The Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI-20)

Efficient Inference of Optimal Decision Trees

Florent Avellaneda
Computer Research Institute of Montreal
florent.avellaneda@crim.ca

Abstract

Inferring a decision tree from a given dataset is a classic problem in machine learning. This problem consists of building, from a labelled dataset, a tree where each node corresponds to a class and a path between the tree root and a leaf corresponds to a conjunction of features to be satisfied in this class. Following the principle of parsimony, we want to infer a minimal tree consistent with the dataset. Unfortunately, inferring an optimal decision tree is NP-complete for several definitions of optimality. For this reason, the majority of existing approaches rely on heuristics, and the few existing exact approaches do not work on large datasets. In this paper, we propose a novel approach for inferring an optimal decision tree with a minimum depth based on the incremental generation of Boolean formulas. The experimental results indicate that it scales sufficiently well and the time it takes to run grows slowly with the size of datasets.

Introduction

In machine learning, the problem of classification consists of inferring a model of a given system from its observations (also called *training examples*) that make it possible to identify which class of a set of classes a new observation belongs to. When the training examples used to infer a model are assigned to the classes to which they belong, it is called supervised learning. Many machine learning models exist to solve this problem, such as support vector machines (Cortes and Vapnik 1995), artificial neural networks (McCulloch and Pitts 1943), decision trees (Breiman et al. 1984). Since inferring such models is complicated and training examples are generally numerous, most existing inference algorithms are heuristic in the sense that the algorithms infer models without any guarantee of optimality.

Although heuristic-based techniques generally work well, there are often cases where a new example is not correctly recognized by the model. In the context of critical systems in which errors are not allowed, such models are problematic. This precision requirement is frequently associated with the demand that models must also be understandable. This area, known as eXplainable Artificial Intelligence (XAI), aims at inferring models capable of explaining their own behaviour. XAI has been the subject of several studies (Goebel et al. 2018; Li et al. 2018; Van Lent, Fisher, and Mancuso 2004; Angelino et al. 2017), and several events (IJCAI XAI Workshop; NIPS IML Symposium; ACM FAT*). One approach to obtain an explainable model is to use decision trees because the reasons for classification are clearly defined (Quinlan 1986).

To obtain accurate and explainable models, we are interested in the inference of optimal decision trees. The optimality of a decision tree is generally defined by the simplicity of the tree based on the principle of parsimony. The basic simplicity criteria are the depth of the tree and the number of nodes. The problem of learning an optimal decision tree is known to be NP-complete for these two basic criteria but also for several other definitions of optimality (Hyafil and Rivest 1976; Hancock et al. 1996) and it is also hard to approximate up to any constant factor under the assumption $P \neq NP$ (Sieling 2008; Alekhnovich et al. 2004). For these reasons, the majority of known algorithms for inferring decision trees are heuristics trying to minimize the number of nodes without guaranteeing any optimality (Kass 1980; Breiman et al. 1984; Quinlan 1986; 2014).

Despite the complexity of the problem and the efficiency of the first attempts to infer optimal decision trees, two studies have recently proposed approaches focused on improving the performance of algorithms inferring decision trees with a minimum number of nodes (Bessiere, Hebrard, and O'Sullivan 2009; Narodytska et al. 2018). Bessiere, Hebrard, and O'Sullivan propose a SAT formulation, but experiments show that the method only works for trees of up to fifteen nodes. Narodytska et al. propose a new SAT formulation that significantly improves the performance of optimal decision tree inference. Thus, with their formulation, Narodytska et al. were able to build a decision tree with a minimum number of nodes in 13 seconds for the "Mouse" dataset, while this required 9.6 minutes with the SAT formulation from Bessiere, Hebrard, and O'Sullivan. Narodytska et al. note that their paper is the first presentation of an optimal decision tree inference method working on well-known datasets.

In this paper, we also use a SAT solver to infer optimal

trees, but our approach has two differences from the two aforementioned papers. The first is that we will generate the Boolean formulas incrementally. Instead of attempting to process all the training examples at once, we iteratively infer a decision tree from their subset (initially it is an empty set) and use active inference to refine it when it is not consistent with the training examples. The second difference is that we choose the depth of the tree as a criterion of simplicity. This criterion of simplicity allows for more efficient Boolean formulations. Hence, our algorithm can process the "Mouse" dataset in only 20 milliseconds and well-known datasets that were considered too large to infer optimal decision trees can now be processed by our algorithm.

Other studies were carried out on a problem similar to ours: inferring decision trees with a given depth such that the total classification error on the training examples is minimal (Bertsimas and Shioda 2007; Bertsimas and Dunn 2017; Verwer and Zhang 2019). These approaches use, as we do, the depth of the tree as a criterion of simplicity, but address the problem of complexity by setting a maximum depth. We show that our approach can infer decision trees with a better accuracy.

The paper is organized as follows. First, to formalize the approach, we provide definitions related to decision trees. Then, we give a new Boolean formulation for passive inference of a decision tree from a set of training examples. Next, we propose an incremental way of generating the Boolean formulas which ensures that the proposed approach scales to large datasets. Finally, we report several experiments comparing our approach to other approaches and summarize our main findings.

Definitions

Let $\mathcal{E} = \{e_0, ..., e_{n-1}\}$ be a set of *training examples*, that is, Boolean valuations of a set $\mathcal{F} = \{f_0, ..., f_{m-1}\}$ of *features*, and let $\mathcal{E}_0, ..., \mathcal{E}_{c-1}$ be a partition of $\mathcal{E}$ into classes. Note that even if we only consider binary features, we can easily handle non-binary features by encoding them in a binary way. A non-Boolean feature f that can have x different values can be represented by x Boolean features: each Boolean feature represents a value that f can take. Moreover, if the feature f is ordered, like numbers, then each Boolean feature can represent the operator $\leq$ as the following examples shows.

Example 1. *Let $\mathcal{E}$ be a set of training examples where each example has a single integer feature f. Let $\mathcal{E}_0 = \{(1), (3)\}$ and $\mathcal{E}_1 = \{(4), (5)\}$ be the partition of $\mathcal{E}$ into two classes. Then, we can transform $\mathcal{E}_0$ and $\mathcal{E}_1$ into $\mathcal{E}'_0$ and $\mathcal{E}'_1$ such that each example has four Boolean features f'_0, f'_1, f'_2, f'_3. If the feature f'_0 is true, it means that the example is smaller or equal to 1 for feature f; if the feature f'_1 is true, it that the example is smaller or equal to 3 for feature f, etc. Thus, with this transformation we obtain $\mathcal{E}'_0 = \{(1,0,0,0), (1,1,0,0)\}$ and $\mathcal{E}'_1 = \{(1,1,1,0), (1,1,1,1)\}$.*

In the following, we denote by $e[f]$ the Boolean valuation of the feature $f \in \mathcal{F}$ in the example $e \in \mathcal{E}$. A *decision tree* is a binary tree where each node is labelled with a single feature and each leaf is labelled with a single class. A decision tree is said to be *perfect* if all leaves have the same depth and all internal nodes have two children. Formally, we denote a perfect decision tree by $T = (V, V')$ where $V \in \mathcal{F}^{2^k-1}$ is the set of internal nodes and $V' \in \{0, 1, ..., c-1\}^{2^k}$ is the set of leaves and k is the depth of the tree. We denote by $V[i]$ the i^{th} node in the tree T and by $V[1]$ the root of the tree. Then we define $V[i \times 2]$ as the left child of $V[i]$ and $V[i \times 2 + 1]$ as the right child. In a similar way, if $i \geq 2^{k-1}$, we define the leaf $V'[(i - 2^{k-1}) \times 2]$ as the left child of $V[i]$ and the leaf $V'[(i - 2^{k-1}) \times 2 + 1]$ as the right child. An illustration of this encoding is depicted by Figure 1.

In the rest of this paper, we will say that the node v is a right ancestor of v' if the right child of v is v' or an ancestor of v' (and inversely for the left ancestor). We will also say that v' is a right descendant of v if v is a right ancestor of v' (and inversely for the left descendant).

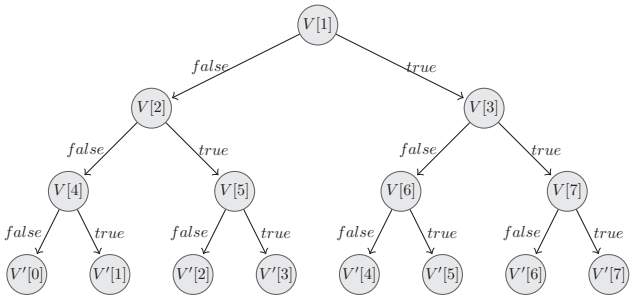

Figure 1: Illustration of nodes index coding.

This way of associating a number with each node and leaf may appear complicated, but it will be useful for our Boolean encoding. We will use the semantics associated with binary coding of node indexes to obtain compact SAT formulas.

If T is a decision tree, and $\mathcal{E}$ is a set of training examples, we say that T is *consistent* with $\mathcal{E}$, denoted $\mathcal{E} \subseteq T$, if each example $e \in \mathcal{E}$ is correctly classified by T. We also say that a system or a decision tree T is *equivalent* to another T' if there is no example classified in different classes by T and T'.

Passive Inference

The exact model inference problem consists of inferring a model equivalent to a given system using observations. The type of the model to be inferred is initially fixed, and we assume that there exists a modelling equivalent to the system to be inferred. There are two types of learning approaches in this area.

The first type is active learning, where the learning algorithm uses an oracle to help correctly infer the model. Generally, the learning algorithm requires that the oracle label new examples and be able to verify if a model corresponds to the system to be inferred. L* is the most well-known algorithm in this category for inferring deterministic finite automata (Angluin 1987). However, such an oracle is often not possible to have in practice. For this reason, there is a second category which is passive inference. This category consists of inferring a model using only a set of examples. Without

oracles, it becomes more difficult to ensure that the inferred model corresponds to the system. However, by following the principle of parsimony and inferring a minimal model, it can generally be shown that if the learning set is representative, then the minimal model will correspond to the system to be inferred.

The notion of "representative" is defined by the notion of a *characteristic sample*:

Definition 1. *Let T be a decision tree with a maximal depth of k and consistent with the training examples $\mathcal{E}$. We say that $\mathcal{E}$ is a* characteristic sample *for T if each decision tree T' with a maximal depth k consistent with $\mathcal{E}$ is equivalent to T.*

If the models to be inferred are decision trees, and a system to be inferred can be represented by a decision tree, a characteristic sample exists:

Theorem 1. *For each decision tree T, there is a characteristic sample $\mathcal{E}$.*

Proof. We can determine a characteristic sample as follows. Initially, we set $\mathcal{E} = \emptyset$. Then, while there is a decision tree T' of depth smaller k such that T and T' are not equivalent, we add to $\mathcal{E}$ the example e that is classified differently by T and T'. Since the number of decision trees with a maximal depth of k is finite, this procedure terminates. □

Note that this theorem still holds when we consider only the perfect decision trees because for each decision tree there exists an equivalent perfect decision tree with the same maximal depth.

In this section, we focus on inferring a decision tree with a maximal depth of k, consistent with training examples $\mathcal{E}$. We will first show how to infer perfect decision trees, and then decision trees that are not necessarily perfect but have a fixed maximum number of nodes.

Inferring perfect decision trees of a fixed depth

Our SAT encoding to infer perfect decision trees of a fixed depth is based on the way the nodes are indexed. As mentioned in Section "Definitions", the index of a node depends on its position in the tree. In particular, the root node corresponds to the node $V[1]$, and for each node $V[i]$, the left child corresponds to $V[i \times 2]$ and the right child to $V[i \times 2 + 1]$. This coding has the capability of providing precise information on the position of a node based on the binary coding of its index. Indeed, reading the binary coding of a node from the highest to the lowest weight bit indicates which branches to take when moving from the root to the node. For example, if the binary coding of i is 1011, then the node $V[i]$ is reached by taking the right branch of the root, then the left branch, and finally the right branch twice.

The idea of our encoding consists of arranging the training examples in the leaves of the tree while respecting the fact that all training examples placed in the same leaf must belong to the same class. Moreover, if an example is placed in a leaf, then all the right ancestors of that leaf can only be labelled by features true for that example (and conversely for the left ancestors).

The encoding idea is formalized using the following types of Boolean variables.

- $X_{i,j}$: If the variable $X_{i,j}$ is true, then the example e_i is assigned to a leaf that is a right descendant of a node located at depth j. If $X_{i,j}$ is false, then e_i is assigned to a leaf that is a left descendant of that node. Note that with this semantics on the variables $X_{i,j}$, we have the property that the binary coding $(X_{i,0}X_{i,1}...X_{i,k-1})$, denoted X_i, corresponds to the index of the leaf where the example e_i belongs, i.e., if $X_i = v$, then the example e_i is assigned to the leaf $V'[v]$. We also denote by $X_i[..a]$ the number formed by the binary coding of $(1X_{i,0}X_{i,1}...X_{i,a-1})$.
- $F_{i,j}$: If $F_{i,j}$ is true, then the node $V[i]$ is labelled with the feature f_j.
- $C_{i,j}$: If $C_{i,j}$ is true, then the leaf $V'[i]$ is labelled with the class j.

We then use the following set of clauses to formulate constraints that a perfect decision tree of depth k should satisfy. For each $i \in [1, 2^k - 1]$, we have the clauses:

$$F_{i,0} \vee F_{i,1} \vee ... \vee F_{i,m-1} \tag{1}$$

These clauses mean that each node should have at least one feature.

For each $i \in [1, 2^k - 1]$ and all features f_a, f_b such that $0 \leq a < b < m$, we have the clauses:

$$\neg F_{i,a} \vee \neg F_{i,b} \tag{2}$$

These clauses mean that each node has at most one feature.

For every i and a such that $e_i[a] = 0$, and each $j \in [0, k-1]$, we have:

$$X_{i,j} \Rightarrow \neg F_{X_i[..j],a} \tag{3}$$

And for every i and a such that $e_i[a] = 1$, and each $j \in [0, k-1]$, we have:

$$\neg X_{i,j} \Rightarrow \neg F_{X_i[..j],a} \tag{4}$$

These formulas add the constraint that some features cannot be found in certain nodes depending on where the training examples are placed in the decision tree. We use the binary coding of the index of a leaf to determine which nodes in the tree are its ancestors. Thus, all the ancestor nodes for which the right branch has been taken cannot be labelled with features that must be false (formula (3)). In the same way, all the ancestor nodes for which the left branch has been taken cannot be labelled with features that must be true (formula (4)). Note that it is not trivial to translate these formulas into clauses, but we show in Algorithm 1 how it could be done. This algorithm performs a depth-first search of the perfect decision tree in a recursive way. The variable q corresponds to the index of the current node and $\neg clause$ constrains such that X_i corresponds to the index of a q successor. Each time the algorithm visits a node q, it adds constraints on the features that can be labelled with this node based on where e_i is placed in the left or right branch of q. If e_1 is placed in the left branch, then q cannot contain a

feature that is true for e_i and vice versa with the right branch.

For each $e_i \in \mathcal{E}_a$ with $a \in [0, c-1]$ and each integer $v \in [0, 2^k - 1]$, we have:

$$(X_i = v) \Rightarrow C_{v,a} \tag{5}$$

And for each $e_i \in \mathcal{E}_a$, each integer $v \in [0, 2^k - 1]$, and $a' \neq a$ we have:

$$(X_i = v) \Rightarrow \neg C_{v,a'} \tag{6}$$

These formulas assign the classes to the leaves according to the placement of the training examples in the decision tree. Again, since it is not trivial to translate these formulas into clauses, we show in Algorithm 2 how it could be performed. This algorithm performs a depth-first search of the perfect decision tree such that when it reaches a leaf, $\neg clauses$ corresponds to the index of that leaf. After that, for each leaf, the algorithm generates the constraints that if e_i is present in that leaf, then the leaf must have the same class as e_i.

Algorithm 1 ($GenerateFeatureConstraints$)

Input: A new example e_i, a clause $clause$, an index node q, the depth of the tree lvl already considered. Initially, $clause = \emptyset$, $q = 1$ and $lvl = 0$.
Output: Clauses for formulas (3) and (4) for a new example e_i

$result \leftarrow true$
if $lvl = k$ **then**
  return $result$
end if
for all $f \in [0, m-1]$ such that $e_i[f] = 0$ **do**
  $result \leftarrow result \wedge (clause \vee \neg X_{i,lvl} \vee \neg F_{q,f})$
end for
$result \leftarrow result \wedge GenerateFeatureConstraints(e_i, (clause \vee \neg X_{i,lvl}), q \times 2 + 1, lvl + 1)$
for all $f \in [0, m-1]$ such that $e_i[f] = 1$ **do**
  $result \leftarrow result \wedge (clause \vee X_{i,lvl} \vee \neg F_{q,f})$
end for
$result \leftarrow result \wedge GenerateFeatureConstraints(e_i, (clause \vee X_{i,lvl}), q \times 2, lvl + 1)$
return $result$

Minimizing the number of nodes

A perfect decision tree has the constraint that all leaves have the same depth. In practice, this constraint leads to a tree with an unnecessarily high number of nodes. For example, there may be an imperfect decision tree consistent with training examples, with the same maximal depth as the minimal perfect tree, but with fewer nodes. Following the principle of parsimony, considering a tree with the same depth but fewer nodes should be a better model.

In this section, we will see how we can add constraints to set a maximum number of nodes. The idea is to limit the number of leaves assigned to a class in the perfect tree.

Theorem 2. *Let T be a decision tree with a maximal depth k and $MaxNodes$ nodes consistent with $\mathcal{E}$. Then there exist a perfect decision tree with depth k and at most $\lfloor MaxNodes/2 \rfloor + 1$ labelled leaves consistent with $\mathcal{E}$.*

Algorithm 2 ($GenerateClassConstraints$)

Input: A new example $e_i \in \mathcal{E}_a$, a clause $clause$, a node number q, an integer lvl and an integer $lvlMax$. Initially, $clause = \emptyset$, $q = 0$ and $lvl = 0$.
Output: Clauses for formulas (5) and (6) for a new example e_i

if $lvl = lvlMax$ **then**
  $result \leftarrow (clause \vee C_{q,a})$
  for $\mathcal{E}_{a'} \neq \mathcal{E}_a$ **do**
    $result \leftarrow result \wedge (clause \vee \neg C_{q,a'})$
  end for
  return $result$
end if
return $GenerateClassConstraints(e_i, clause \vee X_{i,lvl}, q \times 2, lvl + 1, lvlMax) \wedge GenerateClassConstraints(e_i, clause \vee \neg X_{i,lvl}, q \times 2 + 1, lvl + 1, lvlMax)$

Proof. Note that the number of leaves in T is at most $\lfloor MaxNodes/2 \rfloor + 1$. We show that the theorem is true by proposing a procedure that makes the tree perfect without adding labelled leaves. The procedure works iteratively: if all leaves are at the same depth k then T is perfect and we complete the procedure. Otherwise, there is a leaf v that is less than depth k in the tree. In this case, we replace v with a node labelled with the same feature as the root of the tree. One of its children will be v and the other will be a leaf without a label.

Note that with each iteration, the number of nodes increases strictly, but the maximal depth of the tree can never exceed k. Thus, the termination of this procedure is guaranteed. □

Thanks to Theorem 2, we know that searching for a decision tree with a maximum depth of k and with a maximum of $MaxNodes$ nodes is equivalent to searching for a perfect decision tree with a depth of k and limiting the number of labelled leaves to $\lfloor MaxNodes/2 \rfloor + 1$.

To add the constraint of the maximum number of labelled leaves, we add two types of additional variables. The variables U_i which are true if a class is assigned to the leaf i, and the variables $H_{i,0}, H_{i,1}, ..., H_{i,MaxNodes+1}$ which will be used to count, with unary coding, the number of leaves labelled by a class. The variable $H_{i+1,j}$ will be true if there are at least j leaves labelled by a class among the first i leaves.

The clauses encoding the new constraint are as follows:
For each $i \in [0, 2^k - 1]$ and each class $a \in [0, c-1]$, we have the clauses:

$$\neg C_{i,a} \vee U_i \tag{7}$$

These clauses assign the true value to U_i if the leaf i is labelled with a class.

For each $i \in [0, 2^k - 1]$ and each class $j \in [0, MaxNodes + 1]$, we have the clauses:

$$\neg H_{i,j} \vee H_{i+1,j} \tag{8}$$

These clauses propagate the fact that if $H_{i,j}$ is true, then $H_{i+1,j}$ is also true.

For each $i \in [0, 2^k - 1]$ and each class $j \in [0, MaxNodes + 1]$, we have the clauses:

$$\neg U_i \vee \neg H_{i,j} \vee H_{i+1,j+1} \quad (9)$$

These clauses increase the value of H_{i+1} by one if U_i is true. Thus $H_{i+1,j}$ is true if there is at least j leaves labelled by class among the i first leaves.

Finally, we assign the start and end of the counter H :

$$\neg H_{2^{k+1},\lfloor MaxNodes/2 \rfloor + 2} \wedge H_{0,0} \quad (10)$$

The first assignment prohibits having more than $\lfloor MaxNodes/2 \rfloor + 1$ leaves, so $MaxNodes$ nodes. The second assignment sets the counter to 0.

Proposition 1. *The formula for inferring a decision tree of depth k (and a specific number of nodes) from n training examples with m features clustered in c classes requires $O(2^k \times (n+m+c))$ literals, and $O(2^k \times (m^2+m \times n+c))$ clauses.*

Proof. By inspection of the constraints proposed in this section. □

It can be noted that the number of literals and clauses, whether the maximum number of nodes is specified or not, is of the same order of magnitude. However, finding a tree with a minimum number of nodes will take more time because it requires us to search for this number through a dichotomous search.

Incremental Inference

To alleviate the complexity associated with large sets of training examples, we propose an approach which, instead of attempting to process all the training examples $\mathcal{E}$ at once, iteratively infers a decision tree from their subset (initially it is an empty set) and uses active inference to refine it when it is not consistent with one of the training examples. While active inference usually uses an oracle capable of determining to which class an example belongs, we assign this role to the training examples $\mathcal{E}$. Even if such an oracle is restricted because it cannot guess the class for all possible input features, we can demonstrate that it leads to an efficient approach for passive inference from training examples. The approach is formalized in Algorithm 3. Note that this algorithm can easily be used to find a perfect decision tree with a minimal depth. To do this, we can just start with $k = 0$, and increment k until a solution is found. When the minimum depth k is found, we can search among all the decision trees with a minimum depth k, the one with the least node by calling this algorithm dichotomously with $MaxNodes$ between 1 and $2^{k+1} - 1$.

Theorem 3. *Algorithm 3 returns a decision tree consistent with $\mathcal{E}$ with at most $MaxNodes$ nodes and with maximum depth k if it exists.*

Proof. If the algorithm returns a decision tree, it means that the condition of line 4 holds: T is consistent with $\mathcal{E}$. Moreover, because T is a solution for the Boolean formula C then T is a decision tree with at most $MaxNodes$ nodes and with maximum depth k.

If the algorithm returns "No solution", it means that the formula C is unsatisfiable, and there is no decision tree consistent with $\mathcal{E}$ with at most $MaxNodes$ nodes and with the maximum depth k.

The termination of the algorithm is guaranteed by the fact that at each iteration, a new example of $\mathcal{E}$ is considered. Thus, when all examples will be considered, we know that the condition of line 4 will be true. □

Algorithm 3 ($InferDecisionTree$)

Input: The maximum depth k of the tree to infer, the maximal number $MaxNodes$ of nodes of the tree to infer, the set of training examples $\mathcal{E}$ partitioned into c classes $\mathcal{E}_0, ..., \mathcal{E}_{c-1}$.
Output: A decision tree consistent with $\mathcal{E}$ with at most $MaxNodes$ nodes and with maximum depth k if it exists.

1: $C \leftarrow$ formulas (1) and (2)
2: **while** C is satisfiable **do**
3: Let T be a decision tree of a solution of C
4: **if** $\mathcal{E} \subseteq T$ **then**
5: **return** T
6: **end if**
7: Let $e \in \mathcal{E}_a$ be an example mislabeled by T
8: $C \leftarrow C \wedge GenerateFeatureConstraints(e, \emptyset, 1, 0) \wedge GenerateClassConstraints(e, \emptyset, 0, 0, k, a)$
9: **if** $MaxNodes$ is defined **then**
10: $C \leftarrow C \wedge C'$ where C' is a set of clauses described by formulas (7), (8), (9) and (10).
11: **end if**
12: **end while**
13: **return** "No solution"

Benchmarks

We have presented two algorithms to solve two different problems. The first algorithm, denoted DT_depth, finds a perfect decision tree of minimal depth. It uses Algorithm 3 without defining $MaxNodes$. The value of k is initially 1, and while the algorithm is not finding a solution the value k is increased. The second algorithm, denoted DT_size, minimizes the depth of the tree and the number of nodes. It starts by applying DT_depth to learn the minimum depth k required to find a decision tree consistent with the training examples. Then it performs a dichotomic search on the number of nodes allowed between 1 and $2^{k+1} - 1$ to find a decision tree with a minimal number of nodes.

Our two algorithms were implemented[1] in C++ calling the SAT solver MiniSAT (Eén and Sörensson 2003) and we ran experiments on Ubuntu with Intel® Core™ CPU i7-2600K @ 3.40GHz and we limit the memory usage to 2 GB.

We compared our algorithms with the one of Bessiere, Hebrard, and O'Sullivan (2009), denoted $DT2$, and the one of Narodytska et al. (2018), denoted $DT1$. We refer to the

[1] See https://github.com/FlorentAvellaneda/InferDT

results they have published. Note that the characteristics of the machines and tools they used are similar to ours or even better for $DT1$ which uses Intel® Xeon™ 3.50GHz processor with a 2009 SAT solver. Note that additional benchmarks are available at https://github.com/FlorentAvellaneda/InferDT.

The "Mouse" dataset

Our first experiment was performed on the "Mouse" dataset that the authors Bessiere, Hebrard, and O'Sullivan shared with us. This dataset has 70 examples and 45 Boolean features and has the advantage of having been used with both algorithm $DT1$ and $DT2$. In Table 1, we compare the time and accuracy for different algorithms. Each entry in rows DT_size and DT_depth corresponds to the average over 100 runs. The first columns correspond to the name of each algorithm used. The next four columns correspond to inferring a decision tree from the whole dataset. The last column corresponds the 10-fold cross-validations.

Algo	Time (s)	expl	k	n	acc.
$DT2$	577	70	4	15	83.8%
$DT1$	12.9	70	4	15	83.8%
DT_size	0.07	37	4	15	83.5%
DT_depth	**0.02**	33	4	31	**85.8%**

Table 1: Benchmark for "Mouse" dataset.

By analyzing Table 1, we can notice that our incremental approach is very efficient on this dataset. Only 37 examples for DT_size and 33 examples for DT_depth were used to build an optimal decision tree consistent with the entire dataset. Note that even if we disable the incremental approach and consider the 70 examples in the SAT formula, our approach is still much faster than DT1 and DT2 with a resolution time of about 90 ms. We could not compare the accuracy because this data is missing in the two respective papers for $DT2$ and $DT1$.

The "Car" dataset

Another dataset provided to us and used by the authors Bessiere, Hebrard, and O'Sullivan and Narodytska et al. is "Car". This dataset is much more complicated with 1728 examples and 21 Boolean features. To the best of our knowledge, no algorithm has been able to infer an optimal decision tree consistent with the entire dataset. The approach used by the authors Narodytska et al. simplifies the dataset by considering only 10% of the data. Thus, they can infer an optimal decision tree consistent with the 10% of the data selected in 684 sec. Table 2 compares the results of different algorithms. Each entry in rows DT_size and DT_depth corresponds to the average over ten runs. The first four columns correspond to inferring a decision tree from the whole dataset. The last column corresponds the 10-fold cross-validations.

Algo	Time (s)	expl	k	n	acc.
$DT1$	684	173	7	23.67	55%
DT_size	217	635	8	143	**98.8%**
DT_depth	**127**	603	8	511	**98.8%**

Table 2: Benchmark for "Car" dataset.

We can see in Table 2 that out of 1727 examples in the "Car" dataset, our incremental approach uses less than half of it. Although this number is still much higher than the number of examples used by the algorithm $DT1$, we can see that our algorithms run faster. Moreover, since our algorithms ensure that the resulting decision trees are consistent with all training examples, we can see that the accuracy remains very high compared to the $DT1$ algorithm which randomly considers only 10% of training examples.

Other datasets

As mentioned in the introduction of this paper, there is a series of algorithms that address a different but very similar problem to ours: inferring a decision tree with a given depth such that the total classification error on the training examples is minimal. In this section, we compare our results against these algorithms. The datasets we used are extracted from the paper of Verwer and Zhang (2019) and are available at *https://github.com/SiccoVerwer/binoct*. Table 4 shows respectively for each dataset used, the number of examples given, the number of features, the number of features when converted to Boolean and the number of classes.

Dataset	$\lvert\mathcal{E}\rvert$	features	Boolean features	classes
iris	113	4	114	3
Monks-probl-1	124	6	17	2
Monks-probl-2	169	6	17	2
Monks-probl-3	92	6	17	2
wine	178	12	1276	3
balance-scale	625	4	20	3

Table 4: Characteristics of the used datasets.

Each dataset corresponds to a 5-fold cross-validation. Verwer and Zhang compare their approach $BinOCT^*$ to two other approaches. The first one is $CART$ (Breiman et al. 1984), run from sciki-learn with its default parameter setting but with a fixed maximum depth of the trees generated, and the second one is OCT from Bertsimas and Dunn (2017). The depth of trees used is between 2 and 4, but we report in Table 3 the best value among the three depths tested.

It should be noted that only 6 of the 16 datasets present in the Verwer and Zhang paper could be solved. The reason is that the decision trees consistent with some datasets are too large and deep to be inferred. In contrast to the algorithms to which we compare ours, we cannot set a maximum tree depth value because all examples must be correctly classified by the tree we infer.

Note that in Table 3, our algorithms DT_depth and DT_size are very fast even when the trees to be inferred are large. In fact, for the dataset "balance-scale", our algorithms infer decision trees of depth 8 while the other studies limit the depth of the tree to 4.

Dataset	DT_depth			DT_size			$BinOCT^*$	$CART$	OCT
	time (s)	acc.	k	time (s)	acc.	n	acc.	acc.	acc.
iris	0.018	92.9%	3	0.03	93.2%	10.6	**98.4%**	92.4%	93.5%
Monks-probl-1	0.024	90.3%	4.4	0.08	**95.5%**	17	87.1%	76.8%	74.2%
Monks-probl-2	0.19	70.2%	5.8	9.1	**74.0%**	47.8	63.3%	63.3%	54.0%
Monks-probl-3	0.03	78.1%	4.8	0.21	82.6%	23.4	93.5%	**94.2%**	**94.2%**
wine	0.6	89.3%	3	1.2	92.0%	7.8	92.0%	88.9%	**94.2%**
balance-scale	50	**93.0%**	8	183	92.6%	268	78.9%	77.5%	71.6%
Average		85.6%			**88.3%**		85.5%	82.18%	81.1%

Table 3: Benchmark comparing algorithms DT_depth, DT_size, $BinOCT^*$, $CART$ and OCT.

Concerning the accuracy of the trees we inferred, it seems that when the depth is small (< 5) accuracy is similar for all approaches. However, when the depth is bigger, then our algorithms obtain higher accuracy. The most obvious example is the dataset balance-scale where we got 93% accuracy compared to 78.9% for $BinOCT^*$.

Artificial dataset

In this last experiment, we evaluated the time required to infer a decision tree according to the number of learning examples as well as the ability of our algorithms to find decision trees equivalent to the models used to generate the learning examples. To carry out this experiment, we randomly generated 1000 decision trees of a depth 5, with 10 features and 2 classes. We call these trees "generators", and we used them to randomly generate learning examples and check if the models we inferred are equivalent to these generators. We compare our two algorithms to the well-known heuristics *C4.5* (Quinlan 2014) implemented in the Weka tool (Witten et al. 2016) under the name of *J48*. Since we evaluated the percentage of inferred trees equivalent to generators, we deactivated the post-pruning, the MDL correction and the minimum number of instances per leaf in *J48*.

In Figure 2, we depict with solid lines the average time used to infer a decision tree and with dotted lines the percentage of inferred trees equivalent to generators.

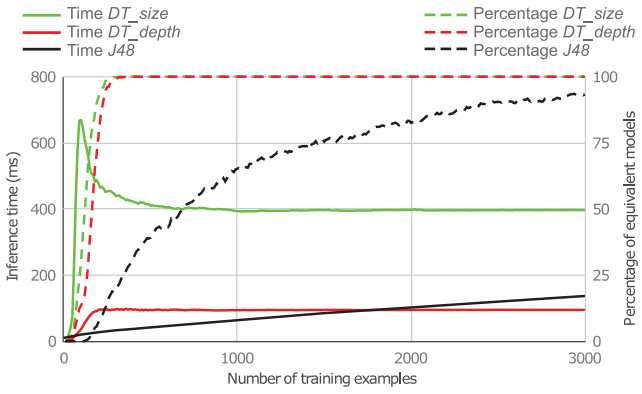

Figure 2: Chart of the average time and accuracy percentage.

In accordance with the principle of parsimony, our algorithms require fewer learning examples than the heuristic approach to infer models that are equivalent to generators. In addition, it is notable that the inference time of our algorithms becomes almost constant when a sufficient number of learning examples are provided. Thus, we find that our DT_{depth} algorithm is faster than the *J48* heuristic when the number of learning examples exceeds 1800, even though with this number of learning examples *J48* has only an 80% chance of inferring trees equivalent to the generators.

Conclusion

We have presented a method that can infer an optimal decision tree for two definitions of optimality. The first definition states that a decision tree consistent with the training example is optimal if it has a minimum depth. The second definition of optimality adds the constraint that the tree, in addition to having a minimum depth, must also have a minimum number of nodes. Although this optimal decision tree inference problem is known to be NP-complete, we proposed an efficient method to solve it.

Our first contribution is an efficient SAT formulation that allows us to infer perfect decision trees for a fixed depth consistent with training examples. We have shown that considering depth as a criterion of simplicity allows a more efficient SAT formula based on a binary coding of the positions of the nodes in the tree. We have also shown how to add constraints in order to set the maximum number of nodes. In this case, the inferred decision tree will no longer necessarily be a perfect tree.

Our second contribution addressed the scalability issue. The previous approach using SAT solver has the disadvantage that the execution time increases significantly with the number of training examples. Thus, we proposed an approach which does not process all the examples at once, instead it does it incrementally.

We evaluated our algorithms using various experiments and compared the execution time and quality of decision trees with other optimal approaches. Experimental results show that our approach performs better than other approaches, with shorter execution times, better prediction accuracy and better scalability. In addition, our algorithms have been able to process datasets for which, to the best of our knowledge, there are no other inference methods capable of producing optimal models consistent with these datasets.

Acknowledgments

I thank Alexandre Petrenko for his input and review of the manuscript and Brigitte Jaumard for her advice on the final version of the document. This work was partially supported by the Ministère de l'Économie et de l'Innovation - Québec.

References

ACM FAT Conference on Fairness, Accountability, and Transparency*.

Alekhnovich, M.; Braverman, M.; Feldman, V.; Klivans, A. R.; and Pitassi, T. 2004. Learnability and automatizability. In *45th Annual IEEE Symposium on Foundations of Computer Science*, 621–630. IEEE.

Angelino, E.; Larus-Stone, N.; Alabi, D.; Seltzer, M.; and Rudin, C. 2017. Learning certifiably optimal rule lists. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 35–44. ACM.

Angluin, D. 1987. Learning regular sets from queries and counterexamples. *Information and computation* 75(2):87–106.

Bertsimas, D., and Dunn, J. 2017. Optimal classification trees. *Machine Learning* 106(7):1039–1082.

Bertsimas, D., and Shioda, R. 2007. Classification and regression via integer optimization. *Operations Research* 55(2):252–271.

Bessiere, C.; Hebrard, E.; and O'Sullivan, B. 2009. Minimising decision tree size as combinatorial optimisation. In *International Conference on Principles and Practice of Constraint Programming*, 173–187. Springer.

Breiman, L.; Friedman, J.; Olshen, R.; and Stone, C. 1984. Classification and regression trees. wadsworth int. *Group* 37(15):237–251.

Cortes, C., and Vapnik, V. 1995. Support-vector networks. *Machine learning* 20(3):273–297.

Eén, N., and Sörensson, N. 2003. An extensible SAT-solver. In *International conference on theory and applications of satisfiability testing*, 502–518. Springer.

Goebel, R.; Chander, A.; Holzinger, K.; Lecue, F.; Akata, Z.; Stumpf, S.; Kieseberg, P.; and Holzinger, A. 2018. Explainable AI: the new 42? In *International Cross-Domain Conference for Machine Learning and Knowledge Extraction*, 295–303. Springer.

Hancock, T.; Jiang, T.; Li, M.; and Tromp, J. 1996. Lower bounds on learning decision lists and trees. *Information and Computation* 126(2):114–122.

Hyafil, L., and Rivest, R. L. 1976. Constructing optimal binary decision trees is np-complete. *Information Processing Letters* 5(1):15–17.

IJCAI workshop on explainable artificial intelligence.

Kass, G. V. 1980. An exploratory technique for investigating large quantities of categorical data. *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 29(2):119–127.

Li, O.; Liu, H.; Chen, C.; and Rudin, C. 2018. Deep learning for case-based reasoning through prototypes: A neural network that explains its predictions. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

McCulloch, W. S., and Pitts, W. 1943. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics* 5(4):115–133.

Narodytska, N.; Ignatiev, A.; Pereira, F.; Marques-Silva, J.; and RAS, I. 2018. Learning optimal decision trees with SAT. In *IJCAI*, 1362–1368.

NIPS interpretable ML symposium.

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

Quinlan, J. R. 2014. *C4.5: programs for machine learning*. Elsevier.

Sieling, D. 2008. Minimization of decision trees is hard to approximate. *Journal of Computer and System Sciences* 74(3):394–403.

Van Lent, M.; Fisher, W.; and Mancuso, M. 2004. An explainable artificial intelligence system for small-unit tactical behavior. In *Proceedings of the National Conference on Artificial Intelligence*, 900–907. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

Verwer, S., and Zhang, Y. 2019. Learning optimal classification trees using a binary linear program formulation. In *33rd AAAI Conference on Artificial Intelligence*.

Witten, I. H.; Frank, E.; Hall, M. A.; and Pal, C. J. 2016. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann.