JMLR: Workshop and Conference Proceedings 1:1–8, 2010 Learnaut 2018

Toward Learning Boolean Functions from Positive Examples

Nicolas Basset BASSTNI@UNIV-GRENOBLE-ALPES.FR
Oded Maler ODED.MALER@UNIV-GRENOBLE-ALPES.FR
Irini-Eleftheria Mens IRINI-ELEFTHERIA.MENS@UNIV-GRENOBLE-ALPES.FR
VERIMAG, CNRS and University of Grenoble-Alpes (UGA), Saint Martin d'Hères, France

Editor: Editor's name

Abstract

In this article we investigate learning of boolean functions with two special focuses: (i) we consider that a sample with only positive examples is given to the learner (and hence no membership queries are allowed); and (ii) the function we consider are true on a sparse set, namely whose cardinality is far smaller than 2^n where n is the number of variables of the problem instance. After motivating this problem, we provide a conceptual solution of how much to generalize beyond the sample and where. We introduce an empirical greedy algorithm to address the problem under study and illustrate it on an example.

Keywords: Learning from positive examples, sparse Boolean functions, DNF, false positives, false negatives

1. Introduction and Motivation

Our initial motivation comes from applying data mining techniques to systems that generates dynamic/temporal behaviors (sequences, signals, traces, time series). These behaviors can be measurements taken from a real system or traces produced by a simulation model of a complex system such as a car engine Jin et al. (2013). In both cases, the system is typically non-deterministic, generating many different behaviors. This can be due to internal non-determinism, external inputs, partial observability, etc. We collect a sample $S = \{w^1, \ldots, w^m\}$ of observed behaviors and want to come up with an approximate characterization of the possible behaviors of the system. Ultimately, we would like to treat dense-time real-valued signals and produces a formula in STL (Signal Temporal Logic, Maler and Nickovic (2004); Maler et al. (2008); Bartocci et al.) which is satisfied by all the elements of S, see Asarin et al. (2011) for a preliminary parametric identification procedure.

We focus in this paper on the fact that we are dealing with a system which is a *generator* of instances of a set, not an acceptor/classifier. We can run simulation runs and obtain positive examples of possible system behaviors, but there is typically no reasonable way, especially when the system is very complex or a black-box, to obtain examples of non-behaviors. Thus our problem falls into the category of learning from *positive examples* where we have a domain D, a characteristic function $f : D \to \mathbb{B} = \{0, 1\}$ of a subset of D and a sample consisting of elements of this subset. We want to come up with a function h, which has some good relationships with f (and with S), expressed typically in terms of misclassification error.

Before we proceed, let us mention two assumptions that we make about the f and S that we believe to hold in many cases independently of the domain. The first is that f is very *sparse*, satisfying $|f| \ll |D|$ where $|.|$ denotes cardinality and f is notationally abused also for the set

© 2010 N. Basset, O. Maler & I.-E. Mens.

$\{x : f(x) = 1\}$. We believe that most interesting concepts are sparse: the set of bitmaps that can be identified as an animal or a letter in the alphabet is a small fraction of the set of bitmaps. Likewise, in the dynamic cyber-physical domain, the set of all behaviors produced by a simulator is a tiny fraction of the set of signals/time series over the corresponding data types. Even the set of possible sequences of sampled values of a thermometer is small relative to all sequences over the temperature domain. In the Boolean case this will exclude for example some classes of functions like k-DNF which are easy to learn. The other assumption is that the sample is small relative to f, $|S| \ll |f|$, a reasonable assumption in most, if not all, learning and statistical inference situations.

It is usual in the well studied PAC learning framework Valiant (1984) and in alternative framework (e.g. allowing membership queries Jackson (1997)) to measure error classification wrt uniform probability on boolean vectors on n variables (see also Bshouty et al. (2005) and reference therein). The error measured is the cardinality of misclassified elements divided by 2^n which hence makes poor sense for sparse functions. Instead, we measure two kinds of errors: (i) false positives (elements that satisfy the output hypothesis function h but not the target function f); and (ii) false negatives (elements that satisfy f but not h). We then compute the proportion of false negative wrt $|f|$ and of false positive wrt $|h|$. These two error measures enable us to correct the bad behavior of the single error measured wrt. uniform distribution over the whole hypercube $\mathbb{B}^n$. Actually, the two error measures we consider are described and get different names such as false positive rate and false negative rate, in several works from different research fields including medical testing, information retrieval, binary classification; but there seems to be no mention of them in the field of algorithmic learning theory.

The absence of negative examples, makes the learning problem less constrained, somewhat ill-posed and solvable in some trivial ways that we want to avoid. Consequently, to the best of our knowledge, this problem has not been studied much in the algorithmic learning theory literature. Learning problems without negative examples are studied in machine learning under the name of one class classification (see e.g. Khan and Madden (2014) for a survey) but we did not found any work that consider learning boolean functions and especially DNF representations as we do. The contribution of this paper is thus twofold: we first provide a conceptual solution to the problem of how much to generalize beyond the sample and where. This solution yields an algorithm that works well empirically in terms of misclassification error on synthetic randomly-generated functions. The algorithm is based on some properties of terms/cubes and efficient operations on them.

The rest of the paper is organized as follows. In Section 2 we introduce Boolean functions, their DNF representation as well as a study of terms/cubes and various operations on terms and show they can be computed efficiently. In Section 3 we define the learning problem and our empirical greedy solution. In section 4, we illustrate our preliminary results on a randomly sampled DNF.

2. Boolean Functions, Terms and Cubes

A Boolean function $f : \mathbb{B}^n \to \mathbb{B}$ is first and foremost a semantic object that can be represented in a non-unique manner by formulas over a set of variables $X = x_1, \ldots x_n$ using operators taken from some basis, for example $\{\wedge, \vee, \neg\}$. The set of functions $\mathcal{F}$ is isomorphic to the set of subsets of the Boolean hyper-cube $\mathbb{B}^n$ which induces a natural partial order $(\mathcal{F}, \subseteq)$ based on set inclusion and an isomorphic lattice structure $(\mathcal{F}, \cup, \cap)$. The top and bottom elements of these structures are, respectively, $\mathbb{B}^n$ (also denoted as $\top$ or true) and $\emptyset$ (also denoted as $\bot$ or false). We use $|f|$ to denote

the (semantic) size of f. Any Boolean function can be written in a disjunctive normal form (DNF, sum of products).

Definition 1 (Disjunctive Normal Form) *A* literal *is either a variable x_i or its negation $\overline{x}_i$ for some i. A* term *is a conjunction of literals. A function is written in a disjunctive normal form if it has the form $r_1 \vee \ldots \vee r_k$ where each r_i is a term. More specifically we say that f is written as a k-term DNF.*

Term/cubes constitute a special class of convex Boolean functions that play a special role in logic minimization Kohavi and Jha (2009) and also in our learning procedure where the basic step is to group a set of sample points and cover them by the smallest cube that contains them all. The inclusion in h of the elements of the cube which are not in the sample, constitutes the act of generalization in our algorithm. In the sequel we make some definitions and observation on those, while freely interchanging the syntactic term *term* and the semantic term *cube*. Thus, a k-term DNF representation of f, covers it by a union of cubes $f = r_1 \cup \ldots \cup r_k$.

Definition 2 (Terms/Cubes) *A term r is a subset of $\mathbb{B}^n$ which is characterized by: 1) The set of variables it ignores (does not care about). We will use $\star$ to denote this fact for a given variable; and 2) The values (0 or 1) it imposes on the other variables. In other words, a term r partitions the variables in X into three classes, r^0, r^1 and $r^\star$.*

Example: for $n = 4$, consider the term r written syntactically in logical style as $x_1 \wedge \overline{x}_3$ or as $x_1\overline{x}_3$. It ignores variables x_2 and x_4 and requires x_1 to be 1 and x_3 to be 0. The variable partition is $\{r^0, r^1, r^\star\} = \{\{x_3\}, \{x_1\}, \{x_2, x_4\}\}$. The term denotes the set $r = \{1000, 1100, 1001, 1101\}$ which is a cube. We will represent terms as elements of $\{0, 1, \star\}^n$ and in our example $(1 \star 0\star)$. ∎

The order of a term, $\omega(r) = |r^\star|$, is the number of $\star$-variables and its semantic size is $|r| = 2^{\omega(r)}$. A minterm is a term of order 0, it cares about all the variables and is semantically a singleton. Terms can be partially-ordered according to set inclusion with $\top = (\star, \ldots, \star) = \mathbb{B}^n$ as a top element and the special symbol $\perp$, denoting the empty set, at the bottom. We use $\mathcal{R}$ to denote the set of all $3^n + 1$ terms. Two terms r and s are *conflicting* if they disagree on at least one coordinate that they both care about. They are *adjacent* if they disagree on exactly one coordinate that they both care about and have the same size.

Cubes, like convex objects, are naturally closed under intersection but not under union. To be more precise, the union of two cubes is a cube in two cases: trivially, when one is included in the other, or if they are adjacent. Still $\mathcal{R}$ can be made lattice using the following operation.

Definition 3 (Join) *The join $r \sqcup s$ of two cubes is the smallest cube containing $r \cup s$.*

One can check that $r \sqcup s$ is uniquely defined and that $(\mathcal{R}, \cap, \sqcup)$ is a lattice. The join operation, which is the Boolean analog of the convex hull of the union, is commutative and associative. The level of semantic enlargement associated with joining two cubes is captured by the following definition.

Definition 4 (Join Cost) *For two terms r and s, let*

$$\begin{aligned} c(r, s) &= |r \sqcup s| - |r \cup s| \\ &= |r \sqcup s| - |r| - |s| + |r \cap s| \end{aligned}$$

We note that in the context of sparse functions, we will often deal with cubes whose intersection is empty.

The Hamming distance d on $\mathbb{B}^n$ can be lifted to $\mathcal{R}$ using the Hausdorff distance between sets.

$$\overrightarrow{\delta}(r,s) = \max_{x \in r} \min_{y \in s} d(x,y) \qquad \delta(r,s) = \max\{\overrightarrow{\delta}(r,s), \overrightarrow{\delta}(s,r)\}.$$

Intuitively, the further r and s are according to δ, the larger is their join cost. Various tests and operations on the semantics of terms can be performed efficiently based on their syntactic ternary representation.

Proposition 5 (Operations on Terms) *Inclusion, intersection, join, join cost and distance between terms can be computed in $O(n)$.*

Proof Let r and s be two terms. One can check that all operations and relations can be done component-wise as follows:

$$\begin{array}{lcll} r \subseteq s & = & \bigwedge_i r_i \subseteq s_i & \\ r \cap s & = & \begin{cases} \bot & \text{if they are conflicting} \\ (r_1 \cap s_1, \ldots, r_n \cap s_n) & \text{otherwise} \end{cases} & \\ r \sqcup s & = & (r_1 \sqcup s_1, \ldots, r_n \sqcup s_n) & \\ \overrightarrow{\delta}(r,s) & = & \sum_{i=1}^{n} \overrightarrow{\delta}(r_i, s_i) & \end{array}$$

The component-wise operations are defined as follows.

$\subseteq$	0	1	$\star$
0	1	0	1
1	0	1	1
$\star$	0	0	1

$\cap$	0	1	$\star$
0	0		0
1		1	1
$\star$	0	1	$\star$

$\sqcup$	0	1	$\star$
0	0	$\star$	$\star$
1	$\star$	1	$\star$
$\star$	$\star$	$\star$	$\star$

$\overrightarrow{\delta}$	0	1	$\star$
0	0	1	0
1	1	0	0
$\star$	1	1	0

■

3. Learning: Complexity and Cardinality Trade-offs

We now come back to the learning problem that we define (still not fully formally) as follows.

Definition 6 (Learning from Positive Examples) *Let f be a sparse Boolean function and let $S \subset f$ be a sample of positive examples, with $|S| = m \ll |f|$. Find a function h which is a good approximation of f.*

The objective evaluation of h is done in terms of the misclassification error (false positive and false negatives) compared to f. This is done by counting misclassifications errors and compare this to the size of f or h to have a probabilistic interpretation.

Definition 7 (Misclassification Errors) *The misclassification errors of a learned function h relative to a target function f are*

- *False negative: $\mathcal{E}_-(f,h) = |f \cap \overline{h}|/|f| = p(\overline{h}|f)$: the proportion of f elements that h misses;*
- *False positive: $\mathcal{E}_+(f,h) = |\overline{f} \cap h|/|h| = p(\overline{f}|h)$; the proportion of h elements which are not in f.*

Note that we use measures adapted to sparse functions that are not standard in the context of algorithmic learning theory where the commonly-used measure is the probability of under uniform distribution over $\mathbb{B}^n$, namely $\mathcal{E}(f,h) = (|f \cap \overline{h}| + |\overline{f} \cap h|)/2^n$. With this latter measure, the function $\bot$ which accepts nothing gives trivially an ε- approximation of f for any sparse function satisfying $|f| \leq \varepsilon 2^n$. For instance if there are 100 variables and $\varepsilon = 2^{-50}$, a function f whose satisfaction set "only" contains 2^{50} elements is considered as neglectible[1].

Note that $\mathcal{E}_+$ can only be used for a posteriori evaluation of the algorithm, not by the algorithm itself, which does not observe negative examples, while an empirical estimation of $\mathcal{E}_-$, based on a testing sample T can be used by the algorithm.

Taking sample compatibility as a basic requirement for h, and unlike the case of mixed positive and negative examples, we can be S-compatible by letting h be the universal function $\top$ that accepts all elements of $\mathbb{B}^n$. On the other extreme, we can select the function f_s which accepts exactly the sample and nothing beyond that. Any function situated between the two in terms of set inclusion is sample compatible and unless we add some additional considerations, there is no compelling reason to prefer one such function over another. We have too much freedom in selecting $|h|$, and even if had such a cardinality target, the question of where to generalize beyond S remains.

We approach the problem by approximately computing complexity and cardinality trade-offs. Let $||h||$ denote the syntactic complexity of h, that is, the number of terms in its DNF representation. Our procedure starts from f_s, which can be written as a DNF with $|S|$ terms, and then it progressively reduces the number of terms, replacing them by larger ones, while trying to generalize as little as possible. Our algorithm gives an approximate solution to the following hard problem.

Definition 8 (Minimal Cardinality k-Term DNF) *Given a sample S, $|S| = m$, and $k \leq m$, find a sample-compatible function h such that $||h|| \leq k$ and $|h|$ is minimal. We denote $|h|$ by $\alpha(S,k)$.*

Clearly, $\alpha(S, k+1) \leq \alpha(S,k)$ and $S \subseteq S'$ implies $\alpha(S,k) \leq \alpha(S',k)$.

To have a decision version of Problem 8, rather than minimizing $|h|$, one asks whether it is bounded from above by a given constant M. This problem is NP-hard since the particular case where $M = |S|$ corresponds to require that h is a k-term DNF that assigns truth values exactly to the set S. This latter problem is well known to be NP-complete (see Allender et al. (2008) and references therein). Yet, we do not know if our problem belongs to NP but we know that it belongs to #P Valiant (1979). Indeed, after guessing a partition of S that defines h in polynomial time, we evaluate $|h|$ by counting the number of boolean vectors that satisfy the DNF formula h. This latter counting problem is known to be #P-complete.

Algorithm 1 can provide a greedy approximation of $\alpha(S,k)$ for every $k \in [1..m]$, thus approximating the Pareto front of $(||h||, |h|)$ trade-offs. We start with f_s which is semantically minimal,

1. We recall also that the usual complexity requirement in (e.g. PAC) learning is to be polynomial in $1/\varepsilon$. So requiring error bound $\varepsilon = 2^{-50}$ leads to a problem instance very far from being tractable.

$|f_s| = |S|$. Then iteratively, we select two terms that minimize $c(r_1, r_2)$, join them into one term $r = r_1 \sqcup r_2$ which replaces them in the new function. We have thus reduced $||h||$ by 1 and enlarged $|h|$ as little as possible. We repeat the process until a stopping criterion is met. When this criterion is $k = 1$, we obtain the single term $\sqcup S$.

Algorithm 1 Finding S-compatible k-Term DNFs with Small Size

Input: A sample S, $|S| = m$.
Output: A sequence $h^m, h^{m-1}, \ldots$ of functions such that h^k is a k-term S-compatible DNF.
$k = m$
$h = S$ $\{h = f_s$ represented as a set of m minterms of sample points$\}$
repeat
  output(h, k)
  $k = k - 1$
  $(r_1, r_2) = \arg\min_{r,r' \in h} c(r, r')$ {find two terms that minimize join cost}
  $r = r_1 \sqcup r_2$ {join the two terms}
  $h = h - \{r_1, r_2\} \cup \{r\}$ {replace the two terms by their join}
until $k = 1$

Note that for each k, the formula $h = r_1 \vee \ldots \vee r_k$ obtained by our algorithm can be represented by a partition $\pi = \{S_1, \ldots, S_k\}$ of S such that for every i, $r_i = \sqcup S_i$. In the following we show that the set of formulas generated by such partitions contains the optimal solution.

Proposition 9 (Joins of Sample) *For every k, there exists an optimal solution for Problem 8 of the form $h = r_1 \vee \ldots \vee r_k$ and a partition $\pi = \{S_1, \ldots, S_k\}$ of S such that for every i, $r_i = \sqcup S_i$.*

Sketch of Proof: We will show that any solution not satisfying the above can be transformed to a solution that does, without increasing the semantics. For every i, let $T_i = r_i \cap S$ be the sample elements included in r_i. Then $r'_i = \sqcup T_i \subseteq r_i$ and it provides the same coverage of T_i as does r_i.

The set $\pi' = \{T_1, \ldots, T_k\}$ is a semi-partition and not necessarily a partition. To make it a partition, we select for each $x \in S$ a unique T_i among those in which it is included and delete it from the other sets. We thus obtain a partition $\pi = \{S_1, \ldots, S_k\}$ with $S_i \subseteq T_i$ for every i and hence $\sqcup S_i \subseteq \sqcup T_i$. Since π is a partition we have $S \subseteq \bigcup_i \sqcup S_i$. ∎

This result reduces the problem of finding an optimal formula of size k to finding a k-partition of S that minimizes the volume of the generated rectangles and opens the way for applying other heuristic combinatorial optimization algorithms such as stochastic local search.

4. Illustration of the learning algorithm

Algorithm 1 should be modified a bit to become a learning algorithm, by providing the stopping criterion other than $k = 1$.

Figure 1 (Left) illustrates the behavior of the algorithm on a sample of size 100 uniformly drawn from a function f defined by a 10-term DNF over 50 variables given as follow:

$$
\begin{array}{ll}
000000*00000*0000000*10*000000000000000*00*000 & + \\
0*000*00*000010*00*00*000**000000000000000**000000 & + \\
00*000000*000000010**0*001*1010*101000000*0*0*0010 & + \\
000000*100*0001001000**00*00*00000*0*1000010*0*100 & + \\
00001000011*1001110*001*000*010*0*0*0100*1*001100* & + \\
1*00**011000*0011*1*00*0000*1100*0011000010100*010 & + \\
10001001*01010010101**000**01*1*00011*000111100**0 & + \\
11*1***00111*11111*111*01010111111*111*1101*011111 & + \\
110101110*1*00*111110*00*0*1011110011*0*0101*11*01 & + \\
1111*111*1111111*1*1111*11*11*111*1*1111111111*111 &
\end{array}
$$

In the plots we see the empirical estimation of the false negative and false positive errors:

$$p(\bar{h}|f) = |T_1 \cap \bar{h}|/|T_1| \text{ and } p(\bar{f}|h) = |T_2 \cap \bar{f}|/|T_2|$$

for positive testing sample $T_1 \subseteq f$ of size 1000 and a negative evaluation sample $T_2 \subseteq h$ of size 100.[2]

Initially, $\mathcal{E}_+ = 0$ and $\mathcal{E}_-$ is large, close to 1, and they advance in their respective opposing directions as the algorithm goes along. It is interesting to observe that after the number of terms in h reaches 10, and we move to the next step to obtain a 9-term DNF, there is a sharp increase in cardinality and in $\mathcal{E}_+$. Of course, our algorithm does not know $||h|| = k$ in advance and it needs a more objective criterion. It seems that our stopping criterion will be based on the following two:

1. The empirical estimation of the false negatives on a test set $\mathcal{E}_-(f,h) \approx |T_1 \cap \bar{h}|/|T_1|$. We will stop when it goes below some small threshold;

2. The growth of the volume $|h|$ of the function. Intuition is that a sharp growth occurs when we merge distant cubes which should not be merged, and this is likely to increase the false positive error $\mathcal{E}_+$ that we cannot observe directly.

Figure 1 (Right) gives in logarithmic scale the size of the hypothesis $|h|$ as a function of the number of steps for the same example as before. We can clearly see the elbow in the curve when the number of terms goes below 10, which is the number of terms in the target DNF. Note that if we reverse the x-axis we obtain an over-approximation of the Pareto-front for the trade-off between syntactic size (number of terms $||h||$) and semantic size ($|h|$).

We plan to look more closely on what makes functions easier or harder to learn. Our first intuition based on experiments is that in a "good" k-term DNF, the terms are well-separated and the distance between them is larger than their diameter. They are easy to cluster. In bad ones, the terms are closer and may intersect.

References

Eric Allender, Lisa Hellerstein, Paul McCabe, Toniann Pitassi, and Michael Saks. Minimizing disjunctive normal form formulas and ac^0 circuits given a truth table. *SIAM Journal on Computing*, 38(1):63–84, 2008.

2. Sample T_2 is smaller for local technical reasons because it has to be generated anew each time as h changes.

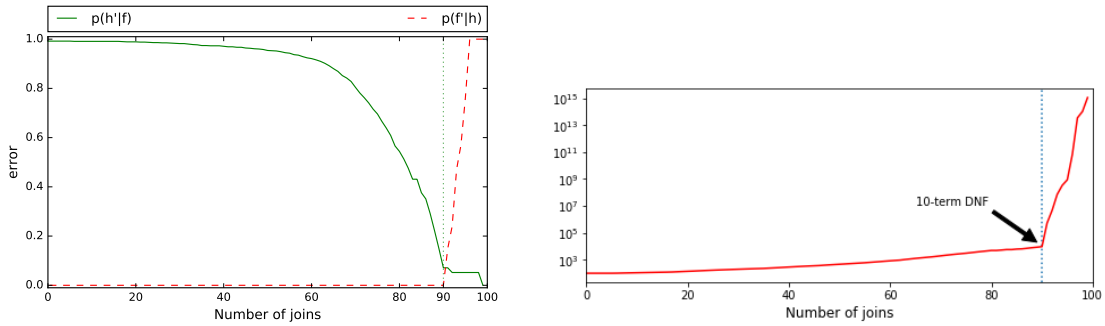

Figure 1: The evolution of misclassification errors (Left) and of the size $|h|$ (Right) as the number of terms is reduced. Note the "phase-transition"-like behavior when the number of terms in h coincides with the number of terms in f

Eugene Asarin, Alexandre Donzé, Oded Maler, and Dejan Nickovic. Parametric identification of temporal properties. In *Runtime Verification*, pages 147–160. Springer, 2011.

Ezio Bartocci, Jyotirmoy Deshmukh, Alexandre Donzé, Georgios Fainekos, Oded Maler, Dejan Nickovic, and Sriram Sankaranarayanan. Specification-based monitoring of cyber-physical systems: A survey on theory, tools and applications.

Nader H Bshouty, Elchanan Mossel, Ryan ODonnell, and Rocco A Servedio. Learning dnf from random walks. *Journal of Computer and System Sciences*, 71(3):250–265, 2005.

Jeffrey C Jackson. An efficient membership-query algorithm for learning dnf with respect to the uniform distribution. *Journal of Computer and System Sciences*, 55(3):414–440, 1997.

Xiaoqing Jin, Alexandre Donzé, Jyotirmoy V. Deshmukh, and Sanjit A. Seshia. Mining Requirements from Closed-loop Control Models. In *HSCC*, 2013.

Shehroz S Khan and Michael G Madden. One-class classification: taxonomy of study and review of techniques. *The Knowledge Engineering Review*, 29(3):345–374, 2014.

Zvi Kohavi and Niraj K Jha. *Switching and finite automata theory*. Cambridge University Press, 2009.

Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In *FORMATS/FTRTFT*, pages 152–166, 2004.

Oded Maler, Dejan Nickovic, and Amir Pnueli. Checking temporal properties of discrete, timed and continuous behaviors. In *Pillars of Computer Science*, pages 475–505, 2008.

Leslie G Valiant. The complexity of computing the permanent. *Theoretical computer science*, 8(2): 189–201, 1979.

Leslie G Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.