Halfspace Learning, Linear Programming, and Nonmalicious Distributions

Philip M. Long*
Computer Science Department
Duke University
P.O. Box 90129
Durham, North Carolina 27708

September 7, 1994

Abstract

We study the application of a linear programming algorithm due to Vaidya to the problem of learning halfspaces in Baum's nonmalicious distribution model. We prove that, in n dimensions, this algorithm learns up to ϵ accuracy with probability $1-\epsilon$ in

$$O\left(\frac{n^2}{\epsilon}\log^2\frac{n}{\epsilon}+n^{3.38}\log\frac{n}{\epsilon}\right)$$

time. For many combinations of the parameters, this compares favorably with the best known bound for the perceptron algorithm, which is $O(n^2/\epsilon^5)$.

Keywords: Computational learning theory, analysis of algorithms.

*Supported by Air Force Office of Scientific Research grant F49620–92–J–0515 and a Lise Meitner Fellowship from the Fonds zur Förderung der wissenschaftlichen Forschung (Austria).

1 Introduction

In part due to their interpretation as artificial neurons [13, 15], a great deal has been written about the analysis of halfspaces in different computational models of learning. In this paper, we analyze the use of one of Vaidya's linear programming algorithms [18] to learn halfspaces in a variant of the PAC model [19] due to Baum [2], called the *non-malicious distribution (NMD) model.*

In the NMD model, a halfspace $H \subseteq \mathbb{R}^n$ is chosen randomly. There is a straightforward reduction from the problem of learning arbitrary halfspaces to learning homogeneous halfspaces, i.e., those that go through the origin (see, e.g. [5]). We will therefore restrict our attention to homogeneous halfspaces. The halfspace H to be learned is then chosen by picking its normal vector uniformly from the unit sphere. (If one works through the reduction from general halfspaces, this gives rise to the distribution on general halfspaces obtained by choosing $(w_1, ..., w_n, b)$ uniformly from the unit ball in $\mathbb{R}^{n+1}$, and setting the halfspace to be learned to be $\{\vec{x} : \vec{w} \cdot \vec{x} \geq b\}$.) It is further assumed that there is a probability distribution D over $\mathbb{R}^n$ that is unknown to the learner, but that the learner may (in unit time) sample points $\vec{x}_1, ..., \vec{x}_m$ independently at random according to D. Furthermore, the learner can find out whether each of these points is in the hidden halfspace H. The goal of the learner is to output an approximation $\hat{H}$ of H. The accuracy of $\hat{H}$ is measured by the probability that another point x drawn from D will be in the symmetric difference of $\hat{H}$ and H, i.e., that $\hat{H}$ is "wrong" about x. The learner is given an input $\epsilon > 0$, and is required to ensure that with probability at least $1 - \epsilon$, with respect to the random choice of H and $\vec{x}_1, ..., \vec{x}_m$, the accuracy of $\hat{H}$ is better than ϵ. This model takes a Bayesian viewpoint, as has been done in [9, 14] and elsewhere.

In Baum's original formulation, there were two separate parameters measuring the accuracy and the probability that the given accuracy was achieved. However, combining the two simplifies our bounds greatly, and it seems that in practice the two should at least be close. Both being extremely sure of having a mediocre hypothesis, and being fairly sure of having an outstanding hypothesis seem unusual goals.[1]

Baum [2] showed that the perceptron algorithm [15] accomplishes the above in $O(n^2/\epsilon^5)$ time. In this paper, we show that an algorithm based on linear programming [18],[2] requires

$$O\left(\frac{n^2}{\epsilon}\log^2\frac{n}{\epsilon} + n^{3.38}\log\frac{n}{\epsilon}\right)$$

time.

For many values of n and ϵ, this represents a substantial improvement. For example, when n and $1/\epsilon$ are within a constant factor, the linear programming bound of this paper grows like $O(n^{3.38}\log n)$, which compares favorably with the $O(n^7)$ bound in this case for the perceptron algorithm.

We make use of $O(n^{2.38})$ time matrix multiplication [4] as a subroutine. Since the constants in the analysis of this matrix multiplication algorithm, as well as those for other $o(n^3)$ algorithms, are quite large, some claim that in practice the time required for multiplying $n \times n$ matrices grows like

[1] The proof of results for the PAC model [8] that show that the dependence of the requirements on the inverse of the "confidence" can always be brought down to being logarithmic cannot be easily modified for this model, since, loosely speaking, here the probability of failure depends on the random choice of the target as well as the random sample.

[2] The idea of using linear programming for recovering a "separating" halfspace is quite old (see [5]), and its use to generate polynomial time learning algorithms originates with Blumer, Ehrenfeucht, Haussler and Warmuth [3].

$O(n^3)$. However, replacing the $n^{2.38}$ in our learning bounds with n^3 only changes the $n^{3.38}$ in our bound to an n^4. An $O(n^4 \log n)$ bound would still be a significant improvement over $O(n^7)$.

The most natural application of linear programming techniques to the problem of learning halfspaces is to draw a sample $\vec{x}_1, ..., \vec{x}_m$, and then find a halfspace $\hat{H}$ for which each of $\vec{x}_1, ..., \vec{x}_m$ is in $\hat{H}$ if and only if it is in the halfspace H to be learned, since this requirement amounts to m linear constraints on the normal vector of $\hat{H}$ [3]. However, in order to simply plug in to the time bounds for solving linear inequalities (with m inequalities and n unknowns), one must make assumptions on the precision of the $\vec{x}_j$'s. (It is open whether there is a polynomial time algorithm for the unit-cost model for linear programming. Such an algorithm would imply unit-cost polynomial time algorithms for learning in the NMD model, as well as Valiant's PAC model [19] (see [3]).)

In the NMD model, we have found that a different approach doesn't require any assumptions on the precision of the $\vec{x}_j$'s, and yields significantly improved time bounds.[3]

The iterations in Vaidya's linear programming algorithm can be viewed as what are called *queries* in another model of learning (due to Angluin [1]) often called the *equivalence query model* [12]. Implicit in Vaidya's analysis of his algorithm are bounds the number of iterations made by this algorithm in terms of the volume of the solution set of the system of linear inequalities [18]. These bounds can therefore also be viewed as equivalence query bounds. Using conversions from this model to the PAC model [1, 10, 11], one can see that, for arbitrary $\epsilon > 0$, if the volume of the set of normal vectors defining halfspaces that "agree" with the target halfspace H on $\vec{x}_1, ..., \vec{x}_m$ is not unreasonably small, then, with high probability, a randomized variant of Vaidya's algorithm quickly finds a setting of weights whose associated halfspace $\hat{H}$ satisfies that the fraction of elements of $\vec{x}_1, ..., \vec{x}_m$ which lie in the symmetric difference of $\hat{H}$ and H is at most $\epsilon/2$. Applying results of Blumer, Ehrenfeucht, Haussler and Warmuth, for $m = O((n/\epsilon)\log(1/\epsilon))$, this is good enough. Finally, we show that, with high probability, the volume of the set of normal vectors defining halfspaces which "agree" with the halfspace to be learned on $\vec{x}_1, ..., \vec{x}_m$ is not too small.

2 Preliminaries

Let $\mathbb{N}$ denote the positive integers and $\mathbb{R}$ the reals. For sets S_1 and S_2, let $S_1 \Delta S_2$ denote the symmetric difference of S_1 and S_2. We use the unit cost RAM model of computation (see, e.g. [17]), where it is assumed that the standard operations on real numbers can be done in unit time.

Choose $X = \cup_n X_n$, and for all n, let $\mathcal{C}_n$ be a class of subsets of X_n, and $\mathcal{C} = \cup_n \mathcal{C}_n$. Such a $\mathcal{C}$ is called a *concept class*. An *example* of $C \in \mathcal{C}_n$ is a pair $(x, \chi_C(x)) \in X_n \times \{0,1\}$, where χ_C is C's characteristic function. A subset C' of X_n is *consistent* with $(x, \chi_C(x))$ if $\chi_C(x) = \chi_{C'}(x)$. A *sample* is a finite sequence of examples. We say $C' \subseteq X_n$ is consistent with a sample if it is consistent with each example in the sample.

A *batch learning algorithm* A takes as inputs an $n \in \mathbb{N}$, a desired accuracy $\epsilon > 0$, and a sequence of elements of[4] $X_n \times \{0,1\}$. A then outputs an algorithm for computing a function $h : X_n \to \{0,1\}$. Usually, the algorithm can be understood from the description of h, and we will refer to the two interchangeably in the rest of this paper.

We begin with the PAC model [19]. For a function $\phi : \mathbb{R} \times \mathbb{N} \to \mathbb{N} \cup \{\infty\}$, we say A PAC-learns $\mathcal{C}$ in time ϕ if and only if there is a function $\psi : \mathbb{R} \times \mathbb{N} \to \mathbb{N} \cup \{\infty\}$ such that for all input parameters

[3] In fact, for many combinations of the parameters, this approach yields improved time bounds for Valiant's PAC model, but the improvement is less dramatic.

[4] In this paper, an encoding for the elements of X will always be obvious.

n and ϵ, if $m = \psi(1/\epsilon, n)$:

- $m \leq \phi(1/\epsilon, n)$,
- whenever A receives n, ϵ, and examples $(x_1, y_1), ..., (x_m, y_m)$, A halts within time $\phi(1/\epsilon, n)$ and outputs a hypothesis h, whose running time (on inputs in X_n) is also at most $\phi(1/\epsilon, n)$,
- for all probability distributions D on X_n, and for all $C \in \mathcal{C}_n$, if A is given m examples of C generated independently according to D, the probability that the output h of A satisfies $\mathbf{Pr}_{u \in D}(h(u) \neq \chi_C(u)) \leq \epsilon$ ("has accuracy better than ϵ") is at least $1 - \epsilon$.

Now we turn to the NMD model [2]. Suppose, for each $n \in \mathbb{N}$, $\mathcal{P}_n$ is a probability distribution on $\mathcal{C}_n$, and $\mathcal{P} = \{\mathcal{P}_n : n \geq 1\}$. In this case, we say A PAC-learns $\mathcal{C}$ with respect to $\mathcal{P}$ in the NMD model in time ϕ if and only if there is a function $\psi : \mathbb{R} \times \mathbb{N} \to \mathbb{N}$ such that for all input parameters n and ϵ if $m = \psi(1/\epsilon, n)$:

- $m \leq \phi(1/\epsilon, n)$,
- for any $x_1, ..., x_m$, the expectation, over a randomly chosen C according to $\mathcal{P}_n$, of the running time of A on inputs n, ϵ, and $(x_1, \chi_C(x_1)), ..., (x_m, \chi_C(x_m))$ is at most $\phi(1/\epsilon, n)$, and the running time of the resulting hypotheses h (on inputs in X_n) is at most $\phi(1/\epsilon, n)$,
- for all probability distributions D on X_n, if $(x_1, ..., x_m)$ and C are chosen independently at random from D^m and $\mathcal{P}_n$, and $(x_1, \chi_C(x_1)), ..., (x_m, \chi_C(x_m))$ is given to A, the probability that the output h of A satisfies $\mathbf{Pr}_{u \in D}(h(u) \neq \chi_C(u)) \leq \epsilon$ is at least $1 - \epsilon$.

We will also discuss concept classes not indexed by n. It should be obvious how to adjust the above definitions for this case.

Finally, we define the equivalence-query model [1]. In the equivalence-query model, when the algorithm is learning a class $\mathcal{C}$, it sequentially produces *queries* which are (representations of) elements of $\mathcal{C}$. When the algorithm correctly guesses the function C to be learned, the learning process is over. Otherwise, if the algorithm has queried $\hat{C}$, it receives $x \in \hat{C} \Delta C$. A learning algorithm for this model is called a *query algorithm*.

For each n, let HALF_n be the set of all homogeneous halfspaces in $\mathbb{R}^n$, i.e.

$$\{\{\vec{x} \in \mathbb{R}^n : \vec{w} \cdot \vec{x} \geq 0\} : \vec{w} \in \mathbb{R}^n\}.$$

Let $\mathrm{HALF} = \cup_n \mathrm{HALF}_n$.

3 The analysis

We will make use of the following lemma, which is implicit in the analysis of Littlestone.

Lemma 1 ([11]) *Choose $X, \mathcal{C} \subseteq 2^X$. Suppose for any $x \in X, C \in \mathcal{C}$, the question of whether $x \in C$ can be computed in β time. If there is a query algorithm B for $\mathcal{C}$ for which the time taken by B between queries is at most α, and the algorithm makes at most γ queries, then there is a PAC learning algorithm for $\mathcal{C}$ which outputs hypotheses from $\mathcal{C}$ whose time requirement is*

$$O\left(\gamma\left(\alpha + \frac{\beta \ln \frac{\gamma}{\epsilon}}{\epsilon}\right)\right)$$

and which uses

$$O\left(\frac{1}{\epsilon}\left(\gamma + \log\frac{1}{\epsilon}\right)\right)$$

random examples.

The following notation will be useful. Fix some n. For each $\vec{w} \in \mathbb{R}^n$, let

$$H_{\vec{w}} = \{\vec{x} \in \mathbb{R}^n : \vec{w} \cdot \vec{x} \geq 0\}.$$

For some finite set $S \subseteq \mathbb{R}^n$ and some $\vec{w} \in \mathbb{R}^n$, let $H_{\vec{w},S} = H_{\vec{w}} \cap S$. Also, define

$$E_{S,\vec{w}} = \{\vec{u} \in \mathbb{R}^n : H_{\vec{u},S} = H_{\vec{w},S}, ||\vec{u}||_2 \leq 1\}.$$

Informally, $E_{S,\vec{w}}$ represents the set of all halfspaces that are indistinguishable from $H_{\vec{w}}$ on the basis of S. Finally, for a finite set $S \subseteq \mathbb{R}^n$, and $v \geq 0$, let

$$\text{HALF}_{S,v} = \{H_{\vec{w},S} : \vec{w} \in \mathbb{R}^n, \text{volume}(E_{S,\vec{w}}) \geq v\}.$$

Now we are ready for another lemma, which is implicit in the analysis of Vaidya [18], if his linear programming algorithm is viewed as a query algorithm in the manner of Maass and Turán [12].

Lemma 2 ([18, 12]) *For any finite $S \subseteq \mathbb{R}^n$, there is a query algorithm MTV, such that, for all $v > 0$, MTV learns $HALF_{S,v}$ while using at most $O(n^{2.38})$ time between queries, and makes at most*

$$O\left(n \log n + \log\frac{1}{v}\right)$$

queries.

We also make use of a special case of a result of Blumer, Ehrenfeucht, Haussler, and Warmuth.

Lemma 3 ([3]) *There is a constant c such that, for all $\epsilon < 1/4$, then if a (randomized) batch learning algorithm for learning $HALF_n$ draws*

$$m \geq \frac{cn}{\epsilon}\log\frac{1}{\epsilon}$$

random examples, and with probability at least $1 - \epsilon/2$ outputs a halfspace $\hat{H}$ such that the fraction of examples in the symmetric difference of $\hat{H}$ and the halfspace H to be learned is at most $\epsilon/2$, then $\hat{H}$ has accuracy better than ϵ with probability at least $1 - \epsilon$.

The following bound will also be useful.

Lemma 4 (see [16, 6, 3]) *Choose $m, n \in \mathbb{N}$, and $S = \{\vec{x}_1, ..., \vec{x}_m\} \in \mathbb{R}^n$. Then*

$$|\{T \subseteq S : \exists H \in HALF_n, H \cap S = T\}| \leq \left(\frac{em}{n}\right)^n.$$

Finally, we record a trivial lemma.

let m be the least power of 2 not less than $\frac{cn}{\epsilon}\log\frac{1}{\epsilon}$, where c is as in Lemma 3;

draw $\vec{x}_1, ..., \vec{x}_m \in I\!R^n$ independently at random from D;

$S := \{\vec{x}_1, ..., \vec{x}_m\}$;

$v := \frac{\epsilon}{4}\left(\frac{2}{em}\right)^n$;

convert algorithm MTV (from Lemma 2) for learning $\mathrm{HALF}_{S,v}$ into a PAC algorithm A' using the transformation of Lemma 1;

simulate A' using the uniform distribution on $(\vec{x}_1, ..., \vec{x}_m)$ to, with probability at least $1 - \epsilon/4$, find a halfspace $\hat{H}$ such that $\frac{1}{m}|\{j : \vec{x}_j \in \hat{H}\Delta H\}| \leq \epsilon/2$;

output $\hat{H}$;

Figure 1: Algorithm A from Theorem 6

Lemma 5 *Choose a set X, and a probability distribution D over X. Suppose $E_1, ..., E_r$ form a partition for X, and for each $x \in X$, i_x is the index of the element of the partition containing x. Then for all $c \geq 0$,*

$$\mathbf{Pr}_{x\in D}(\mathbf{Pr}(E_{i_x}) \leq c) \leq cr.$$

We put these together to prove the main result of this paper.

Theorem 6 *If, for each integer $n \geq 2$, $\mathcal{P}_n$ is the distribution over $HALF_n$ obtained by choosing the normal vector $\vec{w}$ uniformly from the unit ball, then there is an algorithm A that PAC learns HALF with respect to $\mathcal{P} = \{\mathcal{P}_n\}$ in the NMD model in time*

$$O\left(\frac{n^2}{\epsilon}\log^2\frac{n}{\epsilon} + n^{3.38}\log\frac{n}{\epsilon}\right).$$

Proof: Choose n. Consider the algorithm A described in Figure 1.

Fix S for the moment. Then since the volume of the unit ball is more than $2^n/n!$,

$$\mathbf{Pr}_{\vec{w}}(\mathrm{volume}(E_{S,\vec{w}}) \leq v) \leq \mathbf{Pr}_{\vec{w}}\left(\mathbf{Pr}_{\vec{u}}(\vec{u} \in E_{S,\vec{w}}) \leq \frac{n!v}{2^n}\right).$$

Combining Lemma 4 and Lemma 5 with the above, we get

$$\mathbf{Pr}_{\vec{w}}(\mathrm{volume}(E_{S,\vec{w}}) \leq v) \leq \frac{n!}{2^n}v\left(\frac{em}{n}\right)^n \leq \left(\frac{em}{2}\right)^n v.$$

By the choice of v, $\mathbf{Pr}_{\vec{w}}(\mathrm{volume}(E_{S,\vec{w}}) \leq v) \leq \epsilon/4$. By Fubini's Theorem (see [7, volume 2, page 120]), since this holds for arbitrary S, it holds for randomly chosen S as well. Thus, the probability with respect to the random choice of $\vec{x}_1, ..., \vec{x}_m$ and of the target H that $H \cap \{\vec{x}_1, ..., \vec{x}_m\}$ is not in $\mathrm{HALF}_{\{\vec{x}_1,...,\vec{x}_m\},v}$ is at most $\epsilon/4$. By construction, this implies that the probability that $\hat{H}$ does not satisfy $\frac{1}{m}|\{j : \vec{x}_j \in \hat{H}\Delta H\}| \leq \epsilon/2$ is at most $\epsilon/2$. Applying Lemma 3 completes the proof of the correctness of the algorithm.

To analyze the time used by the algorithm, it is tempting to simply plug into the time bounds of Lemma 1. However, in that lemma, it is assumed that the algorithm may obtain random examples

in unit time. However, when we simulate A', we must generate random examples for it. Neverless, one may trivially sample from the uniform distribution on $\{1, ..., m\}$ in $O(\log m)$ time, and therefore the total time spent sampling is at most

$$O\left(\left(\frac{\log m}{\epsilon}\right)\left(n \log n + \log \frac{1}{v} + \log \frac{1}{\epsilon}\right)\right)$$

by the sample size bound of Lemma 1, together with Lemma 2. Substituting and simplifying yields that this time is

$$O\left(\frac{n \log^2 \frac{n}{\epsilon}}{\epsilon}\right). \tag{1}$$

To bound the time spent otherwise, we plug into Lemma 1 together with Lemma 2, getting

$$O\left(\left(n \log n + \log \frac{1}{v}\right)\left(n^{2.38} + \frac{n \ln \frac{n \log n + \log \frac{1}{v}}{\epsilon}}{\epsilon}\right)\right).$$

Expanding the definitions of v and m, simplifying, and observing that the resulting quantity dominates (1) completes the proof. □

4 Acknowledgements

We especially thank Nicolò Cesa-Bianchi, Yoav Freund, Hans Hagauer, David Haussler and Manfred Opper.

References

[1] D. Angluin. Queries and concept learning. *Machine Learning*, 2:319–342, 1988.

[2] E. Baum. The perceptron algorithm is fast for nonmalicious distributions. *Neural Computation*, 2:248–260, 1990.

[3] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *JACM*, 36(4):929–965, 1989.

[4] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Proceedings of the 19th ACM Symposium on the Theory of Computation*, 1987.

[5] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* Wiley, 1973.

[6] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer Verlag, 1984.

[7] W. Feller. *An Introduction to Probability and its Applications*, volume 1. John Wiley and Sons, third edition, 1968.

[8] D. Haussler, M. Kearns, N. Littlestone, and M. Warmuth. Equivalence of models for polynomial learnability. *Information and Computation*, 95:129–161, 1991.

[9] D. Haussler, M. Kearns, and R. Schapire. Bounds on the sample complexity of bayesian learning using information theory and the VC-dimension. *The 1991 Workshop on Computational Learning Theory*, pages 61–74, 1991.

[10] M. Kearns, M. Li, L. Pitt, and L. Valiant. Recent results on boolean concept learning. In P. Langley, editor, *Proceedings of the Fourth International Workshop on Machine Learning*, pages 337–352, Irvine, California, June 1987. (published by Morgan Kaufmann, Los Altos California).

[11] N. Littlestone. From on-line to batch learning. *The 1989 Workshop on Computational Learning Theory*, pages 269–284, 1989.

[12] W. Maass and G. Turán. How fast can a threshold gate learn? Technical Report 321, Graz Technical University, 1991. Previous versions appeared in FOCS89 and FOCS90.

[13] W. S. McCulloch and W. Pitts. A logical calculus of ideas imminent in neural nets. *Bulletin of Mathematical Biophysics*, 5:115–137, 1943.

[14] M. Opper and D. Haussler. Calculation of the learning curve of Bayes optimal classification algorithm for learning a perceptron with noise. In *Computational Learning Theory: Proceedings of the Fourth Annual Workshop*, pages 75–87. Morgan Kaufmann, 1991.

[15] F. Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan Books, Washington, D. C., 1962.

[16] N. Sauer. On the density of families of sets. *J. Combinatorial Theory (A)*, 13:145–147, 1972.

[17] R. Tarjan. *Data Structures and Network Algorithms*. SIAM, 1983.

[18] P. Vaidya. A new algorithm for minimizing convex functions over convex sets. *Proceedings of the 30th Annual Symposium on the Foundations of Computer Science*, pages 338–343, 1989.

[19] L. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.