4(b) shows the overhead at the variation of m for three possible orderings of selection at line 8 of Algorithm 2. One is the the ordering stated by* ***DTdistinct****, based on $rk_{frontier}()$. The second one is the reversed order, namely $a_n, \ldots, a_1$ for $rk_{frontier}()$ being $a_1, \ldots, a_n$. The third one is based on assigning a static index $i \in [1, N]$ to features a_i's, and then ordering over i. The $rk_{frontier}()$ ordering used by* ***DTdistinct*** *is impressively effective, with a ratio of almost* 1 *everywhere.*

The effectiveness of the $rk_{frontier}()$ ordering will be confirmed in the experimental section. Figure 4(b) also reports the ratio of the number of trees in an exhaustive search (which are 2^n for n features) over the number of distinct trees. Smaller m's lead to a smaller ratio, because built trees are larger in size and hence there are more distinct ones. Thus, for small m values, pruning duplicate trees does not guarantee alone a considerably more efficient enumeration than exhaustive search. The next property will help in such cases.

Property 3: feature-incremental tree building. The construction of each single decision tree at line 1 of Algorithm 2 can be speed up by Remark 2. The decision tree T' at a child node of the search tree differs from the decision tree T built at the father node by one missing attribute a_i. The construction of T' can then benefit from this observation. In the implementation of Algorithm 2, we first recursively clone T and then re-build only sub-trees rooted at nodes whose split attribute is a_i. This requires maintaining in memory the trees built along recursive calls, which gives rise to the linear space complexity (in the number of trees) of the algorithm. However, it allows for incrementally building T' from T.

Example 7 (Ctd.) *Figure 5(a) shows the average elapsed time required to built a decision tree w.r.t. the size of feature subset, for the fixed parameter $m = 8$. The exhaustive search requires an average time linear in the size of the feature subset. Due to incremental tree building,* ***DTdistinct*** *requires instead a sub-linear time.*

Property 4: multi-core parallelization. The loop at lines 8–12 of Algorithm 2 has no strong dependency between iterations, and it can be easily parallelized on multi-core platforms. Our implementation of Algorithm 2 runs the loop at lines 8–12 in task parallel threads. The construction of the tree at line 1 is also executed using nested task parallelism. For fair comparison, the implementation of exhaustive search exploits nested parallelism as well in the enumeration and in the construction of trees.

Example 8 (Ctd.) *Figure 5(b) contrasts the total elapsed times of exhaustive search and* ***DTdistinct****. For small values of m, the number of trees built by exhaustive search approaches the number of distinct decision trees (see Figure 4(b)). Nevertheless, the running time of* ***DTdistinct*** *is constantly better than the exhaustive search. This improvement is due to the incremental building of decision trees. The computational efficiency in terms of absolute elapsed time is, in addition, due the effectiveness of parallel implementation, which in the example at hand runs on an low-cost 8-core machine.*

5. Optimal and Acceptable Feature Subsets

According to the wrapper model for feature selection (John et al., 1994; Kohavi and John, 1997), we assume that the available training set is split into a *building* set, used to build decision trees $T = DT(S)$ on subsets S of features F, and a *search* set, used to evaluate performances of the decision tree and, *a fortiori*, of the subset S. We assume that performances are evaluated through the misclassification error $err(T)$ of T on the search set. In our experiments, misclassification error is computed using the C4.5's distribution imputation method[3]. A feature subset is optimal if it has the minimum error among all subsets. It is δ-acceptable, if its error is lower or equal than the optimal one plus δ.

Definition 10 *Let F be a set of features. We define $err_{opt} = min_{S \subseteq F} err(DT(S))$, and call it the optimal error. For $\delta \geq 0$, a feature subset $S_{acc} \subseteq F$ is δ-acceptable if:*

$$err(DT(S_{acc})) \leq \delta + err_{opt}.$$

When $\delta = 0$, we call it an optimal feature subset. Finally, $DT(S_{acc})$ is called a δ-acceptable decision tree.

The *δ-acceptable feature subset problem* consists of finding a δ-acceptable feature subset. In particular, for $\delta = 0$, it consists of finding an optimal feature subset.

3. Predictions of instances with no missing value follows a path from the decision tree root to a leaf node. For instances with missing value of the attribute tested at a decision node, several options are available (Saar-Tsechansky and Provost, 2007; Twala, 2009). In C4.5, all branches of the decision node are followed, and the prediction of a leaf in a branch contributes in proportion to the weight of the branch's child node (fraction of cases reaching the decision node that satisfy the test outcome of the child). The class value predicted for the instance is the one with the largest total contribution.

Algorithm 3 DTaccept$_\delta(R, S)$ finds a δ-acceptable feature subset S_{acc} in $R \bowtie Pow(S)$.

```
1: // e_acc is initialized outside to ∞
2: build tree T = DT(R ∪ S)
3: U ← features(T)
4: if err(T) ≤ e_acc then
5:    e_acc ← err(T)
6:    S_acc ← U
7: end if
8: U ← greedy_δ(U, R, S)
9: R' ← R ∪ (S ∩ U)
10: S' ← S \ U
11: for a_i ∈ S ∩ U order by rk_frontier(T) do
12:    R' ← R' \ {a_i}
13:    DTaccept(R', S')
14:    S' ← S' ∪ {a_i}
15: end for
```

Algorithm 4 greedy$_\delta(U, R, S)$

```
1: W ← U
2: for a_i ∈ S ∩ U order by rk_frontier(T) do
3:    Ŵ ← W \ {a_i}
4:    if e_acc ≤ lberr(R, S ∩ Ŵ) + δ then
5:       W ← Ŵ
6:    end if
7: end for
8: return W
```

Algorithm 3 builds on the procedure for the enumeration of distinct decision trees by implementing a further pruning of the search space. In particular, a call **DTaccept$_\delta(R, S)$** finds a δ-acceptable feature subset S_{acc} among all subsets in $R \bowtie Pow(S)$. The global variable e_{acc} stores the error of the best δ-acceptable tree found so far, and it is initialized outside the call to ∞. The structure of **DTaccept$_\delta$** follows the one of Algorithm 2, from which it differs in two main points.

The first difference regards lines 4-8, which update the best acceptable decision tree found so far instead of just outputting T. The update takes place if the error of T is lower or equal[4] than the best error found so far, so that the best acceptable decision tree among those found is retained. e_{acc} and the acceptable set of attributes S_{acc} are also updated.

The second difference regards the set U of used features which, at line 8, is possibly changed (to a smaller one) by the call **greedy$_\delta(U, R, S)$**. Such a function tries and relax the pruning of features subsets in formula (2) in order to include additional attributes. Let $S = \{a_1, \ldots, a_n\}$. In particular, we aim at finding a minimal set $W = \{a_1, \ldots, a_k\} \subseteq S \cap U$

4. We break ties in favor of smaller feature subsets.

such that:

$$e_{acc} \leq err(DT(V)) + \delta \quad \text{for all } V \in \bigcup_{i=n,\ldots,k+1}(R \cup \{a_1,\ldots,a_{i-1}\}) \bowtie Pow(\{a_{i+1},\ldots,a_n\}). \tag{3}$$

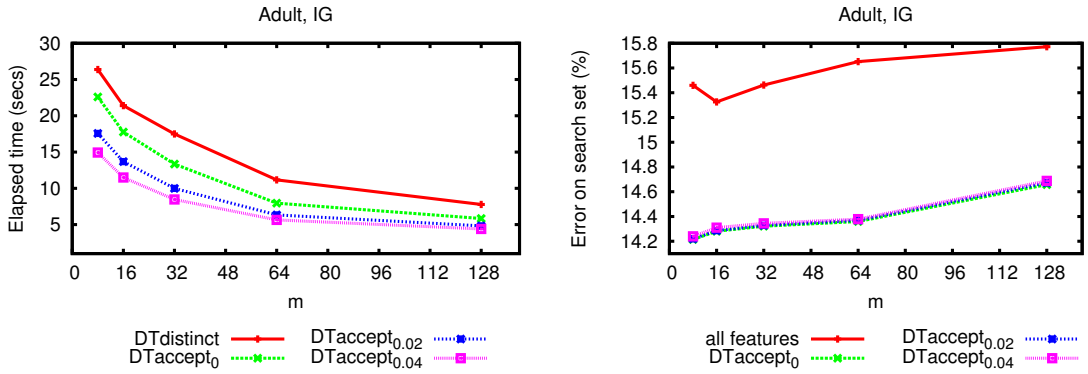
If such a condition holds[5], we can prune from the search space the feature subsets in the quantifier of the condition, because even in the case that a tree with optimal error is pruned this way, the best error found so far e_{acc} is within the δ bound from the error of the pruned decision tree. Practically, this means that we can continue using W in the place of U in the rest of the search – and, in fact, lines 9–15 of $\mathbf{DTaccept}_\delta$ coincide with lines 6–12 of **DTdistinct**. The search space is pruned if $W \subset U$, because the loop at lines 11-15 iterates over a smaller set of features. The approach that $\mathbf{greedy}_\delta$ adopts for determining $\{a_1,\ldots,a_k\}$ is a greedy one, which tries and remove one candidate feature from $S \cap U$ at a time. The order of removal is by[6] $rk_{frontier}(T)$ as in the main loop of **DTdistinct**. The function $\mathbf{greedy}_\delta$ relies on a lower-bound function $lberr()$ for which the test condition[7] $e_{acc} \leq lberr(R, \{a_1,\ldots,a_k\}) + \delta$ is required to imply that (3) holds. This is true when:

$$lberr(R, \{a_1,\ldots,a_k\}) \leq err(DT(V)) \quad \text{for all } V \in \bigcup_{i=n,\ldots,k+1}(R \cup \{a_1,\ldots,a_{i-1}\}) \bowtie Pow(\{a_{i+1},\ldots,a_n\}). \tag{4}$$

hence, the adjective "lower-bound" function for $lberr()$. Our lower-bound function is defined as follows for a candidate $\{a_1,\ldots,a_k\}$. We start from the decision tree $T = DT(R \cup S) = DT(R \cup (S \cap U))$ and remove sub-trees in T rooted at frontier nodes with split features in $(S \cap U) \setminus \{a_1,\ldots,a_k\}$. Let T' be the partial tree obtained and call the removed frontier nodes the "to-be-expanded" nodes. The features used in T' belong to $R \cup \{a_1,\ldots,a_k\}$, which is included in any V quantified over in (4). Due to the Assumptions 4-5, any tree $DT(V)$ may differ from T only at the nodes to-be-expanded and their sub-trees, i.e., T' is a sub-tree of $DT(V)$. At the best, the misclassification error[8] of the sub-trees in V that expand T' will be 0[9]. Thus, we have $err(T') \leq err(DT(V))$, where $err(T')$ adds 0 as misclassification error at nodes to-be-expanded. Since T' is defined only starting from T, and not from any specific V, we then define $lberr(R, \{a_1,\ldots,a_k\}) = err(T')$ and have that (4) holds.

Example 9 *Consider again the sample decision tree T at Figure 3(a). Assume $R = \emptyset$ and $S = S \cap U = \{a_1, a_2, a_3, a_4\}$. Also, let e_{acc} the best misclassification error on the*

5. A direct extension of (2) is to require $err(DT(R \cup S)) \leq err(DT(V)) + \delta$, i.e., that the error of the last built tree is within the δ bound from the error of any tree over quantified attributes V. The relaxed condition (3) allows for a better pruning, namely that the best error found so far is within the δ bound.
6. The rationale is to try and remove first features that lead to minimal changes in the decision tree, and, consequently, to minimal differences in its misclassification error. This is a direct generalization of the approach of **DTdistinct** of removing unused features, which lead to no change in misclassification.
7. Cf. line 4 of Algorithm 4, where $\{a_1,\ldots,a_k\}$ is $S \cap \hat{W}$.
8. For cases in the search set with missing values, the sub-trees in V that expand T' will contribute to the weight of the actual class value only.
9. Since decision tree error is always not lower than the Bayes error (Devroye et al., 1996), a better lower bound can be obtained at each node to-be-expanded as: $1 - \frac{1}{|I|}\sum_{\mathbf{x} \in I} g(\mathbf{x})$. Here, I includes the instances in the search set reaching the node to-be-expanded, and $g(\mathbf{x}) = max_c\ k^c_{\mathbf{x}}/n_{\mathbf{x}}$, where $n_{\mathbf{x}}$ is the number of instances in I whose predictive attribute values are the same as for $\mathbf{x}$, and $k^c_{\mathbf{x}}$ is the number of such instances that also have class value equal to c. Unfortunately, the computational cost of calculating this lower bound function makes it impractical.



(a) Elapsed running time.

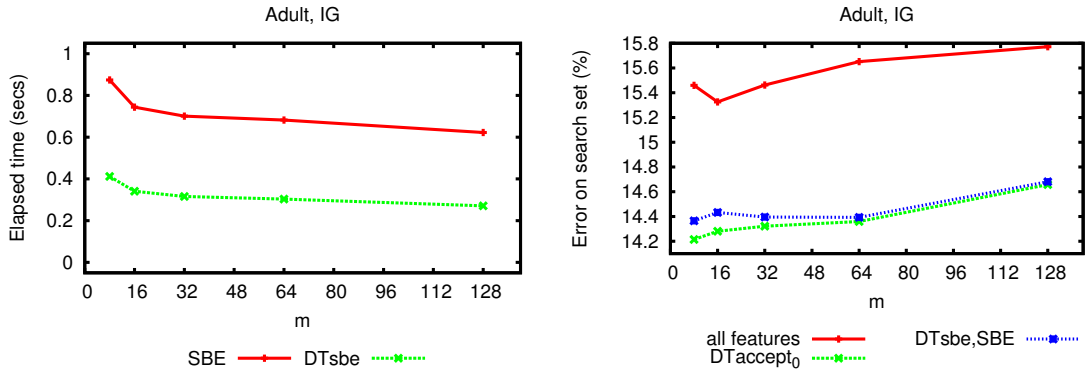
(b) Misclassification error on search set.

Figure 6: **DTaccept**$_\delta$ elapsed times and errors on the Adult dataset.

search set found so far. ***greedy***$_\delta$ *will consider removing attributes in the order provided by* $rk_{frontier}(T)$*, which is* a_4, a_2, a_3, a_1 *(see Example 4). At the first step, the sub-tree rooted at* a_4 *is tentatively removed, and replaced with an oracle with zero error on the search set, as shown in Figure 3(b). If the error* $lb = lberr(\emptyset, \{a_4\})$ *of such a tree is such that* $e_{acc} \leq lb + \delta$*, we can commit the removal of* a_4*. Assume this is the case. In the next step, the sub-tree rooted at* a_2 *is also tentatively removed and replaced with an oracle. Again, if the error* $lb = lberr(\emptyset, \{a_4, a_2\})$ *of such a tree is such that* $e_{acc} \leq lb + \delta$*, we can commit the removal of* a_2*. Assume this is not the case, and* a_2 *is not removed. In the third step, the sub-tree rooted at* a_3 *is tentatively removed and replaced with an oracle, as shown in Figure 3(c). If the error* $lb = lberr(\emptyset, \{a_4, a_3\})$ *of such a tree is such that* $e_{acc} \leq lb + \delta$*, we can commit the removal of* a_3*. Assume this is the case. In the last step, we try and remove the sub-tree rooted at* a_1*, and replace it with an oracle. The error of such a tree is* $lb = lberr(\emptyset, \{a_4, a_3, a_1\}) = 0$*. Condition* $e_{acc} \leq lb + \delta$ *does not hold (otherwise, it would have been satisfied at the second step as well). In summary,* ***greedy***$_\delta$ *returns* $\{a_2, a_1\}$*, whilst* $\{a_4, a_3\}$ *can be safely not iterated over at step 11 of Algorithm 3.*

The calculation of $err(T')$ can be computationally more expensive than constructing the distinct decision trees for the pruned sets V in (4), especially in presence of cases with missing values (due to the costly distribution imputation method). Thus, in our implementation, the **greedy**$_\delta$ function is called only if there is chance of removing some features from $S \cap U$. Such a chance is estimated in an efficient way by considering the removal of the feature ranked first by $rk_{frontier}(T)$. Such a feature has the smallest frontier, which can be used as an estimate of the reduced misclassification error when removing the feature. If we cannot even remove such a feature because the resulting tree would have an estimated error lower than $e_{acc} - \delta$, we do not run the whole **greedy**$_\delta$ function.

Example 10 *Reconsider Example 5. Figure 6(a) shows the elapsed running times of* ***DTaccept***$_\delta$***(***$\emptyset$***,*** F***)*** *where* F *is the set of all features of the Adult dataset. Results are shown for several values of the parameter* δ*. It is worth noting that, for* $\delta = 0$*, the elapsed time is smaller than the enumeration of distinct trees by* ***DTdistinct****. In fact, when we search for*



(a) Elapsed running time.

(b) Misclassification error on search set.

Figure 7: **SBE** and **DTsbe** elapsed times and errors on the Adult dataset.

*an optimal decision tree, **DTaccept**$_0$ prunes from the search space those (distinct) decision trees for which the lower bound on the misclassification error is higher than the best error found during the search. Figure 6(b) shows the misclassification error on the search set of the decision tree built on features returned by **DTaccept**$_\delta$ and on all features F. The difference between the error of $DT(F)$ and the optimal error ($\delta = 0$) provides the range of possible errors of features subsets selected by heuristics approaches. Notice that the actual error of acceptable features found for $\delta = 0.02$ and $\delta = 0.04$ is very close to the optimal error, and distant by the theoretical limits of $+2\%$ and $+4\%$ respectively.*

As a final notice, we observe that our approach can be easily adapted to other variants of the feature selection problem. One variant consists of regularizing the misclassification error with a penalty ϵ for every feature in a subset. In such a case, the only changes in Algorithm 3 would be: the test at line 4 becomes $err(T) + |U| \cdot \epsilon \leq e_{acc}$; the assignment at line 5 becomes $e_{acc} \leftarrow err(T) + |U| \cdot \epsilon$. Regarding the lower bound function $lberr(R, S \cap \hat{W})$ called at line 4 of **greedy**$_\delta$, by adding the penalty $|S \cap \hat{W}| \cdot \epsilon$ we still obtain a lower bound on the regularized misclassification of trees built from subsets V pruned in (4).

6. A White-Box Optimization of SBE

On the practical side, **DTaccept**$_\delta$ does not scale to large dimensional datasets. Moreover, acceptability/optimality on the search set may be obtained at the cost of overfitting (Doak, 1992; Reunanen, 2003) and instability (Nogueira and Brown, 2016). We will discuss these issues in the experimental section. The ideas underlying our approach, however, can impact also on the efficiency of well-performing heuristics searches. In particular, we consider here the cornerstone sequential backward elimination (**SBE**) heuristics. It starts building a decision tree T using the set S of all features. For every $a \in S$, a decision tree T_a is built using features in $S \setminus \{a\}$. If no T_a's has a lower or equal error on the search set than T, the algorithm stops returning S as the subset of selected features. Otherwise, the procedure is repeated removing $\hat{a}$ from S, where $T_{\hat{a}}$ is the tree with the smallest error. In summary, features are eliminated one at a time in a greedy way. **SBE** is a black-box approach. The

procedure applies to any type of classifier. A white-box optimization can be devised for decision tree classifiers that satisfy the assumptions of Section 4.1. The optimization relies on Lemma 6. Let $S \setminus U$ be the set of features not used in the current decision tree T. For $a \in S \setminus U$, it turns out that $T_a = T$. Thus, only trees T_a for $a \in U$ need to be built and evaluated at each step, saving the construction of $|S \setminus U|$ decision trees. The optimization can also be iterated at the inner steps. Assume that the tree T has been obtained by removing an attribute $\hat{b}$ in the previous iteration of the algorithm, i.e., $T = T'_{\hat{b}}$. If for an attribute $a \in U$, T'_a does not use $\hat{b}$, i.e., $\hat{b} \notin \mathit{features}(T'_a)$, then removing a from $T = T'_{\hat{b}}$ turns out to produce T'_a again, which does not need to be recomputed. A final optimization regards incremental tree building, which can be adopted as in the case of **DTdistinct**. We call the resulting white-box optimizated algorithm **DTsbe**.

Example 11 *Figure 7(a) contrasts the elapsed running times of **SBE** and **DTsbe** on the Adult dataset. The efficiency improvement of **DTsbe** over **SBE** is consistently in the order of 2×. Figure 7(b) shows instead their search errors, which are obviously the same. Errors are very close to the optimal error provided by* $\boldsymbol{DTaccept}_0$.

7. Experiments

7.1 Datasets and Experimental Settings

We perform experiments on 17 small and large dimensional benchmark datasets publicly available from the UCI ML repository (Lichman, 2013). Some of the datasets have been used in the NIPS 2003 challenge on feature selection, and are described in detail by Guyon et al. (2006a). Table 1 reports the number of instances and features of the datasets. Following (Kohavi, 1995; Reunanen, 2003), we adopt repeated stratified 10-fold cross validation, with a repetition of 10 times[10]. For each hold-out fold, feature selection is performed by splitting the 9-fold training set into 70% building set and 30% search set using stratified random sampling. The building set is used for constructing decision trees, and the search set for evaluating them. The search error of a feature selection strategy is the average misclassification error of the feature subset selected by the strategy on the 100 search sets. For a given selected feature subset, the actual decision tree is built on the whole 9-fold training set, and evaluated on the hold-out 1-fold test set. The cross-validation error is the average misclassification error on the 100 hold-out folds. Thus, optimal feature subsets are those with minimium search error, but not necessarily with minimum cross-validation error. Different feature selection strategies are applied to the same triples of building, search, and test sets. A baseline for comparison in cross-validation will be the decision tree $DT(F)$ built on all available features F over the 9-fold training set without any feature selection.

Information Gain (IG) is used as quality measure in node splitting. No form of tree simplification (Breslow and Aha, 1997) is considered in this section. Appendix A reports additional results about the impact of C4.5 error-based pruning (EBP) and the Gain Ratio (GR) quality measure. The same conclusions of this section extend to those settings.

10. Cross-validation is a nearly unbiased estimator (Kohavi, 1995), yet highly variable for small datasets. Kohavi's recommendation is to adopt a stratified version of it. Variability of the estimator is accounted for by adopting repetitions (Kim, 2009). An alternative method was proposed by Fan (2016).

Table 1: Experimental datasets, and average elapsed running times (seconds). IG and m=2.

dataset	inst.	feat.	DTsbe	SBE	ratio	DTaccept$_0$
Adult	48,842	14	0.56	1.12	0.503	33.22
Letter	20,000	16	0.38	0.69	0.545	216.86
Hypo	3,772	29	0.07	0.25	0.275	70.99
Ionosphere	351	34	0.01	0.10	0.112	1.19
Soybean	683	35	0.05	0.22	0.249	1955.21
Anneal	898	38	0.01	0.09	0.132	33.40
Census	299,285	40	31.13	107.76	0.289	>1h
Sonar	208	60	0.01	0.41	0.029	21.88
Coil2000	9,822	85	2.17	>1h	-	>1h
Clean1	476	166	0.14	21.62	0.007	>1h
Clean2	6,598	166	4.02	164.07	0.025	>1h
Madelon	2,600	500	10.38	>1h	-	>1h
Isolet	7,797	617	208.62	>1h	-	>1h
Gisette	7,000	5,000	69.47	>1h	-	>1h
P53mutants	31,420	5,408	483.93	>1h	-	>1h
Arcene	900	10,000	21.70	>1h	-	>1h
Dexter	2,600	20,000	979.05	>1h	-	>1h

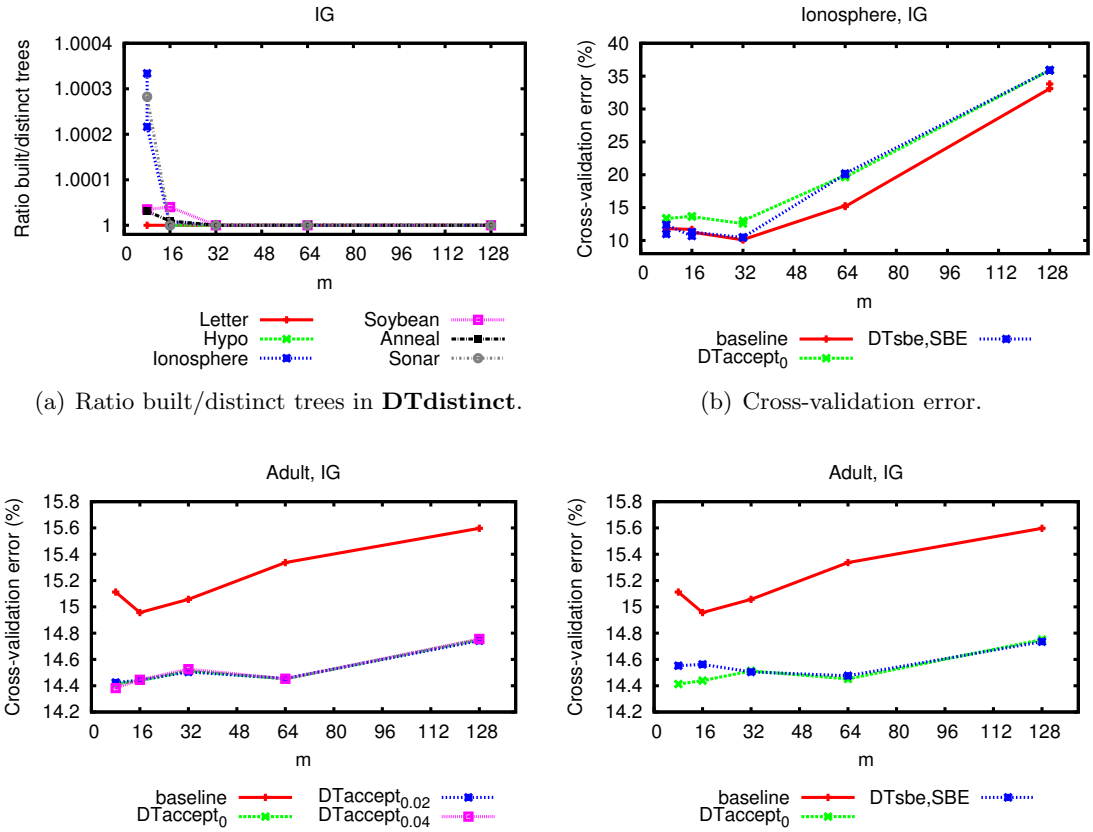
All procedures described in this paper are implemented by extending the YaDT system (Ruggieri, 2002, 2004; Aldinucci et al., 2014). It is a state-of-the-art main-memory C++ implementation of C4.5 with many algorithmic and data structure optimizations as well as with multi-core data parallelism in tree building. The extended YaDT version is publicly available from the author's home page: http://pages.di.unipi.it/ruggieri. Test were performed on a PC with Intel 8 cores i7-6900K@3.7 GHz, without hyperthreading, 16 Gb RAM, and Windows Server 2016 OS.

7.2 On the Search Space Visit of DTdistinct

For the Adult dataset, the order $rk_{frontier}(T)$ used in **DTdistinct** was impressively effective in producing a visit of the feature subset space with a negligible fraction of duplicates (see Figure 4(b)). Figure 8(a) shows the ratio of the number of built trees over the number of distinct trees for other small to medium dimensionality datasets – namely, those for which **DTdistinct** terminates within a time-out of 1h. The ratios are below 1.00035, i.e., there are at most 0.035% duplicate trees built by **DTdistinct**.

7.3 Elapsed Running Times

Table 1 compares the elapsed running times of **SBE** and its computational optimization **DTsbe**. For all datasets, the tree building stopping parameter m is set to the small value 2, which is the default for C4.5. The ratio of elapsed times shows a speedup of up to 100×. For large dimensional datasets, the black-box **SBE** does not even terminate within a time-out of 1h. This is a relevant result for machine learning practitioners, extending the applicability of the **SBE** heuristics, a key reference in feature selection strategies.



(a) Ratio built/distinct trees in **DTdistinct**.

(b) Cross-validation error.

(c) Cross-validation errors.

(d) Cross-validation error.

Figure 8: Procedure performances.

Table 1 also reports the elapsed times for **DTaccept**$_0$, which finds an optimal feature subset. **DTaccept** makes it possibile to search for optimal feature subsets in a reasonable amount of time for datasets with up to 60 features.

7.4 Complete Search or Heuristics? Comparison on Accuracy

Optimality of feature subsets found by complete search is with reference to the search set, but there is no guarrantee that it generalizes to unseen cases, i.e., on cross-validation errors.

Figures 8(c) and 8(d) show cross-validation errors on the Adult dataset for decision trees constructed on all available features (baseline), and on features selected by **DTaccept**$_0$ (optimal feature subset), **DTaccept**$_{0.02}$ and **DTaccept**$_{0.04}$ (0.02 and 0.04-acceptable feature subsets), and **SBE** (same as **DTsbe**). For such a dataset, the optimal feature subset has the best cross-validation error, the 0.02- and 0.04-acceptable feature subsets are very close to it, and **SBE** has a similar performance except for low m values.

Figures 8(b) highlights a different result. For the Ionosphere dataset and large m values, the best performance is for the baseline, then for **SBE**, and finally for the optimal feature subset. Ionosphere is a small dataset, hence using all instances for training (particularly for

Table 2: Search and cross-validation errors (mean ± stdev). IG and m=2. Best method in **bold**. "⋆" labels methods not different from the best one at 95% significance level.

	search errors			cross-validation errors		
dataset	**baseline**	**DTsbe**	**DTaccept$_0$**	**baseline**	**DTsbe**	**DTaccept$_0$**
Adult	16.14 ± 0.30	14.29 ± 0.27	**14.14 ± 0.21**	15.72 ± 0.45	14.44 ± 0.47	**14.36 ± 0.44**
Letter	14.76 ± 0.46	13.93 ± 0.45	**13.82 ± 0.41**	12.52 ± 0.74	12.35 ± 0.73⋆	**12.34 ± 0.79**
Hypo	0.59 ± 0.32	0.31 ± 0.19	**0.27 ± 0.17**	**0.45 ± 0.35**	0.57 ± 0.45	0.58 ± 0.47
Ionosph.	11.31 ± 2.85	5.67 ± 1.86	**2.15 ± 1.16**	12.20 ± 4.82	**9.86 ± 5.60**	12.22 ± 5.10
Soybean	16.51 ± 2.92	8.07 ± 1.86	**5.91 ± 1.21**	13.51 ± 4.17	12.23 ± 4.02⋆	**11.68 ± 4.03**
Anneal	1.59 ± 0.83	0.97 ± 0.52	**0.69 ± 0.44**	**0.90 ± 0.90**	1.58 ± 1.19	2.18 ± 1.72
Census	6.17 ± 0.07	**4.96 ± 0.09**	-	6.04 ± 0.11	**4.99 ± 0.13**	-
Sonar	27.21 ± 6.04	14.60 ± 4.00	**2.19 ± 1.50**	**26.24 ± 9.49**	27.16 ± 9.27⋆	27.72 ± 9.84⋆
Coil2000	9.22 ± 0.44	**7.11 ± 0.32**	-	9.06 ± 0.67	**8.73 ± 0.61**	-
Clean1	22.77 ± 3.74	**10.10 ± 2.50**	-	**17.15 ± 5.30**	19.42 ± 6.55	-
Clean2	4.14 ± 0.60	**1.53 ± 0.39**	-	3.11 ± 0.64	**2.83 ± 0.74**	-
Madelon	30.02 ± 3.72	**15.72 ± 2.10**	-	25.44 ± 3.95	**22.72 ± 3.38**	-
Isolet	18.55 ± 0.91	**11.83 ± 0.64**	-	17.13 ± 1.33⋆	**16.88 ± 1.29**	-
Gisette	6.62 ± 0.64	**3.08 ± 0.40**	-	6.23 ± 0.93	**5.75 ± 0.86**	-
P53mut.	0.61 ± 0.08	**0.31 ± 0.04**	-	0.60 ± 0.14	**0.52 ± 0.11**	-
Arcene	48.54 ± 3.37	**31.28 ± 2.36**	-	48.40 ± 5.38⋆	**48.20 ± 5.53**	-
Dexter	47.99 ± 1.79	**32.04 ± 1.41**	-	48.85 ± 3.54⋆	**48.24 ± 3.13**	-
wins/⋆	0/0	10/0	7/0	4/3	10/3	3/1

large m values) results in a better strategy than splitting training into building and search sets for feature selection.

Table 2 reports the search errors and the cross-validation errors for all datasets. Here and in the following tables, the best method is shown in bold. Other methods are labelled with "⋆" if the null hypothesis in a paired difference t-test with the best method at 95% significance level cannot be rejected. Two conclusions can be made from the table.

First, heuristics search performs good in wrapper feature selection. In fact, the search error of **DTsbe** (or equivalently, of **SBE**) is close to the optimal error in 3 cases out of 7, and better than the baseline in all cases. Only for Ionosphere and Sonar, it considerably departs from the optimal error. Regarding the baseline, in 10 cases the search error is around twice (or even more) the one of the heuristics/complete feature selection strategies. E.g., for Sonar, it is 12 times the optimal error.

Second, heuristics search generalizes to unseen cases better than baseline and complete search. Globally, **DTsbe** wins in 10 cases and it is not statistically different from the winner in other 3 cases. Consider the datasets for which **DTaccept$_0$** terminates within the time-out. Both the optimal feature subset and the baseline win in 3 cases. The heuristics search wins in 1 case only, but in 3 other cases it is not statistically different from the winner. Thus, no clear overall winner/loser can be determined in such cases. Comparing **DTsbe** and **DTaccept$_0$** between them, **DTsbe** wins in 2 cases (Ionosphere and Anneal), loses in 3 cases, and it is not statistically different in 4 cases (we count here Hypo, where the baseline wins). Thus we cannot conclude that complete search leads to better performances. Rather,

Table 3: Number of features and sample Pearson's correlation coefficient in cross-validation (mean ± stdev). IG and m=2.

dataset	number of features			$\phi_{Pearson}$		
	baseline	**DTsbe**	**DTaccept$_0$**	**baseline**	**DTsbe**	**DTaccept$_0$**
Adult	13.89 ± 0.31	6.63 ± 1.56	**5.81 ± 1.18**	**0.80 ± 0.40**	0.46 ± 0.27	0.54 ± 0.23
Letter	16.00 ± 0.00	11.30 ± 1.34	**10.51 ± 1.29**	**1.00 ± 0.00**	0.56 ± 0.24	0.68 ± 0.21
Hypo	15.57 ± 2.16	**6.51 ± 1.21**	7.35 ± 1.49	**0.80 ± 0.15**	0.74 ± 0.14	0.62 ± 0.17
Ionosph.	11.37 ± 1.54	**5.36 ± 1.41**	7.01 ± 1.45	**0.42 ± 0.16**	0.36 ± 0.21	0.05 ± 0.19
Soybean	27.66 ± 0.88	**15.24 ± 2.04**	15.65 ± 1.47*	**0.88 ± 0.11**	0.30 ± 0.18	0.30 ± 0.18
Anneal	10.06 ± 0.49	**6.75 ± 0.93**	8.71 ± 1.58	**0.95 ± 0.06**	0.65 ± 0.21	0.34 ± 0.20
Census	37.94 ± 0.24	**9.28 ± 3.25**	-	**0.98 ± 0.06**	0.51 ± 0.17	-
Sonar	13.14 ± 1.55	**6.80 ± 1.64**	9.00 ± 1.35	**0.25 ± 0.16**	0.17 ± 0.19	0.03 ± 0.14
Coil2000	62.42 ± 1.75	**40.50 ± 4.29**	-	**0.78 ± 0.06**	0.43 ± 0.09	-
Clean1	28.40 ± 3.01	**14.63 ± 2.05**	-	**0.31 ± 0.12**	0.16 ± 0.12	-
Clean2	78.98 ± 4.42	**37.31 ± 4.40**	-	**0.31 ± 0.09**	0.19 ± 0.10	-
Madelon	112.92 ± 9.78	**48.57 ± 6.95**	-	**0.23 ± 0.05**	0.19 ± 0.06	-
Isolet	238.61 ± 7.59	**106.57 ± 6.50**	-	**0.36 ± 0.04**	0.27 ± 0.04	-
Gisette	128.72 ± 4.34	**56.68 ± 5.04**	-	**0.23 ± 0.04**	0.19 ± 0.06	-
P53mut.	73.43 ± 4.66	**22.62 ± 3.89**	-	**0.20 ± 0.05**	0.09 ± 0.06	-
Arcene	69.24 ± 2.23	**35.19 ± 3.98**	-	**0.04 ± 0.03**	0.03 ± 0.03	-
Dexter	285.04 ± 8.58	**124.99 ± 8.40**	-	**0.37 ± 0.03**	0.27 ± 0.04	-
wins/$\star$	0/0	15/0	2/1	17/0	0/0	0/0

our results reinforce the conclusions of Doak (1992); Quinlan and Cameron-Jones (1995); Reunanen (2003) that oversearching increases overfitting.

7.5 Complete Search or Heuristics? Comparison on Feature Stability

Feature selection is typically a multi-objective problem. In addition to the selection of a feature subset with minimum error, one may be interested in other performance measures, e.g., in minimizing the size of the subset or in minimizing the variability due to perturbations in the training set (stability). Table 3 reports on the left hand side the number of features actually used by the 100 decision trees built during cross-validation. The baseline method shows that the embedded feature selection used in tree construction uses a restricted number of features w.r.t. the available ones. **DTsbe** leads to use half or even less features. It is the best performing selection strategy with regard to such a measure. **DTaccept$_0$** wins in 2 datasets out of 7. While the differences with **DTsbe** are statistically significant, they are not large – about 1-2 features.

Table 3 reports on the right hand side the mean and standard deviation of the sample Pearson's correlation coefficient over feature subsets used by decision trees[11] built during cross-validation. The mean value corresponds to the $\hat{\boldsymbol{\Phi}}_{\boldsymbol{Pearson}}$ measure of stability introduced by Nogueira and Brown (2016). Differently from other measures of stability, it

11. Stability is calculated on the subset of features *used* by decision trees, not on the feature subsets *selected* by a strategy. This allows for measuring variability of the baseline approach, which otherwise would result to have zero variability, and to contrast it to the feature selection strategies.

is unbiased w.r.t. dimensionality of the dataset. From the table, we can conclude that the baseline method has the highest stability of selected features (also with the smallest variance), where the value 1 means that any two distinct folds in cross-validation produce decision trees that use the same subset of features. Selection strategies have a considerably lower stability, in some cases half of the values of the baseline. Moreover, **DTsbe** wins over $\mathbf{DTaccept}_0$ in 4 cases out of 7 and is equivalent in another. $\mathbf{DTaccept}_0$ has a better stability only for the 2 lower dimensional datasets. In summary, we can conclude that, in the case of decision tree classifiers, oversearching increases instability.

8. Conclusions

We have introduced an original pruning algorithm of the search space of feature subsets which allows for enumerating all and only the distinct decision trees. The lattice of feature subsets is explored by distinguishing features that must be necessarily used from those that may be possibly used. The order of the visit is impressively effective, and it relies on an estimation of the chances of generating distinct trees. Based on the enumeration of distinct decision trees, we introduced an algorithm for finding δ-acceptable feature subsets, which depart by at most δ from the optimal misclassification error. Coupled with a few computational optimizations and a multi-core parallel implementation, this makes it possible to investigate properties of complete search for datasets of up to 60 features. Beyond such a limit, we contributed by exploiting ideas and optimizations proposed in the paper to a white-box computational improvement of the sequential backward elimination heuristics, which extends its practical applicability to large dimensional datasets. Experimental results reinforce, in the case of decision trees, previous findings that oversearching increases overfitting, and, in addition, they highlight that oversearching also increases instability.

References

M. Aldinucci, S. Ruggieri, and M. Torquati. Decision tree building on multi-core using FastFlow. *Concurrency and Computation: Practice and Experience*, 26(3):800–820, 2014.

E. Amaldi and V. Kann. On the approximation of minimizing non zero variables or unsatisfied relations in linear systems. *Theoretical Computer Science*, 209:237–260, 1998.

A. Blum and P. Langley. Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 97(1-2):245–271, 1997.

V. Bolón-Canedo, N. Sánchez-Maroño, and A. Alonso-Betanzos. A review of feature selection methods on synthetic data. *Knowledge and Information Systems*, 34(3):483–519, 2013.

L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth Publishing Company, 1984.

L. A. Breslow and D. W. Aha. Simplifying decision trees: A survey. *The Knowledge Engineering Review*, 12:1–40, 1997.

R. Caruana and D. Freitag. Greedy attribute selection. In *Proc. of the Int. Conf. on Machine Learning (ICML 1994)*, pages 28–36. Morgan Kaufmann, 1994.

R. Caruana and A. Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proc. of the Int. Conf. on Machine Learning (ICML 2006)*, volume 148, pages 161–168. ACM, 2006.

T. M. Cover. On the possible ordering on the measurement selection problem. *Trans. Systems, Man, and Cybernetics*, 9:657–661, 1977.

M. Dash and H. Liu. Feature selection for classification. *Intelligent Data Analysis*, 1(1-4): 131–156, 1997.

M. Fernández Delgado, E. Cernadas, S. Barro, and D. Gomes Amorim. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*, 15(1):3133–3181, 2014.

L. Devroye, L. Györfi, and G. A. Lugosi. *A Probabilistic Theory of Pattern Recognition*. Springer, 1996.

J. Doak. An evaluation of feature selection methods and their application to computer security. Technical Report CSE-92-18, University of California Davis, 1992.

L. Fan. Accurate robust and efficient error estimation for decision trees. In *Proc. of the Int. Conf. on Machine Learning (ICML 2016)*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 239–247, 2016.

I. Foroutan and J. Sklansky. Feature selection for automatic classification of non-gaussian data. *Trans. Systems, Man, and Cybernetics*, 17(2):187–198, 1987.

I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

I. Guyon, S. Gunn, A. Ben-Hur, and G. Dror. Design and analysis of the NIPS2003 challenge. In I. Guyon, M. Nikravesh, S. Gunn, and L. A. Zadeh, editors, *Feature Extraction: Foundations and Applications*, pages 237–263. Springer, 2006a.

I. Guyon, M. Nikravesh, S. Gunn, and L. A. Zadeh, editors. *Feature Extraction: Foundations and Applications*, volume 207 of *Studies in Fuzziness and Soft Computing*. Springer, 2006b.

G. H. John, R. Kohavi, and K. Pfleger. Irrelevant features and the subset selection problem. In *Proc. of the Int. Conf. on Machine Learning (ICML 1994)*, pages 121–129. Morgan Kaufmann, 1994.

J.-H. Kim. Estimating classification error rate: Repeated cross-validation, repeated hold-out and bootstrap. *Computational Statistics & Data Analysis*, 53(11):3735–3745, 2009.

R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proc. Int. Joint Conf. on Artificial Intelligence (IJCAI 1995)*, pages 1137–1145. Morgan Kaufmann, 1995.

R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97 (1-2):273–324, 1997.

T. N. Lal, O. Chapelle, J. Weston, and A. Elisseeff. Embedded methods. In I. Guyon, M. Nikravesh, S. Gunn, and L. A. Zadeh, editors, *Feature Extraction: Foundations and*

Applications, pages 137–165. Springer, 2006.

M. Lichman. UCI machine learning repository, 2013. http://archive.ics.uci.edu/ml.

H. Liu and L. Yu. Toward integrating feature selection algorithms for classification and clustering. *IEEE Transactions on Knowledge and Data Engineering*, 17(4):491–502, 2005.

H. Liu, H. Motoda, and M. Dash. A monotonic measure for optimal feature selection. In *Proc. of the European Conf. on Machine Learning (ECML 1998)*, volume 1398 of *Lecture Notes in Computer Science*, pages 101–106. Springer, 1998.

S. K. Murthy. Automatic construction of decision trees from data: A multi-disciplinary survey. *Data Mining and Knowledge Discovery*, 2:345–389, 1998.

S. Nogueira and G. Brown. Measuring the stability of feature selection. In *Proc. of Machine Learning and Knowledge Discovery in Databases (ECML-PKDD 2016) Part II*, volume 9852 of *LNCS*, pages 442–457, 2016.

J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.

J. R. Quinlan and M. Cameron-Jones. Oversearching and layered search in empirical learning. In *Proc. of Int. Joint Conf. on Artificial Intelligence (IJCAI 1995)*, pages 1019–1024. Morgan Kaufmann, 1995.

J. Reunanen. Overfitting in making comparisons between variable selection methods. *Journal of Machine Learning Research*, 3:1371–1382, 2003.

S. Ruggieri. Efficient C4.5. *IEEE Transactions on Knowledge and Data Engineering*, 14: 438–444, 2002.

S. Ruggieri. YaDT: Yet another Decision tree Builder. In *Proc. of Int. Conf. on Tools with Artificial Intelligence (ICTAI 2004)*, pages 260–265. IEEE, 2004.

S. Ruggieri. Subtree replacement in decision tree simplification. In *Proc. of the SIAM Conf. on Data Mining (SDM 2012)*, pages 379–390. SIAM, 2012.

S. Ruggieri. Enumerating distinct decision trees. In *Proc. of the Int. Conf. on Machine Learning (ICML 2017)*, number 70 in JMLR Workshop and Conference Proceedings, pages 2960–2968, 2017.

M. Saar-Tsechansky and F. Provost. Handling missing values when applying classification models. *Journal of Machine Learning Research*, 8:1625–1657, 2007.

S. S. Skiena. *The Algorithm Design Manual*. Springer, 2 edition, 2008.

B. Twala. An empirical comparison of techniques for handling incomplete data using decision trees. *Applied Artificial Intelligence*, 23(5):373–405, 2009.

Table 4: Cross-validation errors (mean ± stdev). IG+EBP and m=2.

dataset	**baseline**	**DTsbe**	**DTaccept$_0$**
Adult	14.62 ± 0.36	14.15 ± 0.45*	**14.15 ± 0.44**
Letter	11.92 ± 0.70*	11.88 ± 0.75*	**11.87 ± 0.77**
Hypo	**0.45 ± 0.34**	0.58 ± 0.45	0.55 ± 0.44
Ionosphere	12.14 ± 4.91	**9.78 ± 5.53**	12.31 ± 5.24
Soybean	**10.37 ± 3.38**	11.26 ± 3.62	11.48 ± 4.06
Anneal	**0.90 ± 0.79**	1.35 ± 1.13	2.36 ± 1.88
Census	5.14 ± 0.08	**4.93 ± 0.09**	-
Sonar	**25.95 ± 9.54**	26.68 ± 9.45*	27.66 ± 9.81*
Coil2000	5.98 ± 0.12*	**5.97 ± 0.05**	-
Clean1	**17.38 ± 5.47**	19.40 ± 6.35	-
Clean2	3.09 ± 0.64	**2.78 ± 0.72**	-
Madelon	25.30 ± 4.00	**21.85 ± 3.51**	-
Isolet	16.72 ± 1.30	**16.38 ± 1.33**	-
Gisette	6.05 ± 0.94	**5.47 ± 0.83**	-
P53mutants	0.54 ± 0.12	**0.49 ± 0.10**	-
Arcene	48.48 ± 5.43*	**48.03 ± 5.39**	-
Dexter	48.98 ± 3.58	**48.21 ± 2.99**	-
wins/$\star$	5/3	10/3	2/1

Appendix A. Additional Experimental Results

First, we consider tree simplification, a well-known strategy to address the common problems of overfitting the training data and of trading accuracy for simplicity. We applied the C4.5 Error Based Pruning (EBP) (Breslow and Aha, 1997) approach on the decision trees built from the feature subset selected by the strategies considered in the experimental section.Table 4 reports the resulting cross-validation errors. The winners/starred strategies are almost identical to Table 2. Only for the Soybean dataset, the winner changes from **DTaccept$_0$** to the baseline. Misclassification errors are generally smaller for all strategies (for Coil2000 the improvement is vast), confirming that tree simplification effectively improves generalization (Ruggieri, 2012).

Next, we consider whether the results extend to quality measures other than IG. Tables 5 and 6 are the equivalent of Tables 2 and 3 respectively for the Gain Ratio (GR) splitting criterion (see also footnote 1). Notice that now **DTaccept$_0$** does not terminate on the Soybean dataset within the time-out of 1h. For cross-validation errors, the baseline wins in 3 more cases (Ionosphere, Isolet, and Soybean), thus lowering the gap with the search heuristics. This can be attributed to the better discriminative power of the gain ratio. Feature selection remain a winner strategy if the objective consists of both maximizing accuracy and minimizing the number of used features. Compared to Table 3, the number of used features is slightly higher for GR, but stability is slightly better.

Table 5: Search and cross-validation errors (mean ± stdev). GR and m=2.

dataset	search errors			cross-validation errors		
	baseline	**DTsbe**	**DTaccept$_0$**	**baseline**	**DTsbe**	**DTaccept$_0$**
Adult	14.87 ± 0.24	13.69 ± 0.29	**13.50 ± 0.19**	14.59 ± 0.43	14.00 ± 0.47	**13.80 ± 0.45**
Letter	14.78 ± 0.50	13.66 ± 0.46	**13.50 ± 0.41**	12.58 ± 0.76	12.17 ± 0.80*	**12.16 ± 0.79**
Hypo	0.52 ± 0.22	0.30 ± 0.18	**0.28 ± 0.17**	**0.48 ± 0.33**	0.59 ± 0.45	0.66 ± 0.44
Ionosph.	11.29 ± 2.77	5.79 ± 2.21	**1.92 ± 1.13**	**10.05 ± 4.95**	10.14 ± 5.04*	11.02 ± 5.07*
Soybean	10.30 ± 2.49	**5.46 ± 1.42**	-	**8.39 ± 3.69**	9.25 ± 3.33	-
Anneal	1.52 ± 0.86	0.80 ± 0.46	**0.43 ± 0.39**	**1.46 ± 1.24**	1.91 ± 1.48	2.45 ± 1.94
Census	5.61 ± 0.06	**4.96 ± 0.22**	-	5.52 ± 0.10	**5.02 ± 0.29**	-
Sonar	28.70 ± 6.62	14.75 ± 4.84	**2.02 ± 1.35**	**25.30 ± 9.07**	26.60 ± 9.29*	29.03 ± 9.93
Coil2000	9.21 ± 0.46	**7.29 ± 0.28**	-	9.17 ± 0.60	**8.98 ± 0.67**	-
Clean1	21.92 ± 3.88	**10.97 ± 2.48**	-	**16.52 ± 5.98**	18.84 ± 5.72	-
Clean2	4.25 ± 0.69	**1.52 ± 0.38**	-	3.09 ± 0.75*	**3.00 ± 0.80**	-
Madelon	33.56 ± 3.72	**17.17 ± 2.37**	-	29.68 ± 4.35	**26.02 ± 4.35**	-
Isolet	18.10 ± 0.91	**11.46 ± 0.59**	-	**16.42 ± 1.33**	16.85 ± 1.40	-
Gisette	7.13 ± 0.64	**3.13 ± 0.41**	-	6.59 ± 0.76	**5.86 ± 0.92**	-
P53mut.	0.62 ± 0.07	**0.31 ± 0.03**	-	0.59 ± 0.11	**0.52 ± 0.10**	-
Arcene	48.65 ± 3.25	**31.40 ± 2.67**	-	48.77 ± 5.15	**47.26 ± 5.61**	-
Dexter	48.87 ± 1.78	**32.01 ± 1.62**	-	48.46 ± 3.04*	**48.18 ± 3.01**	-
wins/$\star$	0/0	11/0	6/0	7/2	8/3	2/1

Table 6: Number of features and sample Pearson's correlation coefficient in cross-validation (mean ± stdev). GR and m=2.

dataset	number of features			$\phi_{Pearson}$		
	baseline	**DTsbe**	**DTaccept$_0$**	**baseline**	**DTsbe**	**DTaccept$_0$**
Adult	14.00 ± 0.00	7.60 ± 1.69	**6.59 ± 0.70**	**1.00 ± 0.00**	0.30 ± 0.27	0.46 ± 0.26
Letter	16.00 ± 0.00	11.30 ± 1.70	**10.12 ± 1.13**	**1.00 ± 0.00**	0.53 ± 0.28	0.73 ± 0.18
Hypo	14.01 ± 2.76	**6.58 ± 1.40**	7.70 ± 1.84	**0.82 ± 0.18**	0.74 ± 0.13	0.56 ± 0.18
Ionosph.	10.95 ± 1.68	**5.31 ± 1.49**	6.94 ± 1.43	**0.42 ± 0.16**	0.37 ± 0.22	0.05 ± 0.20
Soybean	26.04 ± 1.15	**15.89 ± 2.33**	-	**0.76 ± 0.12**	0.36 ± 0.16	-
Anneal	15.92 ± 1.32	**7.30 ± 1.60**	11.12 ± 2.26	**0.85 ± 0.09**	0.63 ± 0.21	0.33 ± 0.20
Census	40.00 ± 0.00	**15.25 ± 7.89**	-	**1.00 ± 0.00**	0.32 ± 0.20	-
Sonar	12.97 ± 1.55	**6.51 ± 1.43**	9.24 ± 1.49	**0.23 ± 0.17**	0.15 ± 0.19	0.03 ± 0.14
Coil2000	64.68 ± 1.64	**44.99 ± 3.59**	-	**0.83 ± 0.06**	0.39 ± 0.10	-
Clean1	28.05 ± 2.38	**15.37 ± 1.88**	-	**0.25 ± 0.12**	0.17 ± 0.12	-
Clean2	82.25 ± 4.51	**40.80 ± 3.76**	-	**0.35 ± 0.09**	0.23 ± 0.09	-
Madelon	132.34 ± 9.00	**54.24 ± 7.09**	-	**0.23 ± 0.05**	0.19 ± 0.06	-
Isolet	241.98 ± 6.51	**112.96 ± 6.45**	-	**0.37 ± 0.04**	0.27 ± 0.04	-
Gisette	159.53 ± 5.18	**65.64 ± 6.53**	-	**0.29 ± 0.04**	0.19 ± 0.06	-
P53mut.	75.24 ± 4.73	**20.47 ± 4.10**	-	**0.21 ± 0.06**	0.10 ± 0.07	-
Arcene	83.23 ± 3.27	**41.76 ± 4.45**	-	**0.08 ± 0.03**	0.04 ± 0.03	-
Dexter	314.21 ± 7.57	**133.20 ± 11.34**	-	**0.33 ± 0.02**	0.23 ± 0.03	-
wins/$\star$	0/0	15/0	2/0	17/0	0/0	0/0