Decision Trees: More Theoretical Justification for Practical Algorithms (Extended Abstract)*

Amos Fiat and Dmitry Pechyony**

School of Computer Science, Tel-Aviv University, Tel-Aviv, Israel
{fiat,pechyony}@tau.ac.il

Abstract. We study impurity-based decision tree algorithms such as CART, C4.5, *etc.*, so as to better understand their theoretical underpinnings. We consider such algorithms on special forms of functions and distributions. We deal with the uniform distribution and functions that can be described as unate functions, linear threshold functions and read-once DNF.
For unate functions we show that that maximal purity gain and maximal influence are logically equivalent. This leads us to the exact identification of unate functions by impurity-based algorithms given sufficiently many noise-free examples. We show that for such class of functions these algorithms build minimal height decision trees. Then we show that if the unate function is a read-once DNF or a linear threshold functions then the decision tree resulting from these algorithms has the minimal number of nodes amongst all decision trees representing the function.
Based on the statistical query learning model, we introduce the noise-tolerant version of practical decision tree algorithms. We show that when the input examples have small classification noise and are uniformly distributed, then all our results for practical noise-free impurity-based algorithms also hold for their noise-tolerant version.

1 Introduction

Introduced in 1983 by Breiman *et al.* [3], decision trees are one of the few knowledge representation schemes which are easily interpreted and may be inferenced by very simple learning algorithms. The practical usage of decision trees is enormous (see [21] for a detailed survey). The most popular practical decision tree algorithms are CART ([3]), C4.5 ([22]) and their various modifications. The heart of these algorithms is the choice of splitting variables according to maximal purity gain value. To compute this value these algorithms use various impurity functions. For example, CART employs the Gini index impurity function and C4.5 uses an impurity function based on entropy. We refer to this family of algorithms as "impurity-based".

* The full version of the paper, containing all proofs, can be found online at http://www.cs.tau.ac.il/~pechyony/dt_full.ps

** Dmitry Pechyony is a full-time student and thus this paper is eligible for the "Best Student Paper" award according to conference regulations.

Despite practical success, most commonly used algorithms and systems for building decision trees lack strong theoretical basis. It would be interesting to obtain the bounds on the generalization errors and on the size of decision trees resulting from these algorithms given some predefined number of examples.

1.1 Theoretical Justification of Practical Decision Tree Building Algorithms

There have been several results justifying theoretically practical decision tree building algorithms. Kearns and Mansour showed in [16] that if the function, used for labelling nodes of tree, is a weak approximator of the target function then the impurity-based algorithms for building decision tree using Gini index, entropy or the new index are boosting algorithms. This property ensures distribution-free PAC learning and arbitrary small generalization error given sufficiently input examples. This work was recently extended by Takimoto and Maruoka [23] for functions having more than two values and by Kalai and Servedio [14] for noisy examples.

We restrict ourselves to the input of uniformly distributed examples. We provide new insight into practical impurity-based decision tree algorithms by showing that for unate boolean functions, the choice of splitting variable according to maximal exact purity gain is equivalent to the choice of variable according to the maximal influence. Then we introduce the algorithm **DTExactPG**, which is a modification of impurity-based algorithms that uses exact probabilities and purity gain rather that estimates. The main results of our work are stated by the following theorems (assuming f is unate):

Theorem 1 *The algorithm **DTExactPG** builds a decision tree representing $f(x)$ and having minimal height amongst all decision trees representing $f(x)$. If $f(x)$ is a boolean linear threshold function or a read-once DNF, then the tree built by the algorithm has minimal size amongst all decision trees representing $f(x)$.*

Theorem 2 *Let h be the minimal depth of decision tree representing $f(x)$. For any $\delta > 0$, given $O\big(2^{9h} \ln^2 \frac{1}{\delta}\big) = poly(2^h, \ln \frac{1}{\delta})$ uniformly distributed noise-free random examples of $f(x)$, with probability at least $1-\delta$, CART and C4.5 build a decision tree computing $f(x)$ exactly. The tree produced has minimal height amongst all decision trees representing $f(x)$. If $f(x)$ is read-once DNF or a boolean linear threshold function then the resulting tree has the minimal number of nodes amongst all decision trees representing $f(x)$.*

In case the input examples have classification noise with rate $\eta < \frac{1}{2}$ we introduce a noise-tolerant version of impurity-based algorithms and obtain the same result as for noise-free case:

Theorem 3 *For any $\delta > 0$, given $O\big(2^{9h} \ln^2 \frac{1}{\delta}\big) = poly(2^h, \ln \frac{1}{\delta})$ uniformly distributed random examples of $f(x)$ corrupted by classification noise with constant*

Function	Exact Influence	Exact Purity Gain	CART, C4.5, *etc.* $poly(2^h)$ uniform noise-free examples	Modification of CART, C4.5, *etc.*, $poly(2^h)$ uniform examples with small classification noise
Unate	min height	min height	min height	min height
Boolean LTF	min size	min size	min size	min size
Read-once DNF	min size	min size	min size	min size

Fig. 1. Summary of bounds on the size of decision trees, obtained in our work.

Algorithm	Model, Distribution	Running Time	Hypothesis	Bounds on the Size of DT	Function Learned
Jackson and Servedio [13]	PAC, uniform	$poly(2^h)$	Decision Tree	none	almost any DNF
Impurity-Based Algorithms (Kearns and Mansour [16])	PAC, any	$poly((\frac{1}{\epsilon})^{\frac{c}{\gamma^2}})$	Decision Tree	none	any function satisfying Weak Hypothesis Assumption
Bshouty and Burroughs [4]	PAC, any	$poly(2^n)$	Decision Tree	at most min-sized DT representing the function	any
Kushilevitz and Mansour [18], Bshouty and Feldman [5], Bshouty et al. [6]	PAC, examples from uniform random walk	$poly(2^h)$	Fourier Series	N/A	any
Impurity-Based Algorithms (our work)	PC (exact, identification), uniform	$poly(2^h)$	Decision Tree	minimal height	unate
				minimal size	read-once DNF, boolean LTF

Fig. 2. Summary of decision tree noise-free learning algorithms.

rate η, with probability at least $1-\delta$, a noise-tolerant version of impurity-based algorithms builds a decision tree representing $f(x)$. The tree produced has minimal height amongst all decision trees representing $f(x)$. If $f(x)$ is read-once DNF or a boolean linear threshold function then the resulting tree has the minimal number of nodes amongst all decision trees representing $f(x)$.

Figure 1 summarizes the bounds on the size of decision trees, obtained in our work.

1.2 Previous Work

Building minimal height and minimal number of nodes decision tree consistent with all given examples is NP-hard ([12]). The single polynomial-time deterministic approximation algorithm known today for approximating the height of decision trees is the simple greedy algorithm ([20]), achieving the factor $O(\ln(m))$ (m is the number of input examples). Combining the results of [11] and [8] it can be shown that the depth of decision tree cannot be approximated within a factor $(1-\epsilon)\ln(m)$ unless $NP \subseteq DTIME(n^{O(\log\log(n))})$. Hancock et al. showed in [10] that the problem of building minimal number of nodes decision tree cannot be approximated within a factor $2^{\log^{\delta}\mathrm{OPT}}$ for any $\delta < 1$, unless $\mathrm{NP} \subset \mathrm{RTIME}[2^{poly\log n}]$.

Blum et al. showed in [2] that decision trees cannot even be weakly learned in polynomial time from statistical queries dealing with uniformly distributed examples. Thus, there is no modification of existing decision tree learning algorithms to yield efficient polynomial-time statistical query learning algorithms for arbitrary functions. This result is an evidence for the difficulty of weak learning (and thus also PAC learning) of decision trees of arbitrary functions in the noise-free and noisy settings.

Figure 2 summarizes the best results obtained by theoretical algorithms for learning decision trees from noise-free examples. Most of them may be modified, to obtain corresponding noise-tolerant versions.

Kearns and Valiant ([17]) proved that distribution-free weak learning of read-once DNF using any representation is equivalent to several cryptographic problems widely believed to be hard. Mansour and Schain give in [19] an algorithm for proper PAC-learning of read-once DNF in polynomial time from random examples taken from any maximum entropy distribution. This algorithm may be easily modified to obtain polynomial-time probably correct learning in case the underlying function has decision tree of logarithmic depth and input examples are uniformly distributed, matching the performance of our algorithm in this case. Using both membership and equivalence queries Angluin et al. showed in [1] the polynomial-time algorithm for exact identification of read-once DNF by read-once DNF using examples taken from any distribution.

Boolean linear threshold functions are polynomially properly PAC learnable from both noise-free examples (folk result) and examples with small classification noise ([7]). In both cases the examples may be taken from any distribution.

1.3 Structure of the Paper

In Section 2 we give relevant definitions. In Section 3 we introduce a new algorithm **DTInfluence** for building decision trees using an oracle for influence and prove several properties of the resulting decision trees. In Section 4 we prove Theorem 1. In Section 5 we prove Theorem 2. In Section 6 we introduce the noise-tolerant version of impurity-based algorithms and prove Theorem 3. In Section 7 we outline directions for further research.

2 Background

In this paper we use standard definitions of PAC ([24]) and statistical query ([15]) learning models. All our results are in the PAC model with zero generalization error. We denote this model by PC (Probably Correct).

2.1 Boolean Functions

A boolean function (*concept*) is defined as $f : \{0,1\}^n \to \{0,1\}$ (for boolean formulas, *e.g.* read-once DNF) or as $f : \{-1,1\}^n \to \{0,1\}$ (for arithmetic formulas, *e.g.* boolean linear threshold functions). Let x_i be the i-th variable or *attribute*. Let $x = (x_1, \ldots, x_n)$, and $f(x)$ be the *target* or *classification*. The vector $(x_1, x_2, \ldots, x_n, f(x))$, is called an *example*. Let $f_{x_i=a}(x)$, $a \in \{0,1\}$ be the function $f(x)$ restricted to $x_i = a$. We refer to the assignment $x_i = a$ as a *restriction*. Given the set of restrictions $R = \{x_{i_1} = a_1, \ldots, x_{i_k} = a_k\}$, the restricted function $f_R(x)$ is defined similarly. $x_i \in R$ iff there exists a restriction $x_i = a \in R$, where a is any value.

A *literal* $\tilde{x}_i$ is a boolean variable x_i itself or its negation $\bar{x}_i$. A *term* is a conjunction of literals and a *DNF (Disjunctive Normal Form) formula* is a disjunction of terms. Let $|F|$ be the number of terms in the DNF formula F and $|t_i|$ be the number of literals in the term t_i. Essentially F is a set of terms $F = \{t_1, \ldots, t_{|F|}\}$ and t_i is a set of literals, $t_i = \{\tilde{x}_{i_1}, \ldots, \tilde{x}_{i_{|t_i|}}\}$. The term t_i is *satisfied* iff $\tilde{x}_{i_1} = \ldots = \tilde{x}_{i_{|t_i|}} = 1$.

If for all $1 \le i \le n$, $f(x)$ is monotone w.r.t. x_i or $\overline{x}_i$ then $f(x)$ is a *unate function*. A DNF is *read-once* if each variable appears at most once. Given a weight vector $\overrightarrow{a} = (a_1, \ldots, a_n)$, such that for all $1 \le i \le n$, $a_i \in \Re$, and a threshold $t \in \Re$, the boolean linear threshold function (LTF) $f_{a,t}$ is $f_{a,t}(x) = \sum_{i=1}^{n} a_i x_i > t$.

Let e_i be the vector of n components, containing 1 in the i-th component and 0 in all other components. The *influence* of x_i on $f(x)$ under distribution $\mathcal{D}$ is $I_f(i) = \Pr_{x\sim\mathcal{D}}[f(x) \neq f(x \oplus e_i)]$. We use the notion of *influence oracle* as an auxiliary tool. The influence oracle runs in time $O(1)$ and returns the exact value of $I_f(i)$ for any f and i.

2.2 Decision Trees

In our work we restrict ourselves to binary univariate decision trees for boolean functions. So the definitions given below are adjusted to this model and are not generic. A *decision tree* T is a rooted DAG consisting of *nodes* and *leaves*. Each node in T, except the root, has in-degree 1 and out-degree 2. The in-degree of the root is 0. Each leaf has in-degree 1 and out-degree 0. The edges of T and leaves are labelled with 1 and 0.

The classification of an input to f is done by traversing the tree from the root to some leaf. Every node s of T contains some test $x_i = 1?$ and the variable x_i is called a *splitting variable*. The left (right) son of s is also called the 0-son (1-son) and is referred to as s_0 (s_1). Let $c(l)$ be the label of the leaf l. Upon arriving to the node s, we pass the input x to the $(x_i = 1?)$-son of s. The classification given to the input x by the T is denoted by $c_T(x)$. The path from the root to the node s corresponds to the set of restrictions of values of variables leading to s. Similarly, the node s corresponds to the restricted function $f_R(x)$. In the sequel we use the identifier s of the node and its corresponding restricted function interchangeably.

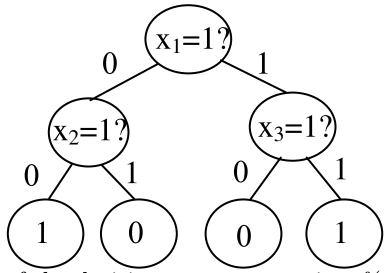

Fig. 3. Example of the decision tree representing $f(x) = x_1x_3 \vee \overline{x_1x_2}$

DTApproxPG$(s,\ X,\ R,\ \phi)$

if all examples arriving to s have the same classification **then**
    Set s as a leaf with that value.
else
    Choose $x_i = \arg\max_{x_i \in X}\{\widehat{PG}(f_R, x_i, \phi)\}$ to be a splitting variable.
    Run **DTApproxPG**$(s_1,\ X - \{x_i\},\ R \cup \{x_i = 1\},\ \phi)$.
    Run **DTApproxPG**$(s_0,\ X - \{x_i\},\ R \cup \{x_i = 0\},\ \phi)$.
end if

Fig. 4. DTApproxPG algorithm - generic structure of all impurity-based algorithms.

The *height* of T, $h(T)$, is the maximal length of path from the root to any node. The *size* of T, $|T|$, is the number of nodes in T. A decision tree T *represents* $f(x)$ iff $f(x) = c_T(x)$ for all x. An example of a decision tree, representing the function $f(x) = x_1x_3 \vee \overline{x_1x_2}$, is shown in Fig. 3.

The function $\phi(x) : [0,1] \to \Re$ is an *impurity function* if it is concave, $\phi(x) = \phi(1-x)$ for any $x \in [0,1]$ and $\phi(0) = \phi(1) = 0$. Examples of impurity functions are the Gini index $\phi(x) = 4x(1-x)$ ([3]), the entropy function $\phi(x) = -x\log x - (1-x)\log(1-x)$ ([22]) and the new index $\phi(x) = 2\sqrt{x(1-x)}$ ([16]). Let $s_a(i)$, $a \in \{0,1\}$, denote the a-son of s that would be created if x_i is placed at s as a splitting variable. For each node s let $\Pr[s_a(i)]$, $a \in \{0,1\}$, denote the probability that random example from the uniform distribution arrives at $s_a(i)$ given that it has already arrived at s. Let $p(s)$ be the probability that a positive example arrives to the node s. The *impurity sum* (IS) of x_i at s using impurity function $\phi(x)$ is $\mathrm{IS}(s, x_i, \phi) = \Pr[s_0(i)]\phi(p(s_0(i))) + \Pr[s_1(i)]\phi(p(s_1(i)))$. The *purity gain* (PG) of x_i at s is: $\mathrm{PG}(s, x_i, \phi) = \phi(p(s)) - \mathrm{IS}(s, x_i, \phi)$. The estimated values of all these quantities are $\widehat{PG}$, $\widehat{IS}$, *etc.*

Figure 4 gives the structure of all impurity-based algorithms. The algorithm takes four parameters: s, identifying current tree's node, X, standing for the set of attributes available for testing, R, which is a set of function's restrictions leading to s and ϕ, identifying the impurity function. Initially s is set to the root node, X contains all attribute variables and R is an empty set.

Since the value of $\phi(p(s))$ is attribute independent, the choice of maximal $\widehat{\mathrm{PG}}(s, x_i, \phi)$ is equivalent to the choice of minimal $\widehat{\mathrm{IS}}(s, x_i, \phi)$. For uniformly distributed examples $\Pr[s_0(i)] = \Pr[s_1(i)] = 0.5$. Thus if impurity sum is computed exactly, then $\phi(p(s_0(i)))$ and $\phi(p(s_1(i)))$ have equal weight. We define the *balanced impurity sum* of x_i at s as $\mathrm{BIS}(s, x_i, \phi) = \phi(p(s_0(i))) + \phi(p(s_1(i)))$.

```
DTInfluence(s, X, R)
if ∀x_i ∈ X, I_{f_R}(i) = 0 then
    Set classification of s as a classification of any example arriving to it.
else
    Choose x_i = arg max_{x_i∈X}{I_{f_R}(i)} to be a splitting variable.
    Run DTInfluence(s_1, X − {x_i}, R ∪ {x_i = 1}).
    Run DTInfluence(s_0, X = {x_i}, R ∪ {x_i = 0}).
end if
```

Fig. 5. DTInfluence algorithm.

3 Building Decision Trees Using an Influence Oracle

In this section we introduce a new algorithm, **DTInfluence** (see Fig. 5), for building decision trees using an influence oracle. This algorithm greedily chooses the splitting variable with maximal influence. Clearly, the resulting tree consisting of only relevant variables. The algorithm takes three parameters, s, X and R, having the same meaning and initial values as in the algorithm **DTApproxPG**.

Lemma 1 *Let $f(x)$ be any boolean function. Then the decision tree T built by the algorithm **DTInfluence** represents $f(x)$ and has no node such that all examples arriving to it have the same classification.*

Proof See online full version ([9]). □

Lemma 2 *If $f(x)$ is a unate function with n relevant variables then any decision tree representing $f(x)$ and consisting only of relevant variables has height n.*

Proof See online full version ([9]). □

Corollary 3 *If $f(x)$ is a unate function then the algorithm **DTInfluence** produces a minimal height decision tree representing $f(x)$.*

Proof Follows directly from Lemma 2. □

3.1 Read-Once DNF

Let $f(x)$ be a boolean function which can be represented by a read-once DNF F. In this section we prove the following lemma:

Lemma 4 *For any $f(x)$ which can be represented by a read-once DNF, the decision tree built by the algorithm **DTInfluence** has the minimal number of nodes amongst all decision trees representing $f(x)$.*

The proof of the Lemma 4 consists of two parts. In the first part of the proof we introduce another algorithm, called **DTMinTerm** (see Figure 6). Then we prove Lemma 4 for the algorithm **DTMinTerm**. In the second part of the proof we show that the trees built by **DTMinTerm** and **DTInfluence** are the same.

DTMinTerm(s, F)

if $\exists\ t_i \in F$ such that $t_i = \emptyset$ **then**
  Set s as a positive leaf.
else
  if $F = \emptyset$ **then**
    Set s as a negative leaf.
  else
    Let $t_{min} = \arg\min_{t_i \in F}\{|t_i|\}$. $t_{min} = \{\tilde{x}_{m_1}, \tilde{x}_{m_2}, \ldots, \tilde{x}_{m_{|t_{min}|}}\}$.
    Choose any $\tilde{x}_{m_i} \in t_{min}$. Let $t^{'}_{min} = t_{min} \backslash \{\tilde{x}_{m_i}\}$.
    if $\tilde{x}_{m_i} = x_{m_i}$ **then**
      Run **DTMinTerm**$(s_1, F\backslash\{t_{min}\} \cup \{t^{'}_{min}\})$, **DTMinTerm**$(s_0, F\backslash\{t_{min}\})$.
    else
      Run **DTMinTerm**$(s_0, F\backslash\{t_{min}\} \cup \{t^{'}_{min}\})$, **DTMinTerm**$(s_1, F\backslash\{t_{min}\})$.
    end if
  end if
end if

Fig. 6. DTMinTerm algorithm.

Assume we are given read-once DNF formula F. We change the algorithm **DTInfluence** so that the splitting rule is to choose any variable x_i in the smallest term $t_j \in F$. The algorithm stops when the restricted function becomes constant (true or false). The new algorithm, denoted by **DTMinTerm**, is shown in Figure 6. The initial value of the first parameter of the algorithm is the same as in **DTInfluence**, and the second parameter is initially set to function's DNF formula F. The following three lemmata are proved in the online full version ([9]).

Lemma 5 *Given the read-once DNF formula F representing the function $f(x)$, the decision tree T, built by the algorithm **DTMinTerm** represents $f(x)$ and has the minimal number of nodes among all decision trees representing $f(x)$.*

Lemma 6 *Let $x_l \in t_i$ and $x_m \in t_j$. If $|t_i| > |t_j|$ then $I_f(l) < I_f(m)$ and if $|t_i| = |t_j|$ then $I_f(l) = I_f(m)$.*

Lemma 7 *Let $X' = \{x_{i_1}, \ldots, x_{i_k}\}$ be a set of variables presenting in the terms of minimal length of some read-once DNF F. For all $x \in X'$ there exists a minimal sized decision tree for $f(x)$ with splitting variable x in the root.*

Proof (Lemma 4): It follows from Lemmata 6 and 7 that the trees produced by the algorithms **DTMinTerm** and **DTInfluence** have the same size. Combining this result with the results of Lemmata 1 and 5, the current lemma follows. $\Box$

3.2 Boolean Linear Threshold Functions

In this section we prove the following lemma:

DTCoeff$(s,\, X,\, t_s)$

if $\sum_{x_i \in X} |a_i| \leq t_s$ or $-\sum_{x_i \in X} |a_i| > t_s$ **then**
  The function is constant. s is a leaf.
else
  Choose a variable x_i from X, having the largest $|a_i|$.
  Run **DTCoeff**$(s_1,\, X - \{x_i\},\, t_s - a_i)$ and **DTCoeff**$(s_0,\, X - \{x_i\},\, t_s + a_i)$.
end if

Fig. 7. DTCoeff algorithm.

x_i	x_j	other variables	function value
v	v'	$w_1, w_2, w_3, \ldots, w_{n-2}$	t_1
$-v$	v'	$w_1, w_2, w_3, \ldots, w_{n-2}$	t_2
v	$-v'$	$w_1, w_2, w_3, \ldots, w_{n-2}$	t_3
$-v$	$-v'$	$w_1, w_2, w_3, \ldots, w_{n-2}$	t_4

Fig. 8. Structure of the truth table G_w from $G(i,j)$.

Lemma 8 *For any linear threshold function $f_{a,t}(x)$, the decision tree built by the algorithm **DTInfluence** has the minimal number of nodes among all decision trees representing $f_{a,t}(x)$.*

The proof of the Lemma 8 consists of two parts. In the first part of the proof we introduce another algorithm, called **DTCoeff** (see Fig. 7). Then we prove the Lemma 8 for the algorithm **DTCoeff**. In the second part of the proof we show that the trees built by **DTCoeff** and **DTInfluence** have the same size.

The difference between **DTCoeff** and **DTinfuence** is in the choice of splitting variable. **DTCoeff** chooses the variable with the largest $|a_i|$, and stops when the restricted function becomes constant (true or false). The meaning and initial values of the first two parameters of the algorithm are the same as in **DTInfluence**, and the third parameter is initially set to the function's threshold t.

Lemma 9 *Given the coefficient vector a, the decision tree T built by the algorithm **DTCoeff** represents $f_{a,t}(x)$ and has minimal number of nodes among all decision trees representing $f_{a,t}(x)$.*

Proof Appears in the online full version ([9]) $\square$

We now prove a sequence of lemmata connecting the influence and the coefficients of variables in the threshold formula. Let x_i and x_j be two different variables in $f(x)$. For each of the 2^{n-2} possible assignments to the remaining variables we get a 4 row truth table for different values of x_i and x_j. Let $G(i,j)$ be the multi set of 2^{n-2} truth tables, indexed by the assignment to the other variables. *I.e.*, G_w is the truth table where the other variables are assigned values $w = w_1, w_2, \ldots, w_{n-2}$. The structure of a single truth table is shown in Fig. 8. In this figure, and generally from now on, v and v' are constants in $\{-1, 1\}$. Observe that $I_f(i)$ is proportional to the sum over the 2^{n-2} G_w's in $G(i,j)$ of the number of times $t_1 \neq t_2$ plus the number of times $t_3 \neq t_4$. Similarly, $I_f(j)$ is proportional to the sum over the 2^{n-2} G_w's in $G(i,j)$ of the number of times $t_1 \neq t_3$ plus the number of times $t_2 \neq t_4$. We use these observations in the proof of the following lemma (see online full version [9]):

Lemma 10 *If $I_f(i) > I_f(j)$ then $|a_i| > |a_j|$.*

Note that if $I_f(i) = I_f(j)$ then there may be any relation between $|a_i|$ and $|a_j|$. The next lemma shows that choosing the variables with the same influence in any order does not increase the size of the resulting decision tree. For any node s, let X_s be the set of all variables in X which are untested on the path from the root to s. Let $\hat{X}(s) = \{x_1, \ldots x_k\}$ be the variables having the same non-zero influence, which in turn is the largest influence among the influences of variables in X_s.

Lemma 11 *Let T_i (T_j) be the smallest decision tree one may get when choosing any $x_i \in \hat{X}_s$ ($x_j \in \hat{X}_s$) at s. Let $|T_{opt}|$ be the size of the smallest tree rooted at s. Then $|T_i| = |T_j| = |T_{opt}|$.*

Proof The proof in by induction on k. For $k = 1$ the lemma trivially holds. Assume the lemma holds for all $\ell < k$. Next we prove the lemma for k. Consider two attributes x_i and x_j from $\hat{X}(s)$ and possible values of targets in any truth table $G_w \in G(i,j)$. Since the underlying function is a boolean linear threshold and $I_f(i) = I_f(j)$, targets may have 4 forms:

- Type A. All rows in G_w have target value 0.
- Type B. All rows in G_w have target value 1.
- Type C. Target value f in G_w is defined as $f = (a_i x_i > 0 \text{ and } a_j x_j > 0)$.
- Type D. Target value f in G_w is defined as $f = (a_i x_i > 0 \text{ or } a_j x_j > 0)$.

Consider the smallest tree T testing x_i at s. There are 3 cases to be considered:

1. Both sons of x_i are leaves. Since $I_f(i) > 0$ and $I_f(j) > 0$ there is at least one $G_w \in G(i,j)$ having a target of type C or D. Thus no neither x_i nor x_j cannot determine the function and this case is impossible.
2. Both sons of x_i are non-leaves. By the inductive hypothesis there exist right and left smallest subtrees of x_i, each one rooted with x_j. Then x_i and x_j may be interchanged to produce an equivalent decision tree T' testing x_j at s and having the same size.
3. Exactly one of the sons of x_i is a leaf.

Let us consider the third case. By the inductive hypothesis the non-leaf son of s tests x_j. It is not hard to see (see online full version [9]) that in this case $G(i,j)$ contains either truth tables with targets of type A and C or truth tables with targets of type B and D (otherwise both sons of x_i are non-leaves). In both these cases some value of x_j determines the value of the function. Therefore if we place the test $x_j = 1$? at s, then exactly one of its sons is a leaf. Thus it can be easily verified that testing x_j and then x_i or testing x_i and then x_j results in a tree of the same size (see [9]). $\Box$

DTExactPG(s, X, R, ϕ)

if all examples arriving at s have the same classification **then**
  Set s as a leaf with that value.
else
  Choose $x_i = \arg\max_{x_i \in X}\{PG(f_R, x_i, \phi)\}$ to be a splitting variable.
  Run **DTExactPG**($s_1, X - \{x_i\}, R \cup \{x_i = 1\}, \phi$).
  Run **DTExactPG**($s_0, X - \{x_i\}, R \cup \{x_i = 0\}, \phi$).
end if

Fig. 9. **DTExactPG** algorithm.

Proof(Lemma 8) Combining Lemmata 9, 10 and 11 we obtain that there exists a smallest decision tree having the same splitting rule as that of **DTInfluence**. Combining this result with Lemma 1 concludes the proof. □

4 Optimality of Exact Purity Gain

In this section we introduce a new algorithm for building decision tree, **DTExactPG**, (see Fig. 9) using exact values of purity gain. The proofs presented in this section are independent of the specific form of impurity function and thus are valid for all impurity functions satisfying the conditions defined in section 2.2.

The next lemma follows directly from the definition of the algorithm:

Lemma 12 *Let $f(x)$ be any boolean function. Then the decision tree T built by the algorithm **DTExactPG** represents $f(x)$ and there exists no inner node such that all inputs arriving at it have the same classification.*

Lemma 13 *For any boolean function $f(x)$, uniformly distributed x, and any node s, $p(s_0(i))$ and $p(s_1(i))$ are symmetric relative to $p(s)$: $|p(s_1(i)) - p(s)| = |p(s_0(i)) - p(s)|$ and $p(s_1(i)) \neq p(s_0(i))$*

Proof Appears in the full version of the paper ([9]).□

Lemma 14 *For any unate boolean function $f(x)$, uniformly distributed input x, and any impurity function ϕ, $I_f(i) > I_f(j) \leftrightarrow PG(f, x_i, \phi) > PG(f, x_j, \phi)$.*

Proof Since x is distributed uniformly, it is sufficient to prove $I_f(i) > I_f(j) \leftrightarrow BIS(f, x_i, \phi) < BIS(f, x_j, \phi)$. Let d_i be number of pairs of examples differing only in x_i and having different target value. Since all examples have equal probability $I_f(i) = \frac{d_i}{2^{n-1}}$.

Consider a split of node s according to x_i. All positive examples arriving at s may be divided into two categories:

1. Flipping the value of i-th attribute does not change the target value of example. Then the first half of such positive examples passes to s_1 and the second half passes to s_0. Consequently such positive examples contribute equally to the probabilities of positive examples in s_1 and s_0.

2. Flipping the value of i-th attribute changes the target value of example. Consider such pair of positive and negative examples, differing only in x_i. Since $f(x)$ is unate, either all positive example in such pairs have $x_i = 1$ and all negative examples in such pairs have $x_i = 0$, or all positive example in such pairs have $x_i = 0$ and all negative examples in such pairs have $x_i = 1$. Consequently either all such positive examples pass to s_1 or all such positive examples pass to s_0. Thus such examples increase the probability of positive examples in one of the nodes $\{s_1, s_0\}$ and decrease the probability of positive examples in the other.

Observe that the number of positive examples in the second category is essentially d_i. Thus $I_f(i) > I_f(j) \leftrightarrow \max\{p(s_1(i)), p(s_0(i))\} > \max\{p(s_1(j)), p(s_0(j))\}$. By Lemma 13, for all i, $p(s_1(i))$ and $p(s_0(i))$ are symmetric relative to $p(s)$. Therefore, if $\max\{p(s_1(i)), p(s_0(i))\} > \max\{p(s_1(j)), p(s_0(j))\}$ then the probabilities of x_i are more distant from $p(s)$ than those of x_j. Consequently, due to concavity of impurity function, $BIS(f, x_j, \phi) > BIS(f, x_i, \phi)$. $\square$

Proof Sketch (of Theorem 1) The first part of the theorem follows from Lemmata 14, 12 and 2. The second part of the theorem follows from Lemmata 14, 6, 7, 11, 4, 8, 3 and 12. See online full version [9] for a complete proof. $\square$

5 Optimality of Approximate Purity Gain

The purity gain computed by practical algorithms is not exact. However under some conditions approximate purity gain suffices. The proof of this result is based on the following lemma (proved in the online full version [9]):

Lemma 15 *Let $f(x)$ be a boolean function which can be represented by decision tree of depth h and x is distributed uniformly then $Pr(f(x) = 1) = \frac{r}{2^h}$, $r \in \mathbb{Z}$, $0 \leq r \leq 2^h$.*

Proof Sketch (Theorem 2) From Lemma 15 and Theorem 1, to obtain the equivalence of exact and approximate purity gains we need to compute all probabilities within accuracy at least $\frac{1}{2 \cdot 2^h}$ (h is the minimal height of decision tree representing the function). We show that accuracy $poly(\frac{1}{2^h})$ suffices for the equivalence. See the online full version ([9]) for complete proof. $\square$

6 Noise-Tolerant Probably Correct Learning

In this section we assume that each input example is misclassified with probability (noise rate) $\eta < 0.5$. We introduce a reformulation of the practical impurity-based algorithms in terms of statistical queries. Since our noise-free algorithms learn probably correctly, we would like to obtain the same results of probable correctness with noisy examples. Our definition of PC learning with noise is that

DTStatQuery$(s,\, X,\, R,\, \phi,\, h)$

if $\widehat{\Pr}[f_R = 1](\frac{1}{2 \cdot 2^h}) > 1 - \frac{1}{2 \cdot 2^h}$ **then**
  Set s as a positive leaf.
else
  if $\widehat{\Pr}[f_R = 1](\frac{1}{2 \cdot 2^h}) < \frac{1}{2 \cdot 2^h}$ **then**
    Set s as a negative leaf.
  else
    Choose $x_i = \arg\max_{x_i \in X} \widehat{PG}(f_R, x_i, \phi, \frac{1}{2^{4h}})$ to be a splitting variable.
    Run **DTStatQuery**$(s_1,\, X - \{x_i\},\ R \cup \{x_i = 1\},\, \phi,\, h)$.
    Run **DTStatQuery**$(s_0,\, X - \{x_i\},\ R \cup \{x_i = 0\},\ \phi,\, h)$.
  end if
end if

Fig. 10. DTStatQuery algorithm.

the examples are noisy yet, nonetheless, we insist upon zero generalization error. Previous learning algorithms with noise (*e.g.* [15]) allow a non-zero generalization error.

Let $\widehat{\Pr}[f_R = 1](\alpha)$ be the estimate of $\Pr[f_R = 1]$ within accuracy α. The algorithm **DTStatQuery**, which is a reformulation of **DTApproxPG** in terms of statistical queries, is shown at Figure 10.

Lemma 16 *Let $f(x)$ be unate boolean function. Then, for any impurity function, **DTStatQuery** builds a minimal height decision tree representing $f(x)$. If $f(x)$ is read-once DNF or a boolean linear threshold function then the resulting tree has also minimal number of nodes amongst all decision trees representing $f(x)$.*

Proof Follows from Lemma 15 and Theorem 2. See full version of the paper ([9]) for a complete proof. □

Kearns shows in [15] how to simulate statistical queries from examples corrupted by small classification noise. This simulation involves the estimation of η. [15] shows that if statistical queries need to be computed within accuracy ϵ then η should be estimated within accuracy $\Delta/2 = \Omega(\epsilon)$. Such an estimation may be obtained by taking $\lceil \frac{1}{2\Delta} \rceil$ estimations of η of the form $i\Delta$. Running the learning algorithm using each time different estimation we obtain $\lceil \frac{1}{2\Delta} \rceil$ hypotheses $h_1, \ldots, h_{\lceil \frac{1}{2\Delta} \rceil}$. By the definition of Δ, amongst these hypotheses there exists at least one hypothesis h_j having the same generalization error as the statistical query algorithm. Then [15] describes a procedure how to recognize the hypothesis having generalization error of at most ϵ. The naïve approach to recognize the minimal sized decision tree having zero generalization error amongst $h_1, \ldots, h_{\lceil \frac{1}{2\Delta} \rceil}$ is to apply the procedure of [15] with $\epsilon = \frac{1}{2 \cdot 2^n}$. However in this case this procedure requires about 2^n noisy examples. Next we show how to recognize minimal size decision tree with zero generalization error using only $poly(2^h)$ uniformly distributed noisy examples.

Let $\gamma_i = \Pr_{EX^{\eta}(\mathcal{U})}[h_i(x) \neq f(x)]$ be the generalization error of h_i over the space of noisy examples. Clearly, $\gamma_i \geq \eta$ for all i, and $\gamma_j = \eta$. Moreover among

$\lceil \frac{1}{2\varDelta} \rceil$ estimations $\hat{\eta}_i = i\varDelta$ of η ($i = 0, \ldots, \lceil \frac{1}{2\varDelta} \rceil - 1$) there exists $i = j$ such that $|\eta - j\varDelta| \leq \varDelta/2$. Therefore our current goal is to find such j.

Let $\hat{\gamma}_i$ be the estimation of γ within accuracy $\varDelta/4$. Then $|\gamma_j - j\varDelta| < \frac{3\varDelta}{4}$. Let $H = \{i \mid |\hat{\gamma}_i - i\varDelta| < \frac{3\varDelta}{4}\}$. Clearly $j \in H$. Therefore if $|H| = 1$ then H contains only j. Consider the case of $|H| > 1$. Since for all i $\gamma_i \geq \eta$, if $i \in H$ then $i \geq j - 1$. Therefore one of the two minimal values in H is j. Let i_1 and i_2 be two minimal values in H. If h_{i_1} and h_{i_2} are the same tree then clearly they are the one with smallest size representing the function. If $|i_1 - i_2| > 1$ then, using the argument $i \in H \rightarrow i \geq j - 1$, we get that $j = \min\{i_1, i_2\}$. If $|i_1 - i_2| = 1$ and $|\hat{\gamma}_{i_1} - \hat{\gamma}_{i_2}| \geq \frac{\varDelta}{2}$, then, since the accuracy of $\hat{\gamma}$ is $\varDelta/4$, $j = \min\{i_1, i_2\}$. The final subcase to be considered is $|\hat{\gamma}_{i_1} - \hat{\gamma}_{i_2}| < \frac{\varDelta}{2}$ and $|i_1 - i_2| = 1$. In this case $\hat{\eta} = (\hat{\gamma}_{i_1} + \hat{\gamma}_{i_2})/2$ estimates the true value of η within accuracy $\varDelta/2$. Thus running the learning algorithm with the value $\hat{\eta}$ for noise rate produces the same tree as the one produced by statistical query algorithm.

It can be shown (see [9]) that to recognize hypothesis with zero generalization error all estimations should done within accuracy $poly(\frac{1}{2^h})$. Thus its sample complexity is the same as in **DTApproxPG**. Consequently, Theorem 3 follows.

7 Future Research

Immediate directions for further research include: analysis of the case with small (less that $poly(2^h)$) number of examples, extensions to other distributions, to other classes of boolean functions, to continuous and general discrete attributes and to multivariate decision trees. It would be interesting to find classes of functions for which **DTInfluence** algorithm approximates the size of decision tree within some small factor. Moreover we would like to compare our noise-tolerant version of impurity-based algorithms vs. pruning methods. Finally since influence and impurity gain are logically equivalent, it would be interesting to use the notion of purity gain in the field of analysis of boolean functions.

Acknowledgements

We thank Yishay Mansour for his great help with all aspects of this paper. We also thank Adam Smith who greatly simplified and generalized an earlier version of Theorem 1.

References

1. D. Angluin, L. Hellerstein and M. Karpinski. Learning Read-Once Formulas with Queries. *Journal of the ACM*, 40(1):185-210, 1993.
2. A. Blum, M. Furst, J. Jackson, M. Kearns, Y. Mansour and S. Rudich. Weakly Learning DNF and Characterizing Statistical Query Learning Using Fourier Analysis. In *Proceedings of the 26th Annual ACM Symposium on the Theory of Computing*, pages 253-262, 1994.

3. L. Breiman, J.H. Friedman, R.A. Olshen and C.J. Stone. *Classification and Regression Trees*. Wadsworth International Group, 1984.
4. N.H. Bshouty and L. Burroughs. On the Proper Learning of Axis-Parallel Concepts. *Journal of Machine Learning Research*, 4:157-176, 2003.
5. N.H. Bshouty and V. Feldman. On Using Extended Statistical Queries to Avoid Membership Queries. *Journal of Machine Learning Research*, 2:359-395, 2002.
6. N.H. Bshouty, E. Mossel, R. O'Donnel and R.A. Servedio. Learning DNF from Random Walks. In *Proceedings of the 44th Annual Symposium on Foundations of Computer Science*, 2003.
7. E. Cohen. Learning Noisy Perceptron by a Perceptron in Polynomial Time. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, pages 514-523, 1997.
8. U. Feige. A Threshold of $\ln n$ for Approximating Set Cover. *Journal of the ACM* 45(4):634-652, 1998.
9. A. Fiat and D. Pechyony. Decision Trees: More Theoretical Justification for Practical Algorithms. Available at http://www.cs.tau.ac.il/~pechyony/dt_full.ps
10. T. Hancock, T. Jiang, M. Li and J. Tromp. Lower bounds on Learning Decision Trees and Lists. *Information and Computation*, 126(2):114-122, 1996.
11. D. Haussler. Quantifying Inductive Bias: AI Learning Algorithms and Valiant's Learning Framework. *Artificial Intelligence*, 36(2): 177-221, 1988.
12. L. Hyafil and R.L. Rivest. Constructing Optimal Binary Decision Trees is NP-Complete. *Information Processing Letters*, 5:15-17, 1976.
13. J. Jackson, R.A. Servedio. Learning Random Log-Depth Decision Trees under the Uniform Distribution. In *Proceedings of the 16th Annual Conference on Computational Learning Theory*, pages 610-624, 2003.
14. A. Kalai and R.A. Servedio. Boosting in the Presence of Noise. In *Proceedings of the 35th Annual Symposium on the Theory of Computing*, pages 195-205, 2003.
15. M.J. Kearns. Efficient Noise-Tolerant Learning from Statistical Queries. *Journal of the ACM*, 45(6):983-1006, 1998.
16. M.J. Kearns and Y. Mansour. On the Boosting Ability of Top-Down Decision Tree Learning Algorithms. *Journal of Computer and Systems Sciences*, 58(1):109-128, 1999.
17. M.J. Kearns, L.G. Valiant. Cryptographic Limitations on Learning Boolean Formulae and Finite Automata. *Journal of the ACM*, 41(1):67-95, 1994.
18. E. Kushilevitz and Y. Mansour. Learning Decision Trees using the Fourier Spectrum. *SIAM Journal on Computing*, 22(6):1331-1348, 1993.
19. Y. Mansour and M. Schain. Learning with Maximum-Entropy Distributions. *Machine Learning*, 45(2):123-145, 2001.
20. M. Moshkov. Approximate Algorithm for Minimization of Decision Tree Depth. In *Proceedings of 9th International Conference on Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing*, pages 611-614, 2003.
21. S.K. Murthy. Automatic Construction of Decision Trees from Data: A Multi-Disciplinary Survey. *Data Mining and Knowledge Discovery*, 2(4): 345-389, 1998.
22. J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.
23. E. Takimoto and A. Maruoka. Top-Down Decision Tree Learning as Information Based Boosting. *Theoretical Computer Science*, 292:447-464, 2003.
24. L.G. Valiant. A Theory of the Learnable. *Communications of the ACM*, 27(11):1134-1142, 1984.