2003

On Approximating Weighted Sums with Exponentially Many Terms

Deepak Chawla
University of Nebraska-Lincoln, dchawla@dg-rtp.dg.com

Lin Li
University of Nebraska-Lincoln, lili@cse.unl.edu

Stephen Scott
University of Nebraska-Lincoln, sscott2@unl.edu

On Approximating Weighted Sums with Exponentially Many Terms[*]

Deepak Chawla[†], Lin Li, and Stephen Scott
Dept. of Computer Science
University of Nebraska
Lincoln, NE 68588-0115
dchawla@dg-rtp.dg.com; {lili, sscott}@cse.unl.edu

February 2, 2003
Technical Report #TR-UNL-CSE-2003-1

Abstract

Multiplicative weight-update algorithms such as Winnow and Weighted Majority have been studied extensively due to their on-line mistake bounds' logarithmic dependence on N, the total number of inputs, which allows them to be applied to problems where N is exponential. However, a large N requires techniques to efficiently compute the weighted sums of inputs to these algorithms. In special cases, the weighted sum can be exactly computed efficiently, but for numerous problems such an approach seems infeasible. Thus we explore applications of Markov chain Monte Carlo (MCMC) methods to estimate the total weight. Our methods are very general and applicable to any representation of a learning problem for which the inputs to a linear learning algorithm can be represented as states in a completely connected, untruncated Markov chain. We give theoretical worst-case guarantees on our technique and then apply it to two problems: learning DNF formulas using Winnow, and pruning classifier ensembles using Weighted Majority. We then present empirical results on simulated data indicating that in practice, the time complexity is much better than what is implied by our worst-case theoretical analysis.

Keywords: Markov chain Monte Carlo approximation, Winnow, Weighted Majority, multiplicative weight updates, DNF learning, boosting

[*] A preliminary version [6] of this paper appeared in COLT 2001.
[†] Now at EMC Corporation, Raleigh-Durham, NC.

1 Introduction

Multiplicative weight-update algorithms (e.g. [19, 22, 5]) have been studied extensively due to their on-line mistake bounds' logarithmic dependence on N, the total number of inputs. (These bounds can be translated into PAC sample complexity bounds via a simple procedure [20].) This *attribute efficiency* allows them to be applied to problems where N is exponential in the input size, which is the case in many applications, including using Winnow [19] to learn DNF formulas in unrestricted domains and using the Weighted Majority algorithm (WM [22]) to predict nearly as well as the best pruning of a classifier ensemble (from e.g. boosting). However, a large N requires techniques to efficiently compute the weighted sums of inputs to WM and Winnow. One method of doing this is to exploit commonalities among the inputs, partitioning them into a polynomial number of *groups* such that given a single member of each group, the total weight contribution of that group can be efficiently computed [23, 11, 12, 13, 28, 35]. But many WM and Winnow applications do not appear to exhibit such structure, so it seems that a brute-force implementation is the only option to guarantee complete correctness[1]. Thus we explore applications of Markov chain Monte Carlo (MCMC) methods to estimate the total weight without the need for special structure in the problem. Our methods are very general and applicable to any representation of a learning problem for which the inputs to the linear learning algorithm can be represented as states in a completely connected, untruncated[2] Markov chain. In this paper we apply our results to two such problems, described below.

First we study learning DNF formulas (e.g. [3]) using Winnow[3] [19] and not using membership queries. We implicitly enumerate all possible DNF terms and use Winnow to learn a monotone disjunction over these terms, which it can do while making $O(k \log N)$ prediction mistakes, where k is the number of relevant terms and N is the total number of terms. So a brute-force implementation of Winnow makes a polynomial number of errors on arbitrary examples (i.e. with no distributional assumptions) and does not require membership queries. However, a brute-force implementation requires exponential time to compute the weighted sum of the inputs. So we apply our MCMC-based results to estimate this sum.

Next we investigate pruning a classifier ensemble (from e.g. boosting), which can reduce overfitting and time for evaluation [24, 36]. We use the Weighted Majority algorithm (WM) [22], using all possible prunings as experts. WM is guaranteed to not make many more prediction mistakes than the best expert, so we know that a brute-force WM will perform nearly as well as the best pruning. However, the exponential number of prunings motivates us to use an MCMC approach to approximate the weighted sum of the experts' predictions.

MCMC methods [14] have been applied to problems in approximate summation, where the goal is to approximate $W = \sum_{x \in \Omega} w(x)$, where w is a positive function and Ω is a finite set of combinatorial structures. It involves defining an ergodic Markov chain $\mathcal{M}$ with state space Ω and

[1] For additive weight-update algorithms such as the Perceptron algorithm, kernels can often be used to exactly compute the weighted sums (e.g. [7, 33, 27]), though a kernel function might not exist for the desired mapping of features.

[2] We believe that the untruncated requirement can be removed by generalizing results of Morris and Sinclair [26] (see Section 4.2).

[3] We also study the application of our methods to learning DNF via Rosenblatt's Perceptron [30] algorithm, though this is done only for contrast with Winnow since exact sums for DNF learning via Perceptron can be computed with kernels [16].

stationary distribution π. Then we repeatedly simulate $\mathcal{M}$ to draw samples almost according to π. Under appropriate conditions, these samples yield accuracy guarantees. E.g. in approximate summation, sometimes one can guarantee that the estimate of the sum is within a factor ϵ of the true value (w.h.p.). When this is true and the estimation algorithm requires only polynomial time, the algorithm is called a *fully polynomial randomized approximation scheme* (FPRAS). In certain cases a similar argument can be made about combinatorial optimization problems, i.e. that the algorithm's solution is within a factor of ϵ of the true maximum or minimum. In this paper we combine two approximators for application to WM and Winnow. The first is for the *approximate knapsack problem* [26, 8], where given a positive real vector $\vec{w}$ and real number b, the goal is to estimate $|\Omega|$ within a multiplicative factor of ϵ, where $\Omega = \{\vec{p} \in \{0,1\}^n : \vec{w} \cdot p(\vec{x}) \leq b\}$. The other is for computing the sum of the weights of a *weighted matching* of a graph: for a graph G and $\lambda \geq 0$, approximate $Z_G(\lambda) = \sum_{k=0}^{n} m_k \lambda^k$, where m_k is the number of matchings in G of size k and n is the number of nodes. This problem has applications to the *monomer-dimer problem* of statistical physics, and has a FPRAS [14].

While we have thoroughly analyzed our approach in the context of these two problems, our results do not guarantee efficient algorithms for learning DNF and for finding the best pruning[4]. But we do provide theoretical machinery that could potentially be applied to analyze algorithms that learn e.g. restricted cases of DNF, including subclasses of DNF formulas and/or specific distributions over examples. Further, our experimental results provide interesting insights into the algorithms' behaviors and show that the weighted sums can be approximated well despite the pessimistic worst-case bounds. Couple this with the fact that good approximations of the weighted sums are not always necessary to accurately simulate Winnow and WM (since we are only interested in the predictions made based on these weighted sums, not the sums themselves), and our results have potential to be effective tools in theory and in practice. These promising results have inspired more research to further study the efficacy of our algorithms both theoretically and empirically.

The rest of this paper is organized as follows. In Section 2 we give background on the on-line learning model and summarize related work in learning DNF formulas, pruning ensembles of classifiers, and MCMC methods. Section 3 presents our algorithm and Markov chain, and proves general bounds on the accuracy and time complexity of our estimation procedure. In Section 4 we apply these results to the problems of learning DNF formulas with Winnow and Perceptron and pruning ensembles with weighted majority. Then some empirical results appear in Section 5. Finally, we conclude in Section 6 with a description of future and ongoing work.

2 Related work

2.1 The on-line learning model

In this paper we consider *on-line* learning algorithms, where learning proceeds in a series of *trials*[5]. In trial t, an example $\vec{X}_t$ is presented to the learning algorithm A, which makes a prediction[6] $\hat{\ell}_t$

[4] This is not surprising since is unlikely that an efficient distribution-free DNF-learning algorithm exists [2, 1].

[5] In Sections 3 and 4, our results focus on only the current trial, so we omit the subscript t unless it is not clear from context.

[6] In this paper we assume $\ell_t, \hat{\ell}_t \in \{-1, +1\}$.

of $\vec{X}_t$'s label. After this prediction is made, A is told the true label ℓ_t, which A uses to update its hypothesis before making future predictions. If $\ell_t \neq \hat{\ell}_t$, we say that A made a *prediction mistake*. If M is the set of examples for which a mistake is made, the goal is to minimize $|M|$ on any sequence of adversarily-generated examples $\mathcal{X} = (\vec{X}_1, \ldots, \vec{X}_\tau)$. Below we overview the on-line learning algorithms Winnow [19], Perceptron [30], and Weighted Majority (WM) [22].

Winnow maintains a weight vector $\vec{w} \in \Re^{+N}$ (N-dimensional positive real space). Upon receiving an instance $\vec{X}_t \in [0,1]^N$, Winnow makes its prediction $\hat{\ell}_t = +1$ if $W_t = \vec{w}_t \cdot \vec{X}_t \geq \theta$ and -1 otherwise ($\theta > 0$ is a threshold). Given the true label ℓ_t, the weights are updated as follows: $w_{t+1,i} = w_{t,i}\, \alpha^{X_i\left(\ell_t - \hat{\ell}_t\right)/2}$ for some $\alpha > 1$. If $w_{t+1,i} > w_{t,i}$ we call the update a *promotion* and if $w_{t+1,i} < w_{t,i}$ we call it a *demotion*. Littlestone [19] showed that if each example is labeled by some monotone disjunction of K of its N inputs, then Winnow will never make more than $O(K \log N)$ prediction mistakes on any sequence of examples. This makes Winnow a natural tool to apply to learning DNF since by enumerating all 3^n possible terms as inputs to Winnow, K-term DNF can be learned while making only $O(Kn)$ prediction mistakes. However, the time complexity of running Winnow this way is exponential in n.

Similar to Winnow, the Perceptron algorithm maintains a weight vector $\vec{w} \in \Re^N$. Upon receiving an instance $\vec{X}_t \in [0,1]^N$, it makes its prediction $\hat{\ell}_t = +1$ if $W_t = \vec{w}_t \cdot \vec{X}_t \geq \theta$ and -1 otherwise[7]. Given the true label ℓ_t, the weights are updated as follows: $w_{t+1,i} = w_{t,i} + \alpha X_{t,i} \left(\ell_t - \hat{\ell}_t\right)/2$ for some $\alpha > 0$. In contrast to Winnow, the Perceptron algorithm can be forced to make $\Omega(KN)$ mistakes on monotone K-disjunctions over N inputs [18], making it inappropriate for learning DNF (see also Khardon et al. [16]). However, the additive nature of the weight updates allows us to get much better bounds for applying MCMC methods (Section 4.1.2).

Inputs to the Weighted Majority algorithm [22] are themselves predictions of "experts" on the current example[8] $\vec{x}_t$. Each such expert e_i in the pool has its own weight w_i (initialized to 1), and when a new example $\vec{x}_t$ is given to each expert in the pool, expert e_i sends to WM its prediction $X_{t,i} = e_i(\vec{x}_t) \in \Re$, where the sign of $X_{t,i}$ indicates e_i's predicted label and $|X_{t,i}|$ can be thought of as e_i's confidence (though some experts may restrict themselves to predictions from $\{-1, +1\}$). WM then takes a weighted combination of the predictions and predicts $\hat{\ell}_t = +1$ if $W_t = \vec{w}_t \cdot \vec{X}_t \geq 0$ and -1 otherwise. Upon receiving the correct label ℓ_t, if WM makes a prediction mistake, it reduces the weights of all experts that predicted incorrectly by dividing them by some constant $\alpha > 1$. It has been shown that if the best expert in the pool makes at most ν mistakes, then WM has a mistake bound[9] of $O(\nu + \log N)$. Applying this to predicting nearly as well as the best pruning of an ensemble is straightforward. By placing each possible pruning into the pool, we get a pool size of $N = 2^n$ and thus a mistake bound of $O(\nu + n)$. However, the time complexity of a straightforward implementation of this algorithm is exponential in n.

[7] For additive weight update algorithms like Perceptron, often the threshold is included in the weight vector as $w_{t,0}$, corresponding to an extra attribute $X_{t,0} = 1$. The dot product is then compared to 0 rather than θ.

[8] Throughout this paper, lower case $\vec{x}$ and $\vec{y}$ will represent examples in the original space, while capital $\vec{X}$ and $\vec{Y}$ represent the examples mapped to a new space, which is the input space of Winnow, Perceptron, and WM.

[9] Note that stronger results on predicting with expert advice were given by Cesa-Bianchi et al. [5] using a more complex algorithm, but these are only better than WM's by a constant factor. Thus for simplicity, we use WM.

2.2 Learning DNF formulas

Learning DNF formulas has been heavily studied, but positive learning-theoretic results exist only in restricted cases, including assuming a uniform distribution over examples (e.g. [3]) or assuming that the number of terms is bounded by $O(\log n)$, where n is the number of variables [4]. In both of these cases, the algorithms require, in addition to labeled examples, *membership queries*, i.e. they need to be able to present arbitrary examples to an oracle and be told their labels.

In contrast, directly applying Winnow to this problem by enumerating all possible terms and learning a monotone disjunction over them does not require any restrictions or the use of membership queries. Since there are only 3^n possible DNF terms over n variables, Winnow's mistake bound on this problem is $O(Kn)$, where K is the number of relevant terms in the target function. However, the time complexity to make a prediction on each example is exponential in this case if a brute-force approach is taken. Indeed, Khardon et al. [16] showed that if P $\neq$ #P, then there is no polynomial time algorithm to exactly simulate Winnow over exponentially many conjunctive features for learning even monotone DNF. Further, while they did provide a kernel allowing them to exactly compute Perceptron's weighted sums when learning DNF, they also gave an exponential lower bound on the number of mistakes that kernel perceptron makes in learning DNF: $2^{\Omega(n)}$.

2.3 Pruning ensembles of classifiers

Prior work in pruning ensembles of classifiers [24, 36] (produced by boosting) has been conducted for two reasons. First, the time required to evaluate a complete ensemble is prohibitive in some applications. Second, despite some evidence to the contrary [31, 29], boosting can be prone to overfitting. The methods of Margineantu and Dietterich [24] and Tamon and Xiang [36] not only sought subsets of the ensemble with high prediction accuracy, but also with high *diversity*, i.e. hypotheses with high accuracy on different portions of the instance space. The approaches they used included simple ones like early stopping, ones that utilized divergence measures such as Kullback-Leibler divergence or the κ statistic, and methods that used prediction error, sometimes combined with a divergence measure.

To address the concern of overfitting, one can use the WM algorithm, using all possible prunings as "experts" in a pool. Since WM is guaranteed to not perform much worse (in terms of number of on-line prediction mistakes) than the best expert in the pool, we know that a brute-force implementation of this algorithm is guaranteed to not perform much worse than the best pruning. However, a brute-force implementation of WM would take time exponential in the number of hypotheses in the ensemble. So we use an MCMC approach to approximate the weighted sum of the experts' predictions.

2.4 Markov chain Monte Carlo methods

MCMC methods [14] have been applied to problems in combinatorial optimization and approximate summation, where the goal is to approximate weighted sum $W = \sum_{x \in \Omega} w(x)$, where w is a positive function defined on Ω and Ω is a very large, finite set of combinatorial structures. The process involves defining an ergodic Markov chain $\mathcal{M}$ with state space Ω and stationary distribution π. Then we repeatedly simulate $\mathcal{M}$ some number of steps to draw several samples (almost) according to π. Under appropriate conditions, these samples yield accuracy guarantees, e.g. in approximate summation, sometimes one can guarantee that the estimate of the sum is within a factor ϵ of the true value (w.h.p.). When this is true and the estimation algorithm requires only polynomial time, the algorithm is called a *fully polynomial randomized approximation scheme* (FPRAS). In certain cases a similar argument can be made about combinatorial optimization problems, i.e. that the algorithm's solution is within a factor of ϵ of the true maximum or minimum.

A well-studied problem with an MCMC solution is the *approximate knapsack problem*, where one is given a positive real vector $\vec{w}$ and real number b. The goal is to estimate $|\Omega|$ within a multiplicative factor of ϵ, where $\Omega = \{\vec{p} \in \{0,1\}^n : \vec{w} \cdot \vec{p} \leq b\}$. Dyer et al. [8] gave a Markov chain for this problem and argued that a polynomial (in n and $1/\epsilon$) number of samples from it were sufficient to accurately estimate $|\Omega|$. Later, Morris and Sinclair [26] showed that it is sufficient to simulate the chain for a polynomial number of steps to obtain each sample (i.e. that the chain is *rapidly mixing*), thus giving a FPRAS for the knapsack problem.

Another problem with a FPRAS [14] is computing the sum of the weights of a *weighted matching* with parameter λ. For a graph G and $\lambda \geq 0$, approximate $Z_G(\lambda) = \sum_{k=0}^{n} m_k \lambda^k$, where m_k is the number of matchings in G of size k and n is the number of nodes. This problem has applications to the monomer-dimer problem of statistical physics. In the next section, we show how to combine the knapsack solution with the matching solution to approximate the weighted sums of inputs of linear learning algorithms.

3 Our general algorithm and Markov chain

For a state $\vec{p} \in \Omega$ and an input example $\vec{x}$, let $p(\vec{x})$ denote $\vec{p}$ evaluated at $\vec{x}$ (e.g. when learning DNF with binary $p(\vec{x})$, $\vec{p}$ is a term, $\vec{x}$ is an assignment to the variables, and $p(\vec{x}) = 1$ if $\vec{x}$ satisfies $\vec{p}$ and 0 otherwise). Now define $\mathcal{M}$ as a Markov chain with state space Ω that makes transitions from state $\vec{p} \in \Omega$ to state $\vec{q} \in \Omega$ by the following rules.

1. With probability $1/2$ let $\vec{q} = \vec{p}$. Otherwise:
2. Let $\vec{p}^{\,'}$ be a neighbor of $\vec{p}$ selected uniformly at random
3. If $\vec{p}^{\,'} \in \Omega$, then let $\vec{p}^{\,''} = \vec{p}^{\,'}$, else let $\vec{p}^{\,''} = \vec{p}$.
4. With probability $\min\left\{1, p^{''}(\vec{x})\, w_{\vec{p}^{\,''}} / \left(p(\vec{x})\, w_{\vec{p}}\right)\right\}$, let $\vec{q} = \vec{p}^{\,''}$, else let $\vec{q} = \vec{p}$. Here $w_{\vec{p}}$ is the weight of node $\vec{p}$ in the learning algorithm.

Thus $\mathcal{M}$ is a random walk where the transition probabilities favor nodes with higher weights. In our applications, Ω will take on many forms depending on the application and the choice of $p(\vec{x})$. Such forms include a boolean hypercube truncated by a hyperplane, an untruncated boolean hypercube (in which case Step 3 is unnecessary), and a non-boolean hypercube.

Lemma 1 *If every state in Ω can be reached from every other state, then $\mathcal{M}$ is ergodic with stationary distribution*

$$\pi_t(\vec{p}) = \frac{p(\vec{x}_t)\, w_{\vec{p}}}{W_t} \quad ,$$

where $W_t = \sum_{\vec{p}\in\Omega} p(\vec{x}_t)\, w_{\vec{p}}$, i.e. the weighted sum of inputs over all states (inputs) in Ω when example $\vec{x}_t$ is the current example.

Proof: Since all states in Ω can communicate, $\mathcal{M}$ is irreducible. Also, the self-loop of step 1 ensures aperiodicity. Finally, $\mathcal{M}$ is reversible since the transition probabilities

$$P(\vec{p},\vec{q}) = \frac{\min\{1, q(\vec{x}_t)\, w_{\vec{q}} / \left(p(\vec{x}_t)\, w_{\vec{p}}\right)\}}{2n} = \frac{\min\{1, \pi_t(\vec{q})/\pi_t(\vec{p})\}}{2n}$$

(here n is the number of neighbors) satisfy the detailed balance condition $\pi_t(\vec{p})P(\vec{p},\vec{q}) = \pi_t(\vec{q})P(\vec{q},\vec{p})$. So $\mathcal{M}$ is ergodic with the stated stationary distribution. □

For each new trial in an on-line algorithm, the weighted sums we estimate are potentially different. Thus we must conduct a new estimation procedure (with a new Markov chain) for each trial. To simplify notation, for the rest of this paper we will let the index t of each trial be implicit, omitting any subscripts unless necessary. Further, in each trial our algorithm defines multiple Markov chains, each assuming that the weight updates of previous trials were made using a different[10] learning rate α_i. Hence the weight $w_{\vec{p}}$ of a node $\vec{p}$ (and hence the stationary distribution π and the sum of weights W) will be functions of both α_i and t, so we use the subscript of α_i to denote these differences, leaving the t implicit.

Let B be a bound on the sum of the magnitude of the updates on all the weights up to the current trial, i.e. $B \geq \sum_{\vec{x}\in M} |Z_{\vec{x}}|$, where M is the set of all examples for which an update was made before the current trial, and $|Z_{\vec{x}}|$ is the maximum absolute value of any update made for any input in response to example $\vec{x}$. (E.g. in Winnow, $B =$ the sum of all promotions and demotions made on all examples for which a prediction mistake was made up to the current trial.) Now let r be the smallest integer s.t. $(1 + 1/B)^{r-1} \geq \alpha$ and $r \geq 1 + \log_2 \alpha$ (so $r \leq 2 + B \ln \alpha$). Also, let $\zeta = 1/\left(\alpha^{1/(r-1)} - 1\right) \geq B$ and $\alpha_i = (1 + 1/\zeta)^{i-1} = \alpha^{(i-1)/(r-1)}$ for $1 \leq i \leq r$ (so $\alpha_r = \alpha$).

Now define $f_i(\vec{p}) = w_{\alpha_{i-1},\vec{p}}/w_{\alpha_i,\vec{p}}$, where $\vec{p}$ is chosen according to π_{α_i}. Then

$$\mathrm{E}\left[f_i\right] = \sum_{\vec{p}\in\Omega} \left(\frac{w_{\alpha_{i-1},\vec{p}}}{w_{\alpha_i,\vec{p}}}\right) \frac{p(\vec{x})\, w_{\alpha_i,\vec{p}}}{W(\alpha_i)} = \frac{W(\alpha_{i-1})}{W(\alpha_i)}.$$

So we can estimate $W(\alpha_{i-1})/W(\alpha_i)$ by sampling states $\vec{p}$ from $\mathcal{M}$ and computing the sample mean of the $f_i(\vec{p})$. Note that

$$W(\alpha) = \left(\frac{W(\alpha_r)}{W(\alpha_{r-1})}\right)\left(\frac{W(\alpha_{r-1})}{W(\alpha_{r-2})}\right)\cdots\left(\frac{W(\alpha_2)}{W(\alpha_1)}\right) W(\alpha_1),$$

So for each value $\alpha_2, \ldots, \alpha_r$, we run S independent simulations of $\mathcal{M}$, and let X_i be the sample mean of $w_{\alpha_{i-1}}/w_{\alpha_i}$. Then our estimate[11] is

$$\hat{W}(\alpha) = W(\alpha_1) \prod_{i=2}^{r} 1/X_i \ . \qquad (1)$$

[10] Note that, however, the actual sequence of updates made will be the same regardless of α_i. This sequence of updates is determined by running the learning algorithm with the original learning rate α.

[11] When we apply our results to the Perceptron algorithm in Section 4.1.2, we will also use $\alpha_0 = 0$ and update the product of ratios accordingly.

In order to complete our computation of $\hat{W}$, we must also estimate $W(\alpha_1)$. Due to the definition of α_1, this is straightforward. For learning DNF using Winnow and binary $p(\vec{x})$, we set $\alpha_1 = 1$, and $W(\alpha_1) = |\Omega| = 2^n$ (see Section 4.1.1). For the ensemble pruning problem, $W^+(\alpha_1) = |\Omega^+|$, where Ω^+ is the set of prunings that predict $+1$ on the input $\vec{x}$. This cannot be efficiently computed exactly, so we must estimate it with the FPRAS of Morris and Sinclair [26] (Section 4.2).

The following theorem bounds the error of our algorithm's estimates of W. The theorem is based on *variation distance*, which is a distance measure between a Markov chain's simulated and stationary distributions, defined as $\max_{U \subseteq \Omega} |P^\tau(\vec{p}, U) - \pi(U)|$, where $P^\tau(\vec{p}, \cdot)$ is the distribution of a chain's state at simulation step τ given that the simulation started in state $\vec{p} \in \Omega$, and π is the chain's stationary distribution.

Theorem 2 *Assume $a \leq f_i, \hat{f}_i \leq b$ for all i, where $\hat{f}_i$ is the same as f_i but with samples drawn according to the distribution yielded by simulating $\mathcal{M}$. Let the sample size $S = \lceil 130\, r\, b/(a\epsilon^2) \rceil$ and $\mathcal{M}$ be simulated long enough for each sample such that the variation distance between the empirical distribution and π_{α_i} is $\leq \epsilon a/(5br)$ for each i. Also, assume that $W(\alpha_1)$ can be computed exactly. Then $\hat{W}(\alpha)$ satisfies*

$$\Pr\left[(1-\epsilon)\, W(\alpha) \leq \hat{W}(\alpha) \leq (1+\epsilon)\, W(\alpha)\right] \geq 3/4 \ .$$

In addition, the $3/4$ can be made arbitrarily close to $1 - \delta$ for any $\delta > 0$.

Proof: Let the distribution $\hat{\pi}_{\alpha_i}$ be the one resulting from simulating $\mathcal{M}$, and assume that the variation distance $\|\hat{\pi}_{\alpha_i} - \pi_{\alpha_i}\| \leq \epsilon a/(5br)$. Now consider the random variable $\hat{f}_i$, which is the same as f_i except that the terms are selected according to $\hat{\pi}_{\alpha_i}$. Since $\hat{f}_i \in [a, b]$, $\left|\mathrm{E}\left[\hat{f}_i\right] - \mathrm{E}\left[f_i\right]\right| \leq \epsilon a/(5r)$, which implies $\mathrm{E}\left[f_i\right] - \epsilon a/(5r) \leq \mathrm{E}\left[\hat{f}_i\right] \leq \mathrm{E}\left[f_i\right] + \epsilon a/(5r)$. Factoring out $\mathrm{E}\left[f_i\right]$ from both sides and noting that $1/\,\mathrm{E}\left[f_i\right] \leq 1/a$ yields

$$\left(1 - \frac{\epsilon}{5r}\right) \mathrm{E}\left[f_i\right] \leq \mathrm{E}\left[\hat{f}_i\right] \leq \left(1 + \frac{\epsilon}{5r}\right) \mathrm{E}\left[f_i\right] \ . \tag{2}$$

This allows us to conclude that $\mathrm{E}\left[\hat{f}_i\right] \geq \mathrm{E}\left[f_i\right]/2$. Since $\hat{f}_i \leq b$, we get $\mathrm{Var}\left[\hat{f}_i\right] \leq b\, \mathrm{E}\left[\hat{f}_i\right]$, yielding

$$\frac{\mathrm{Var}\,[\hat{f}_i]}{\left(\mathrm{E}\,[\hat{f}_i]\right)^2} \leq \frac{b}{\mathrm{E}\,[\hat{f}_i]} \leq \frac{2b}{\mathrm{E}\,[f_i]} \leq 2b/a \ . \tag{3}$$

Let $X_i^{(1)}, \ldots, X_i^{(S)}$ be a sequence of S independent copies of $\hat{f}_i$, and let $\bar{X}_i = \left(\sum_{j=1}^{S} X_i^{(j)}\right)/S$. Then $\mathrm{E}\left[\bar{X}_i\right] = \mathrm{E}\left[\hat{f}_i\right]$ and $\mathrm{Var}\left[\bar{X}_i\right] = \mathrm{Var}\left[\hat{f}_i\right]/S$. The estimator of $W(\alpha)$ is $W(\alpha_1)/X = W(\alpha_1)/\prod_{i=2}^{r} \bar{X}_i$. Since the $\bar{X}_i$'s are independent, $\mathrm{E}\left[X\right] = \prod_{i=2}^{r} \mathrm{E}\left[\bar{X}_i\right] = \prod_{i=2}^{r} \mathrm{E}\left[\hat{f}_i\right]$ and $\mathrm{E}\left[X^2\right] = \prod_{i=2}^{r} \mathrm{E}\left[\bar{X}_i^2\right]$. Let $\rho = \prod_{i=2}^{r} W(\alpha_{i-1})/W(\alpha_i)$, (i.e. what we are estimating with X) and $\hat{\rho} = \mathrm{E}\left[X\right]$. Then applying Equation (2) gives

$$\left(1 - \frac{\epsilon}{5r}\right)^r \rho \leq \hat{\rho} \leq \rho\, \left(1 + \frac{\epsilon}{5r}\right)^r .$$

Since $\lim_{r\to\infty}(1+\epsilon/(5r))^r = e^{\epsilon/5} \le 1+\epsilon/4$ and $(1-\epsilon/(5r))^r$ is minimized at $r=1$, we get

$$\left(1-\frac{\epsilon}{4}\right)\rho \le \hat{\rho} \le \left(1+\frac{\epsilon}{4}\right)\rho \ .$$

Since $\mathrm{Var}\,[X] = \mathrm{E}\,[X^2] - (\mathrm{E}\,[X])^2$, we have

$$\begin{aligned}
\frac{\mathrm{Var}\,[X]}{(\mathrm{E}\,[X])^2} &= \prod_{i=1}^{r-1}\left(1+\frac{\mathrm{Var}\left[\bar{X}_i\right]}{\left(\mathrm{E}\left[\bar{X}_i\right]\right)^2}\right)-1 \\
&\le \left(1+\frac{2b}{aS}\right)^{r-1}-1 \qquad \text{(By Equation (3))} \\
&\le \left(1+\frac{\epsilon^2}{65r}\right)^{r}-1 \le \exp\left(\epsilon^2/65\right)-1 \le \epsilon^2/64 \ .
\end{aligned}$$

The last inequality holds since $\exp(x/65) \le 1+1/64$ for $x\in[0,1]$. We now apply Chebyshev's inequality to X with standard deviation $\epsilon\hat{\rho}/8$:

$$\Pr\left[|X-\hat{\rho}| > \epsilon\hat{\rho}/4\right] \le 1/4 \ .$$

So with probability at least $3/4$ we get

$$\left(1-\frac{\epsilon}{4}\right)\hat{\rho} \le X \le \left(1+\frac{\epsilon}{4}\right)\hat{\rho} \ ,$$

which implies that with probability at least $3/4$

$$\frac{(1+\epsilon)}{\rho} \ge \frac{1}{(1-\epsilon/4)^2\rho} \ge 1/X \ge \frac{1}{(1+\epsilon/4)^2\rho} \ge \frac{(1-\epsilon)}{\rho} \ , \qquad (4)$$

completing the proof of the first part of the theorem. Making the approximation with probability $\ge 1-\delta$ for any $\delta > 0$ is done by rerunning $O(\ln 1/\delta)$ times the procedure for estimating X and taking the median of the results [15]. □

It is also possible to extend Theorem 2 to the case where $W(\alpha_1)$ cannot be exactly computed, but can be accurately estimated.

Corollary 3 *Assume $a \le f_i, \hat{f}_i \le b$ for all i. Let the sample size $S = \lceil 130\, r\, b/(a\epsilon^2)\rceil$, $W(\alpha_1)$'s estimate be within $\epsilon/2$ of its true value with probability $\ge 3/4$, and $\mathcal{M}$ be simulated long enough for each sample such that the variation distance between the empirical distribution and π_{α_i} is $\le \epsilon a/(10br)$ for all i. Then $\hat{W}(\alpha)$ satisfies*

$$\Pr\left[(1-\epsilon)W(\alpha) \le \hat{W}(\alpha) \le (1+\epsilon)W(\alpha)\right] \ge 1/2.$$

In addition, the $1/2$ can be made arbitrarily close to $1-\delta$ for any $\delta > 0$.

Proof: The analysis is the same as in the proof of Theorem 2, except we now must accommodate another source of error. First, substitute $\epsilon/2$ for ϵ in Equation (4). Given the accuracy of $\hat{W}(\alpha_1)$ with probability at least $3/4$, we get

$$\frac{W(\alpha_1)(1-\epsilon/2)^2}{\rho} \le \frac{\hat{W}(\alpha_1)}{X} \le \frac{W(\alpha_1)(1+\epsilon/2)^2}{\rho} \ ,$$

with probability at least $1/2$. This completes the proof (the constants in S remain unchanged). Similar to Theorem 2, both estimates can be run multiple times and the median taken in order to reduce the probability of failure. $\square$

We now bound the mixing time of $\mathcal{M}$ by using the *canonical paths* method [34]. In this method, we treat $\mathcal{M}$ as a directed graph with vertices Ω and edges $E = \{(\vec{p}, \vec{q}) \in \Omega \times \Omega : Q(\vec{p}, \vec{q}) > 0\}$, where $Q(\vec{p}, \vec{q}) = \pi_\alpha(\vec{p})\, P(\vec{p}, \vec{q})$. For each ordered pair $(\vec{p}, \vec{q}) \in \Omega \times \Omega$, we specify a canonical path $\gamma_{\vec{p},\vec{q}} \in \Gamma$ from $\vec{p}$ to $\vec{q}$ in the graph (Ω, E) that corresponds to a set of legal transitions in $\mathcal{M}$ from $\vec{p}$ to $\vec{q}$. We measure how heavily any one edge in E is loaded with canonical paths by

$$\bar{\rho} = \bar{\rho}(\Gamma) = \max_{e \in E} \left\{ \frac{1}{Q(e)} \sum_{\gamma_{\vec{p},\vec{q}} \ni e} \pi_\alpha(\vec{p})\, \pi_\alpha(\vec{q})\, |\gamma_{\vec{p},\vec{q}}| \right\} . \tag{5}$$

We start with a result from Sinclair [34], restated by Jerrum and Sinclair [14].

Theorem 4 *[34, 14] Let $\mathcal{M}$ be a finite, reversible, ergodic Markov chain with loop probabilities $P(\vec{p}, \vec{p}) \geq 1/2$ for all $\vec{p}$. Let Γ be a set of canonical paths with maximum edge loading $\bar{\rho} = \bar{\rho}(\Gamma)$. Then the mixing time of $\mathcal{M}$ satisfies $\tau_{\vec{p}}(\epsilon) \leq \bar{\rho}\,(\ln 1/\pi(\vec{p}) + \ln 1/\epsilon)$ for any choice of initial state $\vec{p}$, i.e. after simulating $\mathcal{M}$ for $\bar{\rho}\,(\ln 1/\pi(\vec{p}) + \ln 1/\epsilon)$ steps starting in $\vec{p}$, the variation distance between $\hat{\pi}_{\alpha_i}$ and π_{α_i} is at most ϵ.*

In general, $\Omega = \prod_{i=1}^{n} \{0, \ldots, k_i\}$ for some integers $k_1, \ldots, k_n$. Without loss of generality we will let $\Omega = \{0, \ldots, k\}^n$ for some positive integer k. Then there is an edge from node $\vec{p} = (p_1, \ldots, p_i, \ldots, p_n)$ to $\vec{p}' = (p_1, \ldots, p_i', \ldots, p_n)$, i.e. an edge exists between each pair of nodes that differ in at most one position (self-loops also exist). For our proof, we assume that the hypercube is untruncated, which is necessary to ensure that no canonical paths leave the chain. However, it is likely that mixing time bounds also exist for truncated hypercubes. Such a bound could probably be derived by the recent work of Morris and Sinclair [26] who give an FPRAS for a truncated boolean hypercube that has a uniform distribution.

Let $\vec{p} = (p_1, \ldots, p_n)$ and $\vec{q} = (q_1, \ldots, q_n)$ be arbitrary states of Ω. The canonical path $\gamma_{\vec{p},\vec{q}}$ consists of n edges, where edge i is $((q_1, \ldots, q_{i-1}, p_i, p_{i+1}, \ldots, p_n), (q_1, \ldots, q_{i-1}, q_i, p_{i+1}, \ldots, p_n))$, i.e. position i is changed from $\vec{p}_i$ to $\vec{q}_i$. So some edges of $\gamma_{\vec{p},\vec{q}}$ might be loops. Now focus on a particular oriented edge

$$e = (\vec{a}, \vec{a}') = ((a_1, \ldots, a_i, \ldots, a_n), (a_1, \ldots, a_i', \ldots, a_n)) \quad .$$

We will now bound Equation (5) for e, which yields a bound on $\bar{\rho}$ and allows us to apply Theorem 4. Let $\mathrm{cp}(e) = \{(\vec{p}, \vec{q}) : \gamma_{\vec{p},\vec{q}} \ni e\}$ be the set of endpoints of canonical paths that use edge e. We use Jerrum and Sinclair's [14] mapping $\vec{\eta}_e : \mathrm{cp}(e) \to \Omega$, defined as follows: if $(\vec{p}, \vec{q}) = ((p_1, \ldots, p_n), (q_1, \ldots, q_n)) \in \mathrm{cp}(e)$, then

$$\vec{\eta}_e(\vec{p}, \vec{q}) = (b_1, \ldots, b_n) = (p_1, \ldots, p_{i-1}, a_i, q_{i+1}, \ldots, q_n) \quad .$$

Note that $\vec{p} = (b_1, \ldots, b_{i-1}, a_i, a_{i+1}, \ldots, a_n)$ and $\vec{q} = (a_1, \ldots, a_{i-1}, a_i', b_{i+1}, \ldots, b_n)$. Since $\vec{p}$ and $\vec{q}$ can be unambiguously recovered from $\vec{\eta}_e(\vec{p}, \vec{q})$, the mapping $\vec{\eta}_e$ is injective.

We are now ready to state the mixing time bound.

Theorem 5 *For all $\vec{p}, \vec{q} \in \Omega$ assume*

$$\pi_\alpha(\vec{p})\,\pi_\alpha(\vec{q}) \leq g\,\pi_\alpha(\vec{a}')\,\pi_\alpha(\vec{\eta}_e(\vec{p},\vec{q}))$$

for some function $g = g(n, K, k, \alpha)$. Also assume that for all neighbors $\vec{a}$ and $\vec{a}'$ in Ω, $\max\{\pi_\alpha(\vec{a})/\pi_\alpha(\vec{a}'), \pi_\alpha(\vec{a}')/\pi_\alpha(\vec{a})\} \leq h = h(n, K, k, \alpha)$. Then a simulation of $\mathcal{M}$ that starts at node $\vec{p}$ and is of length

$$T = 2kn^2\,g\,h\left(\ln\left(\frac{W(\alpha)}{w_{\alpha,\vec{p}}}\right) + \ln\left(1/\epsilon'\right)\right)$$

will draw samples from $\hat{\pi}_\alpha$ such that $\|\hat{\pi}_\alpha - \pi_\alpha\| \leq \epsilon'$.

Proof: Since $Q(e) = \min\left\{\pi_\alpha(\vec{a}), \pi_\alpha(\vec{a}')\right\}/(2kn)$, we get

$$\begin{aligned}
\pi_\alpha(\vec{p})\,\pi_\alpha(\vec{q}) &\leq \frac{2kn\,g\,Q(e)}{\min\left\{\pi_\alpha(\vec{a}), \pi_\alpha(\vec{a}')\right\}}\,\pi_\alpha(\vec{a}')\,\pi_\alpha(\vec{\eta}_e(\vec{p},\vec{q})) \\
&= 2kn\,g\,Q(e)\,\max\left\{1, \pi_\alpha(\vec{a}')/\pi_\alpha(\vec{a})\right\}\pi_\alpha(\vec{\eta}_e(\vec{p},\vec{q})) \\
&\leq 2kn\,g\,h\,Q(e)\,\pi_\alpha(\vec{\eta}_e(\vec{p},\vec{q})).
\end{aligned}$$

Given the above inequality, we can now bound $\bar{\rho}$. Since $|\gamma_{\vec{p},\vec{q}}| = n$, we get

$$\frac{1}{Q(e)}\sum_{\gamma_{\vec{p},\vec{q}}\ni e}\pi_\alpha(\vec{p})\,\pi_\alpha(\vec{q})\,|\gamma_{\vec{p},\vec{q}}| \;\leq\; 2kn^2gh\sum_{\gamma_{\vec{p},\vec{q}}\ni e}\pi_\alpha(\vec{\eta}_e(\vec{p},\vec{q})) \leq 2kn^2gh.$$

The last inequality holds because $\vec{\eta}_e$ is injective and π_α is a probability distribution. Applying Theorem 4 completes the proof. □

Corollary 6 *For Markov chains for which g and h are polynomial in n and K, our algorithm is an FPRAS (we assume k and α are constants).*

4 Example applications

4.1 Learning DNF formulas

We consider generalized DNF representations, where the instance space is $\prod_{i=0}^{n-1}\{1, \ldots, k_i\}$ and the set of terms is $\prod_{i=0}^{n-1}\{0, \ldots, k_i\}$, where k_i is the number of values for feature i. A term $\vec{p} = (p_0, \ldots, p_{n-1})$ is satisfied by example $\vec{x} = (x_0, \ldots, x_{n-1})$ iff $\forall\, p_i > 0$, $p_i = x_i$. So $p_i = 0 \Rightarrow x_i$ is irrelevant for term $\vec{p}$ and $p_i > 0 \Rightarrow x_i$ must equal p_i for $\vec{p}$ to be satisfied.

We present algorithms to learn this concept class that are based on Littlestone's Winnow [19] and Rosenblatt's Perceptron [30] algorithms. The inputs to the linear threshold units learned by these algorithms consist of the entire set of DNF terms over the original set of n inputs. For reasons that will become clear, we look at different versions of the function $p(\vec{x})$, which measures the "degree" to which $\vec{x}$ satisfies $\vec{p}$. These versions include $p(\vec{x})$ being a threshold function, a logistic function, and a linear function.

None of our approaches in this section give complete, efficient solutions to the problem of learning DNF in the on-line model, but they do give new mechanisms that could be refined for potential application to restricted subclasses of learning DNF, e.g. restricted classes of DNF or specific distributions over the examples.

4.1.1 Winnow

Recalling the definition of Winnow in Section 2.1, if the initial weight vector is the all 1s vector, the weight of term $\vec{p}$ is $w_{\vec{p}} = \alpha^{z_{\vec{p}}}$, where $z_{\vec{p}} = \sum_{\vec{x} \in M} \ell_{\vec{x}}\, p(\vec{x})$, M is the set of examples for which a prediction mistake is made and $\ell_{\vec{x}} \in \{-1, +1\}$ is example $\vec{x}$'s label. We now state one of Littlestone's results for Winnow [21], which we will use to bound the number of prediction mistakes it makes for the variations of our algorithm.

Theorem 7 *[21] Let $(\vec{Y}_j, \ell_j) \in [0,1]^N \times \{0,1\}$ for $j = 1, \ldots, t$ (t is the index of the current trial). Suppose that there exist $\vec{\mu} \geq 0$ and $0 < \delta < 1$ such that whenever $\ell_j = 1$ we have $\vec{\mu} \cdot \vec{Y}_j \geq 1$ and whenever $\ell_j = 0$ we have $\vec{\mu} \cdot \vec{Y}_j \leq 1 - \delta$. Now suppose Winnow sees as inputs $\mathcal{X} = (\vec{X}_j, \ell_j)$ where each $X_{ij} \in [0,1]$, and define $\vec{E}_j = (|X_{j1} - Y_{j1}|, \ldots, |X_{jN} - Y_{jN}|)$. Then the number of mistakes made by Winnow on $\mathcal{X}$ with $\alpha = 1 + \delta/2$ and $\theta = N$ is at most*

$$8/\delta^2 + \max\left(0, \frac{14}{\delta^2} \sum_{i=1}^{N} \mu_i \ln\left(\mu_i\, \theta\right)\right) + \frac{4}{\delta} \sum_{j=1}^{t} \vec{\mu} \cdot \vec{E}_j \ .$$

We will examine three versions of our algorithm, differing in the values that Winnow receives as its inputs. In the following, we say that a variable x_i in example $\vec{x}$ *matches* its corresponding variable p_i in term $\vec{p}$ if $p_i = 0$ or $p_i = x_i$. We let $m_{\vec{x},\vec{p}} \in \{0, \ldots, n\}$ denote the number of variables in $\vec{x}$ that match their corresponding variables in $\vec{p}$ (we drop the subscript $\vec{x}$ when it is clear from context).

1. *Binary* $p(\vec{x})$ means that Winnow input $X_{\vec{p}} = 1$ if $m_{\vec{p}} = n$ and 0 otherwise.

2. *Logistic* $p(\vec{x})$ means that Winnow input $X_{\vec{p}}$ is

   $$p(\vec{x}) = \frac{2}{1 + e^{-a(m_{\vec{p}} - n)}} \ , \tag{6}$$

   where a is a positive constant. Thus $p(\vec{x}) \in (0,1]$ and grows as $\vec{p}$ becomes more satisfied by $\vec{x}$ (it equals 1 iff it is completely satisfied).

3. *Linear* $p(\vec{x})$ means that Winnow input $X_{\vec{p}}$ is

   $$p(\vec{x}) = \frac{1 + m_{\vec{p}}}{n + 1} \ . \tag{7}$$

   Thus $p(\vec{x}) \in (0,1]$ and grows as $\vec{p}$ becomes more satisfied by $\vec{x}$ (it equals 1 iff it is completely satisfied).

We defined $p(\vec{x}) > 0$ for all $\vec{x}$ in order to ensure that for a given $\vec{x}$, every term in Ω contributes something to the weighted sum, and the hypercube is untruncated, which allows us to apply Theorem 5. Using binary $p(\vec{x})$ also yields an untruncated hypercube, as explained below.

Before considering these three cases individually, we state some common results for them all. First note that for logistic and linear $p(\vec{x})$, Ω consists of the entire set of possible terms, since each term gives to Winnow a value $p(\vec{x}) > 0$. Thus in the chain defined in Section 3, every state can be reached from every other state. Further, for binary $p(\vec{x})$, we can completely ignore all but the 2^n terms that are satisfied by $\vec{x}$, implying that our state space $\Omega = \{0,1\}^n$, and thus it is also completely connected. Therefore it is obvious that Lemma 1 applies to all our Markov chains. We now discuss the application of Theorem 2. All that is required to apply this result is to bound the range of f and $\hat{f}$. Since f and $\hat{f}$ are independent of $p(\vec{x})$, the same result applies to all three versions of our algorithm.

Lemma 8 *When applying binary, logistic, or linear Winnow to learn DNF, for all* i, $1/e \leq f_i, \hat{f}_i \leq e$.

Proof: First note that the only difference between f_i and $\hat{f}_i$ is the probability distribution that generates the terms that define them, i.e. their ranges are the same. Thus we focus on bounding f_i only. Let $z_{\vec{p}}$ be the net number of promotions of term $\vec{p}$. Then

$$f_i(\vec{p}) = \frac{w_{\alpha_{i-1},\vec{p}}}{w_{\alpha_i,\vec{p}}} = \left(\frac{\alpha_{i-1}}{\alpha_i}\right)^{z_{\vec{p}}} = \left(\frac{(1+1/\zeta)^{i-2}}{(1+1/\zeta)^{i-1}}\right)^{z_{\vec{p}}} = (1+1/\zeta)^{-z_{\vec{p}}} \quad .$$

Recall that from its definition, $\zeta \geq B \geq |z_{\vec{p}}|$ for all $\vec{p}$ (we also assume that $\zeta > 0$, which can be accomplished artificially before the first update by setting $\zeta = 1$). If $z_{\vec{p}} < 0$, then $1 \leq (1+1/\zeta)^{-z_{\vec{p}}} \leq e$. If $z_{\vec{p}} \geq 1$, then $1/e \leq (1+1/\zeta)^{-z_{\vec{p}}} \leq 1$. □

Note that $W(\alpha_1) = W(1)$ is simply $\sum_{\vec{p}\in\Omega} p(\vec{x})$. For the binary case, this is simply the number of terms satisfied by $\vec{x}$, which equals 2^n. For the linear and logistic cases, it can be computed exactly if we assume that $k_i = k$ for all i. Under this assumption, the number of terms that match exactly $i \in \{0,\ldots,n\}$ variables in the example $\vec{x}$ is $2^i\binom{n}{i}(k-1)^{n-i}$, since there are $\binom{n}{i}$ positions to place the matched variables, each matched position p_j can equal 0 or x_j, and each unmatched position $p_{j'}$ can take on any value from $\{1,\ldots,k\} \setminus \{x_{j'}\}$. Thus for the logistic case, we get

$$W(1) = \sum_{i=0}^{n} 2^i \binom{n}{i} \frac{2(k-1)^{n-i}}{1+e^{-a(i-n)}} \quad , \tag{8}$$

and for the linear case, we get

$$W(1) = \sum_{i=0}^{n} 2^i \binom{n}{i} \frac{(i+1)(k-1)^{n-i}}{n+1} = \frac{(k+1)^n + 2n(k+1)^{n-1}}{n+1} \quad . \tag{9}$$

Thus all three can efficiently be computed exactly. By applying this and substituting Lemma 8's bounds into Theorem 2, we get the following.

Corollary 9 *When applying Winnow to learn generalized DNF (with* $k_i = k$ *for all* i *for the logistic and linear cases), let the sample size* $S = \lceil 130\, r\, e^2/\epsilon^2 \rceil$ *and* $\mathcal{M}$ *be simulated long enough for each sample such that the variation distance between the empirical distribution and* $\pi_{\alpha_{i,t}}$ *is* $\leq \epsilon/(5e^2 r)$. *Then* $\hat{W}(\alpha)$ *satisfies*

$$\Pr\left[(1-\epsilon)W(\alpha) \leq \hat{W}(\alpha) \leq (1+\epsilon)W(\alpha)\right] \geq 3/4 \quad .$$

In addition, the $3/4$ *can be made arbitrarily close to* $1-\delta$ *for any* $\delta > 0$.

As stated in the following corollary, our algorithms' behaviors are the same as Winnow's for a straightforward brute-force implementation if the weighted sums are not too close to θ for any input. This is true with probability at least $1 - \delta_t'$ for each trial t, so setting $\delta_t' = \delta/2^t$ yields a total probability of failure of at most[12] $\sum_{t=1}^{\infty} \delta/2^t = \delta$. Finally, note that it is easy to extend the corollary to tolerate a bounded number of trials with weighted sums that are near θ by thinking of such potential mispredictions as noise and applying Theorem 7.

Corollary 10 *Using the assumptions of Corollary 9, if $W_t(\alpha) \notin [\theta/(1+\epsilon), \theta/(1-\epsilon)]$ for all trials t, then with probability at least $1 - \delta$, the number of mistakes made by Winnow on any sequence of examples is as bounded by Theorem 7 (see Sections 4.1.1.1–4.1.1.3 and Lemma 11).*

A hurdle that must be overcome to get an efficient algorithm is S's polynomial dependence on $1/\epsilon$ in Corollary 9, even though Winnow might at times have W/θ exponentially close[13] to 1, requiring exponentially small ϵ. It is open whether this can be addressed in an average-case analysis of Winnow when learning restricted concept classes under specific distributions.

We now explore bounding the mixing times of the Markov chains. Note that the bounds are based on worst-case analyses and assume that the maximum number of weight updates (as bounded by Theorem 7's mistake bound) have been made. Prior to making that number of updates (e.g. near the start of training), the mixing time bounds will be lower since the stationary distribution π of $\mathcal{M}$ will be closer to uniform (in fact, before the first update, π is uniform). We can get mixing time bounds for these earlier cases by substituting the number of prediction mistakes made so far for the mistake bounds.

4.1.1.1 Binary $p(\vec{x})$

It is straightforward to apply Theorem 7 to the binary case. Let the vector $\vec{\mu}$ be 0 for each irrelevant term and 1 for each relevant term. Then when $\ell = 1$, at least one relevant term must be satisfied, so $\vec{\mu} \cdot \vec{Y} \geq 1$. Further, if $\ell = 0$, then no relevant terms are satisfied and $\vec{\mu} \cdot \vec{Y} = 0 \leq 1 - \delta$ for $\delta = 1$. Assuming all examples are noise-free, applying Theorem 7 yields a mistake bound of $|M| \leq 8 + 14K \ln N$. So if $k \geq k_i$ for all i, then using the at most $(k+1)^n$ possible terms as Winnow's inputs, it can learn K-term generalized DNF with at most $8 + 14Kn\ln(k+1)$ prediction mistakes.

Unfortunately, with the binary case it is very difficult to find non-trivial bounds on g and h from Theorem 5, due to the discontinuity of $p(\vec{x})$. Bounding both g and h requires bounding the ratios of the weights of nodes in Ω. For binary $p(\vec{x})$, these weights directly depend on how often the nodes predicted 1 when a prediction mistake was made, but it is difficult to relate how often this occurs for a node $\vec{p}$ to how often this occurs for another node $\vec{q}$, even if $\vec{p}$ and $\vec{q}$ are neighbors. Certainly trivial bounds can be made by using the mistake bound, which implies that $g \leq \alpha^{2|M|}$ and $h \leq \alpha^{|M|}$. On the other hand, when we consider logistic and linear $p(\vec{x})$, we can relate the nodes' weights and get better bounds on g and h.

[12] Recall from the proof of Theorem 2 that only $O(\log 1/\delta')$ runs of the estimation procedure are needed to reduce the probability of failure to δ'.

[13] The potential problem of $W_t/\theta = 1$ can be avoided by using a threshold of $\theta + \alpha^{-(|M|+1)}$, where $|M|$ is the mistake bound from applying Theorem 7. Obviously W can never equal this new threshold.

4.1.1.2 Logistic $p(\vec{x})$

The mistake bound of this application of Winnow is similar to that of the straightforward version with binary inputs.

Lemma 11 *When using Equation 6 with $a = \ln(60Kn\ln k)$ to specify the inputs, where $k = \max_i\{k_i\}$, the number of prediction mistakes $|M|$ made by Winnow when learning DNF is at most*

$$8.88 + 15.54Kn\ln k \ .$$

Proof: We start by finding $\vec{\mu}$ and δ that satisfy the conditions of Theorem 7. For each of the K relevant terms (Winnow inputs), set the corresponding value in $\vec{\mu}$ to a constant μ, which we will define later. Set all other values in $\vec{\mu}$ to 0. In the worst case, when an example $\vec{x}$ is positive, it satisfies exactly one relevant term $\vec{p}$ and does not at all satisfy any of the other relevant terms. Then $p(\vec{x}) = 1$ and $q(\vec{x}) = 2/(1+e^{an})$ for all other relevant terms $\vec{q}$. Thus it suffices to set μ such that

$$\mu + \frac{2(K-1)\mu}{1+e^{an}} \geq 1 \ .$$

After some algebra we see that it suffices to set $\mu = (1+e^{an})/(2K+e^{an}-1)$.

Now we find δ. In the worst case, for a negative example $\vec{x}$, we will have each relevant term $\vec{p}$ almost fully satisfied, i.e. $m_{\vec{p}} = n-1$. Hence $p(\vec{x}) = 2/(1+e^a)$. So we need δ such that $1-\delta \geq 2K\mu/(1+e^a)$. Substituting μ yields

$$\delta \leq 1 - \frac{2K(1+e^{an})}{2K(1+e^a)+e^{an}(1+e^a)-e^a-1} \ .$$

This expression decreases with increasing n, so to find an appropriate δ, it suffices to take its limit as $n \to \infty$. Applying l'Hôpital's rule shows that $\delta = 1 - 2K/(1+e^a)$ is sufficient, which is positive so long as $a > \ln(2K-1)$. We assume that all examples are noise-free, so $\vec{E}_j = \vec{0}$ for all j. Now applying Theorem 7 yields a mistake bound of

$$\begin{aligned} & \left(\frac{(1+e^a)^2}{(1+e^a)^2 - 4K(e^a+1-K)}\right)\left(8+14K\left(\frac{1+e^{an}}{2K+e^{an}-1}\right)\left(\ln\left(\frac{1+e^{an}}{2K+e^{an}-1}\right)+\ln N\right)\right) \\ \leq\ & \left(\frac{(1+e^a)^2}{(1+e^a)^2-4K(1+e^a)}\right)(8+14K\ln N) = \left(\frac{1+e^a}{1+e^a-4K}\right)(8+14K\ln N) \\ \leq\ & 1.11(8+14K\ln N) \ , \end{aligned}$$

since $K, n \geq 1$ and $k \geq 2$. Noting that $N \leq k^n$ completes the proof. □

We now work towards a mixing time bound for the chain.

Lemma 12 *Let $\Omega = \{0, \ldots, k\}^n$. Then for all $\vec{p}, \vec{q} \in \Omega$,*

$$\pi_\alpha(\vec{p})\,\pi_\alpha(\vec{q}) \leq 4\,\alpha^{8.88+15.54Kn\ln k}\,\pi_\alpha(\vec{a}')\,\pi_\alpha(\vec{\eta}_e(\vec{p},\vec{q})) \ .$$

Proof: Let $\vec{q}_{1\ldots i}$ denote $(q_1, \ldots, q_i)$, and similarly for $\vec{p}_{1\ldots i}$. Then $m_{\vec{p}} = m_{\vec{p}_{1\ldots i}} + m_{\vec{p}_{i+1\ldots n}}$, $m_{\vec{q}} = m_{\vec{q}_{1\ldots i}} + m_{\vec{q}_{i+1\ldots n}}$, $m_{\vec{\eta}_e(\vec{p},\vec{q})} = m_{\vec{p}_{1\ldots i}} + m_{\vec{q}_{i+1\ldots n}}$, and $m_{\vec{a}'} = m_{\vec{q}_{1\ldots i}} + m_{\vec{p}_{i+1\ldots n}}$. Further, all four of these values are in $\{0, \ldots, n\}$. This yields

$$\begin{aligned}
&\frac{\pi_\alpha(\vec{p})\,\pi_\alpha(\vec{q})}{\pi_\alpha(\vec{a}')\,\pi_\alpha(\vec{\eta}_e(\vec{p},\vec{q}))} = \frac{p(\vec{x})q(\vec{x})\,\alpha^{z_{\vec{p}}+z_{\vec{q}}}}{\vec{\eta}_e(\vec{p},\vec{q})(\vec{x})\,\vec{a}'(\vec{x})\,\alpha^{z_{\vec{\eta}_e(\vec{p},\vec{q})}+z_{\vec{a}'}}} \\
= \;& \left(\frac{\alpha^{z_{\vec{p}}+z_{\vec{q}}}}{\alpha^{z_{\vec{\eta}_e(\vec{p},\vec{q})}+z_{\vec{a}'}}}\right) \\
\cdot \;& \left(\frac{1+\exp\left(-a\left(m_{\vec{p}_{1\ldots i}}+m_{\vec{q}_{i+1\ldots n}}-n\right)\right)+\exp\left(-a\left(m_{\vec{q}_{1\ldots i}}+m_{\vec{p}_{i+1\ldots n}}-n\right)\right)+\exp\left(-a\left(m_{\vec{p}}+m_{\vec{q}}-2n\right)\right)}{1+\exp\left(-a\left(m_{\vec{p}_{1\ldots i}}+m_{\vec{p}_{i+1\ldots n}}-n\right)\right)+\exp\left(-a\left(m_{\vec{q}_{1\ldots i}}+m_{\vec{q}_{i+1\ldots n}}-n\right)\right)+\exp\left(-a\left(m_{\vec{p}}+m_{\vec{q}}-2n\right)\right)}\right) \\
< \;& 4\,\alpha^{z_{\vec{p}}+z_{\vec{q}}-z_{\vec{\eta}_e(\vec{p},\vec{q})}-z_{\vec{a}'}} \quad ,
\end{aligned}$$

where the last inequality follows from each term in the numerator being strictly less than the entire denominator. Now let C be the exponent of the α term. Then we have[14]

$$\begin{aligned}
C &= z_{\vec{p}} + z_{\vec{q}} - z_{\vec{\eta}_e(\vec{p},\vec{q})} - z_{\vec{a}'} \\
&= \sum_{\vec{x}\in M} \frac{2\ell_{\vec{x}}}{1+\exp\left(-a\left(m_{\vec{x},\vec{p}_{1\ldots i}}+m_{\vec{x},\vec{p}_{i+1\ldots n}}-n\right)\right)} + \frac{2\ell_{\vec{x}}}{1+\exp\left(-a\left(m_{\vec{x},\vec{q}_{1\ldots i}}+m_{\vec{x},\vec{q}_{i+1\ldots n}}-n\right)\right)} \\
&\quad - \frac{2\ell_{\vec{x}}}{1+\exp\left(-a\left(m_{\vec{x},\vec{p}_{1\ldots i}}+m_{\vec{x},\vec{q}_{i+1\ldots n}}-n\right)\right)} - \frac{2\ell_{\vec{x}}}{1+\exp\left(-a\left(m_{\vec{x},\vec{q}_{1\ldots i}}+m_{\vec{x},\vec{p}_{i+1\ldots n}}-n\right)\right)} \quad .
\end{aligned}$$

Each term of the above summation is between -1 and 1, so a worst-case upper bound is $|M| \leq 8.88 + 15.54Kn\ln k$. □

Lemma 13 *For all neighbors $\vec{p}$ and $\vec{q} \in \Omega$,*

$$\max\{\pi_\alpha(\vec{p})/\pi_\alpha(\vec{q}), \pi_\alpha(\vec{q})/\pi_\alpha(\vec{p})\} \leq \alpha\, 60Kn\ln k \quad .$$

Proof: Since $\vec{p}$ and $\vec{q}$ are neighbors, they only differ in one position, so $|m_{\vec{p}} - m_{\vec{q}}| \leq 1$. Then

$$\begin{aligned}
\frac{\pi_\alpha(\vec{p})}{\pi_\alpha(\vec{q})} &= \left(\frac{p(\vec{x})}{q(\vec{x})}\right)\alpha^{z_{\vec{p}}-z_{\vec{q}}} = \left(\frac{1+\exp\left(a\,n - a\,m_{\vec{q}}\right)}{1+\exp\left(a\,n - a\,m_{\vec{p}}\right)}\right)\alpha^{z_{\vec{p}}-z_{\vec{q}}} \\
&\leq \left(\frac{1+\exp\left(a\,n - a\left(m_{\vec{p}}-1\right)\right)}{1+\exp\left(a\,n - a\,m_{\vec{p}}\right)}\right)\alpha^{z_{\vec{p}}-z_{\vec{q}}} = \left(\frac{1+e^a\,\exp\left(a\,n - a\,m_{\vec{p}}\right)}{1+\exp\left(a\,n - a\,m_{\vec{p}}\right)}\right)\alpha^{z_{\vec{p}}-z_{\vec{q}}} \\
&\leq \left(\frac{1+\exp\left(a\,n - a\,m_{\vec{p}}\right)}{1+\exp\left(a\,n - a\,m_{\vec{p}}\right)}\right)e^a\,\alpha^{z_{\vec{p}}-z_{\vec{q}}} = e^a\,\alpha^{z_{\vec{p}}-z_{\vec{q}}} \quad .
\end{aligned}$$

[14] When the subscript $\vec{x}$ is omitted from m, then m counts the number of matches with the current example. In the summations over $\vec{x} \in M$, $m_{\vec{x}}$ represents the number of matches with example $\vec{x}$.

We now consider $z_{\vec{p}} - z_{\vec{q}}$:

$$\begin{aligned}
z_{\vec{p}} - z_{\vec{q}} &= \sum_{\vec{x} \in M} \frac{2\ell_{\vec{x}}}{1 + \exp\left(a\, n - a\, m_{\vec{x},\vec{p}}\right)} - \frac{2\ell_{\vec{x}}}{1 + \exp\left(a\, n - a\, m_{\vec{x},\vec{q}}\right)} \\
&= 2 \sum_{\vec{x} \in M} \ell_{\vec{x}} \left(\frac{\exp\left(a\, n - a\, m_{\vec{x},\vec{q}}\right) - \exp\left(a\, n - a\, m_{\vec{x},\vec{p}}\right)}{1 + \exp\left(a\, n - a\, m_{\vec{x},\vec{q}}\right) + \exp\left(a\, n - a\, m_{\vec{x},\vec{p}}\right) + \exp\left(2\, a\, n - a\left(m_{\vec{x},\vec{p}} + m_{\vec{x},\vec{q}}\right)\right)} \right) \\
&\leq 2 \sum_{\vec{x} \in M} \ell_{\vec{x}} \left(\frac{\exp\left(a\, n - a\, m_{\vec{x},\vec{p}} + a\right) - \exp\left(a\, n - a\, m_{\vec{x},\vec{p}}\right)}{1 + \exp\left(a\, n - a\, m_{\vec{x},\vec{p}} + a\right) + \exp\left(a\, n - a\, m_{\vec{x},\vec{p}}\right) + \exp\left(2\, a\, n - a\left(m_{\vec{x},\vec{p}} - 1 + m_{\vec{x},\vec{q}}\right)\right)} \right) \\
&= 2 \sum_{\vec{x} \in M} \ell_{\vec{x}} \left(\frac{\exp\left(a\, n - a\, m_{\vec{x},\vec{p}}\right)\left(e^a - 1\right)}{1 + \exp\left(a\, n - a\, m_{\vec{x},\vec{p}}\right)\left(e^a + 1\right) + e^a \, \exp\left(2\, a \, \left(n - m_{\vec{x},\vec{p}}\right)\right)} \right) \\
&\leq 2 \sum_{\vec{x} \in M} \frac{\ell_{\vec{x}}}{1 + e^a \, \exp\left(2\, a \, \left(n - m_{\vec{x},\vec{p}}\right)\right)} \leq \frac{2\,|M|}{1 + e^a} \;,
\end{aligned}$$

where the third line follows from the fact that there are only three ways to relate $m_{\vec{x},\vec{p}}$ and $m_{\vec{x},\vec{q}}$ for a specific $\vec{x}$. If $m_{\vec{x},\vec{p}} = m_{\vec{x},\vec{q}} + 1$, then that term of the summation equals the bound. If $m_{\vec{x},\vec{p}} = m_{\vec{x},\vec{q}} - 1$, then that term of the summation is negative, which is less than the bound. If $m_{\vec{x},\vec{p}} = m_{\vec{x},\vec{q}}$, then that term of the summation is 0, which is less than the bound.

Since $a = \ln(60Kn\ln k)$, we get $z_{\vec{p}} - z_{\vec{q}} \leq 1$. Finally, we note that a symmetric argument can be made for $\pi_\alpha(\vec{q})/\pi_\alpha(\vec{p})$. □

We now apply Theorem 5 to bound the mixing time of this Markov chain.

Corollary 14 *When learning generalized DNF using Winnow and a logistic $p(\vec{x})$, a simulation of $\mathcal{M}$ that starts at any node and is of length*

$$T_i = 480kn^3\, \alpha_i^{9.88 + 15.54Kn\ln k}\, K \ln k \left(n \ln k + 2(8.88 + 15.54Kn\ln k)\ln \alpha_i + \ln\left(1/\epsilon'\right)\right)$$

will draw samples from $\hat{\pi}_{\alpha_i}$ such that $\|\hat{\pi}_{\alpha_i} - \pi_{\alpha_i}\| \leq \epsilon'$.

Proof: Lemmas 12 and 13 bound g and h, which we substitute directly into Theorem 5. Also, note that $W(\alpha_i) \leq k^n \alpha_i^{8.88+15.54Kn\ln k}$ and $w_{\alpha_i,\vec{p}} \geq \alpha_i^{-(8.88+15.54Kn\ln k)}$, completing the proof. □

Unfortunately, our worst-case mixing time bound of this chain is exponential in n and K. In fact, a straightforward brute-force means to compute the sum of the weights can be done in $\Theta(k^n)$ time, whereas our bound of T_i grows with $\alpha_i^{9.88+15.54Kn\ln k} \geq k^{15.54Kn\ln \alpha_i} \geq k^{6.30n}$ when $\alpha_i = \alpha = 3/2$ (a popular value for α). However, Corollary 14 is based on a worst-case, adversary-based analysis, especially Lemma 12 which puts Winnow's mistake bound in the exponent of α even though it is possible that C is much less. It is open whether sub-exponential bounds can be achieved by applying a different analysis to some special cases of restricted concept classes and distributional assumptions. Further, in Section 5 we show that in practice, our algorithm performs much better than the worst-case theoretical results imply, especially considering that highly accurate estimates of the weighted sums are not needed so long as we know which side of the threshold the sum lies on.

4.1.1.3 Linear $p(\vec{x})$

Applying Theorem 7 to the linear case is not as straightforward as it was for the logistic case. As in the proof of Lemma 11, we set the entries in $\vec{\mu}$ corresponding to relevant terms to μ and the remaining entries to 0. When $\ell = 1$, in the worst case exactly one relevant term matches all n variables in $\vec{x}$ and the remaining $K-1$ relevant terms match 0. Then we get

$$\vec{\mu} \cdot \vec{Y} = \mu \left(1 + \frac{K-1}{n+1} \right) = \mu \left(\frac{n+K}{n+1} \right) \quad ,$$

which has to be ≥ 1. When $\ell = 0$, the worst case has all K relevant terms matching $n-1$ variables of $\vec{x}$, yielding

$$\vec{\mu} \cdot \vec{Y} = \mu \left(\frac{nK}{n+1} \right) \geq \mu \left(\frac{n+K}{n+1} \right)$$

for $n, K \geq 2$. Thus it is impossible to get a $\delta > 0$ for the linear case unless we treat such worst-case examples as noise and use a nonzero $\vec{E}$. Following this idea, we assume that when $\ell = 1$, all relevant inputs to Winnow are at least $\gamma_1/(n+1)$ (i.e. all relevant inputs match at least $\gamma - 1$ variables in $\vec{x}$), with of course at least one such input $= 1$. Further, we assume that when $\ell = 0$, all relevant inputs are at most $\gamma_0/(n+1)$. Then it is easy to show that setting

$$\mu = \frac{n+1}{n+1+\gamma_1(K-1)}$$

and

$$\delta = \frac{n+1+\gamma_1(K-1) - \gamma_0 K}{n+1+\gamma_1(K-1)}$$

satisfies the conditions of Theorem 7. For example, if $\gamma_1 = 3n/4$ and $\gamma_0 = n/2$, then $\mu = (4n+4)/(n+3nK+4)$, $\delta = (n+nK+4)/(n+3nK+4) \geq 1/3$, and Theorem 7's mistake bound is

$$\begin{aligned} |M| &\leq 72 + 126 \left(\frac{4K(n+1)}{n+3Kn+4} \right) \ln \left(\frac{4N(n+1)}{n+3Kn+4} \right) + 36 \sum_{j=1}^{t} \vec{\mu} \cdot \vec{E}_j \\ &\leq 72 + 252 \ln N + 36 \sum_{j=1}^{t} \vec{\mu} \cdot \vec{E}_j \leq 72 + 252 n \ln k + 36 \sum_{j=1}^{t} \vec{\mu} \cdot \vec{E}_j \quad , \end{aligned} \qquad (10)$$

if $n \geq 2$. Of course, this is of little value as an adversarial bound, since the third term is summed over all trials and an adversary could make each term of this summation positive. But a bound of this form might be useful under appropriate distributional assumptions.

We now bound the mixing time for the linear case.

Lemma 15 *Let $\Omega = \{0, \ldots, k\}^n$. Then for all $\vec{p}, \vec{q} \in \Omega$,*

$$\pi_\alpha(\vec{p})\, \pi_\alpha(\vec{q}) \leq \frac{(1+n/2)^2}{n+1} \pi_\alpha(\vec{a}')\, \pi_\alpha(\vec{\eta}_e(\vec{p}, \vec{q})) \quad .$$

Proof: Using the same notation introduced in the first paragraph of Lemma 12's proof, we get

$$\begin{aligned}
&\frac{\pi_\alpha(\vec{p})\,\pi_\alpha(\vec{q})}{\pi_\alpha(\vec{a}')\,\pi_\alpha(\vec{\eta}_e(\vec{p},\vec{q}))} = \frac{p(\vec{x})\,q(\vec{x})\,\alpha^{z_{\vec{p}}+z_{\vec{q}}}}{\vec{\eta}_e(\vec{p},\vec{q})(\vec{x})\,\vec{a}'(\vec{x})\,\alpha^{z_{\vec{\eta}_e(\vec{p},\vec{q})}+z_{\vec{a}'}}} \\
= \;& \left(\frac{\alpha^{z_{\vec{p}}+z_{\vec{q}}}}{\alpha^{z_{\vec{\eta}_e(\vec{p},\vec{q})}+z_{\vec{a}'}}}\right)\left(\frac{(1+m_{\vec{p}_{1\ldots i}}+m_{\vec{p}_{i+1\ldots n}})(1+m_{\vec{q}_{1\ldots i}}+m_{\vec{q}_{i+1\ldots n}})}{(1+m_{\vec{p}_{1\ldots i}}+m_{\vec{q}_{i+1\ldots n}})(1+m_{\vec{q}_{1\ldots i}}+m_{\vec{p}_{i+1\ldots n}})}\right) \\
\leq \;& \alpha^{z_{\vec{p}}+z_{\vec{q}}-z_{\vec{\eta}_e(\vec{p},\vec{q})}-z_{\vec{a}'}}\left(\frac{(1+n/2)^2}{n+1}\right) \quad .
\end{aligned}$$

The last inequality holds since the second term is maximized by setting $m_{\vec{p}_{1\ldots i}} = m_{\vec{q}_{i+1\ldots n}} = 0$ and $m_{\vec{q}_{1\ldots i}} m_{\vec{p}_{i+1\ldots n}} = n/2$. Now let C be the first term's exponent:

$$\begin{aligned}
C &= z_{\vec{p}} + z_{\vec{q}} - z_{\vec{\eta}_e(\vec{p},\vec{q})} - z_{\vec{a}'} \\
&= \sum_{\vec{x}\in M} \ell_{\vec{x}} \Bigg(\frac{1+m_{\vec{x},\vec{p}_{1\ldots i}}+m_{\vec{x},\vec{p}_{i+1\ldots n}}+1+m_{\vec{x},\vec{q}_{1\ldots i}}+m_{\vec{x},\vec{q}_{i+1\ldots n}}}{n+1} \\
&\qquad - \frac{1+m_{\vec{x},\vec{p}_{1\ldots i}}+m_{\vec{x},\vec{q}_{i+1\ldots n}}+1+m_{\vec{x},\vec{q}_{1\ldots i}}+m_{\vec{x},\vec{p}_{i+1\ldots n}}}{n+1} \Bigg) \\
&= 0 \quad .
\end{aligned}$$

□

Lemma 16 *For all neighbors $\vec{p}$ and $\vec{q} \in \Omega$,*

$$\max\{\pi_\alpha(\vec{p})/\pi_\alpha(\vec{q}), \pi_\alpha(\vec{q})/\pi_\alpha(\vec{p})\} \leq 2\alpha^{|M|/(n+1)} \quad ,$$

where $|M|$ is the learner's mistake bound.

Proof: Since $\vec{p}$ and $\vec{q}$ are neighbors, they only differ in one position, so $|m_{\vec{p}} - m_{\vec{q}}| \leq 1$. Then

$$\frac{\pi_\alpha(\vec{p})}{\pi_\alpha(\vec{q})} = \left(\frac{p(\vec{x})}{q(\vec{x})}\right)\alpha^{z_{\vec{p}}-z_{\vec{q}}} = \left(\frac{1+m_{\vec{p}}}{1+m_{\vec{q}}}\right)\alpha^{z_{\vec{p}}-z_{\vec{q}}} \leq \left(\frac{2+m_{\vec{q}}}{1+m_{\vec{q}}}\right)\alpha^{z_{\vec{p}}-z_{\vec{q}}} \leq 2\,\alpha^{z_{\vec{p}}-z_{\vec{q}}} \quad .$$

We now consider $z_{\vec{p}} - z_{\vec{q}}$:

$$z_{\vec{p}} - z_{\vec{q}} = \sum_{\vec{x}\in M} \ell_{\vec{x}}\,(1 + m_{\vec{x},\vec{p}} - 1 - m_{\vec{x},\vec{q}}) \leq |M|/(n+1) \quad .$$

Finally, we note that a symmetric argument can be made for $\pi_\alpha(\vec{q})/\pi_\alpha(\vec{p})$. □

Corollary 17 *When learning generalized DNF using Winnow and a linear $p(\vec{x})$, a simulation of $\mathcal{M}$ that starts at any node and is of length*

$$T_i = 4kn(n^2/4+n+1)\alpha_i^{|M|/(n+1)}\left(n\ln k + 2|M|\ln\alpha_i + \ln(1/\epsilon')\right)$$

(where $|M|$ is the learner's mistake bound) will draw samples from $\hat{\pi}_{\alpha_i}$ such that $\|\hat{\pi}_{\alpha_i} - \pi_{\alpha_i}\| \leq \epsilon'$.

Proof: Lemmas 15 and 16 bound g and h, which we substitute directly into Theorem 5. Also, note that $W(\alpha_i) \leq k^n \alpha_i^{|M|}$ and $w_{\alpha_i,\vec{p}} \geq \alpha_i^{-|M|}$, completing the proof. □

Note that if $|M| = O(n \log K)$ (e.g. if the third term of Equation (10) is $O(n \log K)$), then the chain's mixing time is polynomial in all relevant parameters if k (the number of values each variable can take on) is a constant, yielding an FPRAS under the conditions of Corollary 9.

4.1.2 Perceptron

We now consider applying our technique to Rosenblatt's Perceptron [30] algorithm. The purpose of our analysis is primarily for contrast to the Winnow case since[15] Khardon et al. [16] give a kernel function to efficiently exactly compute the weighted sums when applying Perceptron to learning DNF. But they also give an exponential lower bound on the number of mistakes that kernel perceptron makes in learning DNF: $2^{\Omega(n)}$. Thus their results do not imply an efficient DNF-learning algorithm.

We refer the reader to Section 2.1 for an overview of the Perceptron algorithm. Note that if the initial weight vector is the all 0s vector, then term $\vec{p}$'s weight is $w_{\vec{p}} = \alpha\, z_{\vec{p}}$, where $z_{\vec{p}} = \sum_{\vec{x} \in M} \ell_{\vec{x}}\, p(\vec{x})$, M is the set of examples for which a prediction mistake is made and $\ell_{\vec{x}} \in \{-1, +1\}$ is example $\vec{x}$'s label.

Since our technique is only capable of estimating positive functions, we cannot allow the Perceptron's weights to be negative. Thus to each weight in "standard" Perceptron, we add a positive constant c, yielding a new weight for term $\vec{p}$ of $w_{\vec{p}} = c + \alpha z_{\vec{p}}$, where $c = 3\alpha|M| \geq 3\alpha \max_{\vec{p} \in \Omega}\{|z_{\vec{p}}|\}$. The dot product of this new weight vector with the Perceptron inputs will then be compared to the new threshold c.

We use the same definitions and algorithm (given in Section 3) that were used for Winnow, except the "base" value of α_i is now $\alpha_0 = 0$ rather than $\alpha_1 = 1$. So our weight estimate is the same as given in Equation (1), but the product runs from $i = 1$ to r rather than starting at $i = 2$, and we multiply it by $W(\alpha_0)$. As with Winnow, this latter quantity is easily computed exactly: For binary $p(\vec{x})$, $W(\alpha_0) = c\, 2^n$. For logistic $p(\vec{x})$, if $k_i = k$ for all i, we get

$$W(0) = c \sum_{i=0}^{n} 2^i \binom{n}{i} \frac{2(k-1)^{n-i}}{1 + e^{-a(i-n)}} \ ,$$

and for the linear case, we get

$$W(0) = c \sum_{i=0}^{n} 2^i \binom{n}{i} \frac{(i+1)(k-1)^{n-i}}{n+1} = \frac{c\,(k+1)^n + 2c\,n(k+1)^{n-1}}{n+1} \ .$$

Thus all three can efficiently be computed exactly.

We now make the following argument about f_i and $\hat{f}_i$ for all three versions of $p(\vec{x})$.

Lemma 18 *When applying binary, logistic, or linear Perceptron to learn DNF, for all i,*

$$2/3 \leq f_i, \hat{f}_i \leq 2 \ .$$

Proof: We focus on the actual random variables, since the estimates (the "hat" variables) have the same range. Since these variables for all three versions have $p(\vec{x})$ in the numerator and denominator, they all equal

$$f_i(\vec{p}) = \frac{c + z_{\vec{p}}\,\alpha_{i-1}}{c + z_{\vec{p}}\,\alpha_i} \ .$$

[15] Note, however, that other applications of Perceptron for which no kernels are available might be amenable to an MCMC-based approach to estimate the dot products.

If $z_{\vec{p}} \geq 0$, then (since $\alpha_{i-1} < \alpha_i$) obviously $f_i(\vec{p}) \leq 1$. Also, since $c \geq 2\alpha z_{\vec{p}}$ and $\alpha \geq \alpha_i$,

$$f_i(\vec{p}) \geq \frac{c + z_{\vec{p}}\,\alpha_{i-1}}{c + c\,\alpha_i/(2\alpha)} \geq \frac{c + z_{\vec{p}}\,\alpha_{i-1}}{3c/2} = \frac{2}{3}\left(1 + \frac{z_{\vec{p}}\,\alpha_{i-1}}{c}\right) \geq 2/3 \ .$$

If $z_{\vec{p}} < 0$, then obviously $f_i(\vec{p}) > 1$. Also,

$$f_i(\vec{p}) \leq \frac{c + z_{\vec{p}}\,\alpha_{i-1}}{c/2} = 2 + \frac{2z_{\vec{p}}\,\alpha_{i-1}}{c} \leq 2 \ .$$

Thus $f_i(\vec{p}) \in [2/3, 2]$ for all i. □

This leads to the following corollary.

Corollary 19 *When applying Perceptron to learn generalized DNF (with $k_i = k$ for all i for the logistic and linear cases), let the sample size $S = \lceil 390\, r/\epsilon^2 \rceil$ and $\mathcal{M}$ be simulated long enough for each sample such that the variation distance between the empirical distribution and π_{α_i} is $\leq \epsilon/(5e^2 r)$. Then $\hat{W}(\alpha)$ satisfies*

$$\Pr\left[(1-\epsilon)\, W(\alpha) \leq \hat{W}(\alpha) \leq (1+\epsilon)\, W(\alpha)\right] \geq 3/4 \ .$$

In addition, the $3/4$ can be made arbitrarily close to $1-\delta$ for any $\delta > 0$.

To bound the number of prediction mistakes the Perceptron algorithm makes in learning DNF, we apply a result from Gentile and Warmuth [10].

Theorem 20 *[10] Let $(\vec{Y}_j, \ell_j) \in [0,1]^N \times \{0,1\}$ for $j = 1, \ldots, t$, let $\vec{\mu}$ be an arbitrary weight vector of dimension N, and let M be the set of examples on which the Perceptron algorithm makes a prediction mistake. Then the number of mistakes made by the Perceptron algorithm is*

$$|M| \leq \left(\frac{\|\vec{\mu}\|_2\, \rho}{\gamma_{\vec{\mu},M}}\right)^2 \ ,$$

where $\|\cdot\|_2$ is the 2-norm, $\rho \geq \|\vec{Y}\|_2$ for all $\vec{Y} \in M$, and

$$\gamma_{\vec{\mu},M} = \sum_{\vec{Y}_j \in M} \ell_j\, \vec{\mu} \cdot \vec{Y}_j$$

is the average margin *of $\vec{\mu}$.*

4.1.2.1 Binary $p(\vec{x})$

Applying Theorem 20 to the binary case is straightforward. We let $\vec{\mu}$ be 0 in all places except those corresponding to a relevant attribute, which are set to 1 (so there are K 1s in $\vec{\mu}$). In addition, we add an $(N+1)$th position to $\vec{\mu}$, setting it to $-1/2$. This position will correspond to a 1 added to each example seen by the perceptron. Thus we get $\|\vec{\mu}\|_2 = \sqrt{K + 1/4}$. Since all Perceptron inputs are from $\{0,1\}$ for the binary case, we get $\gamma_{\vec{\mu},M} \geq 1/2$. Further, since exactly $2^n + 1$ inputs are 1 for each example, $\rho = \sqrt{2^n + 1}$. Applying Theorem 20 yields $|M| \leq (4K+1)(2^n+1)$.

As with binary $p(\vec{x})$ with Winnow, binary $p(\vec{x})$ with Perceptron is difficult to analyze to provide non-trivial bounds on the mixing time. Thus we look at the logistic and linear cases.

4.1.2.2 Logistic $p(\vec{x})$

We begin by bounding the number of mistakes Perceptron will make.

Lemma 21 *When using Equation 6 with $a = \ln(60Kn\ln k)$ to specify the inputs, the number of prediction mistakes made by Perceptron when learning DNF is at most*

$$12K(2 + k/(3600\ln^2 k))^n \ .$$

Proof: We use the same $\vec{\mu}$ as we did for the binary case, namely 1s at the K relevant positions, $-1/2$ matching the extra 1 added to each example, and 0s elsewhere. We now bound $\gamma_{\vec{\mu},M}$. If $\ell_j = +1$ for some trial j, then at least one of the relevant terms $\vec{p}$ must send a 1 input to Perceptron. Thus $\ell_j\,\vec{\mu}\cdot\vec{Y}_j \geq 1/2$, where $\vec{Y}_j$ is the vector of inputs to Perceptron (i.e. the outputs of the $p(\cdot)$ functions). If $\ell_j = -1$, then in the worst case each relevant term will be almost completely satisfied by the input example $\vec{x}_j$, i.e. all but one variable in each relevant term will be satisfied. If this happens, then we get

$$\gamma_{\vec{\mu},M} \geq \ell_j\,\vec{\mu}\cdot\vec{Y}_j \geq \frac{1}{2} - \frac{2K}{1+e^a} = \frac{1}{2} - \frac{2K}{1+60Kn\ln k} \geq \frac{1}{2} - \frac{1}{30\ln 2} \geq 0.45 \ ,$$

since $n \geq 1$ and $k \geq 2$.

We now bound ρ. Recall that Equation (8) sums the $p(\vec{x})$ values for the entire set of terms. By substituting $(p(\vec{x}))^2$ for $p(\vec{x})$ in this equation and taking the square root, we exactly get the 2-norm for any input to Perceptron:

$$\begin{aligned}
\|\vec{Y}\|_2 &= \sqrt{\sum_{i=0}^{n} 2^i \binom{n}{i} \frac{4(k-1)^{n-i}}{\left(1+e^{-a(i-n)}\right)^2}} \leq \sqrt{\sum_{i=0}^{n} 2^i \binom{n}{i} \frac{4k^{n-i}}{e^{-2a(i-n)}}} \\
&= 2\sqrt{\sum_{i=0}^{n} 2^i \binom{n}{i} k^{n-i} \left(60Kn\ln k\right)^{2i-2n}} = \frac{2k^{n/2}}{(60Kn\ln k)^n}\sqrt{\sum_{i=0}^{n} \left((7200/k)n^2K^2\ln^2 k\right)^i \binom{n}{i}} \\
&= \sqrt{4\left(\frac{(7200/k)n^2K^2\ln^2 k + 1}{(3600/k)n^2K^2\ln^2 k}\right)^n} < 2\left(2 + k/(3600\ln^2 k)\right)^{n/2} \ ,
\end{aligned}$$

since $n, K \geq 1$. Thus setting $\rho = 2\left(2 + k/(3600\ln^2 k)\right)^{n/2}$ suffices. □

We now bound the mixing time for the Markov chain.

Lemma 22 *Let $\Omega = \{0, \ldots, k\}^n$. Then for all $\vec{p}, \vec{q} \in \Omega$,*

$$\pi_\alpha(\vec{p})\,\pi_\alpha(\vec{q}) \leq 8\,\pi_\alpha(\vec{a}')\,\pi_\alpha(\vec{\eta}_e(\vec{p},\vec{q})) \ .$$

Proof: As in the proof of Lemma 12, let $\vec{q}_{1\ldots i}$ denote $(q_1, \ldots, q_i)$, and similarly for $\vec{p}_{1\ldots i}$. Then $m_{\vec{p}} = m_{\vec{p}_{1\ldots i}} + m_{\vec{p}_{i+1\ldots n}}$, $m_{\vec{q}} = m_{\vec{q}_{1\ldots i}} + m_{\vec{q}_{i+1\ldots n}}$, $m_{\vec{\eta}_e(\vec{p},\vec{q})} = m_{\vec{p}_{1\ldots i}} + m_{\vec{q}_{i+1\ldots n}}$, and $m_{\vec{a}'} = m_{\vec{q}_{1\ldots i}} +$

$m_{\vec{p}_{i+1\ldots n}}$. Further, all four of these values are in $\{0,\ldots,n\}$. This yields

$$\begin{aligned}
&\frac{\pi_\alpha(\vec{p})\,\pi_\alpha(\vec{q})}{\pi_\alpha(\vec{a}')\,\pi_\alpha(\vec{\eta}_e(\vec{p},\vec{q}))} = \frac{p(\vec{x})q(\vec{x})\,(c+\alpha\,z_{\vec{p}})(c+\alpha\,z_{\vec{q}})}{\vec{\eta}_e(\vec{p},\vec{q})(\vec{x})\,\vec{a}'(\vec{x})\,(c+\alpha\,z_{\vec{\eta}_e(\vec{p},\vec{q})})(c+\alpha\,z_{\vec{a}'})} \\
= \;& \left(\frac{(c+\alpha\,z_{\vec{p}})(c+\alpha\,z_{\vec{q}})}{(c+\alpha\,z_{\vec{\eta}_e(\vec{p},\vec{q})})(c+\alpha\,z_{\vec{a}'})}\right) \\
\cdot \;& \left(\frac{1+\exp\left(-a\left(m_{\vec{p}_{1\ldots i}}+m_{\vec{q}_{i+1\ldots n}}-n\right)\right)+\exp\left(-a\left(m_{\vec{q}_{1\ldots i}}+m_{\vec{p}_{i+1\ldots n}}-n\right)\right)+\exp\left(-a\left(m_{\vec{p}}+m_{\vec{q}}-2n\right)\right)}{1+\exp\left(-a\left(m_{\vec{p}_{1\ldots i}}+m_{\vec{p}_{i+1\ldots n}}-n\right)\right)+\exp\left(-a\left(m_{\vec{q}_{1\ldots i}}+m_{\vec{q}_{i+1\ldots n}}-n\right)\right)+\exp\left(-a\left(m_{\vec{p}}+m_{\vec{q}}-2n\right)\right)}\right) \\
< \;& 4\left(\frac{(c+\alpha\,z_{\vec{p}})(c+\alpha\,z_{\vec{q}})}{(c+\alpha\,z_{\vec{\eta}_e(\vec{p},\vec{q})})(c+\alpha\,z_{\vec{a}'})}\right) \quad ,
\end{aligned}$$

where the last inequality comes directly from the proof of Lemma 12. We now bound the second term. Since $p(\vec{x}) \leq 1$, each $(\alpha\,z)$ summation is upper bounded by $\alpha\,|M|$, so the numerator is at most $(c+\alpha\,|M|)^2$. Meanwhile, the denominator is trivially at least c^2. Thus since $c = 3\alpha|M|$, the second term is at most $1 + 2/3 + 1/9 < 2$. □

Lemma 23 *For all neighbors $\vec{p}$ and $\vec{q} \in \Omega$,*

$$\max\{\pi_\alpha(\vec{p})/\pi_\alpha(\vec{q}), \pi_\alpha(\vec{q})/\pi_\alpha(\vec{p})\} \leq 80Kn\ln k \quad .$$

Proof: Since $\vec{p}$ and $\vec{q}$ are neighbors, they only differ in one position, so $|m_{\vec{p}} - m_{\vec{q}}| \leq 1$. Then

$$\begin{aligned}
\frac{\pi_\alpha(\vec{p})}{\pi_\alpha(\vec{q})} &= \left(\frac{p(\vec{x})}{q(\vec{x})}\right)\left(\frac{c+\alpha\,z_{\vec{p}}}{c+\alpha\,z_{\vec{q}}}\right) = \left(\frac{1+\exp\left(a\,n - a\,m_{\vec{q}}\right)}{1+\exp\left(a\,n - a\,m_{\vec{p}}\right)}\right)\left(\frac{c+\alpha\,z_{\vec{p}}}{c+\alpha\,z_{\vec{q}}}\right) \\
&\leq \left(\frac{1+\exp\left(a\,n - a\,(m_{\vec{p}}-1)\right)}{1+\exp\left(a\,n - a\,m_{\vec{p}}\right)}\right)\left(\frac{c+\alpha\,|M|}{c}\right) = \left(\frac{1+e^a\,\exp\left(a\,n - a\,m_{\vec{p}}\right)}{1+\exp\left(a\,n - a\,m_{\vec{p}}\right)}\right)\left(\frac{c+\alpha\,|M|}{c}\right) \\
&\leq \left(\frac{1+\exp\left(a\,n - a\,m_{\vec{p}}\right)}{1+\exp\left(a\,n - a\,m_{\vec{p}}\right)}\right)4e^a/3 = 80Kn\ln k \quad ,
\end{aligned}$$

since $a = \ln(60Kn\ln k)$. Finally, we note that a symmetric argument can be made for $\pi_\alpha(\vec{q})/\pi_\alpha(\vec{p})$. □

We now apply Theorem 5 to bound the mixing time of this Markov chain.

Corollary 24 *When learning generalized DNF using Perceptron and a logistic $p(\vec{x})$, a simulation of $\mathcal{M}$ that starts at any node and is of length*

$$T_i = 640kKn^3\ln k\left(n\ln k + \ln(48K\alpha_i) + n\ln(2 + k/(3600\ln^2 k)) + \ln(1/\epsilon')\right)$$

will draw samples from $\hat{\pi}_{\alpha_i}$ such that $\|\hat{\pi}_{\alpha_i} - \pi_{\alpha_i}\| \leq \epsilon'$.

Proof: Lemmas 22 and 23 bound g and h, which we substitute directly into Theorem 5. Also, note that $W(\alpha_i) \leq k^n\,(c+\alpha_i\,|M|)$ and $w_{\alpha_i,\vec{p}} \geq 1$, completing the proof. □

4.1.2.3 Linear $p(\vec{x})$

We begin by bounding the number of mistakes Perceptron will make. However, like the linear Winnow case, worst-case (adversary) bounds are not possible since the average margin could be forced to be negative. Thus we assume that the examples are such that most of them are linearly separable and have a positive average margin $\gamma_{\vec{\mu},M}$.

Lemma 25 *When using Equation 7 to specify the inputs, and if the average margin $\gamma_{\vec{\mu},M}$ of the sequence of examples is positive, then the number of prediction mistakes made by Perceptron when learning DNF is*

$$|M| \leq \frac{5K\left((K+1)^n + 6n(K+1)^{n-1} + 4n(n-1)(K+1)^{n-2}\right)}{4\gamma^2_{\vec{\mu},M}(n+1)^2} \ .$$

Proof: We use the same $\vec{\mu}$ as we did for the binary and logistic cases, and hence get the same 2-norm for this vector as before. To bound ρ, recall that Equation (9) sums the $p(\vec{x})$ values for the entire set of terms. By substituting $(p(\vec{x}))^2$ for $p(\vec{x})$ in this equation and taking the square root, we exactly get the 2-norm for any input to Perceptron:

$$\begin{aligned} \|\vec{y}\|_2 &= \sqrt{\sum_{i=0}^{n} 2^i \binom{n}{i} \left(\frac{i+1}{n+1}\right)^2} \\ &= \frac{\sqrt{(2+K-1)^n + 4n(2+K-1)^{n-1} + 4n(n-1)(2+K-1)^{n-2} + 2n(2+K-1)^{n-1}}}{n+1} \ , \end{aligned}$$

which completes the proof. □

Now we bound the mixing time.

Lemma 26 *Let $\Omega = \{0, \ldots, k\}^n$. Then for all $\vec{p}, \vec{q} \in \Omega$,*

$$\pi_\alpha(\vec{p})\, \pi_\alpha(\vec{q}) \leq \frac{2(1+n/2)^2}{n+1} \pi_\alpha(\vec{a}')\, \pi_\alpha(\vec{\eta}_e(\vec{p},\vec{q})) \ .$$

Proof: Using the same notation introduced in the first paragraph of Lemma 22's proof, we get

$$\begin{aligned} \frac{\pi_\alpha(\vec{p})\, \pi_\alpha(\vec{q})}{\pi_\alpha(\vec{a}')\, \pi_\alpha(\vec{\eta}_e(\vec{p},\vec{q}))} &= \left(\frac{p(\vec{x})\, q(\vec{x})}{\vec{\eta}_e(\vec{p},\vec{q})(\vec{x})\, \vec{a}'(\vec{x})}\right) \left(\frac{(c+\alpha\, z_{\vec{p}})(c+\alpha\, z_{\vec{q}})}{(c+\alpha\, z_{\vec{\eta}_e(\vec{p},\vec{q})})(c+\alpha\, z_{\vec{a}'})}\right) \\ &\leq \left(\frac{(1+n/2)^2}{n+1}\right) \cdot 2 \ , \end{aligned}$$

using results from the proofs of Lemmas 15 and 22. □

Lemma 27 *For all neighbors $\vec{p}$ and $\vec{q} \in \Omega$,*

$$\max\{\pi_\alpha(\vec{p})/\pi_\alpha(\vec{q}), \pi_\alpha(\vec{q})/\pi_\alpha(\vec{p})\} \leq 8/3 \ .$$

Proof: Since $\vec{p}$ and $\vec{q}$ are neighbors, they only differ in one position, so $|m_{\vec{p}} - m_{\vec{q}}| \leq 1$. Then

$$\frac{\pi_\alpha(\vec{p})}{\pi_\alpha(\vec{q})} = \left(\frac{p(\vec{x})}{q(\vec{x})}\right)\left(\frac{c+\alpha\, z_{\vec{p}}}{c+\alpha\, z_{\vec{q}}}\right) = \left(\frac{1+m_{\vec{p}}}{1+m_{\vec{q}}}\right)\left(\frac{c+\alpha\, z_{\vec{p}}}{c+\alpha\, z_{\vec{q}}}\right) \leq \left(\frac{2+m_{\vec{q}}}{1+m_{\vec{q}}}\right)(4/3) \leq 8/3 \ .$$

Finally, we note that a symmetric argument can be made for $\pi_\alpha(\vec{q})/\pi_\alpha(\vec{p})$. □

Corollary 28 *When learning generalized DNF using Perceptron and a linear $p(\vec{x})$, a simulation of $\mathcal{M}$ that starts at any node and is of length*

$$T_i = \left(\frac{32kn^2(1+n/2)^2}{3(n+1)}\right)(n\ln k + \ln(4\alpha_i\,|M|) + \ln(1/\epsilon'))$$

(where $|M|$ is as stated in Lemma 25) will draw samples from $\hat{\pi}_{\alpha_i}$ such that $\|\hat{\pi}_{\alpha_i} - \pi_{\alpha_i}\| \leq \epsilon'$.

Proof: Lemmas 26 and 27 bound g and h, which we substitute directly into Theorem 5. Also, note that $W(\alpha_i) \leq k^n\,(c + \alpha_i\,|M|)$ and $w_{\alpha_i,\vec{p}} \geq 1$, completing the proof. □

4.2 Pruning ensembles of classifiers

We now apply our methods to pruning an ensemble, produced by e.g. AdaBoost [32]. AdaBoost's output is a set of functions $h_i : \mathcal{X} \rightarrow \Re$, where $i \in \{1, \ldots, n\}$ and $\mathcal{X}$ is the instance space. Each h_i is trained on a different distribution over the training examples and is associated with a parameter $\beta_i \in \Re$ that weights its predictions. Given an instance $\vec{x} \in \mathcal{X}$, the ensemble's prediction is $H(\vec{x}) = \mathrm{sign}\left(\sum_{i=1}^n \beta_i h_i(\vec{x})\right)$. Thus $\mathrm{sign}\left(h_i(\vec{x})\right)$ is h_i's prediction on $\vec{x}$, $|h_i(\vec{x})|$ is its confidence in its prediction, and β_i weights AdaBoost's confidence in h_i. It has been shown that if each h_i has error less than $1/2$ on its distribution, then the error on the training set and the generalization error of $H(\cdot)$ can be bounded. Strong bounds on $H(\cdot)$'s generalization error can also be shown even if the boosting algorithm is run past the point where $H(\cdot)$'s error is zero [31]. However, overfitting can still occur [24], i.e. sometimes better generalization can be achieved if some of the h_i's are discarded. So our goal is to find a weighted combination of all possible prunings that performs not much worse in terms of generalization error than the best single pruning.

To predict nearly as well as the best pruning, we place every possible pruning in a pool (so $N = 2^n$) and run WM. We start by computing W^+ and W^-, which are, respectively, the sums of the weights of the experts predicting a positive and a negative label on example $\vec{x}$. Then WM predicts $+1$ if $W^+ > W^-$ and -1 otherwise. Whenever WM makes a prediction mistake, it reduces the weights of all experts that predicted incorrectly by dividing them by α (see Section 2.1).

As in Section 4.1, using a binary $p(\vec{x})$ makes bounding the mixing time difficult, except in a trivial sense. Thus we use a linear $p(\vec{x})$, which allows us to also incorporate each pruning's confidence in its prediction, and to use that confidence when updating the weights. Given an example $\vec{x} \in \mathcal{X}$, we compute $h_i(\vec{x})$ for all $i \in \{1, \ldots, n\}$. We then use our MCMC procedure to compute $\hat{W}^+$, an estimate of $W^+ = \sum_{\vec{p}\in\Omega^+} p(\vec{x})\, w_{\vec{p}}$, where $p(\vec{x}) = \sum_{i=1}^n p_i\, \beta_i\, h_i(\vec{x}_t)$, $\Omega^+ = \{\vec{p} \in \{0,1\}^n : \sum_{i=1}^n p_i\, \beta_i\, h_i(\vec{x}) \geq 0\}$, $w_{\vec{p}} = \alpha^{z_{\vec{p}}}$, $z_{\vec{p}} = \sum_{\vec{x}\in M} \ell_{\vec{x}}\, p(\vec{x})$, and M is the set of examples for which a prediction mistake was made. A similar procedure is used to compute $\hat{W}^-$. Then WM predicts $+1$ if $\hat{W}^+ > \hat{W}^-$ and -1 otherwise.

Define the Markov chain $\mathcal{M}$ with state space Ω^+ (similarly, Ω^-) and that makes transitions according to the description in Section 3. The chain corresponds to a random walk on the boolean hypercube truncated by a hyperplane. It is easy to show that all pairs of states in Ω^+ (similarly, Ω^-) can communicate. To move from node $\vec{p} \in \Omega^+$ to $\vec{q} \in \Omega^+$, first add to $\vec{p}$ all bits i in $\vec{q}$ and not in $\vec{p}$ that correspond to positions where $\beta_i \, h_i(x) \geq 0$. Then delete from $\vec{p}$ all bits i in $\vec{p}$ and not in $\vec{q}$ that correspond to positions where $\beta_i \, h_i(x) < 0$. Then delete the unnecessary "positive bits" and add the necessary "negative bits". It is easy to see that all states between $\vec{p}$ and $\vec{q}$ are in Ω^+. Thus $\mathcal{M}$ is irreducible and hence ergodic by Lemma 1.

As before, we let B be an upper bound[16] on the sum of all updates made on any pruning. Then it is straightforward to adapt Lemma 8's proof to bound f and $\hat{f}$ for WM when applying Section 3's procedure to estimate $W(\alpha)$. But when applying Equation (1), we must determine $W(\alpha_1) = W(1) = |\Omega^+|$. This is equivalent to counting the number of solutions to a 0-1 knapsack problem, which is it is #P-complete. Thus in order to complete our computation of $\hat{W}$, we must also estimate $|\Omega^+|$. We do this by mapping the problem to the knapsack problem which is summarized in Section 2.4 and shown to have an FPRAS by Morris and Sinclair [26]. If we let the "weight" of item i in our problem be $w_i = \beta_i \, h_i(\vec{x})$ for an example $\vec{x}$, then the only difference between the two problems is that the weights in the $|\Omega^+|$ estimation problem may be negative. We now argue that they are still equivalent and thus we can directly apply the results of Morris and Sinclair. Given a vector $\vec{p} \in \{0,1\}^n$ and a weight vector $\vec{w}$, let $p'_i = 1 - p_i$ if $w_i < 0$ and $p'_i = p_i$ otherwise. Also, let $w'_i = |w_i|$ and $b' = \sum_{w_i < 0} |w_i|$. It is easy to argue that $\sum_{i=1}^n w_i \, p_i \geq 0$ (which is the definition[17] of Ω^+) if and only if $\sum_{i=1}^n w'_i \, p'_i \geq b'$ (which is an instance of the knapsack problem). If we let $s^+ = \sum_{w_i > 0, p_i = 1} w_i$, $s^- = \sum_{w_i < 0, p_i = 1} w_i$, and $s^{'-} = \sum_{w_i < 0, p_i = 0} w_i$, then $b' = s^{'-} - s^-$. This is exactly what is added to both sides of the inequality $\sum_{i=1}^n w_i \, p_i \geq 0$ to get $\sum_{i=1}^n w'_i \, p'_i \geq b'$. Thus we can efficiently estimate $|\Omega^-|$ to within a factor of ϵ, allowing us to apply Corollary 3.

Corollary 29 *When applying WM to learn a weighted combination of ensemble prunings, let the sample size $S = \lceil 130 \, re^2/\epsilon^2 \rceil$, $|\Omega^+|$ (likewise, $|\Omega^-|$) be estimated to within $\epsilon/2$ of its true value with probability $\geq 3/4$ via the procedure outlined above, and $\mathcal{M}$ be simulated long enough for each sample such that the variation distance between the empirical distribution and π_{α_i} is $\leq \epsilon/(10e^2 r)$. Then $\hat{W}(\alpha)$ satisfies*

$$\Pr\left[(1-\epsilon)\, W(\alpha) \leq \hat{W}(\alpha) \leq (1+\epsilon)\, W(\alpha)\right] \geq 1/2 \ .$$

In addition, the $1/2$ can be made arbitrarily close to $1 - \delta$ for any $\delta > 0$.

Note that if $W^+/W^- \notin [\frac{1-\epsilon}{1+\epsilon}, \frac{1+\epsilon}{1-\epsilon}]$ for all trials, then our estimates of W^+ and W^- are (with probability at least $1 - \delta'_t$ for trial t) sufficiently accurate to correctly determine whether or not $W^+ > W^-$. Setting $\delta'_t = \delta/2^t$ yields a total probability of failure of at most[18] $\sum_{t=1}^{\infty} \delta/2^t = \delta$. Thus under these conditions, our version of WM runs identically to the brute-force version, and we can apply WM's mistake bounds. This yields the following corollary.

[16] In contrast to Section 4.1, where B is by definition upper bounded by Winnow's or Perceptron's mistake bound, for this application B could be arbitrarily large since it depends on the predictions of arbitrary hypotheses. Thus for the rest of this section we implicitly assume that B is polynomial in all relevant parameters, i.e. that it is expressed in unary.

[17] By negating all w_i, we can use the same arguments to estimate $|\Omega^-|$.

[18] Recall from the proof of Theorem 2 that only $O(\log 1/\delta')$ runs of the estimation procedure are needed to reduce the probability of failure to δ'.

Corollary 30 *Using the assumptions of Corollary 29, if* $W^+/W^- \notin [\frac{1+\epsilon}{1-\epsilon}, \frac{1-\epsilon}{1+\epsilon}]$ *for all* t*, then with probability at least* $1-\delta$*, the number of prediction mistakes made by this algorithm on any sequence of examples is* $O(\nu+n)$*, where* n *is the number of hypotheses in the ensemble and* ν *is the number of mistakes made by the best pruning.*

Now we bound the mixing time. Bounding g is easy, since when viewed as multisets, $\vec{p} \cup \vec{q} = \vec{a}' \cup \vec{\eta}_e(\vec{p},\vec{q})$, which implies $z_{\vec{p}} + z_{\vec{q}} = z_{\vec{a}'} + z_{\vec{\eta}_e(\vec{p},\vec{q})}$ and $\pi^+_\alpha(\vec{p})\,\pi^+_\alpha(\vec{q}) = \pi^+_\alpha(\vec{a}')\,\pi^+_\alpha(\vec{\eta}_e(\vec{p},\vec{q}))$. Thus $g=1$. Further, if $\vec{p}$ and $\vec{q}$ are neighbors that differ in bit i, then

$$\begin{aligned} z_{\vec{p}} - z_{\vec{q}} &= \sum_{\vec{x}\in M}\left(\sum_{j:p_j=1} \ell_{\vec{x}}\,\beta_j\,h_j(\vec{x}) - \sum_{j:q_j=1} \ell_{\vec{x}}\,\beta_j\,h_j(\vec{x})\right) \\ &\le \beta_i \sum_{\vec{x}\in M} \ell_{\vec{x}}\,h_i(\vec{x}) \ , \end{aligned}$$

which implies that for any neighbors $\vec{p}$ and $\vec{q}$, $\max\{\pi_\alpha(\vec{p})/\pi_\alpha(\vec{q}), \pi_\alpha(\vec{q})/\pi_\alpha(\vec{p})\} \le \alpha^{B_{\max}}$ where $B_{\max} = \max_j \left\{\beta_j \sum_{\vec{x}\in M} \ell_{\vec{x}}\,h_j(\vec{x})\right\}$.

Corollary 31 *When learning a weighted combination of ensemble prunings with Weighted Majority, if* $\Omega = \{0,1\}^n$*, then a simulation of* $\mathcal{M}$ *that starts at any node and is of length*

$$T_i = 2n^2\alpha_i^{B_{\max}}\left(n\ln 2 + n(B_{\max} - B_{\min})\ln\alpha_i + \ln 1/\epsilon'\right)$$

will draw samples from $\hat{\pi}_{\alpha_i}$ *such that* $\|\hat{\pi}_{\alpha_i} - \pi_{\alpha_i}\| \le \epsilon'$*.*

Proof: We substitute our bounds of g and h directly into Theorem 5 and note that $W(\alpha_i) \le 2^n\alpha_i^{nB_{\max}}$ and $w_{\alpha_i,\vec{p}} \ge \alpha_i^{nB_{\min}}$, completing the proof. □

The above mixing time bound is only polynomial if $B_{\max}$ is logarithmic in all relevant parameters, which is unlikely. However, our analysis is handicapped by worst-case assumptions, as with Corollary 14. While it is unlikely that an efficient bound on the mixing time exists for general ensembles with arbitrary classifiers and arbitrary distributions over examples, it is open whether restricted cases could have better bounds. Further, in Section 5 we show that in practice, our algorithm performs much better than the worst-case theoretical results imply, especially considering that highly accurate estimates of the weighted sums are not needed so long as we know whether or not $W^+ > W^-$.

Note that in Corollary 31 we assume that $\Omega = \{0,1\}^n$, i.e. that all prunings classify $\vec{x}$ as positive (or negative), and the chain is an untruncated hypercube. This is because without such an assumption we cannot guarantee that a canonical path between two nodes does not leave Ω at any time. A new approach, employing *balanced, almost-uniform permutations* has recently been pioneered by Morris and Sinclair [26] and applied to the truncated boolean hypercube problem when the chain's stationary distribution is uniform (i.e. for counting the number of solutions to the knapsack problem). It is reasonable that their technique could be generalized to the case of a non-uniform distribution, which would allow us to consider truncated hypercubes for WM and for other algorithms.

4.3 Discussion

In examining the results and proofs related to using Winnow on DNF, we see some interesting differences. Recall from Theorem 5 that there are two functions that must be bounded in order to bound a chain's mixing time: g, which bounds the ratio of $\pi(\vec{p})\pi(\vec{q})$ to $\pi(\vec{a}')\pi(\vec{\eta}_e(\vec{p},\vec{q}))$, and h, which bounds the ratio of weights of neighboring nodes in the chain. Linear $p(\vec{x})$ allows us to bound g with a polynomial since the z's in the exponent of α all cancel out due to the linear nature of the weight updates. In contrast, logistic $p(\vec{x})$ (and probably binary $p(\vec{x})$) in the worst case get charged a multiplicative factor of α for each update, yielding g that is exponential in the mistake bound. However, when bounding h, linear $p(\vec{x})$ allows the difference in the z's to grow with the mistake bound, while logistic $p(\vec{x})$ bounds this by a constant. Further, in the worst case we cannot bound $|M|$ for linear, but we can for logistic.

A more fundamental difference arises when comparing multiplicative weight update algorithms (Winnow and WM) to the additive weight update algorithm (Perceptron). The additive weight updates prevent $\mathcal{M}$'s stationary distribution from deviating very far from uniform (when compared to the MWU algorithms). Thus for Perceptron, $\mathcal{M}$ mixes rapidly, even for the logistic case. However, Lemma 21's bound on the number of mistakes (updates) Perceptron will make in the worst case is exponential, which is corroborated by Khardon et al.'s lower bound [16]. But despite the large bounds on mixing time, we note that our experiments on simulated data (Section 5) indicate that MWU algorithms simulated with MCMC may do well in practice, especially considering that highly accurate estimates of the weighted sums are not needed so long as we know which side of the threshold the sum lies on. Further, an average case analysis of Winnow on restricted classes of functions might result in polynomial mixing time bounds.

A natural extension of the results of Section 4.2 is to generalize the results to multiclass predictions. This comes for free if the boosting algorithm used is AdaBoost.MH from Schapire and Singer [32], which reduces the multiclass boosting problem to a set of binary ones, allowing our results to fit into this framework. Alternatively, AdaBoost.M1 from Freund and Schapire [9] more directly addresses the multiclass problem by having each hypothesis predict its confidence that $\vec{x}$ belongs to class j. Then the ensemble's prediction is the class that maximizes these confidence-rated predictions. That is, each class is tested individually and the one that scores the highest is the predicted class for $\vec{x}$. To adapt our framework to this, rather than simply estimating W^+ and W^-, we estimate W^j for each class j and then predict the class with the maximum. Since each W^j estimate uses a separate boolean hypercube truncated by a single hyperplane (the one that separates prunings that predict class j from those predicting another class), we can bound the mixing time using the same machinery developed in Section 4.2, assuming a version of Theorem 5 exists for truncated cubes.

Since one of the goals of pruning an ensemble of classifiers is to reduce its size, one may adopt one of several heuristics, such as choosing the pruning that has highest weight in WM, the highest ratio of weight to size, or the highest product of weight and diversity, where diversity is measured by e.g. Kullback-Leibler divergence (see Margineantu and Dietterich [24]). Let $f(\vec{p})$ be the function that one wants to maximize. Then the goal is to find the $\vec{p} \in \{0,1\}^n$ that approximately maximizes f. To do this one can define a new Markov chain $\mathcal{M}'$ whose transition probabilities are the same as for $\mathcal{M}$ in Section 3 except that Step 3 is irrelevant (since there is no training example $\vec{x}$) and in Step 4, change the transition probability to $\min\left\{1, \rho^{f(\vec{p}') - f(\vec{p})}\right\}$. The parameter ρ governs the shape of the stationary distribution: $\rho = 1$ implies a uniform distribution over all prunings, while a large value of ρ yields a distribution that peaks at prunings with large $f(\vec{p})$. (This is a special case of simulated annealing [17] where the temperature is held constant.) Lemma 1 obviously holds for $\mathcal{M}'$, but it is an open problem to bound how far from optimal its solution will be. Of course, other combinatorial optimization methods such as genetic algorithms can also be applied here.

Similarly, one issue with our DNF algorithm is that after training, we still require the training examples and running $\mathcal{M}$ to evaluate the hypothesis on a new example. In lieu of this, one can, after training, search (using a modified chain or a GA as described above) for the terms with the largest weights in Winnow. The result is a set of rules, and the prediction on a new example can be a thresholded sum of weights of satisfied rules, using the same threshold θ. The only issue then is to determine how many terms to select. Since each example satisfies exactly 2^n terms, for an example to be classified as positive, the average weight of its satisfied terms must be at least $\theta/2^n$. Thus one heuristic is to choose as many terms as possible with weight at least $\theta/2^n$, stopping when we find “too many” (as specified by a parameter) terms with weight less than $\theta/2^n$. Using this pruned set of rules, no additional false positives will occur, and in fact the number might be reduced. The only concern is causing extra false negatives.

5 Empirical results

5.1 Learning DNF formulas

In our DNF experiments, we defined the instance space to be $\mathcal{X} = \{1, 2\}^n$ and the set of terms to be $\mathcal{P} = \{0, 1, 2\}^n$, i.e. $k_i = 2$ for all i. Recall that a term $\vec{p} = (p_1, \ldots , p_n) \in \mathcal{P}$ is satisfied by example $\vec{x} = (x_1, \ldots , x_n) \in \mathcal{X}$ iff $\forall\, p_i > 0$, $p_i = x_i$. So $p_i = 0 \Rightarrow x_i$ is irrelevant for term $\vec{p}$ and $p_i > 0 \Rightarrow x_i$ must equal p_i for $\vec{p}$ to be satisfied. Even though we do not have an analysis of its mixing time, we used binary $p(\vec{x})$ in our experiments[19]. Even though using binary $p(\vec{x})$ we could define for each new example $\Omega = \{0, 1\}^n$ (i.e. an untruncated boolean hypercube; see Section 4.1.1), we chose to use as Ω a truncated (with a single hyperplane) version of $\mathcal{P}$. We did this for two reasons. First, doing so allows us to evaluate the performance of our MCMC approach when the hypercube is truncated, which is more generally applicable and also currently lacking in theoretical results. Second, this experimental approach gives us a better idea of how quickly the time complexity of brute force grows as a function of n. Comparing this with the time of our MCMC experiments tells us the minimum value of n for which our approach is faster.

We generated random (from a uniform distribution) $K = 5$-term DNF formulas, using $n \in \{10, 15, 20\}$. So the total number of Winnow inputs was $3^{10} = 59049$, $3^{15} = 1.43 \times 10^7$, and $3^{20} = 3.49 \times 10^9$. For each value of n there were nine training/testing set combinations, each with 50 training examples and 50 testing examples. Examples were generated uniformly at random.

Table 1 gives averaged[20] results for $n = 10$, indexed by S and T (“BF” means brute-force). “GUESS” is the average error of the estimates ($= |guess - actual|/actual$). “LOW” is the fraction of guesses that were $< \theta$ when the actual value was $> \theta$, and “HIGH” is symmetric. These are

[19] When we compare actual mixing times to theoretical bounds, we will compare to theoretical bounds for logistic $p(\vec{x})$, which is a reasonable approximation to the binary case.

[20] The number of weight estimations made per row in the table varied due to a varying number of training rounds, but typically was around 3000.

Table 1: Results for $n = 10$ and r chosen as in Section 3.

S	T	**GUESS**	**LOW**	**HIGH**	**PRED**	S_{theo}	T_{theo}
100	100	0.4713	0.0000	0.1674	0.0600	2.23×10^5	1.772×10^{104}
100	200	0.1252	0.0017	0.0350	0.0533	3.16×10^6	1.777×10^{104}
100	300	0.0634	0.0041	0.0172	0.0711	1.23×10^7	1.780×10^{104}
100	500	0.0484	0.0091	0.0078	0.0844	2.11×10^7	1.781×10^{104}
500	100	0.4826	0.0000	0.1594	0.1000	2.13×10^5	1.772×10^{104}
500	200	0.1174	0.0000	0.0314	0.0600	3.60×10^6	1.778×10^{104}
500	300	0.0441	0.0043	0.0145	0.0867	2.55×10^7	1.781×10^{104}
500	500	0.0232	0.0034	0.0064	0.0800	9.16×10^7	1.784×10^{104}
BF					0.0730		

Table 2: Results for $n = 10$, $S = 100$, and $T = 300$.

r	**GUESS**	**LOW**	**HIGH**	**PRED**
5	0.1279	0.0119	0.0203	0.0844
10	0.0837	0.0095	0.0189	0.0867
15	0.0711	0.0058	0.0159	0.0800
20	0.0638	0.0042	0.0127	0.0889
BF				0.0730

the only times our algorithm deviates from brute-force. "PRED" is the prediction error on the test set, and "S_{theo}" and "T_{theo}" are S and T from Corollaries 9 and 14 that guarantee an error of GUESS given the values of r in our simulations using $\alpha = 3/2$. These latter two columns show how pessimistic the worst-case bounds are in contrast to what works in practice.

Both GUESS and HIGH are very sensitive to T but not as sensitive to S. LOW was negligible due to the distribution of weights as training progressed: the term $\vec{p}_e = \vec{0}$ (satisfied by all examples) had high weights. Since all computations started at $\vec{0}$ and the Markov chain $\mathcal{M}$ seeks out nodes with high weights, the estimates tended to be too high rather than too low. But this is less significant as S and T increase. In addition, note that PRED does not appear correlated to the accuracy of the weight guesses, and most of them are very near that for brute-force. We feel that this is coincidental, and that the only way to ensure a good hypothesis is to choose S and T sufficiently large such that GUESS, LOW, and HIGH are small, e.g. $S = 100$ and $T = 300$. For these values, training and testing with $\mathcal{M}$ was slower than brute-force by a factor of over 108.

Since the run time of our algorithm varies linearly in r, we ran some experiments where we fixed r rather than letting it be set as in Section 4.1. We set $S = 100$, $T = 300$ and $r \in \{5, 10, 15, 20\}$. The results are in Table 2. This indicates that for the given parameter values, r can be reduced a little below that which is stipulated in Section 3.

Results for $n = 15$ appear in Table 3. The trends for $n = 15$ are similar to those for $n = 10$. Brute-force is faster than $\mathcal{M}$ at $S = 500$ and $T = 1500$, but only by a factor of 16. As with

Table 3: Results for $n = 15$ and r chosen as in Section 3.

S	T	**GUESS**	**LOW**	**HIGH**	**PRED**	S_{theo}	T_{theo}
500	1500	0.0368	0.0028	0.0099	0.0700	5.01×10^7	4.112×10^{151}
500	1800	0.0333	0.0040	0.0049	0.0675	6.12×10^7	4.112×10^{151}
500	2000	0.0296	0.0035	0.0023	0.0675	7.68×10^7	4.113×10^{151}
1000	1500	0.0388	0.0015	0.0042	0.0650	4.51×10^7	4.111×10^{151}
1000	1800	0.0253	0.0006	0.0038	0.0775	1.06×10^8	4.114×10^{151}
1000	2000	0.0207	0.0025	0.0020	0.0800	1.58×10^8	4.115×10^{151}
BF					0.0800		

Table 4: Results for $n = 15$, $S = 500$, and $T = 1500$, and a training set of size (a) 50 and (b) 100.

(a)

r	**GUESS**	**LOW**	**HIGH**	**PRED**
10	0.0572	0.0049	0.0132	0.1075
20	0.0444	0.0033	0.0063	0.0756
30	0.0407	0.0022	0.0047	0.0822
BF				0.0800

(b)

r	**GUESS**	**LOW**	**HIGH**	**PRED**
10	0.0577	0.0046	0.0478	0.0511
20	0.0456	0.0032	0.0073	0.0733
30	0.0405	0.0044	0.0081	0.0689
BF				0.0356

$n = 10$, r can be reduced to speed up the algorithm, but at a cost of increasing the errors of the predictions (e.g. see Table 4(a)). We ran the same experiments with a training set of size 100 rather than 50 (the test set was still of size 50), summarized in Table 4(b). As expected, error on the guesses changes little, but prediction error is decreased.

For $n = 20$, no exact (brute-force) sums were computed since there are over 3 billion inputs. So we only examined the prediction error of our algorithm. The average prediction error over all runs was 0.11 with $S = 1000$, $T = 2000$, r set as in Section 3, and a training set of size 100. The average value of r used was 55 (range was 26–78), and the run time for $\mathcal{M}$ was over 270 times faster than brute force (brute-force was run on a small number of examples to estimate its time complexity for $n = 20$). Thus for this case our algorithm provides a significant speed advantage. When running our algorithm with a fixed value of $r = 30$ (reducing time per example by almost a factor of 2), prediction error increases to 0.1833.

In summary, even though our experiments are for small values of n, they indicate that relatively small values of S, T, and r are sufficient to minimize our algorithm's deviations from brute-force Winnow. In addition, our algorithm becomes significantly faster than that of brute-force somewhere between $n = 15$ and $n = 20$, which is small for a machine learning problem. However, our implementation is still extremely slow, taking several days or longer to finish training when $n = 20$ (evaluating the learned hypothesis is also slow). Thus we are actively working on optimizations

Table 5: Results for $n = 10$ and r chosen as in Section 3.

S	T	$\lvert\Omega^+\rvert$	X_i	$\hat{W}^+(\alpha)$	DEPARTURE
50	500	0.0423	0.00050	0.0071	0.0000
50	750	0.0332	0.00069	0.0061	0.0000
50	1000	0.0419	0.00068	0.0070	0.0000
75	500	0.0223	0.00067	0.0050	0.0000
75	750	0.0197	0.00047	0.0047	0.0000
75	1000	0.0276	0.00058	0.0055	0.0000
100	500	0.0185	0.00040	0.0047	0.0000
100	750	0.0215	0.00055	0.0050	0.0000
100	1000	0.0288	0.00044	0.0056	0.0000

to speed up learning and evaluation (see Section 6).

5.2 Ensemble pruning

For the Weighted Majority experiments, we used AdaBoost over decision shrubs (depth-2 decision trees) generated by C4.5 to learn hypotheses for an artificial two-dimensional data set (Figure 1). The target concept is a circle and the examples are distributed around its circumference, each point's distance from the circle normally distributed with zero mean and unit variance. By concentrating examples around the circular boundary and limiting each decision tree's depth, we required ensembles of multiple trees to achieve low classification error on the data. We created an ensemble of 10 classifiers and simulated WM with[21] $S \in \{50, 75, 100\}$ and $T \in \{500, 750, 1000\}$ on the set of 2^{10} prunings and compared the values computed for Equation (1) to the true values from brute-force WM. The results are in Table 5: "$|\Omega^+|$" denotes the error of our estimates of $|\Omega^+|$, "X_i" denotes the error of our estimates of the ratios $W^+(\alpha_i)/W^+(\alpha_{i-1})$, and "$\hat{W}^+(\alpha)$" denotes the error of our estimates of $W^+(\alpha)$. Finally, "DEPARTURE" indicates our algorithm's departure from brute-force WM, i.e. in these experiments our algorithm perfectly emulated brute-force. Finally, we note that other results show that for $n = 30$, $S = 200$, and $T = 2000$, our algorithm takes about 4.5 hours/example to run, while brute-force takes about 2.8 hours/example. Thus we expect our algorithm to run faster than brute-force at about $n = 31$ or $n = 32$.

6 Conclusions and future work

MWU algorithms are particularly useful when the number of inputs is very large, since their mistake bounds are logarithmic in the total number of inputs. However, only in specific cases is it known how to compute the weighted sum of inputs efficiently to exploit this attribute efficiency. We presented a general, widely-applicable method based on Markov chain Monte Carlo to es-

[21] The estimation of $|\Omega^+|$ required an order of magnitude larger values of S and T than did the estimation of the ratios to get sufficiently low error rates.

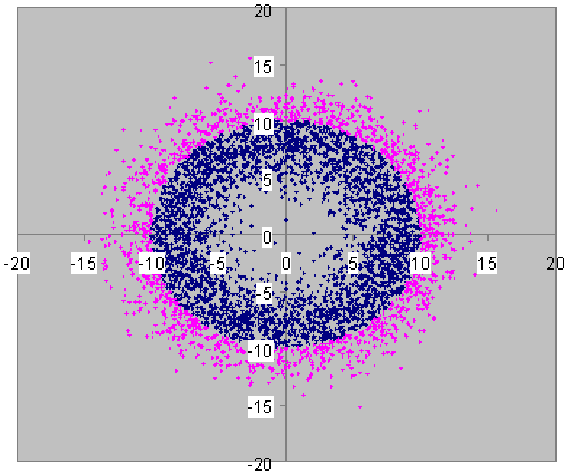

Figure 1: The circle data set.

timate these weighted sums, along with theoretical and empirical analyses of these methods as applied to learning DNF formulas and pruning ensembles of classifiers. Our theoretical results do not yield efficient algorithms for these problems, but they do provide machinery for potentially conducting average-case analyses of these algorithms on e.g. restricted classes of DNF and/or on restricted distributions. Further, as a heuristic, our methods show promise: our experimental results on simulated data, our algorithms perform much better than the worst-case theoretical results imply, especially considering that highly accurate estimates of the weighted sums are not needed so long as we know which side of the threshold the sum lies on.

Ongoing work (submitted) includes a more thorough empirical analysis of this method to speed it up further. Some of our tests include experimenting with other sampling methods besides the Metropolis sampler [25] of Section 3, utilizing “early stopping” methods that stop sampling early when it is known what side of the threshold the weighted sum will fall, and parallelizing the independent Markov chain simulations when drawing samples. We are also applying our results to accelerate an implementation of the algorithm of Goldman et al. [11], which has shown promise in multiple-instance learning. Further, we are looking for other problems for which an MWU algorithm is applicable with an exponential number of inputs. The key is to map the set of inputs to a hypercube (perhaps truncated by a hyperplane), and use that space as the set of states for the Markov chain.

When Morris and Sinclair [26] solved the knapsack problem, they also generalized their result to a hypercube truncated by multiple hyperplanes, rather than the single one that we consider in Section 4.2. Since the stationary distribution of their chain was assumed uniform, a natural question to ask is whether their results can be generalized to non-uniform distributions, and if there are applications of this generalization to learning problems.

There is also the question of how to elegantly choose S and T for empirical use to balance time complexity and precision. While it is important to accurately estimate the weighted sums in order to properly simulate WM and Winnow, some imperfections in simulation can be handled since incorrect simulation decisions can be treated as noise, which Winnow and WM can tolerate. Ideally, the algorithms would intelligently choose S and T based on past performance, perhaps (for Winnow) utilizing the brute-force upper bound of $\alpha\,\theta$ on all weights in a brute-force execution (since no promotions can occur past that point). So $\forall\,\vec{p}$, $z(\vec{p}) \leq 1 + \lfloor \log_{\alpha} \theta \rfloor$. If this bound is exceeded during a run of Winnow, then one can increase S and T and run again.

Acknowledgments

The authors thank Mark Jerrum and Alistair Sinclair for their discussions, Jeff Jackson, Qingping Tao, and the COLT reviewers for their helpful comments and Jeff Jackson for presenting an earlier version of this paper at COLT. This work was supported in part by NSF grants CCR-0092761 and CCR-9877080 with matching funds from UNL-CCIS and a Layman grant, and was completed in part utilizing the Research Computing Facility of the University of Nebraska-Lincoln. Deepak Chawla performed this work at the University of Nebraska.

References

[1] A. Blum, P. Chalasani, and J. Jackson. On learning embedded symmetric concepts. In *Proc. 6th Annu. Workshop on Comput. Learning Theory*, pages 337–346. ACM Press, New York, NY, 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 Twenty-sixth ACM Symposium on Theory of Computing*, 1994.

[3] N. H. Bshouty, J. Jackson, and C. Tamon. More efficient PAC-learning of DNF with membership queries under the uniform distribution. In *Proceedings of the Twelfth Annual Conference on Computational Learning Theory*, pages 286–295, 1999.

[4] Nader H. Bshouty. Simple learning algorithms using divide and conquer. In *Proc. 8th Annu. Conf. on Comput. Learning Theory*, pages 447–453. ACM Press, New York, NY, 1995.

[5] N. Cesa-Bianchi, Y. Freund, D. Helmbold, D. Haussler, R. Schapire, and M. Warmuth. How to use expert advice. *J. of the ACM*, 44(3):427–485, 1997.

[6] D. Chawla, L. Li, and S. D. Scott. Efficiently approximating weighted sums with exponentially many terms. In *Proceedings of the Fourteenth Annual Conference on Computational Learning Theory*, pages 82–98, 2001.

[7] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge University Press, 2000.

[8] M. Dyer, A. Frieze, R. Kannan, A. Kapoor, and U. Vazirani. A mildly exponential time algorithm for approximating the number of solutions to a multidimensional knapsack problem. *Combinatorics, Prob. and Computing*, 2:271–284, 1993.

[9] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and Systems Sciences*, 55(1):119–139, 1997.

[10] C. Gentile and M. K. Warmuth. Linear hinge loss and average margin. In *Advances in Neural Information Processing Systems 11*, pages 225–231, 1998.

[11] S. A. Goldman, S. K. Kwek, and S. D. Scott. Agnostic learning of geometric patterns. *Journal of Computer and System Sciences*, 6(1):123–151, February 2001.

[12] S. A. Goldman and S. D. Scott. Multiple-instance learning of real-valued geometric patterns. *Annals of Mathematics and Artificial Intelligence*, in press. Early version in technical report UNL-CSE-99-006, University of Nebraska.

[13] D. P. Helmbold and R. E. Schapire. Predicting nearly as well as the best pruning of a decision tree. *Machine Learning*, 27(1):51–68, 1997.

[14] M. Jerrum and A. Sinclair. The Markov chain Monte Carlo method: An approach to approximate counting and integration. In D. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, chapter 12, pages 482–520. PWS Pub., 1996.

[15] M. R. Jerrum, L. G. Valiant, and V. V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science*, 43:169–188, 1986.

[16] R. Khardon, D. Roth, and R. Servedio. Efficiency versus convergence of boolean kernels for online learning algorithms. In *Advances in Neural Information Processing Systems 14*, 2001.

[17] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[18] J. Kivinen, M. K. Warmuth, and P. Auer. The perceptron algorithm vs. Winnow: Linear vs. logarithmic mistake bounds when few input variables are relevant. *Artificial Intelligence*, 97(1-2):325–343, 1997.

[19] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

[20] N. Littlestone. From on-line to batch learning. In *Proc. 2nd Annu. Workshop on Comput. Learning Theory*, pages 269–284. Morgan Kaufmann, 1989.

[21] N. Littlestone. Redundant noisy attributes, attribute errors, and linear threshold learning using Winnow. In *Proc. 4th Annu. Workshop on Comput. Learning Theory*, pages 147–156, San Mateo, CA, 1991. Morgan Kaufmann.

[22] N. Littlestone and M. K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, 1994.

[23] W. Maass and M. K. Warmuth. Efficient learning with virtual threshold gates. *Information and Computation*, 141(1):66–83, 1998.

[24] D. D. Margineantu and T. G. Dietterich. Pruning adaptive boosting. In *Proc. 14th International Conference on Machine Learning*, pages 211–218. Morgan Kaufmann, 1997.

[25] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculation by fast computing machines. *J. of Chemical Physics*, 21:1087–1092, 1953.

[26] B. Morris and A. Sinclair. Random walks on truncated cubes and sampling 0-1 knapsack solutions. *SIAM Journal on Computing*, to appear. Early version in FOCS 99.

[27] K.-R. Muller, S. Mika, G. Ratsch, K. Tsuda, and B. Schölkopf. An introduction to kernel-based learning methods. *IEEE Transactions on Neural Networks*, 12(2):181–201, 2001.

[28] F. Pereira and Y. Singer. An efficient extension to mixture techniques for prediction and decision trees. *Machine Learning*, 36(3):183–199, September 1999.

[29] J. R. Quinlan. Bagging, boosting, and C4.5. In *Proceedings of the Thirteenth National Conference on Aritificial Intelligence*, pages 725–730, 1996.

[30] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psych. Rev.*, 65:386–407, 1958. (Reprinted in *Neurocomputing* (MIT Press, 1988).).

[31] R. E. Schapire, Y. Freund, P. Bartlett, and W. S. Lee. Boosting the margin: a new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 26(5):1651–1686, 1998.

[32] R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999.

[33] B. Schölkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2002.

[34] A. Sinclair. Improved bounds for mixing rates of Markov chains and multicommodity flow. *Combinatorics, Probability and Computing*, 1:351–370, 1992.

[35] E. Takimoto and M. Warmuth. Predicting nearly as well as the best pruning of a planar decision graph. In *Proc. of the Tenth International Conference on Algorithmic Learning Theory*, 1999.

[36] C. Tamon and J. Xiang. On the boosting pruning problem. In *Proceedings of the Eleventh European Conference on Machine Learning*, pages 404–412. Springer-Verlag, 2000.