Fixed-Parameter Tractability of Learning Small Decision Trees (full paper)

Anonymous author
Anonymous affiliation

1 Introduction

Decision trees have proved to be extremely useful tools for the describing, classifying, generalizing data [14, 18, 20]. In this paper, we consider decision trees for *classification instances (CIs)*, consisting of a finite set E of *examples* (also called *feature vectors*) over a finite set F of *features*. Each example $e \in E$ is a function $e : F \to \{0, 1\}$ which determines whether the feature f is true or false for e. Moreover, E is given as a partition $E^+ \uplus E^-$ into positive and negative examples. For instance, examples could represent medical patients and features diagnostic tests; a patient is positive or negative corresponding to whether they have been diagnosed with a certain disease or not. CIs are also called *partially* or *incompletely defined Boolean functions*, as we can consider the features as Boolean variables, and examples as truth assignments that evaluate to 0 (for positive examples) or 1 (for negative examples). CIs have been studied as a key concept for the logical analysis of data and in switching theory [2, 4, 3, 5, 6, 13, 16].

Because of their simplicity, decision trees are particularly attractive for providing interpretable models of the underlying CI, an aspect whose importance has been strongly emphasized over the recent years [8, 9, 11, 15, 17]. In this context, one prefers *small trees*, as they are easier to interpret and require fewer tests to make a classification. Small trees are also preferred in view of the parsimony principle (Occam's Razor) since small trees are expected to generalize better to new data [1]. However, finding a small decision tree, as formulated in the following decision problem, is NP-complete [12].

> Minimum Decision Tree Size (DTS): given a CI $E = E^+ \uplus E^-$ and an integer s, is there a decision tree with at most s nodes for E?

Given this complexity barrier, we propose a fixed-parameter algorithm for the problem, which exploits the input CI's hidden structure.

2 Preliminaries

2.1 Parameterized Complexity

We give some basic definitions of Parameterized Complexity and refer for a more in-depth treatment to other sources [7, 10]. Parameterized complexity considers problems in a two-dimensional setting, where a problem instance is a pair (I, k), where I is the main part and k is the parameter. A parameterized problem is *fixed-parameter tractable* if there exists a computable function f such that instances (I, k) can be solved in time $f(k)\|I\|^{O(1)}$.

2.2 Classification Problems

An *example* e is a function $e : \mathit{feat}(e) \to \{0, 1\}$ defined on a finite set $\mathit{feat}(e)$ of *features*. For a set E of examples, we put $\mathit{feat}(E) = \bigcup_{e \in E} \mathit{feat}(e)$. We say that two examples e_1, e_2 *agree* on a feature f if $f \in \mathit{feat}(e_1)$, $f \in \mathit{feat}(e_2)$ and $e_1(f) = e_2(f)$. If $f \in \mathit{feat}(e_1)$, $f \in \mathit{feat}(e_2)$ but $e_1(f) \neq e_2(f)$, we say that the examples *disagree on* f.

A *classification instance (CI)* (also called a *partially defined Boolean function* [13]) $E = E^+ \uplus E^-$ is the disjoint union of two sets of examples, where for all $e_1, e_2 \in E$ we have $\mathit{feat}(e_1) = \mathit{feat}(e_2)$. The examples in E^+ are said to be *positive*; the examples in E^- are said to be *negative*. A set X of examples is *uniform* if $X \subseteq E^+$ or $X \subseteq E^-$; otherwise X is *non-uniform*.

Given a CI E, a subset $F \subseteq \mathit{feat}(E)$ is a *support set* of E if any two examples $e_1 \in E^+$ and $e_2 \in E^-$ disagree in at least one feature of F. Finding a smallest support set, denoted by MSS(E), for a classification instance E is an NP-hard task [13, Theorem 12.2].

We define the *incidence graph* of E, denoted by $G_I(E)$, as the bipartite graph with partition $(E, \mathit{feat}(E))$ having an edge between an example $e \in E$ and a feature $f \in \mathit{feat}(e)$ if $f(e) = 1$.

2.3 Decision Trees

A *decision tree* (DT) (or *classification tree*) is a rooted tree T with vertex set $V(T)$ and arc set $A(T)$, where each non-leaf node (called a *test*) $v \in V(T)$ is labelled with a feature $\mathit{feat}(v)$, each non-leaf node v has exactly two out-going arcs, a *left arc* and a *right arc*, and each leaf is either a *positive* or a *negative* leaf. We write $\mathit{feat}(T) = \{\, v \in V(T) \mid \mathit{feat}(v) \,\}$.

Consider a CI E and a decision tree T with $\mathit{feat}(T) \subseteq \mathit{feat}(E)$. For each node v of T we define $E_T(v)$ as the set of all examples $e \in E$ such that for each left (right, respectively) arc (u, v) on the unique path from the root of T to v we have $e(\mathit{feat}(v)) = 0$ ($e(\mathit{feat}(v)) = 1$, respectively). T *correctly classifies* an example $e \in E$ if e is a positive (negative) example and $e \in E_T(v)$ for a positive (negative) leaf. We say that T *classifies* E (or simply that T is a DT for E) if T correctly classifies every example $e \in E$. See Figure **??** for an illustration of a CI, its incidence graph, and a DT that classifies E.

The size of T is its number of nodes, i.e. $|V(T)|$. We consider the following problem.

Minimum Decision Tree Size (DTS)	
Input:	A classification instance E and an integer s.
Question:	Is there a decision tree of size at most s for E?

3 An FPT-Algorithm for bounded solution size and δ_{max}.

In the following, let E be a CI and $q \notin feat(E)$. A *decision tree pattern*, or simply a *DT pattern*, T is a rooted subcubic tree, where every leaf node is either a *positive* or *negative* leaf and every non-leaf node is labelled with a feature in $feat(E) \cup \{q\}$. For every node v of a DT pattern T, we indicate with $feat_T(v)$ the label associated to that node. Finally we say that an innder node $v \in V(T)$ is a *fixed node* if $feat_T(v) \in feat(E)$ and *non-fixed* otherwise.

A DT pattern T' is an *improvement* for a DT pattern T if $T' = T$ as rooted trees and $feat_{T'}(v) = feat_T(v)$ for every fixed node v of T. A *complete improvement* T' of T is an improvement such that $feat(T') \subseteq feat(E)$. A *threshold assignment* for a DT pattern T is a function th that maps every fixed node $v \in V(T)$ to a natural number $th(v)$. Note that any complete improvement T' of a DT pattern T can be made to a decision tree with a threshold assignment.

Let T be a DT pattern and th be a threshold assignment for T, for each node v of T we define the set of examples that arrive at node v, $E_T(v)$ as follows: $E_T(v)$ is the set of all examples $e \in E$ such that for each left (right, respectively) arc (u, w) on the unique path from the root of T to v either u is a fixed node and $(feat(u))(e) \leq th(u)$ ($(feat(u))(e) > th(u)$, respectively) or u is a non-fixed node. A DT pattern T is *valid* for a set of examples $E' \subseteq E$ if there is threshold assignment for the fixed nodes such that for every positive (negative) example e, $e \in E_T(v)$ for a positive (negative) leaf v.

The definition of $E_T(v)$ is an indication of the behaviour of feature q and of non-fixed nodes. Informally, if any example reaches at a non-fixed node of T then it reach both his

children. While no feature in $feat(E)$ can simulate such behaviour for any threshold, q simultaneously cover the two cases a feature with his threshold does not distinguish any two examples.

3.1 Preprocess

Let E be a CI and T be a DT pattern. For every $v \in V(T)$, we define the set of *expected examples* E_v as follows:

- if v is the root, then $E_v = E$;
- if v is the left child of a fixed node v_p, then $E_v = E_{v_p}[feat(v_p) \leq th_L(v_p) + 1]$;
- if v is the right child of a fixed node v_p, then $E_v = E_{v_p}[feat(v_p) > th_R(v_p) - 1]$;
- if v is a child of a non-fixed node v_p, then $E_v = E_{v_p}$.

Node that the definition of E_v is strictly related with the following: if v is a fixed node, let c_ℓ and c_r be the left, risp. right, child of v, we define two values $th_L(v)$ and $th_R(v)$ as follows:

- let $th_L(v)$ be the maximum value in $D_E(feat(v))$ such that T_{c_ℓ} is valid for $E_v[feat(v) \leq th_L(v)]$;
- let $th_R(v)$ be the minimum value in $D_E(feat(v))$ such that T_{c_r} is valid for $E_v[feat(v) > th_R(v)]$.

Before formally proving in Lemma 3 that we are able to compute E_v and $th_L(v)$, $th_R(v)$ (when v is a fixed node) for every $v \in V(T)$, we want to describe the role of E_v in the proof of Lemma 5.

Let us consider the following situation. Suppose we are trying to find a DT of minimum size for a CI E using at least the features in a given support set S. The first step would be to compute a minimum size DT T^* for E such that $feat(T^*) = S$. Next we analyse the case an optimal DT for E uses not only every feature from S but some additional feature: for this reason we consider DT patterns T of size at most s and such that $feat(T) = S \cup \{q\}$.

Let E be a CI, S be a support set for E and T be a DT pattern of size at most s such that $feat(T) = S \cup \{q\}$. If T is a valid DT pattern for E, then T, and every T' obtained after left/right-contracting every non-fixed node v of T, can be easily extended to a solution.

The following two lemmas cover the case T is not a valid DT pattern for E.

▶ **Lemma 1.** *Let T be a DT pattern that is not valid for E. For every node v of T it holds that T_v is not valid for E_v.*

Proof. Let T be a DT pattern that is not valid for E. We show this statement in a root-to-leaves fashion: first we show the statement holds for the root; then we prove it holds for every other node, given the fact it holds for each of its ancestors (or its parent). Let r be the root of T. By definition $E_r = E$ and $T_r = T$ and so the statement follows directly from the assumption.

Let v be the left child of a fixed node v_p. By the definition of $th_L(v_p)$, the DT pattern T_v is not valid for $E_v = E_{v_p}[feat(v_p) \leq th_L(v_p) + 1]$. Similarly if v is the right child of a fixed node v_p, the DT pattern T_v is not valid for $E_v = E_{v_p}[feat(v_p) > th_R(v_p) - 1]$.

Let v be a child of a non-fixed node v_p. Suppose by contradiction that T_v is valid for E_v. We show that T_{v_p} is valid for E_{v_p} and consequently reaching a contradiction with the assumption: any threshold assignment for the fixed nodes of T_v that is a witness of the validity of T_v for E_v is also threshold assignment for the fixed nodes of T_{v_p} that is a witness of the validity of T_{v_p} for $E_{v_p} = E_v$; note this is true because v_p is a non-fixed node. ◀

▶ **Lemma 2.** *Let T be a DT pattern that is not valid for E. For every fixed node v of T it holds that $th_L(v) < th_R(v)$.*

Proof. Let T be a DT pattern that is not valid for E. Suppose by contradiction that there is a fixed node v^* such that $th_L(v^*) \geq th_R(v^*)$. Let c_ℓ and c_r be the left and right child of v^*. We can set the threshold for $feat(v^*)$ as $th_L(v^*)$ and note that, by definition and the assumption, T_{c_ℓ} is valid for E_{c_ℓ} and T_{c_r} is valid for E_{c_r}. This is a contradiction with Lemma 1 as for every node $v \in V(T)$, T_v is not valid for E_v. ◀

Now we are finally ready to prove we can efficiently compute E_v, $th_L(v)$ and $th_R(v)$ for every node $v \in V(T)$.

▶ **Lemma 3.** *Let E be a CI, let T be a DT pattern of depth at most d. Then there is an algorithm that runs in time $\mathcal{O}(2^{d^2/2} n^{1+o(1)} \log n)$ and computes the set E_v and thresholds $th_L(v)$ and $th_R(v)$ for every node $v \in V(T)$.*

Proof. The idea is to use the recursive algorithm **findLR** illustrated in Algorithm 1. That is, given E, T, the algorithm **findLR** attempts to find the triple $(E_v, th_L(v), th_R(v))$ for every node $v \in V(T)$. Lines 3 to 4: if T consists of a leaf node, the algorithm just report $(E, \texttt{nil}, \texttt{nil})$. Let c_ℓ and c_r be the left, risp. right, child of the root v. Lines 6 to 11: if the root of T is a non-fixed node, the algorithm calls itself recursively to compute on (E, T_{c_ℓ}) and (E, T_{c_r}). Lines 13 to 15: if the root of T is a fixed node v, the algorithm computes the pair (t_ℓ, t_r) for the root using the algorithm **binarySearch** and then calls itself recutsively to compute the triple for $(E[feat(v) \leq t_\ell + 1], T_{c_\ell})$ and $(E[feat(v) > t_r - 1], T_{c_r})$.

A key element for the correctness of **findLR** is the algorithm **binarySearch** illustrated in Algorithm 2. Given E, T, f, c_ℓ and c_r, this algorithm computes the pair (t_ℓ, t_r) for the root of T that has feature f. This sub-routine performs a standard binary search procedure on the array D containing all the values in $D_E(f)$ in ascending order to find maximum t_ℓ and minimum t_r such that T_{c_ℓ} and T_{c_r} can be extended to DT for $E[f \leq t_\ell]$ and for $E[f > t_r]$ respectively. To achieve this, the sub-routine makes at most $log|E|$ calls to **findTH**; note that each of those calls is made for a tree of smaller depth. Lines 3 to 12: the algorithm finds the maximum t_ℓ by calling algorithm **findTH** in Line 6 repeatedly. Lines 13 to 22: the algorithm finds the minimum t_r by calling algorithm **findTH** in Line 16 repeatedly.

A sub-routine used for **binarySearch** is the algorithm **findTH** illustrated in Algorithm 3. This algorithm is very similar to Algorithm 1 but the output is some way much simpler.

The running time of Algorithm 1 can now be obtained by multiplying the number of recursive calls to **findLR** with the time required for one recursive call. To obtain the number of recursive calls first note that if **findLR** is called with DT pattern of depth d, then it makes at most $(2 \log n) + 2$ recursive calls to **findLR** with a pattern of depth at most $d - 1$, where $n = |E|$. Therefore the number $T(n, d)$ of recursive calls for a pattern of depth d is given by the recursion relation $T(n, d) = (2(\log n) + 2)T(n, d - 1)$ starting with $T(n, 0) = 0$. This implies that $T(n, d) \in \mathcal{O}((\log n)^d)$. Finally, the runtime for one recursive call is easily seen to be at most $\mathcal{O}(n \log n)$. Hence, the total runtime of the algorithm is at most $\mathcal{O}((\log n)^d n \log n)$, which because (see also [7, Exercise 3.18]):

$$(\log n)^d \leq 2^{d^2/2} 2^{\log \log d^2/2} = 2^{d^2/2} n^{o(1)}$$

is at most $\mathcal{O}(2^{d^2/2} n^{1+o(1)} \log n)$. ◀

Algorithm 1 Algorithm to compute the triple $(E_v, th_L(v), th_R(v))$ for every node $v \in V(T)$.

Input: CI E, DT pattern T
Output: a triple $(E_v, th_L(v), th_R(v))$ for every node $v \in V(T)$.

function findLR(E, T)
  $r \leftarrow$ "root of T"
  if r is a leaf **then**
    return $(E, \texttt{nil}, \texttt{nil})$
  $c_\ell, c_r \leftarrow$ "left child and right child of r"
  if r is a non-fixed node **then**
    $\lambda_\ell \leftarrow$ findLR(E, T_{c_ℓ})
    $\lambda_r \leftarrow$ findLR(E, T_{c_r})
    if $\lambda_\ell \neq \texttt{nil}$ and $\lambda_r \neq \texttt{nil}$ **then**
      return $(E, \texttt{nil}, \texttt{nil}) \cup \lambda_\ell \cup \lambda_r$
    return nil
  $f \leftarrow feat(r)$
  $(t_\ell, t_r) \leftarrow$ binarySearch(E, T, f, c_ℓ, c_r)
  $\lambda_\ell \leftarrow$ findLR($E[f \leq t_\ell + 1]$, T_{c_ℓ})
  $\lambda_r \leftarrow$ findLR($E[f > t_r - 1]$, T_{c_r})
  return $(E, t_\ell, t_r) \cup \lambda_\ell \cup \lambda_r$

3.2 The algorithm

Now we have computed a set E_v for every node $v \in V(T)$, whether it is a leaf, fixed or non-fixed node. A *pool set* for node $v \in V(T)$ is a set $\Pi(v) \subseteq E_v$, such that if $\Pi(v) \subseteq E_T(v)$ then either

- T_v is not valid for E_v, or
- for any complete improvement T'_v for T_v that that is valid for E_v, there are two elements $e, e' \in \Pi(v)$ and there is a non-fixed node u for T such that $feat_{T'}(u)$ must distinguish e and e'.

For every node $v \in V(T)$, we define $\Pi(v)$ in a leaves-to-root fashion as follows. If v is a negative leaf then $\Pi(v) = \{e^+\}$, where e^+ is any example in $E^+ \cap E_v$; similarly, if v is a positive leaf then $\Pi(v) = \{e^-\}$, where e^- is any example in $E^- \cap E_v$. Let c_ℓ and c_r be the left, resp. right, child of v, then $\Pi(v) = \Pi(c_\ell) \cup \Pi(c_r)$.

Now we want to show that the construction of Π is correct, that is:

▶ **Lemma 4.** *$\Pi(v)$ is a pool set for v for every node $v \in V(T)$.*

Proof. We show this by induction on the depth of T and let v be the root of T. Since $E_T(v) = E$ it is trival to note that $\Pi(v) \subseteq E_T(v)$. We start proving the base case: let T be a pattern of depth 0. Suppose v is negative leaf. Since $E_v = E$ is not uniform, there is an example $e^+ \in E^+ \cap E_v$. The case where v is a positive leaf can be proved in a symmetrical manner.

Now, let T be a pattern of depth at least one and let c_ℓ and c_r be the left and right child of v. Suppose first that v is a fixed node and let $f = feat(v)$. Thanks to Lemma 1, for every $e_\ell \in \Pi(c_\ell)$ and $e_r \in \Pi(c_r)$, we know that $f(e_\ell) < f(e_r)$. This means that either $\Pi(c_\ell) \subseteq E_T(c_\ell)$ or $\Pi(c_r) \subseteq E_T(c_r)$, say that $\Pi(c_i) \subseteq E_T(c_i)$, for $i \in \{\ell, r\}$. Since T_{c_i} has depth smaller than $T_v = T$, by the inductive hypothesis $\Pi(c_i)$ is a pool set for c_i.

Finally suppose v is a non-fixed node. Let us consider any complete improvement T'_v for T_v. For any threshold assignment for T'_v, we have one of the following three cases: either

Algorithm 2 Algorithm to compute the pair $(th_L(r), th_R(r))$ for the root r of T

Input: CI E, DT pattern T, feature f of the root of T, left child c_ℓ of the root of T, right child c_r of the root of T

Output: maximum threshold t_ℓ in $D_E(f)$ for f such that (T_{c_ℓ}, α) can classify every example in $E[f \leq t_\ell]$ and minimum threshold t_r in $D_E(f)$ for f such that (T_{c_r}, α) can classify $E[f > t_r]$

function binarySearch(E, T, f, c_ℓ, c_r)
  $D \leftarrow$ "array containing all elements in $D_E(f)$ in ascending order"
  $L \leftarrow 0$; $R \leftarrow |D_E(f)| - 1$; $b \leftarrow 0$
  while $L \leq R$ **do**
    $m \leftarrow \lfloor (L + R)/2 \rfloor$
    if FINDTH($E[f \leq D[m]]$, T_{c_ℓ}) = TRUE **then**
      $L \leftarrow m + 1$; $b \leftarrow 1$
    else
      $R \leftarrow m - 1$; $b \leftarrow 0$
  if $b = 1$ **then**
    $t_\ell \leftarrow D[m]$
  $t_\ell \leftarrow D[m - 1]$ ▷ assuming that $D[-1] = D[0] - 1$
  $L \leftarrow 0$; $R \leftarrow |D_E(f)| - 1$; $b \leftarrow 0$
  while $L \leq R$ **do**
    $m \leftarrow \lfloor (L + R)/2 \rfloor$
    if FINDTH($E[f > D[m]]$, T_{c_r}) = TRUE **then**
      $R \leftarrow m - 1$; $b \leftarrow 1$
    else
      $L \leftarrow m + 1$; $b \leftarrow 0$
  if $b = 1$ **then**
    $t_r \leftarrow D[m]$
  $t_r \leftarrow D[m + 1]$ ▷ assuming that $D[|D_E(f)|] = D[|D_E(f)| - 1] + 1$
  return (t_r, t_r)

$\Pi(c_\ell) \subseteq E_{T'}(c_\ell)$ or $\Pi(c_r) \subseteq E_{T'}(c_r)$ or there is an example $e_\ell \in \Pi(c_\ell)$ and an example $e_r \in \Pi(c_r)$ such that $e_\ell \in E_{T'}(c_r)$ and $e_r \in E_{T'}(c_\ell)$. In the first two cases the statement is again proven thanks to the inductive hypothesis since T_{c_ℓ} and T_{c_r} have depth smaller than T_v. In the third case, v is a non-fixed node for T such that $feat_{T'}(v)$ distinguishes e_ℓ and e_r. ◀

In particular, let us consider the pool set $\Pi(r)$ for the root r of T, we define $\Pi(T) := \Pi(r)$. In this way given T, we are able to compute the corresponding pool set.

Let S be a support set for a CI E, we stay that $B \subseteq feat(E)$ is a *branching set* for S if for every minimal DT T for E such that $S \subset feat(T)$ then $B \cap (feat(T) \setminus S) \neq \emptyset$.

▶ **Lemma 5.** *There is a $\mathcal{O}(2^{d^2/2} s^{2s+1} n^{1+o(1)} \log n)$ time algorithm that given a support set S computes a branching set R_0 for S of size at most $s^{2s+3}\delta_{\max}$.*

Proof. Let E be a CI, a support set S for E and an integer s. We start by enumerating all DT patterns T of size at most s such that $feat(T) = S \cup \{q\}$. For every such DT pattern T, thanks to Lemma 3, we are able to obtain the set E_v for every node $v \in V(T)$ in time $\mathcal{O}(2^{d^2/2} n^{1+o(1)} \log n)$. In a leaves-to-root fashion, we are able to compute the set $\Pi(v)$ for every node $v \in V(T)$ and ultimately $\Pi(T)$.

Let $R(T)$ be the set of all the features in $feat(E) \setminus S$ that distinguish at least two examples in $\Pi(T)$. The algorithm returns the set of features R_0 obtained by considering the union of

Algorithm 3

Input: CI E, pattern T
Output: `TRUE` if T can classify all examples in E, `FALSE` otherwise

function findTH(E, T)
  $r \leftarrow$ "root of T"
  if r is a leaf **then**
    if E is not uniform **then**
      return `FALSE`
    return `TRUE`
  $c_\ell, c_r \leftarrow$ "left child and right child of r"
  if r is a non-fixed **then**
    $\lambda_\ell \leftarrow$ FINDTH(E, T_{c_ℓ})
    $\lambda_r \leftarrow$ FINDTH(E, T_{c_r})
    if $\lambda_\ell =$ `TRUE` and $\lambda_r =$ `TRUE` **then**
      return `TRUE`
    return `FALSE`
  $f \leftarrow feat(r)$
  $t \leftarrow$ BINARYSEARCH(E, T, f, c_ℓ, c_r)
  $\lambda_\ell \leftarrow$ FINDLR($E[f \leq t_\ell + 1]$, T_{c_ℓ})
  $\lambda_r \leftarrow$ FINDLR($E[f > t_r - 1]$, T_{c_r})
  if $\lambda_r =$ `FALSE` **then**
    return `FALSE`
  return `TRUE`

the sets $R(T)$ over all these DT patterns T of size at most s. By Lemma **??** this algorithm runs in time $\mathcal{O}(2^{d^2/2} s^{2s+1} n^{1+o(1)} \log n)$.

Now we show the size of R_0 is bounded. By construction $|\Pi(T)| \leq |T| \leq s$; for every two distinct elements of $\Pi(T)$, by definition, there are at most $\delta_{\max}$ features that distinguish such two examples. This means that $|R(T)| \leq s^2 \delta_{\max}$ and so R_0 has size at most $s^{2s+3} \delta_{\max}$.

We are left to show that R_0 is a branching set for S. Let T be a minimal DT for E such that $S \subset feat(T)$ and suppose by contradiction that $R_0 \cap (feat(T) \setminus S) = \emptyset$. In particular we have that $R(T) \cap (feat(T) \setminus S) = \emptyset$. This means that for every feature f of T that does not belong to S, f does not distinguish any two elements in $\Pi(T)$. By Lemma 4, $\Pi(T) = \Pi(r)$, where r is the root of T, is a pool set and so T is not valid for E, which is a contradiction. ◀

▶ **Lemma 6** ([19]). *Let E be a CI and let k be an integer. Then there is an algorithm that in time $\mathcal{O}(\delta_{\max}(E)^k |E|)$ enumerates all (of the at most $\delta_{\max}(E)^k$) minimal support sets of size at most k for E.*

▶ **Lemma 7** ([19]). *Let T be a DT of minimum size for E and let S be a support set contained in feat(T). Then, the set $R = feat(T) \setminus S$ is useful.*

▶ **Observation 8** ([19]). *Let T be a DT for a CI E, then feat(T) is a support set of E.*

Proof. Suppose for a contradiction that this is not the case and there is an example $e^+ \in E^+$ and an example $e^- \in E^-$ such that e^+ and e^- agree on all features in $feat(T)$. Therefore, e^+ and e^- are contained in the same leaf node of T, contradicting our assumption that T is a DT. ◀

▶ **Theorem 9** ([19]). *Let E be a CI, $S \subseteq feat(E)$ be a support set for E, and let s and d be integers. Then, there is an algorithm that runs in time $2^{\mathcal{O}(s^2)} \|E\|^{1+o(1)} \log \|E\|$ and computes a DT of minimum size among all DTs T with $feat(T) = S$ and $size(T) \leq s$ if such a DT exists; otherwise* `nil` *is returned.*

▶ **Theorem 10.** Minimum Decision Tree Size *is fixed-parameter tractable prarametrized by* $\delta_{\max} + s$.

Proof. We start by presenting the algorithm for Minimum Decision Tree Size, which is illustrated in Algorithm 4 and Algorithm 5.

Given a CI E and an integer s, the algorithm returns a DT of minimum size among all DTs of size at most s if such a DT exists and otherwise the algorithm returns `nil`. The algorithm **minDT** starts by computing the set $\mathcal{S}$ of all minimal support sets for E of size at most s, which because of Lemma 6 results in a set $\mathcal{S}$ of size at most (). In Line 4 the algorithm then interates over all sets S in $\mathcal{S}$ and calls the function **minDTS** given in Algorithm 5 for E, s, and S, which returns a DT of minimum size among all DTs T for E of size at most s such that $S \subseteq \mathit{feat}(T)$. It then updates the currently best decision tree B if necessary with the DT found by the function **minDTS**. Moreover, if the best DT found after going through all sets in $\mathcal{S}$ has size at most s, it is returned (in Line 9), otherwise the algorithm returns `nil`. Finally, the function **minDTS** given in Algorithm 5 does the following. It first computes a DT T of minimum size that uses exactly the features in S using Lemma 9. It then tries to improve upon T with the help of useful sets. That is, it uses Lemma 5 to compute the branching set R_0. It then interates over all (of the at most ()) features $f \in R_0$ (using the for-loop in Line 4), and calls itself recursively on the support set $S \cup \{f\}$. If this call finds a smaller DT, then the current best DT is updated. Finally, after the for-loop the algorithm either returns a solution if its size is less then s or `nil` otherwise.

Towards showing the correctness of Algorithm 4, consider the case that E has a DT of size at most s and let T be a such a DT of minimum size. Because of Observation 8, $\mathit{feat}(T)$ is a support set for E and therefore $\mathit{feat}(T)$ contains a minimal support set S of size at most s. Because the for-loop in Line 4 of Algorithm 4 interates over all minimal support sets of size at most s for E, it follows that Algorithm 5 is called with parameters E, s, and S. If $\mathit{feat}(T) = S$, then B is set to a DT for E of size $|T|$ in Line 2 of Algorithm 5 and the algorithm will output a DT of size at most $|T|$ for E. If, on the other hand, $\mathit{feat}(T) \setminus S \neq \emptyset$, then because T has minimum size and S is a support set for E with $S \subseteq \mathit{feat}(T)$, we obtain from Lemma 7 that the set $R = \mathit{feat}(T) \setminus S$ is useful for S. Therefore, because of Lemma 5, R has to contain a feature f from the set R_0 computed in Line 3. It follows that Algorithm 5 is called with parameters E, s, and $S \cup \{v\}$. From now onwards the argument repeats and since $R_0 \neq \emptyset$ the process stops after at most $s - |S|$ recursive calls after which a DT for E of size at most $|T|$ will be computed in Line 2 of Algorithm 5. Finally, it is easy to see that if Algorithm 4 outputs a DT T, then it is a valid solution. This is because, T must have been computed in Line 2 of Algorithm 5, which implies that T is a DT for E. Moreover, T has size at most s, because of Line 8 in Algorithm 4.

To analyse the run-time of the algorithm, we first remark that the whole algorithm can be seen as a bounded-depth search tree algorithm, i.e., a branching algorithm with small recursion depth and few branches at every node. In particular, every recursive call adds at least one feature to the set of features bounding the recursion depth to at most s. Moreover, every feature that is added is either added in Line 2 of Algorithm 4, when enumerating all minimal support sets, in which case there are at most $\delta_{\max}(E)$ branches or the feature is added in Line 5 of Algorithm 5, in which case there are at most $|R_0| \leq s^{2s+3}\delta_{\max}(E)$ branches. It follows that the algorithm can be seen as a branching algorithm of depth at most s with at most $s^{2s+3}\delta_{\max}(E) = \max\{s^{2s+3}\delta_{\max}(E), \delta_{\max}(E)\}$ branches at every step. Therefore, the total run-time of the algorithm is at most the number of nodes in the branching tree, i.e., at most $(s^{2s+3}\delta_{\max}(E))^s$, times the maximum time required in one recursive call. Now the maximum time required for one recursive call is dominated

by the time spend in Line 2 of Algorithm 5, i.e., the time required to compute a DT of minimum size using exactly the features in S with the help of Theorem 9, which is at most $2^{\mathcal{O}(s^2)}\|E\|^{1+o(1)}\log\|E\|$. Therefore, we obtain $(s^{2s+3}\delta_{\max}(E))^s 2^{\mathcal{O}(s^2)}\|E\|^{1+o(1)}\log\|E\|$ as the total run-time of the algorithm, which shows that DTS is fixed-parameter tractable parameterized by $s+\delta_{\max}(E)$. ◀

Algorithm 4 Main method for finding a DT of minimum size.

Input: CI E and integer s
Output: DT for E of minimum size (among all DTs of size at most s) if such a DT exists, otherwise `nil`

```
1: function minDT(E, s)
2:     S ← "set of all minimal support sets for E of size at most s using Lemma 6"
3:     B ← nil
4:     for S ∈ S do
5:         T ← MINDTS(E, s, S)
6:         if (T ≠ nil) and (B = nil or |B| > |T|) then
7:             B ← T
8:     if B ≠ nil and |B| ≤ s then
9:         return B
10:    return nil
```

Algorithm 5 Method for finding a DT of minimum size using at least the features in a given support set S.

Input: CI E, integer s, support set S for E with $|S| \leq s$
Output: DT of minimum size among all DTs T for E of size at most s such that $S \subseteq feat(T)$; if no such DT exists, `nil`

```
1: function minDTS(E, s, S)
2:     B ← "compute a DT of minimum size for E using exactly the features in S using Theorem ??"
3:     R_0 ← "compute the branching set R_0 for S using Lemma 5"
4:     for f ∈ R_0 do
5:         T ← MINDTS(E, s, S ∪ {f})
6:         if T ≠ nil and |T| < |B| then
7:             B ← T
8:     if |B| ≤ s then
9:         return B
10:    return nil
```

4 Approximation

4.1 For Size

For every natural number $k \geq 1$, we can define a CI E_k as follows. Let E_k be the CI with exactly 2^k examples $\{e_1, \ldots, e_{2^k}\}$ on k binary features $\{f_1, \ldots, f_k\}$: there is exactly one example for every of the 2^k feature assignments. An example $e \in exam(E_k)$ is a positive example if $|\{f \in feat(E_k) \mid f(e) = 1\}|$ is even and negative otherwise.

Let D_k be the set of all the examples e in $exam(E_k)$ such that $f_i(e) = 1$ for every $i \in [k-2]$ and denote by $\overline{D_k}$ the set $exam(E_k) \setminus D_k$. Now we are ready to define a new feature f^* as follows: $f^*(e) = 1$ if either e is a positive example or $e \in D_k$ and $f^*(e) = 0$ otherwise.

f_1	f_2	f_3	f^*	
0	0	0	1	+
0	0	1	0	−
0	1	0	0	−
0	1	1	1	+
1	0	0	1	−
1	0	1	1	+
1	1	0	1	+
1	1	1	1	−

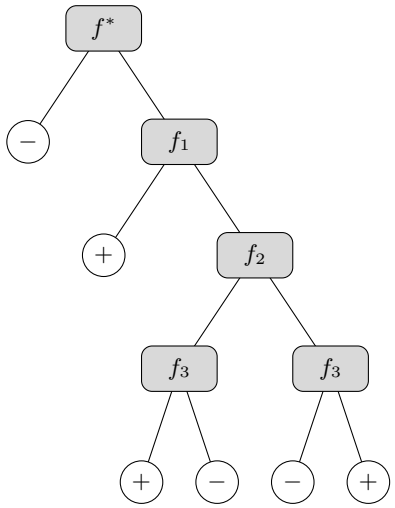

Figure 1

▶ **Lemma 11.** *The features in* $\{f_1, \ldots, f_k\}$ *form the only minimal support set in* $S = \{f_1, \ldots, f_k, f^*\}$ *for* E_k.

Proof. First we show that $\{f_1, \ldots, f_k\}$ is a support set: let $e^- \in E_k^-$ and $e^+ \in E_k^+$, by construction there is one feature $f \in \{f_1, \ldots, f_k\}$ where $f(e^+) \neq f(e^-)$. Now it is time to show that, for any $i \in [k]$, the set $S_i = \{f_1, \ldots, \overline{f_i}, \ldots, f_k, f^*\} = S \setminus \{f_i\}$ is not a support set for E_k. For $i \in [k-2]$, let e_i^- and e_i^+ be the negative and the positive examples such that $f(e_i^-) = f(e_i^+) = 1$ if k is odd ($= 0$ if k is even) for every $f \in S_i$: e_i^- and e_i^+ can not be distinguished by a feature in S_i and so S_i is not a support set (note there is one such pair $exam(E_k)$). For $i \in \{k-1, k\}$, let e_i^- and e_i^+ be the negative and the positive examples in D_k such that $f(e_i^-) = f(e_i^+)$ for every $f \in S_i$: e_i^- and e_i^+ can not be distinguished by a feature in S_i and so S_i is not a support set (note there are two of such pairs in $exam(E_k)$). ◀

▶ **Lemma 12.** *A reduced DT* T *with features in* $\{f_1, \ldots, f_k\}$ *is a DT for* E_k *if and only if* T *is a complete DT of height* $k+1$. *In particular such DT has* $2^{k+2} - 1$ *nodes (*$2^{k+1} - 1$ *of those are inner nodes).*

Proof. In this proof we assume that a leaf is either positive or negative depending on the parity of the number of right arcs present in the unique path from the root to that leaf. We start with the forward direction: let T be a reduced DT that is not a complete DT of height $k+1$. Let P a path of T from the root to a leaf ℓ of length at most k: at most $k-1$ features appear in P and so there exists a feature $f_i \in \{f_1, \ldots, f_k\}$ that does not appear in P. Since S_i is not a support set for E, there exit a negative example e^- and a positive example e^+ that can not be distinguished by S_i, this means that $\{e^-, e^+\} \subseteq E_T(\ell)$ and so T is not a DT for E_k.

In order to prove the backward direction, we assume that T is a reduced and complete DT of height $k+1$ with features in $\{f_1, \ldots, f_k\}$. Let P be a path of T from the root to a leaf ℓ of length $k+1$. Since T is reduced, every feature of $\{f_1, \ldots, f_k\}$ appears exactly once in P. Since $\{f_1, \ldots, f_k\}$ is a support set, there is only one example e_ℓ that ends ℓ, that is $e_\ell \in E_T(\ell)$. From this proof, it follows that every reduced DT T with features in $\{f_1, \ldots, f_k\}$ for E_k has $2^{k+2} - 1$ nodes ($2^{k+1} - 1$ of those are inner nodes). ◀

Let σ be any bijection of the set $[k-2]$ and τ be any of the two bijections of the set

$\{k-1,k\}$ and (arbitrarily) define $\sigma(k-1)=\tau(k-1)$. Let us describe a DT $T_{\sigma,\tau}$ as follows. The root of r has feature f^*. The left child of r is a negative leaf and the right child v_1 has feature $f_{\sigma(1)}$. For every $i\in[k-2]$, the left child of v_i is a positive leaf and the right child v_{i+1} has feature $f_{\sigma(i+1)}$. Finally v_k and v'_k are respectively the left and right child of v_{k-1}, both having feature $f_{\tau(k)}$. The children of v_k and v'_k are leaves that are either positive or negative depending on the parity of the number of right arcs present in the unique path from the root to that leaf. In particular note that $T_{\sigma,\tau}$ has $2k+5$ nodes ($k+2$ of those are inner nodes).

▶ **Lemma 13.** *A DT T' is a DT for E_k of minimum size among those with features in $\{f_1,\ldots,f_k,f^*\}$ if and only if $T'=T_{\sigma,\tau}$, for some permutation σ and τ.*

Proof. We start by doing the forward direction: TO DO

In order to prove the backward direction, we assume that $T'=T_{\sigma,\tau}$, for some permutation σ and τ. By construction, r and its feature f^* send every negative example to its left child c_ℓ, which is a negative leaf, except for the two negative examples in D_k, that is, if $\{e_1^-,e_2^-\}=E_k^-\cap D_k$ then $E_{T'}(c_\ell)=E_k^-\setminus\{e_1^-,e_2^-\}$ and $E_{T'}(v_1)=E_k^+\cup\{e_1^-,e_2^-\}$.

Let e be an example in D_k; by construction, for every $i\in[k-2]$ if $e\in E_{T'}(v_i)$ then $e\in E_{T'}(v_{i+1})$ and by induction we obtain that $e\in E_{T'}(v_{k-1})$. Let e be an example in $\overline{D_k}$ and $j\in[k-2]$ be the minimum integer such that $f_{\sigma(j)}(e)=0$. This means that $e\notin E_{T'}(v_{j+1})$ and e is classified by the left child of the node v_j. We have just proved that $D_k=E_{T'}(v_{k-1})$ and that T' classifies $\overline{D_k}$. Now it is straightforward to show that $T'_{v_{k-1}}$ classifies D_k.

Finally we have to show that $T'=T_{\sigma,\tau}$ has minimum size. Let T'' be a reduced DT for E_k with features in $\{f_1,\ldots,f_k,f^*\}$. If T'' does not have a node with feature f^*, then by Lemma 12 $|T''|>|T_{\sigma,\tau}|$. Let v^* be a node in T'' with feature f^*. Since the set of features of a DT for E_k has to be a support set, thanks to Lemma 11 we can assume that $feat(T'')=\{f_1,\ldots,f_k,f^*\}$. Let j be the minimum level in which there is a node with feature in $\{f_{k-1},f_k\}$. First, let us assume v^* is the root of T''. If $j\geq k-1$, then it is easy to see that $T''=T_{\sigma,\tau}$ for some bijections σ and τ and so $|T''|=|T_{\sigma,\tau}|$. If $j\in\{2,\ldots,k-2\}$ TO CONTINUE

◀

4.2 For Height

For every natural number $k\geq 1$, we can define a CI E_k as follows. Let E_k be the CI with exactly 2^k examples $\{e_1,\ldots,e_{2^k}\}$ on k binary features $\{f_1,\ldots,f_k\}$: there is exactly one example for every of the 2^k feature assignments. An example $e\in exam(E_k)$ is a positive example if $|\{f\in feat(E_k)\mid f(e)=1\}|$ is even and negative otherwise.

Let IN be a subset of examples in $exam(E_k)$ that can be classified by a DT T_{IN} of height $log(k)$ and denote by OUT all the remaining examples, that is OUT$=exam(E_k)\setminus$ IN. Now we are ready to define a new feature f' follows:$f'(e)=0$ if $e\in$ IN and $f'(e)=1$ otherwise. We also introduce another new feature f'' as follows: $f''(e)=0$ if either $e\in$ IN or e is a negative example and $f''(e)=1$ otherwise.

▶ **Lemma 14.** *The features in $\{f_1,\ldots,f_k\}$ form the only minimal support set in $S=\{f_1,\ldots,f_k,f',f''\}$ for E_k.*

Proof. By Lemma 11, the set $\{f_1,\ldots,f_k\}$ is a support set. Now it is time to show that, for any $i\in[k]$, the set $S_i=\{f_1,\ldots,\overline{f_i},\ldots,f_k,f',f''\}=S\setminus\{f_i\}$ is not a support set for E_k.

◀

f_1	f_2	f_3	f'	f''		
0	0	0	0	0	+	
0	0	1	0	0	−	IN
1	0	0	0	0	−	
0	1	0	1	0	−	
0	1	1	1	1	+	
1	0	1	1	1	+	OUT
1	1	0	1	1	+	
1	1	1	1	0	−	

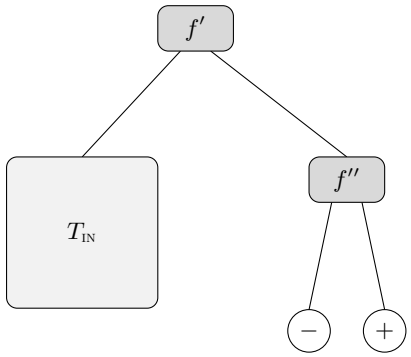

Figure 2

References

1 Christian Bessiere, Emmanuel Hebrard, and Barry O'Sullivan. Minimising decision tree size as combinatorial optimisation. *Proc. CP 2009*, 5732:173–187, 2009.

2 Endre Boros, Yves Crama, Peter L. Hammer, Toshihide Ibaraki, Alexander Kogan, and Kazuhisa Makino. Logical analysis of data: classification with justification. *Annals of Operations Research*, 188(1):33–61, 2011.

3 Endre Boros, Vladimir Gurvich, Peter L. Hammer, Toshihide Ibaraki, and Alexander Kogan. Decomposability of partially defined Boolean functions. *Discrete Applied Mathematics*, 62(1-3):51–75, 1995.

4 Endre Boros, Takashi Horiyama, Toshihide Ibaraki, Kazuhisa Makino, and Mutsunori Yagiura. Finding essential attributes from binary data. *Annals of Mathematics and Artificial Intelligence*, 39:223–257, 2003.

5 Endre Boros, Toshihide Ibaraki, and Kazuhisa Makino. Variations on extending partially defined Boolean functions with missing bits. *Information and Computation*, 180(1):53–70, 2003.

6 Yves Crama, Peter L. Hammer, and Toshihide Ibaraki. Cause-effect relationships and partially defined Boolean function. *Annals of Operations Research*, 16:299–326, 1988.

7 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

8 Adnan Darwiche and Auguste Hirth. On the reasons behind decisions. *Proc. ECAI 2020*, 325:712–720, 2020.

9 Finale Doshi-Velez and Been Kim. A roadmap for a rigorous science of interpretability. *arXiv.org*, 2017.

10 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer Verlag, 2013.

11 Bryce Goodman and Seth R. Flaxman. European union regulations on algorithmic decision-making and a "right to explanation". *AI Magazine*, 38(3):50–57, 2017.

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

13 Toshihide Ibaraki, Yves Crama, and Peter L. Hammer. *Partially defined Boolean functions (in Boolean Functions: Theory, Algorithms, and Applications)*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2011.

14 Daniel T. Larose and Chantal D. Larose. *Discovering Knowledge in Data: An Introduction to Data Mining*. John Wiley & Sons, 2nd edition, 2014.

15 Zachary C. Lipton. The mythos of model interpretability. *Communications of the ACM*, 61(10):36–43, 2018.

16 Edward J. McCluskey. *Introduction to the Theory of Switching Circuits*. Electrical and electronic engineering series. Princeton University series. McGraw-Hill, 1965.

17 Don Monroe. AI, explain yourself. *AI Communications*, 61(11):11–13, 2018.

18 Sreerama K. Murthy. Automatic construction of decision trees from data: A multi-disciplinary survey. *Data Mining and Knowledge Discovery*, 2(4):345–389, 1998.

19 Sebastian Ordyniak and Stefan Szeider. Parameterized complexity of small decision tree learning. In *Proceedings of AAAI'21, the Thirty-Fifth AAAI Conference on Artificial Intelligence*, pages 6454–6462. AAAI Press, 2021.

20 J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.