2004

An Analysis of MCMC Sampling Methods for Estimating Weighted Sums in Winnow

Qingping Tao
University of Nebraska-Lincoln, qtao@cse.unl.edu

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

Technical Report TR-UNL-CSE-2004-0007
Dept. of Computer Science, University of Nebraska

An Analysis of MCMC Sampling Methods for Estimating Weighted Sums in Winnow *

Qingping Tao (qtao@cse.unl.edu) and Stephen D. Scott (sscott@cse.unl.edu)
Department of Computer Science & Engineering, University of Nebraska, Lincoln, NE 68588-0115 USA

April 1, 2004

Abstract. Chawla et al. introduced a way to use the Markov chain Monte Carlo method to estimate weighted sums in multiplicative weight update algorithms when the number of inputs is exponential. But their algorithm still required extensive simulation of the Markov chain in order to get accurate estimates of the weighted sums. We propose an optimized version of Chawla et al.'s algorithm, which produces exactly the same classifications while often using fewer Markov chain simulations. We also apply three other sampling techniques and empirically compare them with Chawla et al.'s Metropolis sampler to determine how effective each is in drawing good samples in the least amount of time, in terms of accuracy of weighted sum estimates and in terms of Winnow's prediction accuracy.

Keywords: DNF, Winnow, Markov Chain Monte Carlo

1. Introduction

Chawla et al. (2004) introduced the use of the Markov chain Monte Carlo (MCMC) method to estimate weighted sums in multiplicative weight update (MWU) algorithms when the number of inputs is exponential. One of their applications was using Littlestone's (1988) Winnow algorithm to learn DNF formulas. Although their preliminary empirical results are much stronger than what their theoretical results implied, they still required extensive simulation of the Markov chain to draw "good" samples (i.e. from close to the chain's stationary distribution) in order to get accurate estimates of the weighted sums. This significantly slowed their algorithm.

We propose an optimized version of Chawla et al.'s algorithm, which often uses less computation time without any loss in classification accuracy. We give two theorems to prove the correctness of our algorithm. We also give lower bounds on how much computation time our algorithm will save.

In our experiments, we empirically compare three MCMC sampling techniques (Gibbs, Metropolized Gibbs and parallel tempering) to Chawla et al.'s Metropolis sampler to determine how effective each is in quickly drawing good samples, in terms of accuracy of weighted sum estimates and in terms of Winnow's prediction accuracy. The experimental results show that

* An early version of this paper appeared as Tao and Scott (2003).

Metropolis sampler has worse performance than Gibbs and Metropolized Gibbs on estimating weighted sums. Gibbs shows better performance than Metropolized Gibbs when the number of variables is large, though there is little difference between them when the number of variables is small. For prediction errors, there is little difference between any MCMC techniques. Also, on the data sets we experimented with, we discovered that all approximations of Winnow have no disadvantage over brute force Winnow (i.e. when weighted sums are computed exactly). Thus generalization accuracy is not compromised by the approximation.

We also extend Chawla et al.'s algorithm to handle generalized (non-boolean) inputs and multi-class outputs. These results are critical in applying MCMC methods to other applications of MWU algorithms with exponentially large feature spaces. For example, the Winnow-based algorithm of Tao and Scott (2004) (adapted from Goldman et al. (2001)) for learning concepts from a generalization of the multiple-instance model (Dietterich et al., 1997)) is efficient for low dimensions, but does not scale well. It is possible that Chawla et al.'s MCMC-based approach can be very useful to make this algorithm (and others) more scalable, but first a thorough empirical analysis of the sampling method is required.

The rest of our paper is as follows. In Section 2 we discuss related work. Section 3 describes Chawla et al.'s MCMC approach for estimating weighted sums and presents our optimized version. Section 4 gives four different MCMC sampling techniques. Section 5 gives several extensions of the basic Winnow algorithm. We then report our experimental results in Section 6 and conclude in Section 7.

2. Related Work

2.1. Learning DNF Formulas Using Winnow

Let $f = P_1 \vee P_2 \vee \ldots \vee P_K$ be the target function, where $P_i = c_{i1} \wedge c_{i2} \wedge \ldots \wedge c_{in}$ is a term and c_{ij} is a constraint on the value of attribute j. If we let attribute j take on values from $\{1, \ldots, k_j\}$, then $c_{ij} = \ell \in \{1, \ldots, k_j\}$ means that for an example $\mathbf{x}$ to satisfy constraint c_{ij}, $x_j = \ell$. If $c_{ij} = 0$, then x_j can be any value from $\{1, \ldots, k_j\}$ and still satisfy the constraint. If $x_j = 0$, then this attribute value is unspecified and only satisfies a "don't care" constraint of $c_{ij} = 0$. In other words, $\mathbf{x}$ satisfies P_i iff for all j, either $x_j = c_{ij}$ or $c_{ij} = 0$. Thus the set of possible terms available for f and the instance space are both $\Omega = \prod_{j=0}^{n-1} \{0, \ldots, k_j\}$. It is easily seen that if example $\mathbf{x}$ has $n_{\mathbf{x}}$ specified values (i.e. $n_{\mathbf{x}}$ values > 0), then there are exactly $2^{n_{\mathbf{x}}}$ terms satisfied by it. The problem of learning conventional DNF formulas (i.e. $k_j = 2$ for all j) has been heavily studied in a learning-theoretic framework, e.g. Bshouty

et al. 1999; Khardon et al. 2001, but positive results exist only in restricted cases, and the general DNF problem remains open[1].

The algorithm Winnow of Littlestone (1988) is a linear threshold learner that uses multiplicative updates to change its weights. Winnow is an *on-line* learning algorithm, which means that learning proceeds in *trials*. At trial t, Winnow receives an input vector $\mathbf{x}'_t$ and makes its prediction $\hat{y}_t = 1$ if $W_t = \mathbf{w}_t \cdot \mathbf{x}'_t \geq \theta$ and 0 otherwise, where $\mathbf{w}_t$ is its weight vector at trial t and θ is the threshold. Then Winnow is told the true label y_t and updates its weight vector as follows: $w_{t+1,i} = w_{t,i}\alpha^{x'_{t,i}(y_t - \hat{y}_t)}$ for some learning rate $\alpha > 1$. If $w_{t+1,i} > w_{t,i}$, we call it a *promotion* and if $w_{t+1,i} < w_{t,i}$, we call it a *demotion*. Littlestone showed that if the target concept labelling the examples can be represented as a monotone disjunction of K of the N total inputs to Winnow (i.e. a disjunction where none of the K inputs are negated in the target function), then Winnow can exactly learn the target concept while making at most $O(K \log N)$ prediction mistakes. Hence Winnow can be used to learn DNF formulas by using all possible terms as its inputs: E.g. if $k_i = 2$ for all i, there will be $N = 3^n$ possible terms, where n is the number of variables in original input vector $\mathbf{x}$. For a particular instance $\mathbf{x}$, input x'_i to Winnow is 1 if $\mathbf{x}$ satisfies term i and 0 otherwise. The number of mistakes that Winnow will make on any sequence of examples is then $O(Kn)$, where K is the number of terms in the target disjunction.

Of course, the algorithm as described above is not efficient, since brute force computation of $W_t = \mathbf{w}_t \cdot \mathbf{x}'_t$ takes time $\Omega(N)$, which is exponential[2] in n. Thus another approach is needed to compute W_t. One possibility is to use kernels, as illustrated by Khardon et al. (2001) for the Perceptron algorithm (i.e. using additive weight updates). However, while they showed that it is possible to efficiently compute the weighted sum for Perceptron when learning DNF, they also showed that in the worst case, their kernel-based algorithm makes $2^{\Omega(n)}$ prediction mistakes. They also argued that unless P = #P, it is impossible to efficiently exactly simulate Winnow for learning DNF. Thus we look to Chawla et al. (2004), who use MCMC methods to *estimate* W_t for Winnow *with high probability*, as opposed to Khardon et al.'s hardness result that says no *deterministic* simulation of Winnow is possible for DNF. While Chawla et al. do not guarantee an efficient DNF algorithm in the learning-theoretic sense, their results yield an effective heuristic for learning DNF. This is in part due to the fact that in order to correctly simulate Winnow, it is not required that the estimate $\hat{W}_t$ be close to W_t, but only that it be *on the same side of the threshold* θ as W_t.

[1] It is unlikely that an efficient distribution-free (PAC) DNF-learning algorithm exists (Blum et al., 1994; Blum et al., 1993).

[2] Note that storing all weights takes $\Omega(N)$ space, but we can compute the weight of a particular term P by evaluating it on the training set, eliminating the need to store the weights.

2.2. MARKOV CHAIN MONTE CARLO METHODS

The Markov chain Monte Carlo method uses a Markov chain to simulate Monte Carlo experiments that provide approximations to quantities by performing statistical sampling experiments. Starting with the work of Metropolis et al. (1953) and Geman and Geman (1984), MCMC methods have been widely used to solve problems in statistical physics and Bayesian statistical inference. One major class of these problems is approximate summation, whose goal is to approximate the sum $W = \sum_{x \in \Omega} w(x)$, where w is a positive function defined on Ω that is a large but finite set of combinatorial structures.

Generally a Markov chain M with state space Ω and stationary distribution π is designed to be *ergodic*, that is, the probability distribution over Ω converges asymptotically to π regardless of the initial state. Then M is repeatedly simulated for T steps and generates samples almost according to π. These S samples are then used to estimate the quantity of interest. Usually we discard states in the first T_0 steps and assume there would be a rapid convergence to π during these steps. This T_0-step procedure is called *burn-in*.

After burn-in, M then would return the element at the state as a sample from the empirical distribution $\hat{\pi}$ at the end of the simulation. Then this process would be restarted S times, which means $T = ST_0$. Rather than repeatedly restarting the whole process, the more common technique is to sample from a single run of the process, which means the Markov chain would be simulated for another S steps after burn-in, each additional step generating a new sample. Thus the total number of steps is $T = T_0 + S$. One benefit of multiple runs is that samples are completely independent, but this requires more computation. Instead, a single run only needs a single burn-in procedure and the samples drawn from it would be good enough if the chain converged to π rapidly. Since efficiency is essential in our experiments, we use a single run rather than multiple runs.

Two well-studied problems with MCMC solutions are the approximate knapsack problem and the problem of approximating the sum of the weights of a weighted matching in a graph (e.g. Jerrum and Sinclair (1996)). Chawla et al. (2004) combined these two sloultions and gave a Metropolis sampler (Metropolis et al., 1953) to approximate the weighted sum in Winnow for learning DNF[3]. Then they evaluated this algorithm on simple simulated data sets. We extend their work by adding optimizations (Section 3.3) and applying three other sampling techniques: Gibbs, Metropolized Gibbs and parallel tempering.

[3] They also used a similar technique to approximate weighted sums in Weighted Majority (Littlestone and Warmuth, 1994) of classifiers created by boosting (Schapire and Singer, 1999) for predicting nearly as well as the best pruning of an ensemble.

3. Estimating Weighted Sums with MCMC

3.1. CHAWLA ET AL.'S MCMC SOLUTION FOR WINNOW

We now describe Chawla et al.'s (2004) MCMC solution for estimating $W_t(\alpha)$, where $W_t(\alpha) = \sum_{P \in \Omega_t} w_{t,P}(\alpha)$, $w_{t,P}(\alpha) = \alpha^{\varpi_t(P)}$ is term P's weight of training[4] Winnow with learning rate α, and $\varpi_t(P) = u_t(P) - v_t(P)$, where $u_t(P)$ is the total number of promotions of term P at time t and $v_t(P)$ is the total number of demotions.

Let $\Omega'_t \subseteq \Omega$ be the set of 2^{n_t} terms that are satisfied by example $\mathbf{x}_t$ ($n_t \leq n$ is the number of non-zero values in $\mathbf{x}_t$). For each term $P' = (p'_1, \ldots, p'_n) \in \Omega'_t$, let $P = (p_1, \ldots, p_{n_t}) \in \Omega_t$ be defined as follows:

1. delete p'_i from P' for all i such that $x_i = 0$ and call the new term P'';
2. set $p_i = 1$ if $p''_i > 0$ and $p_i = 0$ if $p''_i = 0$.

Chawla et al. then build a set of Markov chains $\mathcal{M}_t$ on the state space Ω_t that are based on Metropolis sampler (see Section 4.1). Each chain $M_t(\alpha') \in \mathcal{M}_t$ has a specific learning rate α' and a stationary distribution $\pi_{\alpha',t}(P) = w_{t,P}(\alpha')/W_t(\alpha')$.

They then define $f_{i,t}(P) = w_{t,P}(\alpha_{i-1,t})/w_{t,P}(\alpha_{i,t})$, where $\alpha_{i,t} = (1 + \frac{1}{m_t})^{i-1}$ for $1 \leq i \leq r_t$, r_t is the smallest integer such that $(1 + \frac{1}{m_t})^{r_t - 1} \geq \alpha$, and $m_t = u_t(P_e) + v_t(P_e)$ where $P_e = (0, 0, \ldots, 0)$. Then they get

$$\mathrm{E}[f_{i,t}(P)] = \sum_{P \in \Omega_t} \pi_{\alpha_{i,t},t}(P) f_{i,t}(P) = \frac{W_t(\alpha_{i-1,t})}{W_t(\alpha_{i,t})} \ .$$

So $W_t(\alpha_{i-1,t})/W_t(\alpha_{i,t})$ can be estimated by computing the sample mean of $f_{i,t}(P)$, which allows $W_t(\alpha)$ to be computed since

$$W_t(\alpha) = \left(\frac{W_t(\alpha_{r,t})}{W_t(\alpha_{r-1,t})}\right) \cdots \left(\frac{W_t(\alpha_{2,t})}{W_t(\alpha_{1,t})}\right) W_t(\alpha_{1,t})$$

and $W_t(\alpha_{1,t}) = W(1) = |\Omega_t| = 2^{n_t}$. Therefore, for each value $\alpha_{2,t}, \ldots, \alpha_{r,t}$, S_t samples are drawn from $M_t(\alpha_{i,t})$ after discarding the first $T_{i,t}$ steps. If $X_{i,t}$ is the sample mean of $f_{i,t}(P)$ and $|\mathcal{M}_t| = r_t - 1$, then Chawla et al.'s estimate of $W_t(\alpha)$ is

$$\hat{W}_t(\alpha) = 2^{n_t} \prod_{i=2}^{r_t} 1/X_{i,t} \ .$$

The following results hold for this estimation procedure.

[4] W_t is a function of α since Chawla et al. define their approximation method using several different values of $\alpha_i \in [1, \alpha]$. Note that, however, the actual sequence of updates made (i.e. the values of $\varpi_t(P)$) will be the same regardless of α_i. This single sequence of updates is determined by running the learning algorithm with the original learning rate α. Hence $w_{t,P}(\alpha_i)/w_{t,P}(\alpha_j) = (\alpha_i/\alpha_j)^{\varpi_t(P)}$.

THEOREM 1. *(*Chawla et al., 2004*) Let the sample size $S_t = \lceil 130 r_t e^2/\epsilon^2 \rceil$ and M_t be simulated long enough for each sample such that the variation distance between the empirical distribution and $\pi_{\alpha_{i,t}}$ is at most $\epsilon/(5e^2 r_t)$. Then for any $\delta > 0$, $\hat{W}_t(\alpha)$ satisfies*

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

COROLLARY 2. *(*Chawla et al., 2004*) Using the assumptions of Theorem 1, if $W_t(\alpha) \notin [\frac{\theta}{(1+\epsilon)}, \frac{\theta}{(1-\epsilon)}]$ for all t, then with probability at least $1-\delta$, the number of mistakes made by Winnow on any sequence of examples is at most $8 + 14Kn\ln(k+1)$.*

3.2. WHAT IS THE BEST CHOICE OF r_t?

According to Chawla et al.'s MCMC solution, the computation time of estimating $W_t(\alpha)$ depends on the number of chains $r_t - 1$, the number of burn-in steps T_0 and the sample size S. To reduce the computation time, we need to reduce either T_0 and S, or r_t. We could choose relatively small T_0 and S if all of the chains in $\mathcal{M}_t$ converged to their stationary distributions fast enough. This could be achieved by using a good sampler, which is discussed in Section 4. In this section, we analyze if we could choose a smaller r_t than Chawla et al.'s proposition.

Define $f'_{i,t}(P) = w_{t,P}(\alpha'_{i-1,t})/w_{t,P}(\alpha'_{i,t})$, where $\alpha'_{i,t} = \left(1+\kappa/m'_t\right)^{i-1}$ for $1 \le i \le r'_t$, r'_t is the smallest integer such that $\left(1+\kappa/m'_t\right)^{r'_t-1} \ge \alpha$, and κ and m'_t are positive constants. Obviously, $f_{i,t}(P)$ is a special case of $f'_{i,t}(P)$ when $\kappa = 1$ and $m'_t = m_t$. If we increase κ or use a smaller m'_t, we would decrease r'_t. In Theorem 4 below, we extend Theorem 1.

LEMMA 3. *For any distribution π of Ω_t, if $m'_t \ge \max\{u_t(P_e), v_t(P_e)\}$, for all i and P, $e^{-\kappa} \le f'_{i,t}(P) \le e^{\kappa}$.*

Proof. Let $\wp_t^{max} = \max_{P\in\Omega_t}\{u_t(P) - v_t(P)\}$ and $\wp_t^{min} = \min_{P\in\Omega_t}\{u_t(P) - v_t(P)\}$, i.e. the maximum and minimum number of net promotions. Since P_e is satisfied by any term and $u_t(P_e), v_t(P_e) \ge 0$, $u_t(P_e) \ge \wp_t^{max} \ge \wp_t^{min} \ge -v_t(P_e)$. Therefore, $\hat{m}_t \ge \max\{|\wp_t^{max}|, |\wp_t^{min}|\}$.

For all $P \in \Omega_t$,

$$f'_{i,t}(P) = \frac{w_{t,P}(\alpha'_{i-1,t})}{w_{t,P}(\alpha'_{i,t})} = \left(\frac{\alpha'_{i-1,t}}{\alpha'_{i,t}}\right)^{\varpi(P)} = \left(1+\frac{\kappa}{m'_t}\right)^{\varpi(P)},$$

where $\varpi(P) = u_t(P) - v_t(P)$. Therefore,

$$f'_{i,t}(P) = \left(1+\frac{\kappa}{m'_t}\right)^{\varpi(P)} \le \left(1+\frac{\kappa}{m'_t}\right)^{\wp_t^{max}} \le \left(1+\frac{\kappa}{m'_t}\right)^{m'_t} \le e^{\kappa},$$

and

$$f'_{i,t}(P) = \left(1 + \frac{\kappa}{m'_t}\right)^{\varpi(P)} \geq \left(1 + \frac{\kappa}{m'_t}\right)^{\wp_t^{min}} \geq \left(1 + \frac{\kappa}{m'_t}\right)^{-m'_t} \geq e^{-\kappa} \ . \quad \square$$

By substituting Lemma 3's bounds into Theorem 1, we get the following.

THEOREM 4. *If $m'_t \geq \max\{u_t(P_e), v_t(P_e)\}$, let the sample size $S_t = \lceil 130 e^{2\kappa} \hat{r}_t / \epsilon^2 \rceil$ and M_t be simulated long enough for each sample such that the variation distance between the empirical distribution and $\pi_{\hat{\alpha}_{i,t}}$ is at most $\epsilon/(5e^{2\kappa}\hat{r}_t)$. Then for any $\delta > 0$, $\hat{W}_t(\alpha)$ satisfies*

$$\Pr\left[(1-\epsilon)W_t(\alpha) \leq \hat{W}_t(\alpha) \leq (1+\epsilon)W_t(\alpha)\right] \geq 1 - \delta.$$

According to Theorem 4, we can set $m'_t = \max\{u_t(P_e), v_t(P_e)\}$, which is often less than Chawla et al.'s proposal of $m_t = u_t(P_e) + v_t(P_e)$ (see Section 3.1). But Theorem 4 tells us that if we increase κ by 1 , we would need almost e^2 times the sample size and e^2 times smaller variation distance. This means that the result of our MCMC solution would become worse if we reduce the number of chains without drawing more samples. Thus it seems we could not expect to get as good a result as before with less computation time by increasing κ. But Theorem 4 only gives the worst-case theoretic bounds. In practice, increasing κ might reduce computation time without dramatically affecting the performance of estimations. Also, in the next section we describe ways to use fewer chains without reducing the accuracy of our Winnow simulations.

3.3. Our Optimized MCMC Solution

In Chawla et al.'s MCMC solution, $r_t - 1$ Markov chains need to be simulated. Here we give an optimized solution that is based on the idea that to exactly simulate Winnow, we only need to know what Winnow's prediction is going to be (i.e. on what side of the threshold θ that W will fall on), not what the weighted sum exactly is. So it is possible that we could stop computing our estimate after only a subset of the chains in $\mathcal{M}_t$ has been run.

Let $\wp_t^{max} = \max_{P \in \Omega_t}\{u_t(P) - v_t(P)\}$, $\wp_t^{min} = \min_{P \in \Omega_t}\{u_t(P) - v_t(P)\}$ (i.e. the maximum and minimum number of net promotions), and $\Psi_t = \{2, \ldots, r_t\}$. Given some $\Psi' \subseteq \Psi_t$, we can define the following two conditions:

$$\mathcal{C} \prod_{i \in \Psi'} \frac{W_t(\alpha_{i,t})}{W_t(\alpha_{i-1,t})} \left(\frac{\alpha_{i,t}}{\alpha_{i-1,t}}\right)^{-\wp_t^{min}} \geq \theta, \tag{1}$$

$$\mathcal{D} \prod_{i \in \Psi'} \frac{W_t(\alpha_{i,t})}{W_t(\alpha_{i-1,t})} \left(\frac{\alpha_{i,t}}{\alpha_{i-1,t}}\right)^{-\wp_t^{max}} < \theta, \tag{2}$$

where $\mathcal{C} = 2^{n_t} \prod_{i=2}^{r_t} \left(\frac{\alpha_{i,t}}{\alpha_{i-1,t}}\right)^{\wp_t^{min}}$ and $\mathcal{D} = 2^{n_t} \prod_{i=2}^{r_t} \left(\frac{\alpha_{i,t}}{\alpha_{i-1,t}}\right)^{\wp_t^{max}}$. Now we can prove the following theorem.

THEOREM 5. *If $\exists \Psi' \subseteq \Psi_t$ that satisfies condition (1) , then $W_t(\alpha) \geq \theta$; If $\exists \Psi' \subseteq \Psi_t$ that satisfies condition (2), then $W_t(\alpha) < \theta$.*

Proof. Because $\alpha_{i,t} > \alpha_{i-1,t} > 0$ and $\varpi_t(P) - \wp_t^{min} \geq 0$ for all $P \in \Omega_t$,

$$\sum_{P \in \Omega_t} \alpha_{i,t}^{\varpi_t(P) - \wp_t^{min}} \geq \sum_{P \in \Omega_t} \alpha_{i-1,t}^{\varpi_t(P) - \wp_t^{min}}.$$

So $\frac{W_t(\alpha_{i,t})}{W_t(\alpha_{i-1,t})} \left(\frac{\alpha_{i,t}}{\alpha_{i-1,t}}\right)^{-\wp_t^{min}} \geq 1$. Then

$$W_t(\alpha) = \mathcal{C} \prod_{i=2}^{r_t} \frac{W_t(\alpha_{i,t})}{W_t(\alpha_{i-1,t})} \left(\frac{\alpha_{i,t}}{\alpha_{i-1,t}}\right)^{-\wp_t^{min}} \geq \mathcal{C} \prod_{i \in \Psi'} \frac{W_t(\alpha_{i,t})}{W_t(\alpha_{i-1,t})} \left(\frac{\alpha_{i,t}}{\alpha_{i-1,t}}\right)^{-\wp_t^{min}} \geq \theta.$$

Similarly we can prove the second statement. □

Theorem 5 tells us that it would not always be necessary to run all $r_t - 1$ Markov chains if we were only interested in Winnow's predictions. Instead, we can sometimes limit our simulations to a subset of Markov chains. So what we want is to find such a subset with the smallest size.

Let $\Gamma_1(\Psi_t)$ be the set of all Ψ' that satisfy (1), and $\Gamma_0(\Psi_t)$ be the set of all Ψ' that satisfy (2). We define $\Psi_1^{min} \in \Gamma_1(\Psi_t)$ as a *minimum 1-prediction set* if $|\Psi_1^{min}| \leq |\Psi'|$ for all $\Psi' \in \Gamma_1(\Psi_t)$, and $\Psi_0^{min} \in \Gamma_0(\Psi_t)$ as a *minimum 0-prediction set* if $|\Psi_0^{min}| \leq |\Psi'|$ for all $\Psi' \in \Gamma_0(\Psi_t)$. This leads us to Theorem 6.

THEOREM 6. *If Ψ_1^{min} exists, $\{r_t, r_t - 1, \ldots, r_t - |\Psi_1^{min}| + 1\}$ is a minimum 1-prediction set, and if Ψ_0^{min} exists, $\{2, 3, \ldots, |\Psi_0^{min}| + 1\}$ is a minimum 0-prediction set.*

Proof. Let $\beta = (1 + \frac{1}{m_t})$. Using Cauchy's inequality, we can prove that

$$\begin{aligned}
W_t(\alpha_{i+1,t}) W_t(\alpha_{i-1,t}) &= \sum_{P \in \Omega_t} \alpha_{i+1,t}^{\varpi_t(P)} \sum_{Q \in \Omega_t} \alpha_{i-1,t}^{\varpi_t(Q)} \\
&= \sum_{P \in \Omega_t} \beta^{i \cdot \varpi_t(P)} \sum_{Q \in \Omega_t} \beta^{(i-2)\varpi_t(Q)} \\
&= \sum_{P \in \Omega_t} \left(\beta^{i \cdot \varpi_t(P)/2}\right)^2 \sum_{Q \in \Omega_t} \left(\beta^{(i-2)\varpi_t(Q)/2}\right)^2 \\
&\geq \left(\sum_{P \in \Omega_t} \beta^{(i-1) \cdot \varpi_t(P)}\right)^2 \\
&= W_t(\alpha_{i,t}) W_t(\alpha_{i,t})
\end{aligned}$$

So $\frac{W_t(\alpha_{i+1,t})}{W_t(\alpha_{i,t})} \geq \frac{W_t(\alpha_{i,t})}{W_t(\alpha_{i-1,t})}$ for all $i \in \{2, \ldots, r_t - 1\}$. Now let $\Psi' = \{r_t, r_t - 1, \ldots, r_t - |\Psi_1^{min}| + 1\}$:

$$\mathcal{C} \prod_{i \in \Psi'} \frac{W_t(\alpha_{i,t})}{W_t(\alpha_{i-1,t})} \left(\frac{\alpha_{i,t}}{\alpha_{i-1,t}}\right)^{-\wp_t^{min}} \geq \mathcal{C} \prod_{i \in \Psi_1^{min}} \frac{W_t(\alpha_{i,t})}{W_t(\alpha_{i-1,t})} \left(\frac{\alpha_{i,t}}{\alpha_{i-1,t}}\right)^{-\wp_t^{min}} \geq \theta.$$

Therefore $\Psi' \in \Gamma_1(\Psi_t)$. Since $|\Psi'| = |\Psi_1^{min}|$, then $\Psi' = \{r_t, r_t - 1, \ldots, r_t - |\Psi_1^{min}| + 1\}$ is a minimum 1-prediction set. Similarly we can prove $\{2, 3, \ldots, |\Psi_0^{min}| + 1\}$ is a minimum 0-prediction set. □

According to Theorem 6, if $W_t(\alpha) \geq \theta$ and we simulate Markov chains in the order of $r_t, r_t - 1, \ldots, 2$, and halt when we find a minimum 1-prediction set, we need no more computation than any other sequence of Markov chains. Similarly to get a minimum 0-prediction set, we use no more chains than any other sequence if $W_t(\alpha) < \theta$ and we simulate them in the order of $2, 3, \cdots, r_t$. Then we get an optimized MCMC solution for Winnow as in Table I.

In Table I, we can estimate $\wp_t^{max}$ and $\wp_t^{min}$ with $u_t(P_e)$ and $-v_t(P_e)$ because $u_t(P_e) \geq \wp_t^{max} \geq \wp_t^{min} \geq -v_t(P_e)$. Then we choose one of the two orders ($\{r_t, \cdots, 2\}$ or $\{2, \cdots, r_t\}$) by guessing the most likely prediction y_t'. When we use Winnow to predict an unlabeled example, we could just assume y_t' is 1. When we are training Winnow, we can set y_t' as the class label of training example $\mathbf{x}$. But a better way is that at the tth training iteration, let $y_t' = \hat{y}_{t-1}(\mathbf{x})$, where $\hat{y}_{t-1}$ is the prediction of $\mathbf{x}$ at the $(t-1)$th iteration. The heuristic is that the weighted sum of $\mathbf{x}$ might not change too much after the last time Winnow met $\mathbf{x}$. At the beginning of training, all weights of Winnow are 1. So $W_1(\alpha) = 2_t^n$ for all examples. If $2_t^n \geq \theta$, $y_1' = 1$, otherwise 0.

Another question is how small Ψ_1^{min} and Ψ_0^{min} can be. We know that if $W_t(\alpha)$ is very close to the threshold θ, the chance for our algorithm to stop early is small. Below we give the upper bounds of the sizes of Ψ_1^{min} and Ψ_0^{min}.

THEOREM 7. *If $W_t(\alpha) \geq \theta$, let $W_t(\alpha) = (1+\epsilon)\theta$ where $\epsilon \geq 0$. Then the size of Ψ_1^{min} is at most $r_t - 1 - \ln(1+\epsilon)$; If $W_t(\alpha) < \theta$, let $W_t(\alpha) = (1-\epsilon)\theta$ where $0 < \epsilon < 1$. Then the size of Ψ_0^{min} is at most $r_t - 1 + \ln(1-\epsilon)$.*

Proof. If $W_t(\alpha) \geq \theta$, then Ψ_1^{min} exists. Let $\Psi_1^k = \{r_t, r_t - 1, \ldots, r_t - k + 1\}$ (so $|\Psi_1^k| = k$). According to Theorem 6, Ψ_1^k is a minimum 1-prediction set if k is the minimum value that makes Ψ_1^k satisfy (1).

Notice $W_t(\alpha) = W_t(\alpha_{r_t,t}) = \mathcal{C} \prod_{i=2}^{r_t} \frac{W_t(\alpha_{i,t})}{W_t(\alpha_{i-1,t})} \left(\frac{\alpha_{i,t}}{\alpha_{i-1,t}}\right)^{-\wp_t^{min}}$. Then (1) can be written as

$$\theta \leq \mathcal{C} \prod_{i=r_t-k+1}^{r_t} \frac{W_t(\alpha_{i,t})}{W_t(\alpha_{i-1,t})} \left(\frac{\alpha_{i,t}}{\alpha_{i-1,t}}\right)^{-\wp_t^{min}}$$

Table I. Optimized MCMC solution for Winnow

Given an example $\mathbf{x}$, guess Winnow's most possible prediction is y'_t, as described in the text
if y'_t is 1 **then**
  $\hat{W}_t(\alpha) \leftarrow 2^{n_t} \prod_{i=2}^{r_t} \left(\frac{\alpha_{i,t}}{\alpha_{i-1,t}}\right)^{\wp_t^{min}}$
  for $i = r_t$ to 2 **do**
    compute the sample mean $X_{i,t}$ with $M_t(\alpha_{i,t})$
    $\hat{W}_t(\alpha) \leftarrow \left(\frac{\alpha_{i,t}}{\alpha_{i-1,t}}\right)^{-\wp_t^{min}} \hat{W}_t(\alpha)/X_{i,t}$
    if $\hat{W}_t(\alpha) \geq \theta$ **then**
      STOP and return $\hat{y}_t(\mathbf{x}) \leftarrow 1$.
    end if
  end for
  return $\hat{y}_t(\mathbf{x}) \leftarrow 0$
else
  $\hat{W}_t(\alpha) \leftarrow 2^{n_t} \prod_{i=2}^{r_t} \left(\frac{\alpha_{i,t}}{\alpha_{i-1,t}}\right)^{\wp_t^{max}}$
  for $i = 2$ to r_t **do**
    compute the sample mean $X_{i,t}$ with $M_t(\alpha_{i,t})$
    $\hat{W}_t(\alpha) \leftarrow \left(\frac{\alpha_{i,t}}{\alpha_{i-1,t}}\right)^{-\wp_t^{max}} \hat{W}_t(\alpha)/X_{i,t}$
    if $\hat{W}_t(\alpha) < \theta$ **then**
      STOP and return $\hat{y}_t(\mathbf{x}) \leftarrow 0$.
    end if
  end for
  return $\hat{y}_t(\mathbf{x}) \leftarrow 1$
end if

$$
\begin{aligned}
&= W_t(\alpha) / \prod_{i=2}^{r_t-k} \frac{W_t(\alpha_{i,t})}{W_t(\alpha_{i-1,t})} \left(\frac{\alpha_{i,t}}{\alpha_{i-1,t}}\right)^{-\wp_t^{min}} \\
&= (1+\epsilon)\theta / \left(\frac{W_t(\alpha_{r_t-k,t})}{W_t(\alpha_{1,t})} \left(\frac{\alpha_{r_t-k,t}}{\alpha_{1,t}}\right)^{-\wp_t^{min}} \right).
\end{aligned}
$$

Since[5] $\theta > 0$, $W_t(\alpha_{r_t-k,t}) = \sum_{P\in\Omega_t} \alpha_{r_t-k,t}^{\varpi_t(P)}$, $\alpha_{1,t} = 1$ and $W_t(\alpha_{1,t}) = 2^{n_t}$,

$$\sum_{P\in\Omega_t} \alpha_{r_t-k,t}^{\varpi_t(P)-\wp_t^{min}} \leq (1+\epsilon)2^{n_t} \tag{3}$$

[5] Notice θ is always greater than 0 for standard versions of Winnow that only maintain positive weights.

$$\sum_{P\in\Omega_t} \ (\alpha_{r_t-k,t}^{\varpi_t(P)-\wp_t^{min}} - (1+\epsilon)) \leq 0, \tag{4}$$

(4) is equivalent to (1). So Ψ_1^k is a minimum 1-prediction set if k is the minimum solution of (4). If $1+\epsilon \geq \alpha_{r_t-k,t}^{\varpi_t(P)-\wp_t^{min}}$ for all P, then the above inequality holds. Applying $\alpha_{r_t-k,t} = (1+\frac{1}{m_t})^{r_t-k-1}$ yields

$$\ln(1+\epsilon) \ \geq \ (r_t-k-1)\ln\left(\left(1+\frac{1}{m_t}\right)^{\varpi_t(P)-\wp_t^{min}}\right)$$

$$k \ \geq \ r_t - 1 - \ln(1+\epsilon)/\ln\left(\left(1+\frac{1}{m_t}\right)^{\varpi_t(P)-\wp_t^{min}}\right)$$

Notice $(1+\frac{1}{m_t})^{\varpi_t(P)-\wp_t^{min}} < (1+\frac{1}{m_t})^{m_t} < e$. So $k' = r_t - 1 - \ln(1+\epsilon)$ is a solution of (4). Therefore the minimum solution of (4) must be at most k'. Then we get an upper bound of $|\Psi_1^{min}|$ as $r_t - 1 - \ln(1+\epsilon)$. Similarly we can prove that $r_t - 1 + \ln(1-\epsilon)$ is an upper bound of $|\Psi_0^{min}|$. □

According to Theorem 7, our optimized MCMC solution uses at least $\ln(1+\epsilon)$ fewer chains than that Chawla et al. (2004)'s solution if the prediction is 1 and at least $\ln(1-\epsilon)^{-1}$ fewer chains if the prediction is 0. Also we notice that out solution will use less chains if W_t is farther away from θ, which means it saves more computation time.

4. Sampling from $\pi_{\alpha,t}$

All that remains is efficiently drawing samples from the chains (lines 6 and 15 in Table I). Chawla et al. (2004) applied the Metropolis sampler, which is the most popular MCMC method, to the state space Ω_t. Here we also look at three other MCMC sampling techniques including Gibbs sampler, Metropolized Gibbs sampler, and parallel tempering.

4.1. Metropolis Sampler for Winnow

The Metropolis sampler (Metropolis et al., 1953) can be applied to problems where the state is either continuous or discrete, as long as it is possible to compute the ratio of the probabilities of two states. To draw samples from the stationary distribution, the Metropolis sampler repeatedly considers randomly generated changes to the variables of the current state P and accepts new state Q with probability $\min(1, \pi(Q)/\pi(P))$.

Chawla et al. defined $M_t(\alpha)$ based on the Metropolis sampler. Each transition in $M_t(\alpha)$ selects a single variable $p_i \in P$ at random and proposes a new value $1-p_i$. Then the Metropolis acceptance probability for $M_t(\alpha)$ is

$\min(1, \pi(Q)/\pi(P))$, where $Q = (p_1, \ldots, p_{i-1}, 1 - p_i, \ldots, p_{n_t})$. Then they used the Metropolis sampler for $M_t(\alpha)$ as in Table[6] II.

Table II. The Metropolis sampler for Winnow

1: With probability $1/n_t$ let $Q = P$; otherwise,
2: Select i uniformly at random from $1, \ldots, n_t$ and let $Q^{'} = (p_1, \ldots, p_{i-1}, 1 - p_i, \ldots, p_{n_t})$;
3: Let $Q = Q^{'}$ with probability $\min\{1, \frac{\pi(Q^{'})}{\pi(P)}\}$, where

$$\frac{\pi(Q^{'})}{\pi(P)} = \frac{w_{t,Q'}}{w_{t,P}} = \alpha^{\varpi_t(Q^{'}) - \varpi_t(P)},$$

else let $Q = P$.

4.2. GIBBS SAMPLER FOR WINNOW

The Gibbs sampler (Geman and Geman, 1984) is widely applicable to problems where the variables have conditional distributions of a parametric form that can easily be sampled from. In a single transition of the Gibbs sampler, each variable is replaced with a value picked from its distribution conditioned on the current values of all other components.

For any state $P \in \Omega_t$, each variable p_i only has two possible values: 0 and 1. If $P_0 = (p_1, \cdots, p_i = 0, \cdots, p_{n_t})$ and $P_1 = (p_1, \cdots, p_i = 1, \cdots, p_{n_t})$, the conditional distribution is

$$\pi_{\alpha,t}(p_i \mid P \setminus \{p_i\}) = \frac{\pi_{\alpha,t}(P)}{\pi_{\alpha,t}(P_0) + \pi_{\alpha,t}(P_1)}.$$

Then we define the Gibbs sampler for $M_t(\alpha)$ as in Table III.

The Gibbs sampler has a number of distinct features. The conditional distributions of the Gibbs sampler are constructed on prior knowledge of π. Furthermore, the Gibbs sampler is, by construction, multidimensional. It generates new values for all variables and only after that it outputs a sample. In the Metropolis sampler, the variable to be changed is selected totally at random.

[6] We compute the weights of P and Q' by evaluating them on the training set (see Footnote 2).

Table III. The Gibbs sampler for Winnow

1: $Q \leftarrow P$
2: **for** $i = 1, \cdots, n_t$ **do**
3: let $Q^{'} = (q_1, \ldots, q_{i-1}, 1 - q_i, \ldots, q_{n_t})$;
4: Let $Q = Q^{'}$ with probability $\pi_{\alpha,t}(Q^{'} \mid P \setminus \{p_i\})$, where

$$\begin{aligned} \pi_{\alpha,t}(Q^{'} \mid P \setminus \{p_i\}) &= w_{t,Q'}/(w_{t,Q'} + w_{t,P}) \\ &= 1/(1 + \alpha^{\varpi_t(P) - \varpi_t(Q^{'})}); \end{aligned}$$

5: **end for**

4.3. METROPOLIZED GIBBS SAMPLER FOR WINNOW

The Metropolized Gibbs sampler (Liu, 1996) is a modification of the Gibbs sampler. It has been proven to be statistically more efficient than the Gibbs sampler. The sampler draws a new value p_i' with probability

$$\frac{\pi_{\alpha,t}(p_i' \mid P \setminus \{p_i\})}{1 - \pi_{\alpha,t}(p_i \mid P \setminus \{p_i\})} \ ,$$

and accepts with the Metropolis-Hastings acceptance probability

$$\min \left\{ 1, \frac{1 - \pi_{\alpha,t}(p_i \mid P \setminus \{p_i\})}{1 - \pi_{\alpha,t}(p_i' \mid P \setminus \{p_i\})} \right\} \ .$$

As mentioned in Section 4.2, each component in Ω_t only has two possible values. So the Metropolized Gibbs sampler becomes a Metropolis sampler that repeatedly updates all components in a fixed order. We then build the Metropolized Gibbs sampler for $M_t(\alpha)$ by replacing the acceptance probability in line 4 of Table III with

$$\min \left\{ 1, \frac{\pi_{\alpha,t}(Q^{'} \mid Q \setminus \{q_i\})}{1 - \pi_{\alpha,t}(Q^{'} \mid Q \setminus \{q_i\})} \right\} \ ,$$

where

$$\frac{\pi_{\alpha,t}(Q^{'} \mid Q \setminus \{q_i\})}{1 - \pi_{\alpha,t}(Q^{'} \mid Q \setminus \{q_i\})} = \frac{\pi(Q^{'})}{\pi(Q)} = \alpha^{\varpi_t(Q^{'}) - \varpi_t(Q)} \ .$$

4.4. PARALLEL TEMPERING FOR WINNOW

The idea of parallel tempering (Geyer, 1991) is to artificially ensemble a set of Markov chains with different, but related stationary distributions. It involves two sets of steps: local steps in each chain and swap steps between two chains. Each local step is defined by each Markov chain, such as Metropolis sampler or Gibbs sampler. In each swap step, two chains make an exchange of their current states. For example, chain i is at state P_i and chain j is at state P_j. After a swap step, chain i will be at state P_j and chain j will be at state P_i. Swap steps are introduced to allow greater mobility and faster mixing. After all of the chains make a local step, the sampler attempts to swap the current states of two of the chains. The acceptance probability for swapping states P_i and P_j between two chains i and j is $\min\left\{1, \frac{\pi_i(P_j)\pi_j(P_i)}{\pi_i(P_i)\pi_j(P_j)}\right\}$.

In our MCMC solution, we build $r_t - 1$ parallel Markov chains $M_t(\alpha_i)$ with stationary distributions

$$\pi_{\alpha_i,t}(P) = \frac{\alpha_i^{\varpi_t(P)}}{W_t(\alpha_i)} = \frac{(1+1/m_t)^{(i-1)\varpi_t(P)}}{W_t(\alpha_i)}.$$

Then we get the parallel tempering version of our samplers as in Table IV. In Table IV, we simulate all $r_t - 1$ Markov chains in parallel, while our optimized MCMC solution needs to run these chains in a specific sequence (see Section 3.3). In order to apply our optimized solution, we can partition the sequence into small groups, use parallel tempering in each group and run these groups sequentially. Then we can apply our optimized solution on these groups. But we might not find the best subset of chains as indicated in Theorem 6 since we will only check the constraints after the simulation of a group of chains instead of a single chains.

Table IV. Parallel tempering for Winnow

1: **for** $T = 1, \cdots, S$ **do**
2: With probability ρ_{swap}, randomly choose a neighboring pair of chains, say i and $i+1$, and swap current states P_i and P_{i+1} with probability

$$\min\left\{1, (1+1/m_t)^{\varpi_t(P_i)-\varpi_t(P_{i+1})}\right\}.$$

3: Simulate all $r_t - 1$ Markov chains for a single step via their samplers.
4: Save current states of all $r_t - 1$ Markov chains as samples.
5: **end for**

5. Other Extensions

5.1. Discretizing Real-Valued Attributes

Since our algorithm can only handle discrete-valued attributes, we employ the method of Elomaa and Rousu (1999) to discretize continuous-valued attributes. For each attribute, we sort (in ascending order) its values over all examples and then divide its value range into several intervals according to some evaluation function that estimates the class coherence in a given set of examples. Here we use the Average Class Entropy as the evaluation function. Let $\biguplus S_i$ be a partition of S, the Average Class Entropy of the partition is:

$$ACE(\biguplus_i S_i) = \sum_i \frac{S_i}{S} H(S_i) = \frac{1}{|S|} \sum_i |S_i| H(S_i) \ ,$$

where the entropy function $H(S) = -\sum_{j=1}^{m} P(C_j, S) \log_2 P(C_j, S)$, where m is the number of classes and $P(C_j, S)$ is the proportion of the examples in S that belong to the class C. A good property of Average Class entropy is cumulativity, i.e. the impurity of a partition can be obtained by a weighted summation over the impurities of its subsets. Cumulativity facilitates incremental evaluation of impurity values. So we can apply a dynamic programming scheme that is suggested by Elomaa and Rousu (1999) to discretize real-valued attributes.

An advantage to partitioning a single continuous attribute c into several intervals (i.e. mapping c to a single k-valued attribute) versus finding several thresholds for c (i.e. mapping c to $k-1$ boolean attributes) is that the state space Ω is smaller, as is n. If $k-1$ boolean attributes are used, then $|\Omega| = 3^{k-1} \prod_{i \in other}(k_i + 1)$, where $other$ is the set of other attributes. In contrast, a single k-valued attribute yields $|\Omega| = (k+1) \prod_{i \in other}(k_i + 1)$. In addition, reducing n makes exact computation of W_t (which requires enumerating up to 2^n terms) easier, so it is more likely that we can do exact computation of W_t rather than an approximation, which increases accuracy.

5.2. Handling Missing Data

There is an implicit means in our representation of examples that can be used to handle missing data. If the values of some attributes of example $\mathbf{x}$ are missing in an example, we simply assign 0 to these attributes and define a term P to not be satisfied by $\mathbf{x}$ unless $p_i = 0$ for all i such that $x_i = 0$ (see Section 2.1). We also tried the popular approach of assigning to the missing attribute the most common value in the same class.

5.3. MULTI-CLASS CLASSIFICATION

Since Winnow only can only make binary predictions, we train one Winnow DNF learner for each class, i.e. a "one versus the rest" approach. So given an example $\mathbf{x}$ with a label of class j, $\mathbf{x}$ is presented to Winnow$_j$ as a positive example and to all others as a negative example. After training all Winnows, we take an unlabeled example, estimate the weighted sums of all Winnows on that example , and predict the class with the Winnow that has the maximum weighted sum. In our experiments, we observed that even if each single Winnow has a high error rate, the Winnow of the class that an example belongs to frequently is the one with the highest weighted sum. Therefore our algorithm frequently had low prediction error even when the individual binary classifiers did not.

6. Experimental Results

In our experiments, we evaluated three MCMC sampling techniques: Metropolis, Gibbs and Metropolized Gibbs, each with and without parallel tempering (PT). So we tested 6 samplers. We compared their performance on estimating weighted sums on two data sets: simulated data similar to that used by Chawla et al. (2004) and Voting data from the UCI repository (Blake et al., 2004). We then tested our optimized algorithm on the simulated data and five UCI data sets to find what kind of impact these MCMC samplers have on Winnow's predictions. We also compared the computation costs of our optimized solution with Chawla et al.'s algorithm in terms of the total number of Markov chains simulated.

We started out tests with data similar to Chawla et al.'s simulated data. They used data with $n \in \{10, 15, 20\}$. In our experiment, we used their simulated data generator to generate random 5-term monotone DNF formulas, using $n \in \{10, 15, 20, 25, 30, 35, 40\}$. For each value of n there were 10 training/testing sets, each with 50 training examples and 50 testing examples. In our experiments on UCI data, we partitioned each data set into k blocks, where $k = 10$ if the data set size was ≥ 300. Otherwise the number of blocks was reduced to ensure that each block was of size at least 30. Since our algorithm can only handle discrete-valued attributes, we also discretized continuous-valued attributes.

Neal (1995) pointed out that it takes about T^2 steps to move to a state T steps away because of the random walk nature of MCMC samplers. The farthest distance between two states in Ω_t is n, the number of variables. Thus we set the burn-in time $T_0 = n^2$. Our experiments showed that this burn-in time worked very well. In each experiment, we counted each update of a

single variable of current state as a single sampling step. We used the same number of sampling steps[7] T_s for all six samplers.

6.1. Comparisons of Weighted Sum Estimates

Our first experiments were designed to evaluate how well the weight estimation procedures with different samplers guessed the weighted sums. Since we are interested in estimated weighted sums instead of just predictions in these experiments, we turned off our optimized solution. During Winnow's training, we computed the estimates with six samplers, while we updated weights using the exact weighted sum computed via brute force (i.e. Winnow was trained using the exact weighted sums). So all samplers worked on the same distributions and state spaces. Then we compared them using the measure *Guess Error* given in Chawla et al. (2004), which is the average error of the estimates ($|\hat{W} - W|/W$).

Figure 1 shows the results on Voting, which has 16 variables and 435 examples. We set $T_0 = 256$ and varied $T_s \in \{800, 1600, 3200, 6400, 9600, 12800, 16000\}$. We trained Winnow for 20 rounds on 10 partitions. So Figure 1 represents averages over more than 70000 estimates.

In the figure, there are dramatic drops of Guess Error from 800 to 6400 sampling steps. When $T_s \geq 9600$, T_s has much less effect on Guess Error. This indicates that at that point Markov chains may be close to the stationary distribution π_t. The parallel tempering version of each sampler showed little effect. This is because the number of chains r_t for each estimation increased quickly. It was more than 50 after 8 training iterations and eventually more than 150. So even setting the swap probability to 0.9 did not provide enough swap steps to make a big improvement. For all T_s, Guess Errors of Gibbs and Metropolized Gibbs are always lower than Metropolis. Although Metropolized Gibbs has a lower Guess Error than Gibbs, the difference between these two samplers is very small, especially when $T_s = 9600, 12800, 16000$.

To evaluate the effect of varying the number of attributes n, we measured Guess Error of the six samplers on the simulated data[8]. In Figure 2, T_s is fixed at 10000 and $T_0 = n^2$. Gibbs and Metropolized Gibbs are still always better than Metropolis. The Guess Error of Metropolized Gibbs is lower than Gibbs when $n \leq 25$. But when $n > 25$, Gibbs becomes the best sampler.

[7] The number of samples S drawn by the Metropolis sampler is T_s. But S for the Gibbs and Metropolized Gibbs samplers is T_s/n because they only use states as samples after all n variables have been updated (see Section 4).

[8] It seems it is unpractical to run brute force Winnow when n is bigger than 20. But the simulated data is randomly generated from monotone DNF, so each variable can only be 1 or 0. Since we do not need to consider the variables with 0 value (see Section 3.1), the expected number of variables used by brute force Winnow each trial is $n/2$. So we can run brute force Winnow even when $n = 40$.

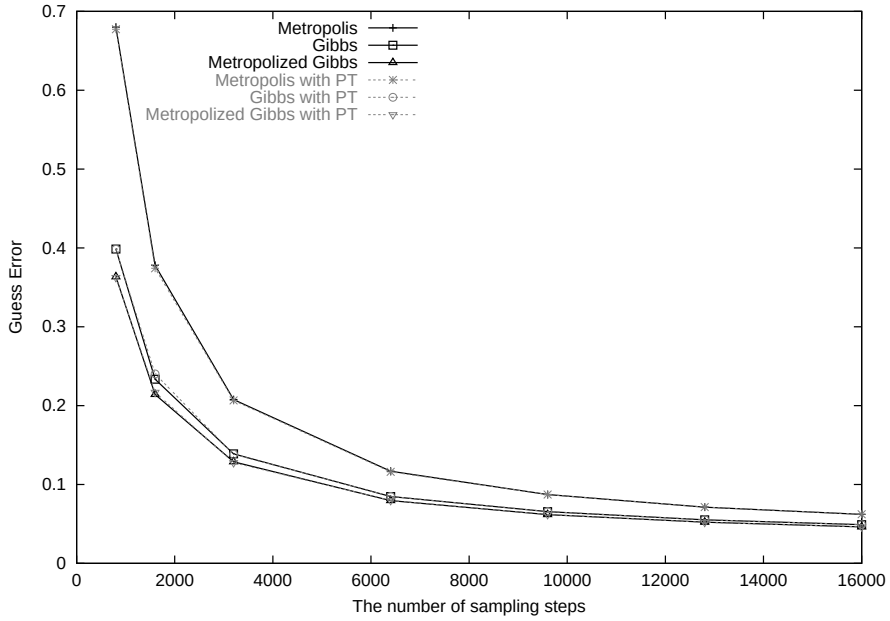

Figure 1. Guess Error vs. the number of sampling steps T_s on Voting.

As with Voting, parallel tempering only improved performance very slightly. Considering results in Figures 1 and 2, we can say that Gibbs sampler is the best choice on average in terms of the accuracy of estimating weighted sums.

6.2. Comparisons of Prediction Error

Now we describe experiments that examine the prediction error of MCMC-based Winnow. In both training and testing, we ran the MCMC-based simulation instead of brute force. We performed k-fold cross-validation on six data sets from the UCI repository and computed 95% confidence intervals (it is not practical to run brute force Winnow on auto and annealing). Since Winnow can only make binary predictions, we train one Winnow DNF learner for each class. After training, we take an unlabeled example, estimate the weighted sums of all Winnows on that example, and predict the class with the Winnow that has the maximum weighted sum. Table V summarizes the results. For each data set, all six samplers have similar performance. All confidence intervals of each data set overlap. Although no sampler showed a significant advantage over others, we notice that prediction errors of all three samplers are often lower than their PT versions. Another interesting fact is that MCMC-based Winnow has even better performance than brute force Winnow on iris and car. On breast cancer and voting data, non-PT samplers showed little difference from brute force Winnow. Thus for the data sets we

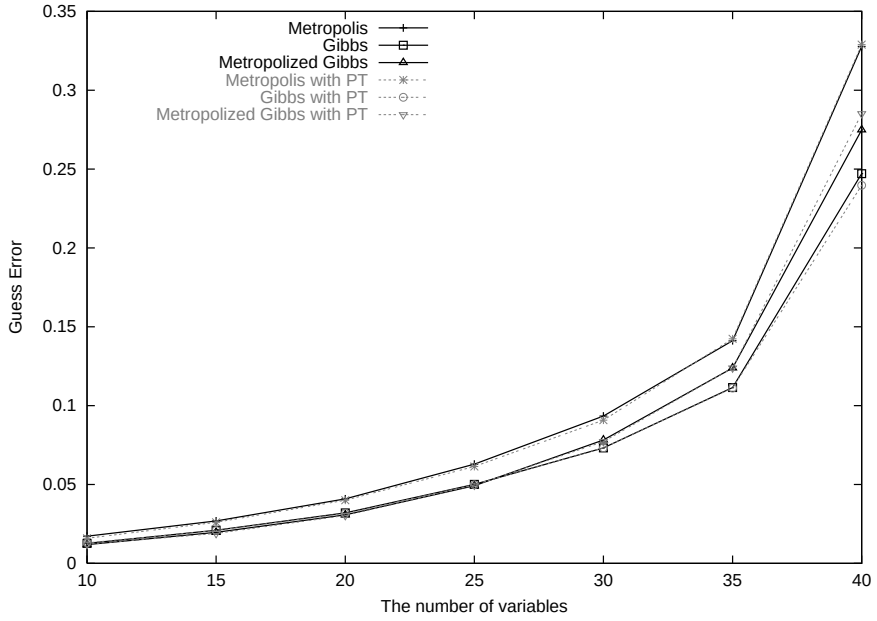

Figure 2. Guess Error vs. the number of variables n on simulated data.

tested, there does not seem to be any disadvantage to using an MCMC-based approximation in place of brute force.

To further investigate these observations, we reran the experiments of Section 6.1. There were 7 experiments for simulated data and 7 experiments for

Table V. Comparisons of prediction errors on UCI data sets ($T_0 = n^2$ and $T_s = 10n^2$).
M - Metropolis, G - Gibbs, MG - Metropolized Gibbs, PT - Parallel Tempering, BF - Brute Force.

Data Sets	iris	car	breast cancer	voting	auto	annealing
n	4	6	9	16	25	38
M	$\mathbf{5.3 \pm 2.1}$	1.7 ± 0.8	31.5 ± 5.0	5.0 ± 2.1	$\mathbf{12.8 \pm 7.5}$	1.0 ± 0.7
G	6.7 ± 3.8	1.9 ± 0.8	$\mathbf{30.9 \pm 5.5}$	5.0 ± 2.4	15.6 ± 7.8	0.6 ± 0.5
MG	6.0 ± 1.7	$\mathbf{1.5 \pm 0.8}$	31.7 ± 5.0	5.0 ± 2.1	16.6 ± 5.5	$\mathbf{0.4 \pm 0.4}$
M+PT	6.0 ± 3.2	1.7 ± 0.9	32.7 ± 4.7	$\mathbf{5.0 \pm 1.6}$	18.6 ± 5.1	0.9 ± 0.7
G+PT	6.7 ± 2.7	$\mathbf{1.5 \pm 0.8}$	$\mathbf{31.5 \pm 3.8}$	5.9 ± 2.5	18.4 ± 4.7	1.3 ± 0.9
MG+PT	6.0 ± 3.2	1.6 ± 0.7	31.8 ± 5.8	5.4 ± 2.5	$\mathbf{18.1 \pm 4.3}$	$\mathbf{0.7 \pm 0.5}$
BF	7.3 ± 3.2	3.3 ± 1.1	33.3 ± 4.9	5.0 ± 2.0	-	-

Table VI. Comparisons of prediction errors on Voting ($n = 16$ and $T_0 = 256$).
M - Metropolis, G - Gibbs, MG - Metropolized Gibbs, PT - Parallel Tempering, BF - Brute Force.

T_s	800	1600	3200	6400	9600	12800	16000
M	5.1±1.7	5.3±1.6	**5.0 ± 2.1**	5.3±1.8	5.0±1.3	4.8±1.6	5.5±2.4
G	**4.6 ± 2.5**	**4.8 ± 1.5**	5.0±2.4	4.8±2.1	6.0±2.5	**4.6 ± 1.8**	**4.3 ± 2.0**
MG	5.1±1.1	5.0±1.5	**5.0 ± 2.1**	**4.6 ± 1.6**	**4.8 ± 1.8**	5.0±2.7	5.9±2.5
M+PT	5.3±1.9	4.1±0.9	**5.0 ± 1.6**	5.5±1.5	**5.0 ± 1.8**	**4.1 ± 2.0**	**3.6 ± 1.9**
G+PT	5.3±1.4	4.8±1.8	5.9±2.8	5.9±2.5	5.4±1.8	4.6±2.2	4.3±2.0
MG+PT	**5.1 ± 1.1**	**3.2 ± 0.9**	5.4±2.5	**4.8 ± 2.7**	5.0±2.5	5.0±2.2	5.5±2.4
BF							5.0±2.0

Table VII. Comparisons of prediction errors on simulated data ($T_0 = n^2$ and $T_s = 10000$).
M - Metropolis, G - Gibbs, MG - Metropolized Gibbs, PT - Parallel Tempering, BF - Brute Force.

n	10	15	20	25	30	35	40
M	1.8±1.6	2.6±2.6	7.4±3.9	4.2±2.3	**6.0 ± 3.0**	**7.2 ± 5.9**	7.0±7.0
G	1.8±1.6	3.0±2.5	7.8±4.3	**3.4 ± 3.0**	8.0±3.5	7.4±4.9	**6.2 ± 3.8**
MG	1.8±1.4	**2.6 ± 2.3**	**7.2 ± 3.5**	3.8±2.9	6.4±3.9	8.8±5.2	7.0±3.9
M+PT	**1.4 ± 1.5**	4.4±2.4	7.4±3.7	3.2±2.8	**6.4 ± 4.1**	**7.0 ± 3.8**	**5.8 ± 4.5**
G+PT	1.6±2.0	3.0±1.8	**5.6 ± 3.1**	4.4±3.3	7.4±4.5	7.2±3.6	6.0±3.4
MG+PT	1.6±1.5	**2.6 ± 2.6**	7.0±4.3	**3.2 ± 2.3**	7.8±4.3	9.6±5.5	6.0±4.1
BF	2.8±2.6	4.6±3.8	7.2±4.2	4.4±4.4	6.2±3.9	7.4±3.5	5.6±4.0

Voting. For Voting, 10-fold cross-validation was done for each experiment. Table VI reports the results. MCMC samplers with PT have higher prediction error than non-PT versions when T_s is less than 9600, except for $T_s = 1600$. When T_s is bigger than 9600, PT versions have equal or even better performance compared to non-PT versions. We suspect this behavior was because when T_s is small, Guess Error of all samplers is high. The estimated value $\hat{W}$ varies a lot at different time simulations. Since parallel tempering allows greater mobility (see Section 4.4), PT would introduce more variation, which makes it is hard for Winnow to learn. But when Guess Error is very low, this variation does not change Winnow's learning procedure according to Theorem 1 and 4. Also we notice that in this experiment, Gibbs has the lowest prediction error four times, which is best in non-PT version. But the best PT sampler is Metropolis with PT. Finally, note that brute force does not have any significant advantage over most approximations, and is often worse.

Table VII reports results for simulated data. Here we fixed T_s and varied n. Each experiment involved 10 training/testing sets. Samplers with PT have better performance when n is 10, 15 and 40, and show little difference on other n. Because Guess Errors of PT and non-PT samplers are almost same, the only reason is the randomness in samplers as we discussed before. MCMC-based Winnow achieved lower error rates than brute force Winnow when n is smaller than 20. But they have higher error rates when $n = 40$ since Guess Error is relatively high. Also we notice that in this experiment, Gibbs has the lowest prediction error for three times, which is best in non-PT version. But the best PT sampler is Metropolis with PT. Again, we see that brute force's error is at least as large as several of the approximations.

From all above results, we found when Guess Error is small, MCMC-based Winnow can get equal or even better performance than brute force Winnow. Parallel Tempering introduces more random behavior in MCMC. Its performance is hard to predict. In this sense, Gibbs sampler, which has the best performance in terms of Guess Error, is a good choice, though its advantage is less clear in terms of prediction error. We do not recommend using its PT version.

6.3. Comparisons of Computation Cost

Here we report speedups of our optimized algorithm (Section 3.3) over Chawla et al.'s solution in terms of the total numbers of Markov chains that are used. In Tables VIII and IX, "MCMC" is the total number of chains used by Chawla et al.'s solution. "Opt MCMC" is the total number of chains used by our algorithm. So $MCMC = \sum_{t=1}^{\tau}(r_t - 1)$ and $Opt\ MCMC = \sum_{t=1}^{\tau} r'_t$, where r'_t is the number of chains used by our optimized algorithm at trial t and τ is the number of trials. "Savings" is the percentage of chains our algorithm saved, that is, $Savings = 1 - Opt\ MCMC/MCMC$. "$Savings_{theory}$" is the percentage of chains our algorithm can save according to the worst-case analysis of Theorem 7. To compute $Savings_{theory}$, we computed the number of chains saved for each trial and add them up over all trials. So $Savings_{theory} = \left(\sum_{t=1}^{\tau} |\ln(Sav_\theta(W_t))|\right)/MCMC$, where $Sav_\theta(W_t) = 1 + \epsilon_t$ if $W_t \geq \theta$, $Sav_\theta(W_t) = 1 - \epsilon_t$ if $W_t < \theta$. Since it is not practical to compute W_t for auto and annealing, we cannot compute $Savings_{theory}$ for those data sets. In the tables, $\bar{\varepsilon}$ is the average over all trials $\varepsilon_t = |(W_t - \theta)/\theta|$, which is the normalized distance between true weighted sum W_t and threshold θ.

Results in Tables VIII and IX confirm that we do not need to run all chains when estimating weighted sums. However, this widely varies with the data set. A possible reason is that the true weighted sums of Winnow are much less or much greater than the threshold for some data sets, and closer for others. When the true weighted sums diverge significantly from the threshold, there

Table VIII. Comparisons of the total number of Markov chains on simulated data (in thousands).

n	$\bar{\varepsilon}$	MCMC	Opt MCMC	Savings (%)	$Savings_{theory}$ (%)
10	0.351	4.6	4.1	11.3	6.4
15	0.365	11.9	11.2	6.3	3.9
20	0.340	20.8	18.7	10.0	7.0
25	0.454	23.5	21.2	9.7	6.8
30	0.363	40.8	37.5	8.1	5.1
35	0.421	58.7	53.0	9.7	6.2
40	0.294	60.9	53.2	12.7	6.8

Table IX. Comparisons of the total number of Markov chains on UCI data sets (in thousands).

Data Sets	$\bar{\varepsilon}$	MCMC	Opt MCMC	Savings (%)	$Savings_{theory}$ (%)
iris	0.472	217.7	200.1	8.0	1.14
car	0.529	14272.7	14032.0	1.7	0.12
breast cancer	0.500	45176.2	45111.5	0.2	0.04
voting	0.587	1185.5	1132.0	4.5	0.48
auto	-	13822.8	13751.7	0.5	-
annealing	-	4855.2	3781.6	22.1	-

are more chances for our algorithm to stop early. As shown in Tables VIII and IX, when $Savings_{theory}$ is big, the true savings is often big.

7. Conclusions and Future Work

We proposed an optimized MCMC solution for estimating weighted sums in Winnow. We showed that it often uses less computation time than Chawla et al.'s (2004) solution without any loss of classification accuracy. Our experimental results confirmed that our algorithm only needs to use a subset of all Markov chains implied by the original solution. We also showed how to get such a subset with the smallest size and gave lower bounds on how many chains our solution can save. We also empirically compared three new MCMC sampling techniques: Gibbs, Metropolized Gibbs and parallel tempering. They all showed better performance than Chawla et al.'s Metropolis sampler in terms of accuracy of weighted sum estimates. We also found that the Gibbs sampler had good performance on average. Further, we found that all approximation algorithms had prediction error that was no worse (and

often better than) a brute force version that exactly computed the weighted sums.

Future work includes applying our algorithm to other algorithms (e.g. the algorithms of Tao and Scott (2004) and Goldman et al. (2001) for multiple-instance learning) and exploring other MCMC techniques such as blocking and over-relaxation (Neal, 1995). In addition, we are investigating the effect of pruning terms (i.e. using only heavy terms) on prediction accuracy, found via both the Markov chain and a genetic algorithm. We are also developing other methods to reduce training time, e.g. by limiting each term (input to Winnow) to be constraints on at most k attributes (i.e. at most k non-0s are allowed per term). This reduces the state space size for exactly computing W_t to $|\Omega_t| = O(n^k)$ as opposed to 2^n. Khardon et al. (2001) discuss this idea in a slightly different context. Also, Khardon et al. give a negative result for exactly simulating Winnow for learning DNF (Section 2.1). Does a similar result exist for randomized algorithms simulating Winnow with high probability?

Acknowledgements

This work was funded in part by NSF grants CCR-0092761 and EPS-0091900 and a grant from the University of Nebraska Foundation. It was also supported in part by NIH Grant Number RR-P20 RR17675 from the IDeA program of the National Center for Research Resources. This work was completed in part utilizing the Research Computing Facility of the University of Nebraska.

References

Blake, C.L. and C. J. Merz: 2004, 'UCI Repository of machine learning databases'. `http://www.ics.uci.edu/~mlearn/MLRepository.html`. University of California, Department of Information and Computer Science, Irvine, CA.

Blum, A., P. Chalasani, and J. Jackson: 1993, 'On learning embedded symmetric concepts'. In: *Proceedings of the Sixth Annual Workshop on Computational Learning Theory*. pp. 337–346, ACM Press, New York, NY.

Blum, A., M. Furst, J. Jackson, M. Kearns, Y. Mansour, and S. Rudich: 1994, 'Weakly Learning DNF and Characterizing Statistical Query Learning Using Fourier Analysis'. In: *Proceedings of Twenty-sixth ACM Symposium on Theory of Computing*. pp. 253–262.

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

Chawla, D., L. Li, and S. D. Scott: 2004, 'On approximating weighted sums with exponentially many terms'. In: *Journal of Computer and System Sciences*, to appear. Early version in COLT '01.

Dietterich, T. G., R. H. Lathrop, and T. Lozano-Perez: 1997, 'Solving the Multiple-Instance Problem with Axis-Parallel Rectangles'. *Artificial Intelligence* **89**(1–2), 31–71.

Elomaa, T. and F. Rousu: 1999, 'General and Efficient Multisplitting of Numerical Attributes'. *Machine Learning* **36**(3), 201–244.

Geman, S. and D. Geman: 1984, 'Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images'. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **6**, 721–741.

Geyer, C.: 1991, 'Markov chain Monte Carlo maximum likelihood'. In: *Computing Science and Statistics: Proc. of the 23rd Symposium on the Interface*. pp. 156–163.

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

Jerrum, M. and A. Sinclair: 1996, 'The Markov chain Monte Carlo method: An approach to approximate counting and integration'. In: D. Hochbaum (ed.): *Approximation Algorithms for NP-Hard Problems*. PWS Pub., Chapt. 12, pp. 482–520.

Khardon, R., D. Roth, and R. Servedio: 2001, 'Efficiency versus Convergence of Boolean Kernels for Online Learning Algorithms'. In: *Advances in Neural Information Processing Systems 14*. pp. 423–430.

Littlestone, N.: 1988, 'Learning Quickly When Irrelevant Attributes Abound: A New Linear-threshold Algorithm'. *Machine Learning* **2**, 285–318.

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

Liu, J.: 1996, 'Peskun's Theorem and A Modified Discrete-State Gibbs Sampler'. *Biometrika* **83**, 681–682.

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

Neal, R. M.: 1995, 'Suppressing random walks in Markov chain Monte Carlo using ordered overrelaxation'. Technical Report 9508, Dept. of Statistics, University of Toronto.

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

Tao, Q. and S. Scott: 2003, 'An analysis of MCMC sampling methods for estimating weighted sums in Winnow'. In: *Artificial Neural Networks in Engineering*. pp. 15–20.

Tao, Q. and S. Scott: 2004, 'A faster algorithm for generalized multiple-instance learning'. In: *Proceedings of the Seventeenth Annual FLAIRS Conference*. To appear.