Learning monotone Boolean functions

Carl Burch*

Question We want to predict the value of a hidden *monotone Boolean* function f: f maps vectors of n bits onto Boolean values, and if $f(x) = 1$ for a vector x, then flipping any bit of x from 0 to 1 keeps $f(x) = 1$.

We would like to predict how the hidden function f will label a random bit vector x. For information about f, our algorithm can get a small number (polynomial in n) random bit vectors and the label f gives them. The question is this: What should we do to maximize our probability of agreeing with $f(x)$ (no matter what f is—the probability is over the random choice of x), and what guarantee can we make about this probability? Guaranteeing $1/2$ is easy—flip a coin—but how much better can we do?

Motivation This question arises in learning settings where we want to predict a hidden function—whether a consumer will like granola bars, for example. We are given some history about the problems. One shopper who regularly buys Cheerios and raisins but not oatmeal liked granola bars; another who buys Cheerios but not not raisins or oatmeal did not. Should we offer granola bars to a shopper who buys only raisins?

To make the problem tractable, we make assumptions about consistency, monotonicity, and uniformity: The function will always label a bit vector the same way (*consistency*); each 1 bit can only help, and no 0 bit can help (*monotonicity*); examples are uniformly distributed among the bit vectors (*uniformity*). Removing these makes the problem much more difficult (consistency) or even impossible (monotonicity and uniformity). (*Impossible* here prohibits guarantees of more than $1/2 + 1/poly(n)$, where $poly(n)$ can be any polynomial in n.)

Results The following table summarizes the history of the problem and our results. The algorithms represent known guarantees; the hardness results represent upper bounds on the most any algorithm can guarantee. (The hardness results are actually stronger: They hold even if the algorithm is allowed to choose the labeled bit vectors.)

author	algorithm	hardness result
[KLV]	$1/2 + O(1/n)$	$1 - \omega(1/\sqrt{n})$
[BT]	$1/2 + O((\log^2 n)/n)$	
us	$1/2 + O(1/\sqrt{n})$	$1/2 + O((\log n)/\sqrt{n})$

Thus our research closes the question of the learnability of monotone Boolean formula to within an $O(\log n)$ factor.

This abstract summarizes the algorithm achieving our result; it does not prove the guarantee, nor does it prove the hardness result. These are available in the full paper, to appear at FOCS '98 coauthored with Avrim Blum and John Langford,

`www.cs.cmu.edu/~cburch/pub/bbl_monotone.abs.html`

Algorithm Our algorithm is surprisingly simple.

1. Draw $O(n)$ random samples labeled by f, and let x be the vector our algorithm should label.
2. If at least a $1/2+0.05/\sqrt{n}$ fraction of samples are labeled 1, then we predict that $f(x)$ is 1. Likewise, if at least a $1/2+0.05/\sqrt{n}$ fraction of samples are labeled 0, then we predict that $f(x)$ is 0.
3. Otherwise, we predict according to the majority of the bits of x. (If more than half of the bits of x are 1, we predict 1. Otherwise we predict 0.)

Future research This research closes the question of the learnability of monotone Boolean formulas, aside from the minorly irritating $O(\log n)$ factor.

A big open question is the learnability of *monotone DNF*. Say we know that f is representable as an *OR* of k terms, each of which is an *AND* of some bits, with no *NOT*s. If we can get $poly(n, k)$ bit vectors labeled according to f, how well can we predict? (The difference from the question addressed by this research is that now the number of labeled bit vectors can be polynomial in the number of terms.)

If the algorithm is allowed to choose the bit vectors, then the chance of correctness can be made arbitrarily close to 1 [Jac]. But how well can we do if the bit vectors are random? The algorithms for monotone Boolean formulas are the best known, and so the simple algorithm described above is the state of the art. There is no hardness result known for this question.

References

[BT] N Bshouty and C Tamon. On the Fourier spectrum of monotone functions. *Journal of the ACM*, 43(4):747–770, Jul 1996.

[Jac] J Jackson. *The Harmonic Sieve: A Novel Application of Fourier Analysis to Machine Learning Theory and Practice*. PhD thesis, CMU, 1995. CMU-CS-95-183.

[KLV] M Kearns, M Li, and L Valiant. Learning Boolean formulas. *Journal of the ACM*, 41(6):1298–1328, Nov 1994.

*Carnegie Mellon University. Joint work with Avrim Blum and John Langford (CMU). Carl Burch is supported in part by a National Science Foundation Graduate Fellowship. E-mail: cburch+@cs.cmu.edu.