Enumerating Distinct Decision Trees

Salvatore Ruggieri [1]

Abstract

The search space for the feature selection problem in decision tree learning is the lattice of subsets of the available features. We provide an exact enumeration procedure of the subsets that lead to *all and only the* distinct decision trees. The procedure can be adopted to prune the search space of complete and heuristics search methods in wrapper models for feature selection. Based on this, we design a computational optimization of the sequential backward elimination heuristics with a performance improvement of up to 100×.

1. Introduction

Feature selection in machine learning classification is an extremely relevant and widely studied topic. Wrapper models for feature selection have shown superior performance in many contexts (Doak, 1992). They explore the lattice of feature subsets. For a given subset, a classifier is built over the features in the subset and an optimality condition is tested. However, complete search of the lattice of feature subsets is know to be NP hard (Amaldi & 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. Such a pruning would be useful not only for complete searches, but also in the case of heuristics searches. 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 contribution is a non-trivial enumeration algorithm 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 they use all attributes in R. Coupled with the tremendous computational optimization of decision tree induction algorithms, our approach makes it possible to increase the limit of practical applicability of theoretically hard complete searches. It also allows to optimize the sequential backward elimination (SBE) search heuristics when specifically designed for decision tree learning, with a speedup of up to 100× compared to a black-box approach. 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 devised in Sect. 3. Next, Sect. 4 introduces a procedure for the enumeration of distinct decision trees as a pruning of the feature subset lattice. A white-box optimization of SBE is described in Sect. 5. Experimental results are shown in Sect. 5. Finally, we summarize the contribution of the paper.

2. Related Work

(Blum & Langley, 1997; Dash & Liu, 1997; Guyon & Elisseeff, 2003; Liu & 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. *Wrappers* approaches optimize induction algorithm performances as part of feature selection (Kohavi & 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

[1]University of Pisa and ISTI-CNR, Pisa, Italy. Correspondence to: Salvatore Ruggieri <ruggieri@di.unipi.it>.

Proceedings of the 34^{th} International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017.

evaluated on the search set. For a dataset with n features, the search space consists of 2^n possible subsets. Search space exploration strategies include (see (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 feature subset 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 & 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 and 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 and Bound (AMB&B) (Foroutan & 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 & Freitag, 1994), which examines 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 4.3. While caching improves 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 4.3 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.

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. Fig. 1 (top) 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:

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

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

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

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

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

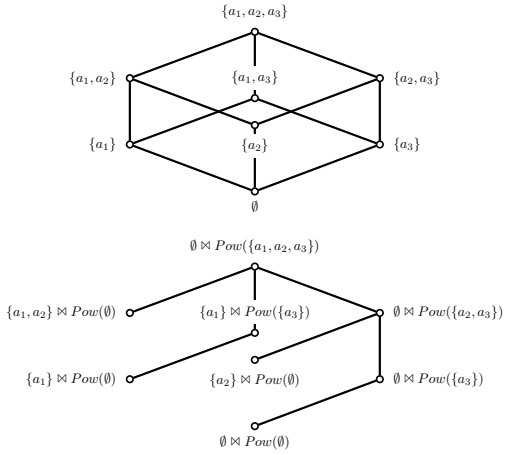

Figure 1. Lattice of subsets and reverse binary counting.

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 Alg. 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 Fig. 1 (bottom). According to line 1 of Alg. 1, the subset outputted at a node labelled as $R \bowtie Pow(S)$ is $R \cup S$. Hence, the output 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 the recursive procedure Alg. 1 will be relevant for the rest of the paper.

Remark 3.2 *A set S'' generated at a non-root node of the search tree of Alg. 1 is obtained by removing an element from the set S' generated at the father node, i.e., $S'' = S' \setminus \{v\}$ for some $v \in S'$.*

The invariant $|R' \cup S'| = |R \cup S|$ readily holds for the loop at lines 4–8 of Alg. 1. Before the recursive call at line 6, an element 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 3.3 *The selection order of $a_i \in S$ at line 4 of Alg. 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 nondeterminism* 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**", Alg. 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.

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

1: **output** $R \cup S$
2: $R' \leftarrow R \cup S$
3: $S' \leftarrow \emptyset$
4: **for** $a_i \in S$ **do**
5: $\quad R' \leftarrow R' \setminus \{a_i\}$
6: $\quad$ subset(R', S')
7: $\quad S' \leftarrow S' \cup \{a_i\}$
8: **end for**

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 $S = \{a_1, \ldots, a_n\}$ be the set of predictive features. We write $T = DT(S)$ to denote the decision tree built from predictive features 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 4.1 *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, used in C4.5 (Quinlan, 1993) and CART algorithms (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 4.2 *If $stop(S, C) = true$ then $stop(S', C) = true$ for every $S' \subseteq S$.*

The assumption states that either: (1) the stopping criterion

[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 4.1, making the selection of the split attribute dependent on the set S. An heuristics that satisfies Assumption 4.1 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 necessarily using R and possibly using S as split features

1: **build tree** $T = DT(R \cup S)$
2: $U \leftarrow$ unused features in T
3: **if** $R \cap U = \emptyset$ **then**
4: **output** T
5: **end if**
6: $R' \leftarrow R \cup (S \setminus U)$
7: $S' \leftarrow S \cap U$
8: **for** $a_i \in S \setminus U$ **order by** $frontier(T, a_i)$ **do**
9: $R' \leftarrow R' \setminus \{a_i\}$
10: DTdistinct(R', S')
11: $S' \leftarrow S' \cup \{a_i\}$
12: **end for**

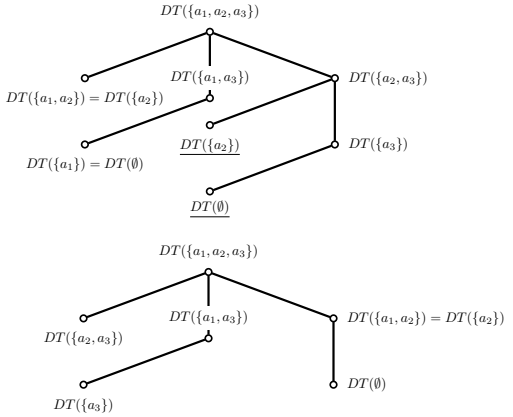

Figure 2. Search spaces of Alg. 2 for different selection orders.

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. (2) applies to criteria which require minimum quality of features for splitting a node. E.g., the C4.5 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 & Freitag, 1994)), states a useful consequence of Assumptions 4.1 and 4.2.

Remark 4.3 *Let $T = DT(S)$, and $\hat{S}$ be the set of split features used in T. For every S' such that $S \supseteq S' \supseteq \hat{S}$, we have $DT(S') = T$.*

If the decision tree T built from S uses only features from $\hat{S}$, then $argmax_{a \in S} f(a, C) = argmax_{\hat{a} \in \hat{S}} f(\hat{a}, C)$ at any decision node of T. Hence, any unused attribute in $S \setminus \hat{S}$ will not change the result of maximizing the quality measure and then, by Assumption 4.1, the split attribute at a decision node. Moreover, by Assumption 4.2, a leaf node in T will remain a leaf node for any $S' \subseteq S$.

4.2. Enumerating Distinct Decision Trees

Let $\hat{S} = \{a_1, \ldots, a_k\}$ be the set of features used in split nodes of the decision tree $T = DT(S)$ built from S, and $S \setminus \hat{S} = \{a_{k+1}, \ldots, a_n\}$ the set of features never selected as split features. By Remark 4.3, the decision tree T is equal to the one built starting from features $a_1, \ldots, a_k$ plus any subset of $a_{k+1}, \ldots, a_n$. In symbols, all the decision trees for attribute subsets in $\{a_1, \ldots, a_k\} \bowtie Pow(\{a_{k+1}, \ldots, a_n\})$ do coincide with T. We will use this observation to remove from the recurrence relation of Lemma 3.1 those sets 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:

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

contains sets of features leading to the same decision tree as $DT(R \cup S)$. The simplified recurrence relation prunes from the the search space features subsets that lead to duplicated decision trees. However, we will show in Ex. 4.4 that such a pruning alone is not sufficient to generate distinct decision trees only, i.e., duplicate trees may still exists.

Alg. 2 provides an enumeration of *all and only the distinct decision trees*. It builds on the generalized subset generation procedure. Line 1 constructs a tree T from features $R \cup S$. Feature in the set 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 $R \cap U = \emptyset$, namely features *required* to be used (i.e., R) are *actually* used in decision splits. This prevents from outputting more than once a decision tree that can be obtained from multiple paths of the search tree.

Example 4.4 *Let $S = \{a_1, a_2, a_3\}$. 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 Fig. 1 (bottom) gives rise to the trees built by **DTdistinct($\emptyset$, S)** as shown in Fig. 2 (top). For instance, the subset $\{a_1, a_2\}$ visited at the node*

labelled $\{a_1, a_2\} \bowtie \emptyset$ in Fig. 1 (bottom), 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 Fig. 2 (top) – 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 Fig. 2 (top), together with its underlined duplicate tree. Unique trees for two or more duplicates can be 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 node underlined will output their decision trees, while the other duplicates will not pass the test at line 3 of Alg. 2.

Remark 3.3 states that the selection order in the recursive calls of **subset()** is not relevant. Alg. 2 adopts a specific order that, while not affecting the result (any order would produce the enumeration of distinct decision trees), impacts on the effectiveness of pruning the search space. We define the frontier $frontier(T, a_i)$ of an attribute a_i in a decision tree T as the sum of the number of cases of the training set that reach a node in T where a_i is the split attribute. The smaller the frontier is the smaller is the impact of removing sub-trees of T rooted at nodes with a_i as split attribute.

Example 4.5 (Ctd.) *The order of selection of a_i's in the visit of Fig. 2 (top) is by descending i's. This order 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. As a consequence, it would be better to have a_3 removed in the rightmost child of a node, which has the largest search sub-space, and hence the best possibilities of pruning. The ordering based on ascending frontier estimates the discriminatory power of a_i by the amount of cases in the training set discriminated by splits using a_i. In our example, such an order would likely be a_1, a_2, and a_3. The search space of **DTdistinct($\emptyset$, S)** is then reported in Fig. 2 (bottom). Notice that there is no duplicate tree here. Also notice 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 is pruned at line 8 of Alg. 2. The sub-space to be searched consists then of only the subset of $\{a_1\}$, not all subsets of $\{a_1, a_2\}$.*

The following non-trivial result holds.

Theorem 4.6 ***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 3.3, the order of selection of $a_i \in S \setminus U$ 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 3.1, 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 $i < j$. By inductive hypothesis, a_i is missing as a predictive attribute in the trees in output from the first call, while 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, if it is outputted at line 4. In fact, T has $|R \cup S \setminus U|$ split features, whilst recursive calls construct decision trees with at most $|R \cup S \setminus U| - 1$ features.

Let us now show that trees pruned at line 7 or at line 4 are already outputted elsewhere, which implies that every distinct decision tree is outputted at least once. First, by Remark 4.3, the trees of the pruned iterations $S \cap U$ at line 7 are the same of the tree of T at line 1. Second, if the tree T is not outputted at line 4, because $R \cap U \neq \emptyset$, 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. Let $R = \{a_1, \ldots, a_n\}$, with $n > 0$, and let $R' = \{a_1, \ldots, a_{i-1}\}$ be such that $a_i \in U$ and $R' \cap U = \emptyset$. There is a sibling node in the search tree corresponding to a call with parameters R' and $S' = \{a_{i+1}, \ldots, a_n\} \cup S$. 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 $R' \bowtie Pow(S')$. □

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

Property 1: linear space complexity. Alg. 2 is computationally linear (per number of trees built) in space in the number of predictive features. 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 forms of caching of duplicates (Caruana & Freitag, 1994). Those approaches would require exponential space, as shown in the next example.

Example 4.7 *Let us consider the well-known Adult dataset[2] (Lichman, 2013), consisting of 48842 cases, 14 predictive features, and a binary class attribute. Fig. 3*

[2] See Sect. 5 for the experimental settings.

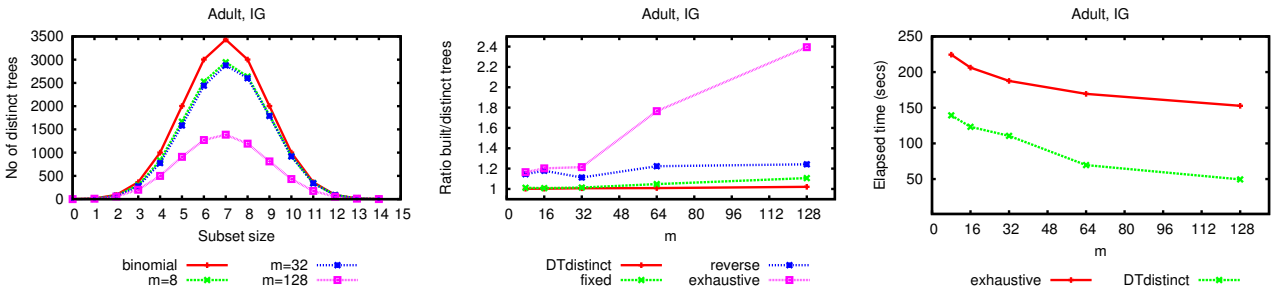

Figure 3. Left: distribution of distinct decision trees. Center: ratio built/distinct decision trees. Right: elapsed times.

(left) shows, for the IG split criterion, the distribution of distinct decision trees w.r.t. the size of attribute subset. The distributions are plotted for various values of the stopping parameter m 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 $|C| < m$).

Property 2: reduced overhead. Our procedure may construct duplicate decision trees at line 1, which, however, are not outputted thanks to the test at line 3. We measure such an overhead of Alg. 2 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. The overhead can be controlled by the attribute selection ordering at line 8.

Example 4.8 (Ctd.) *Fig. 3 (center) shows the overhead at the variation of m for three possible orderings of selection at line 8 of Alg. 2. One is the the ordering stated by **DTdistinct()**, based on ascending frontier. The second one is the reversed order, namely descending frontier. The third one is based on assigning a fixed index i to features a_i's, and then ordering over i. The **DTdistinct()** ordering is impressively effective, with an overhead close to 1 – i.e., the search space is precisely the set of distinct decision trees.*

Fig. 3 (center) also reports the ratio of the number of trees in an exhaustive search (2^n for n features) over the number of distinct trees. Smaller m's lead to a smaller ratio. Thus, for small m values, pruning duplicate trees does not guarantee alone an enumeration more efficient than exhaustive search. The next property will help.

Property 3: feature-incremental tree building. The construction of each single decision tree at line 1 of Alg. 2 can be speed up by Remark 3.2. 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. The construction of T' can then benefit from this observation. We first recursively clone T and then re-build only sub-trees rooted at node where the split attribute is a_i.

Example 4.9 (Ctd.) *Fig. 3 (right) contrasts the elapsed times of exhaustive search and **DTdistinct()**. For smaller values of m, there is an exponential number of duplicated decision trees, but the running time of **DTdistinct()** is still much better than the exhaustive search due to the incremental building of decision trees.*

5. PSBE: A White-Box Optimization of SBE

In wrapper models, the training set is divided into a building set and a search set. A decision tree is built on the building set and its the accuracy is evaluated on the search set. Our enumeration procedure **DTdistinct()** has a direct application, which consists of running a complete wrapper search looking for the feature subset that leads to the most accurate decision tree on the search set. On the practical side, however, using **DTdistinct()** to look for the optimal feature subset is computationally feasible only when the number of predictive features is moderate. Moreover, optimality on the search set may be obtained at the cost of overfitting (Doak, 1992; Reunanen, 2003) and instability (Nogueira & Brown, 2016). The ideas underlying our approach, however, can impact also on the efficiency of heuristics searches.

In particular, we consider here the widely used sequential backward elimination (SBE) heuristics. SBE starts building a decision tree T using the set S of all features. We call T the *top tree*. For every $a_i \in S$, a decision tree T_i is built using features in $S \setminus \{a_i\}$. If no T_i's has a smaller error on the search set than T, the algorithm stops returning S as the subset of selected features. Otherwise, the procedure is repeated removing a_k from S, where T_k 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, not only to decision trees. A white-box optimization can be devised for decision tree classifiers that satisfy the assumptions of Section 4.1. The optimization relies on Remark 4.3. Let U be the set of features not used in the current decision tree T. For $a_i \in U$, it turns out that

Table 1. Experimental results. IG and m=2.

dataset			elapsed time (secs)			cross-validation error (%)		
name	**inst.**	**feat.**	**PSBE**	**SBE**	**ratio**	**top**	**(P)SBE**	**optimal**
Adult	48,842	15	2.318	3.311	0.700	15.74 ± 0.42	14.64 ± 0.46	$\mathbf{14.31 \pm 0.39}$
Letter	20,000	17	1.221	1.717	0.711	12.69 ± 0.75	12.49 ± 0.75	$\mathbf{12.37 \pm 0.71}$
Hypo Thyroid	3,772	31	0.029	0.100	0.290	$\mathbf{0.39 \pm 0.28}$	0.42 ± 0.32	0.46 ± 0.34
Ionosphere	351	35	0.005	0.065	0.077	11.74 ± 5.72	$\mathbf{10.25 \pm 4.61}$	10.72 ± 5.40
Soybean	683	36	0.067	0.212	0.316	13.26 ± 4.19	12.71 ± 4.09	$\mathbf{10.62 \pm 3.67}$
Anneal	898	39	0.004	0.025	0.160	$\mathbf{0.91 \pm 0.97}$	1.14 ± 1.39	1.45 ± 1.26
Sonar	208	61	0.006	0.222	0.027	28.57 ± 8.95	27.01 ± 8.78	$\mathbf{25.69 \pm 9.12}$
Coil2000	9,822	86	3.647	19.578	0.186	9.01 ± 0.61	$\mathbf{8.75 \pm 0.71}$	-
Clean1	476	166	0.053	5.381	0.010	19.50 ± 5.98	$\mathbf{19.28 \pm 7.04}$	-
Clean2	6,598	166	1.585	79.465	0.020	3.20 ± 0.73	$\mathbf{3.02 \pm 0.83}$	-
Madelon	2,600	500	2.647	>1h	-	25.28 ± 3.61	$\mathbf{22.67 \pm 3.28}$	-
Gisette	7,000	5,000	11.738	>1h	-	6.27 ± 0.81	$\mathbf{6.11 \pm 0.92}$	-
p53Mutants	31,420	5,408	150.518	>1h	-	0.60 ± 0.12	$\mathbf{0.54 \pm 0.10}$	-

$T_i = T$. Thus, only trees built from $S \setminus \{a_i\}$ for $a_i \notin U$ need to be considered for backward elimination. This saves the construction of $|S \cap U|$ decision trees at each step of the procedure. We call this optimization PSBE (Pruned SBE).

6. Experiments

6.1. Datasets and Experimental Settings

Table 1 reports the number of instances and of features for small and large standard benchmarks datasets publicly available from (Lichman, 2013). Following (Reunanen, 2003), we adopt 5-repeated stratified 10-fold cross validation in experimenting with wrapper models. 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. Information Gain (IG) is used as quality measure in node splitting. No form of tree simplification (e.g., error-based pruning) is used. The search error is the average misclassification error on the search set. The cross-validation error is the average misclassification error on the hold-out folds for the tree built on the training set using the selected feature subset. Misclassification errors are computed using the C4.5's distribution imputation method (Saar-Tsechansky & Provost, 2007).

All procedures described in this paper were 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 tree building. The extended YaDT version is publicly downloadable from: http://pages.di.unipi.it/ruggieri. Test were performed on a commodity PC with Intel 4 cores i5-2410@2.30 GHz, 16 Gb RAM, and Windows 10 OS.

6.2. How Fast is DTdistinct()?

Or, in other words, how much our pruning approach will make a complete search feasible in practice? Fig. 4 shows the ratio of the number of built trees over the number of distinct trees (left) and the total elapsed time of **DTdistinct()** (center) for low to medium dimensionality datasets – actually, those for which **DTdistinct()** terminates within a timeout of 1h. The ratio ranges from 1 to 1.35, which show that the selection order based on ascending frontier size (line 8 of Alg. 2) is effective in practice. The total elapsed time of the enumeration procedure, shown in Fig. 4 (center), grows exponentially with the inverse of m, the stopping parameter. This is intuitive, since lower m's lead to higher numbers of distinct decision trees, and, as shown in Fig. 3 (left), such numbers approach 2^n – where n is the number of features. However, the total elapsed time of **DTdistinct()** remains within a practically admissible bound for datasets with a moderate number of features. Consider for instance, the Anneal dataset. An exhaustive enumeration would be impossible, since it consists of building $2^{39} \approx 550B$ trees. **DTdistinct()** runs in less than 20 seconds for $m = 16$, and less than 270 seconds for $m = 8$. This is the coupled result of three factors: the pruning approach of Alg. 2, the feature-incremental tree building optimization, and the tremendous efficiency of the state-of-the-art tree induction implementations.

6.3. PSBE vs SBE

Table 1 reports elapsed times that allows for comparing the efficiency of PSBE vs SBE. The m parameter is set to the small value 2 for all datasets. The ratio of elapsed times shows a speedup of up to 100× of PSBE vs SBE. The improvement increases with the number of features. For high-

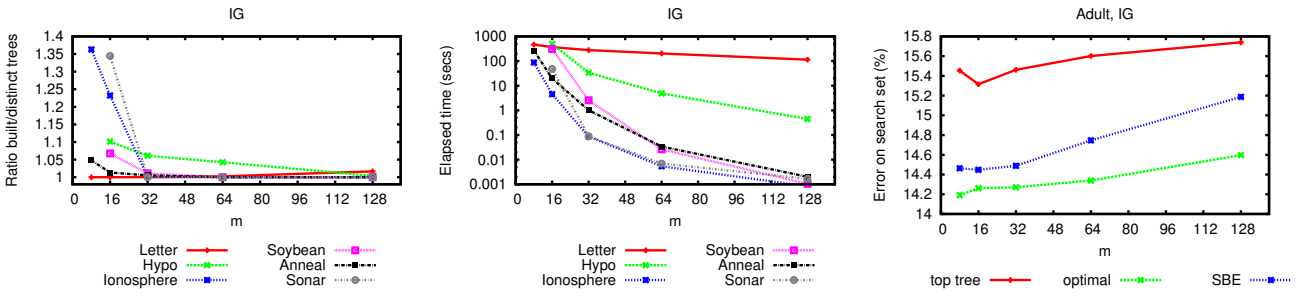

Figure 4. Left: ratio built/distinct trees. Center: elapsed times of **DTdistinct()**. Right: search errors.

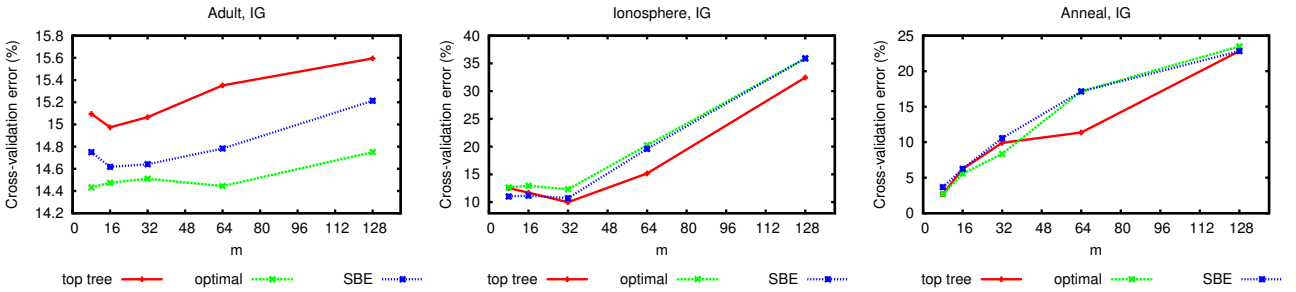

Figure 5. Cross-validation errors.

dimensional datasets, the black-box SBE does not even terminate within a time-out of 1h. The white-box PSBE, instead, runs in about 150 seconds for the highly dimensional dataset p53Mutants. This is a relevant result for machine learning practitioners, extending the applicability of the SBE heuristics.

6.4. Complete Search or Heuristics?

Fig. 4 (right) shows the average search error over the Adult dataset of the decision trees constructed on: (1) all features (*top*); (2) the features selected by SBE (same as PSBE); and (3) the features selected by **DTdistinct()**, namely those with the lowest error on the search set (hence, the name *optimal*). Obviously, SBE is better than top, and optimal is better than SBE. Interestingly, SBE is close to the optimal search error, in particular for small m parameter. Does this generalize to unknown cases? Fig. 5 reports the cross-validation errors over the Adult, Ionosphere and Anneal datasets. For Adult, optimal is better than SBE, which in turn is better than top. For Ionosphere, instead, the optimal tree has the worst performance, the top tree is the best for almost all m's, and SBE is the best for small m's. For Anneal, SBE is the worst, and the top tree is better than the optimal for large m's. Table 1 reports the cross-validation errors for $m = 2$ for all datasets. PSBE, or equivalently SBE as they select the same subset of features, wins over top in most cases. But there is no clear evidence of the superiority of optimal over SBE. This is consistent with the conclusions of (Doak, 1992; Reunanen, 2003) that simple sequential elimination exhibits better generalization performances than more exhaustive searches when considering error on an unseen set of instances.

7. 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. On the theoretical side, this makes it possible to run a complete wrapper procedure for moderate dimensionality datasets. This will allow, for instance, for a deeper investigation of old and new search heuristics by comparing their performances with those of a complete search. On the practical side, ideas and results of the paper have been applied to improve the computational efficiency of the SBE heuristics.

As future work, we will investigate the extension of the proposed approach in presence of decision tree simplification and for ensembles of decision trees (bagging, random forests). Moreover, we will consider the related problem of finding an optimal subset of features, which in the present paper is tackled by simply enumerating all distinct decision trees. Actually, there is no need to explore a sub-space of (distinct) decision trees, if a lower bound for the accuracy of any tree in the sub-space can be computed and such a lower bound is higher than the best error found so far. This idea would build upon the enumeration procedure **DTdistinct()** as a further pruning condition of the search space.

References

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

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

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

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

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

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

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

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

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

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

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

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

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

Liu, H. and Yu, L. Toward integrating feature selection algorithms for classification and clustering. *IEEE Trans. Knowl. Data Eng.*, 17(4):491–502, 2005.

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

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

Nogueira, S. and Brown, G. 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*, pp. 442–457, 2016.

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

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

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

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

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

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

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