Mach Learn (2008) 73: 107–132
DOI 10.1007/s10994-008-5063-9

# Improved MCMC sampling methods for estimating weighted sums in Winnow with application to DNF learning

**Qingping Tao · Stephen D. Scott**

Received: 10 November 2005 / Revised: 17 May 2008 / Accepted: 25 May 2008 /
Published online: 17 July 2008


**Abstract** A Markov chain Monte Carlo method has previously been introduced to estimate weighted sums in multiplicative weight update algorithms when the number of inputs is exponential. However, the original 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 the original algorithm that produces exactly the same classifications while often using fewer Markov chain simulations. We also apply three other sampling techniques and empirically compare them with the original 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. We found that two other samplers (Gibbs and Metropolized Gibbs) were slightly better than Metropolis in their estimates of the weighted sums. For prediction errors, there is little difference between any pair of MCMC techniques we tested. Also, on the data sets we tested, we discovered that all approximations of Winnow have no disadvantage when compared to brute force Winnow (where weighted sums are exactly computed), so generalization accuracy is not compromised by our approximation. This is true even when very small sample sizes and mixing times are used.

**Keywords** DNF · Winnow · Markov chain Monte Carlo

Editor: Zoubin Ghahramani.

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

Q. Tao
GC Image, LLC, 216 N 11th St, Suite 302, Lincoln, NE 68508, USA
e-mail: qtao@gcimage.com

S.D. Scott (✉)
Department of Computer Science & Engineering, University of Nebraska, 256 Avery Hall, Lincoln, NE 68588-0115, USA
e-mail: sscott@cse.unl.edu

## 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 were much stronger than what their theoretical results implied, they still required extensive simulation of the Markov chain to draw "good" samples (i.e. from a distribution similar to that of the chain's stationary distribution) in order to get accurate estimates of the weighted sums. This significantly slowed down their algorithm.

We explore ways to speed up Chawla et al.'s algorithm. We also evaluate whether their algorithm is useful in practice, even when theoretical guarantees are sacrificed. First 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 that our optimized version exactly simulates that of Chawla et al. We also give lower bounds on how much computation time our algorithm saves.

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 will be very useful to make this algorithm (and others) more scalable, but first a thorough empirical analysis of the sampling method is required.

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 the Metropolis sampler was slightly worse than Gibbs and Metropolized Gibbs on estimating weighted sums. For prediction errors, there is little difference between any pair of MCMC techniques we tested. Also, on the data sets we tested, we discovered that all approximations of Winnow have no disadvantage when compared to brute force Winnow (where weighted sums are exactly computed), so generalization accuracy is not compromised by our approximation. This is true even when very small sample sizes and mixing times are used.

The rest of this paper is as follows. In Sect. 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 presents various MCMC sampling techniques. Section 5 gives several extensions of the basic Winnow algorithm. We then report our experimental results in Sect. 6 and conclude in Sect. 7.

## 2 Related work

### 2.1 Learning disjunctive normal form formulas

Let $f = P_1 \vee P_2 \vee \cdots \vee P_K$ be the target function, where $P_i = c_{i1} \wedge c_{i2} \wedge \cdots \wedge c_{in}$ 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 that are $> 0$), then there are exactly $2^{n_{\mathbf{x}}}$ terms satisfied by it.

*Example 2.1* (Monotone DNF) A monotone DNF formula is a DNF formula with no negated variables. So $k_j = 1$ for all $j$. For example, assume $n = 3$, example $\mathbf{x} = (0, 1, 1)$ satisfies $2^2 = 4$ terms: $\phi$ (the always true term), $x_2$, $x_3$, and $x_2 \wedge x_3$.

*Example 2.2* (Conventional DNF) In conventional DNF formulas, $k_j = 2$ for all $j$ and 1 represents[1] "true" and 2 represents "false". For example, assume $n = 3$, example $\mathbf{x} = (0, 1, 2)$ satisfies $2^2 = 4$ terms: $\phi$, $x_2$, $\neg x_3$, and $x_2 \wedge \neg x_3$.

*Example 2.3* (Generalized DNF) In generalized DNF formulas, $k_j$ can have more than 2 values for all $j$. For example, assume $n = 3$, example $\mathbf{x} = (0, 3, 7)$ satisfies $2^2 = 4$ terms: $\phi$, $(x_2 = 3)$, $(x_3 = 7)$, and $(x_2 = 3) \wedge (x_3 = 7)$.

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. 2004; Klivans and Servedio 2004; Khardon and Servedio 2005; Khardon et al. 2005 and references therein), but positive results exist only in restricted cases, and the general DNF problem remains open.[2]

### 2.2 The Winnow algorithm

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 $\theta$ is the threshold. Then Winnow is told the true label $y_t \in \{0, 1\}$ 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 $w_{t+1,i} > w_{t,i}$, the update is called a *promotion* and if $w_{t+1,i} < w_{t,i}$, it is called a *demotion*. Littlestone showed that if the target concept labeling 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 $K$ relevant inputs is negated in the target function), then Winnow can exactly learn the target concept while making at most $O(K \log N)$ prediction mistakes.[3] 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

[1] Recall that we reserve the value 0 for "don't care", hence the use of 2 for "false". I.e. we think of "true" and "false" as attribute values.

[2] It is unlikely that an efficient distribution-free (PAC) DNF-learning algorithm exists (Klivans and Servedio 2004; Blum et al. 1994; Blum et al. 1993).

[3] A more general result was shown by Auer and Warmuth (1998): If the best $K$-disjunction makes $M_{opt}$ mistakes on a sequence of instances, then Winnow's mistake bound is $O(K(M_{opt} + \log N))$.

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 DNF.

It turns out that storing all weights takes $\Omega(N)$ space, but one can compute the weight of a particular term $P$ by evaluating it on the training set, eliminating the need to store the weights. Specifically, in pass $i$ through the entire training set (here we assume that when operating as a batch learner, Winnow iterates multiple times over the fixed training set), the algorithm has a record of the number of prediction errors that Winnow made on each training example in each pass $j < i$ through the training set. Now let $S_P^+$ be the set of positive examples that satisfy term $P$ and $S_P^-$ be the set of negative examples that satisfy term $P$. Then, assuming all weights were initialized to 1, term $P$'s weight is $\alpha^{g(P)}$, where $g(P) = \sum_{x \in S_P^+} mist_i(x) - \sum_{x \in S_P^-} mist_i(x)$ and $mist_i(x)$ is the number of prediction mistakes made by Winnow on training example $x$ in passes 1 through $i-1$.

Even though the algorithm as described above can be space efficient, it is not time efficient, since brute force computation of $W_t = \mathbf{w}_t \cdot \mathbf{x}'_t$ takes time $\Omega(N)$, which is exponential in $n$. Thus another approach is needed to compute $W_t$. One possibility is to use kernels, as illustrated by Khardon et al. (2005) 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.[4] 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* $\theta$ as $W_t$.

### 2.3 Markov chain Monte Carlo

Markov chain Monte Carlo (MCMC) methods use Markov chains 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 $\Omega$, which is a large but finite set of combinatorial structures.

Generally a Markov chain $M$ with state space $\Omega$ and stationary distribution $\pi$ is designed to be *ergodic*, that is, the probability distribution over $\Omega$ converges asymptotically to $\pi$ regardless of the initial state. Then $M$ is repeatedly simulated for $T$ steps and generates samples almost according to $\pi$. 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 $\pi$ during these steps. This $T_0$-step procedure is called *burn-in*.

[4] It is possible, though not automatic, that Khardon et al.'s hardness results may be transferable to Chawla et al.'s randomized approximation setting. Either way, Chawla et al.'s results do not guarantee an efficient algorithm for learning DNF since their Markov chain's mixing time is not guaranteed to be polynomial and the range restriction on the weighted sum is not necessarily guaranteed to hold (see Sect. 3.1).

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 $T = S\,T_0$. 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 $\pi$ rapidly. Since efficiency is essential in our experiments, we use a single run rather than multiple runs.

One requirement for an approximation algorithm to be efficient is for the burn-in time $T_0$ to be polynomial in all relevant parameters ($n$ and $K$ in our application). A Markov chain having this property is called *rapidly mixing,* which means that the chain will be close to its stationary distribution $\pi$ after taking a polynomial-length random walk through $\Omega$. More formally, we measure the difference between $\hat{\pi}_{P,t}$ (the empirical distribution resulting from simulating the chain for $t$ steps starting from state $P \in \Omega$) and the true stationary distribution $\pi$ by *variation distance* (Jerrum and Sinclair 1996):

$$\Delta_P(t) = \max_{\mathcal{U} \subseteq \Omega} \left| \hat{\pi}_{P,t}(\mathcal{U}) - \pi(\mathcal{U}) \right| = \frac{1}{2} \sum_{Q \in \Omega} \left| \hat{\pi}_{P,t}(Q) - \pi(Q) \right|.$$

The number of simulation steps $T$ required for $\Delta_P(T) \leq \epsilon$ for all states $P \in \Omega$ is called the *mixing time* of $M$.

Under appropriate conditions, the approximation algorithms based on the MCMC method yield accuracy guarantees, e.g. in approximate summation, sometimes one can guarantee that the estimate of the sum is within a factor $\epsilon$ of the true value. 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 $\epsilon$ of the true maximum or minimum.

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 solutions and gave a Metropolis sampler (Metropolis et al. 1953) to approximate the weighted sum in Winnow for learning DNF.[5] Then they evaluated this algorithm on simple simulated data sets. We extend their work by adding optimizations (Sect. 3.2), applying other sampling techniques (Gibbs, Metropolized Gibbs and parallel tempering), and empirically evaluating this approach with very small sample sizes and mixing times.

## 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 Winnow with

[5] 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.

learning rate $\alpha$, 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. $W_t$ is a function of $\alpha$ 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 $\alpha_i$. This single sequence of updates is determined by running the learning algorithm with the original learning rate $\alpha$. Hence $w_{t,P}(\alpha_i)/w_{t,P}(\alpha_j) = (\alpha_i/\alpha_j)^{\varpi_t(P)}$.

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 $P' = (p'_1, \ldots, p'_n) \in \Omega'_t$, $P = (p_1, \ldots, p_{n_t}) \in \Omega_t = \{0, 1\}^{n_t}$ is 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 $\Omega_t$ that are based on the Metropolis sampler (see Sect. 4.1). Each chain $M_t(\alpha') \in \mathcal{M}_t$ has a specific learning rate $\alpha'$ 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 $P_e = (0, 0, \ldots, 0)$ (i.e. $m_t$ is the number of prediction mistakes of Winnow so far). Then they get

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

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

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

and $W_t(\alpha_{1,t}) = W(1) = |\Omega_t| = 2^{n_t}$. Therefore, for each value $\alpha_{2,t}, \ldots, \alpha_{r_t,t}$, $S_t$ 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

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

Table 1 shows Chawla et al.'s MCMC solution. The following result holds for this estimation procedure.

**Theorem 1** (Chawla et al. 2004) *Assume $a \leq f_{i,t} \leq b$ for all $i$, let the sample size $S_t = \lceil 130 r_t b/(a\epsilon^2) \rceil$ 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 $\epsilon a/(5br_t)$. Then for any $\delta > 0$, $\hat{W}_t(\alpha)$ satisfies*

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

**Table 1** Chawla et al.'s MCMC solution for Winnow

1: Given an example $\mathbf{x}$ to predict, a learning rate $\alpha$ and $m_t$ (the number of Winnow prediction mistakes so far)
2: $r_t \leftarrow \lceil \frac{\ln(\alpha)}{\ln(1+1/m_t)} \rceil + 1$
3: $\hat{W}_t(\alpha) \leftarrow 2^{n_t}$
4: **for** $i = 2$ to $r_t$ **do**
5: $\quad \alpha_{i,t} \leftarrow (1 + 1/m_t)^{i-1}$
6: $\quad$ compute the sample mean $X_{i,t}$ by sampling from $M_t(\alpha_{i,t})$ (Sect. 4.1)
7: $\quad \hat{W}_t(\alpha) \leftarrow \hat{W}_t(\alpha)/X_{i,t}$
8: **end for**
9: **if** $\hat{W}_t(\alpha) < \theta$ **then**
10: $\quad$ return $\hat{y}_t(\mathbf{x}) \leftarrow 0$
11: **else**
12: $\quad$ return $\hat{y}_t(\mathbf{x}) \leftarrow 1$
13: **end if**

Chawla et al. showed that for Winnow learning DNF, $1/e \leq f_{i,t} \leq e$ for all $i$. Thus we can apply Theorem 1 using $a = 1/e$ and $b = e$.

As stated in Theorem 1, Chawla et al.'s algorithm can yield $\epsilon$-good approximations for $W_t$ if enough samples are drawn after the chain mixes. The implication of this is that if the weighted sums are sufficiently far from Winnow's threshold $\theta$ (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 an MCMC simulation of Winnow on any sequence of examples is at most[6] $8 + 14Kn \ln(k + 1)$, where $k = \max_j\{k_j\}$ and $\{1, \ldots, k_j\}$ is the set of possible values of attribute $j$. Unfortunately, whether the chain is rapidly mixing is unknown, which means an exponential number of burn-in steps may be needed before any sample can be drawn in order for Chawla et al.'s results to hold. These negative theoretical results on Chawla et al.'s algorithm are not surprising, since no efficient algorithm is known to efficiently learn DNF formulas.

In addition to the burn-in time of the chains and the number of samples drawn on Line 6 of Table 1, the time complexity of Chawla et al.'s algorithm for trial $t$ is influenced by the number of chains, which is $r_t - 1 = \lceil \frac{\ln(\alpha)}{\ln(1+1/m_t)} \rceil = \Theta(m_t \ln(\alpha))$. If each $W_t$ is sufficiently far from $\theta$, then we know that $m_t \leq 8 + 14Kn \ln(k + 1)$. Otherwise, it could conceivably grow beyond this bound. We now briefly explore other means of setting $r_t$.

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 $S$. Thus one possible way of reducing the computation time is to reduce $r_t$. However, it turns out that reducing $r_t$ does not gain us anything. Specifically, our analysis in the Appendix (specifically, Corollary 6) shows that, for our bounds to apply, reducing $r_t$ requires a larger sample size and a smaller variation distance between the empirical distribution and the chain's stationary distribution (i.e. longer simulations of the chain to draw samples). This means that the result of our MCMC solution would become worse if we reduce the number of chains without drawing more samples. In the next section we describe ways to use fewer chains without reducing the accuracy of our Winnow simulations.

[6] If instead we have $W_t(\alpha) \in [\frac{\theta}{(1+\epsilon)}, \frac{\theta}{(1-\epsilon)}]$ for some $t$, then the mistake bound may not hold (though further theoretical analysis may yield new mistake bounds). Further, to ensure in the worst case that $W_t$ is never in this range, $\epsilon$ would have to be exponentially small. See Chawla et al. for more discussion on this.

### 3.2 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 $\theta$ 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 of any Winnow input), 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 $\mathcal{C} = 2^{n_t}\prod_{i=2}^{r_t}(\frac{\alpha_{i,t}}{\alpha_{i-1,t}})^{\wp_t^{min}}$ and $\mathcal{D} = 2^{n_t}\prod_{i=2}^{r_t}(\frac{\alpha_{i,t}}{\alpha_{i-1,t}})^{\wp_t^{max}}$. Now we can prove the following theorem.

**Theorem 2** *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 $\frac{W_t(\alpha_{i,t})}{W_t(\alpha_{i-1,t})}(\frac{\alpha_{i,t}}{\alpha_{i-1,t}})^{-\wp_t^{min}} \geq 1$. Then

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

Similarly we can prove the second statement. □

Theorem 2 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 $\Psi'$ that satisfy (1), and $\Gamma_0(\Psi_t)$ be the set of all $\Psi'$ 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 3.

**Theorem 3** *If $\Psi_1^{min}$ exists, $\{r_t, r_t - 1, \ldots, r_t - |\Psi_1^{min}| + 1\}$ is a minimum 1-prediction set, and if $\Psi_0^{min}$ 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

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

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 $\Psi' = \{r_t, r_t - 1, \ldots, r_t - |\Psi_1^{min}| + 1\}$:

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

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 3, 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 $2, 3, \ldots, r_t$. This yields our optimized MCMC solution in Table 2.

In Table 2, 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 ($\langle r_t, r_t - 1, \ldots, 2 \rangle$ or $\langle 2, 3, \ldots, r_t \rangle$) 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 $t$th 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 $W_1(\alpha) = 2^{n_t}$ for all examples. If $2^{n_t} \geq \theta$, $y_1' = 1$, otherwise 0.

Another question is how small $\Psi_1^{min}$ and $\Psi_0^{min}$ can be. Intuitively, if $W_t(\alpha)$ is very close to the threshold $\theta$, the chance for our algorithm to stop early is small. Below we give upper bounds of the sizes of $\Psi_1^{min}$ and $\Psi_0^{min}$.

**Table 2** Optimized MCMC solution for Winnow

Given an example $\mathbf{x}$ to predict, a learning rate $\alpha$ and $m_t$ (the number of Winnow prediction mistakes so far), let $y'_t$ be our guess of Winnow's most likely prediction of $\mathbf{x}$'s label, as described in the text
$r_t \leftarrow \lceil \frac{\ln(\alpha)}{\ln(1+1/m_t)} \rceil + 1$
**if** $y'_t = 1$ **then**
  $\hat{W}_t(\alpha) \leftarrow 2^{n_t} \prod_{i=2}^{r_t} (\frac{\alpha_{i,t}}{\alpha_{i-1,t}})^{\wp_t^{min}}$, where $\alpha_{i,t} = (1 + 1/m_t)^{i-1}$
  **for** $i = r_t$ to 2 **do**
    compute the sample mean $X_{i,t}$ with $M_t(\alpha_{i,t})$
    $\hat{W}_t(\alpha) \leftarrow (\frac{\alpha_{i,t}}{\alpha_{i-1,t}})^{-\wp_t^{min}} \hat{W}_t(\alpha)/X_{i,t}$
    **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**
  $\hat{W}_t(\alpha) \leftarrow 2^{n_t} \prod_{i=2}^{r_t} (\frac{\alpha_{i,t}}{\alpha_{i-1,t}})^{\wp_t^{max}}$
  **for** $i = 2$ to $r_t$ **do**
    compute the sample mean $X_{i,t}$ with $M_t(\alpha_{i,t})$
    $\hat{W}_t(\alpha) \leftarrow (\frac{\alpha_{i,t}}{\alpha_{i-1,t}})^{-\wp_t^{max}} \hat{W}_t(\alpha)/X_{i,t}$
    **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**

**Theorem 4** *If $W_t(\alpha) \geq \theta$, let $W_t(\alpha) = (1+\epsilon)\theta$, where $\epsilon \geq 0$. Then the size of $\Psi_1^{min}$ 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 $\Psi_0^{min}$ is at most $r_t - 1 + \ln(1-\epsilon)$.*

*Proof* If $W_t(\alpha) \geq \theta$, then $\Psi_1^{min}$ exists. Let $\Psi_1^k = \{r_t, r_t - 1, \ldots, r_t - k + 1\}$ (so $|\Psi_1^k| = k$). According to Theorem 3, $\Psi_1^k$ is a minimum 1-prediction set if $k$ is the minimum value that makes $\Psi_1^k$ satisfy (1).

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

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

Notice $\theta$ is always greater than 0 for standard versions of Winnow that only maintain positive weights. Since $\theta > 0$, $W_t(\alpha_{r_t-k,t}) = \sum_{P\in\Omega_t} \alpha_{r_t-k,t}^{\varpi_t(P)}$, $\alpha_{1,t} = 1$, $W_t(\alpha_{1,t}) = 2^{n_t}$,

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

and

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

Equation (4) is equivalent to (1), so $\Psi_1^k$ 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

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

and

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

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 4, our optimized MCMC solution uses at least $\ln(1+\epsilon)$ fewer chains than Chawla et al.'s solution if the prediction is 1 and at least $\ln(1-\epsilon)^{-1}$ fewer chains if the prediction is 0. Also we note that our solution will use fewer chains if $W_t$ is farther away from $\theta$, saving more computation time.

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

All that remains is efficiently drawing samples from the chains (lines 6 and 16 in Table 2). Chawla et al. (2004) applied the Metropolis sampler, which is a very popular MCMC method, to the state space $\Omega_t$. Here we also look at other MCMC sampling techniques, including Gibbs sampler, Metropolized Gibbs sampler, and parallel tempering. Implemented properly, each technique provably yields an ergodic Markov chain with the desired stationary distribution, and thus application of each technique will draw samples from near the stationary distribution.

### 4.1 Metropolis sampler for Winnow

The Metropolis sampler (Metropolis et al. 1953) can be applied to problems where the states of the Markov chain are 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 near the chain's stationary

**Table 3** The Metropolis sampler for Winnow for transition from state $P$ to state $Q$

1: With probability $1/(n_t + 1)$ 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 $\min\{1, \frac{\pi_{\alpha,t}(Q')}{\pi_{\alpha,t}(P)}\}$, where

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

else let $Q = P$.

distribution $\pi$, the Metropolis sampler repeatedly considers random changes to the variables defining the current state $P$, yielding a new state $Q$, which it then probabilistically accepts or rejects (in the latter case, $P$ remains the current state). Specifically, first the Metropolis sampler decides with some non-negligible probability whether to follow a self loop. If it does not follow the self loop, then it randomly chooses one variable of the current state $P$ and changes its value according to a *proposal distribution*, resulting in a new candidate state $Q$. Then state $Q$ is accepted with probability $\min\{1, \pi(Q)/\pi(P)\}$.

Chawla et al. defined the chain $M_t(\alpha)$ with state space $\{0, 1\}^{n_t}$ 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 $M_t(\alpha)$, with transitions defined as in Table 3.[7]

### 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 iterated over and is replaced with a value picked from its distribution conditioned on the current values of all other variables. In this case, there is no notion of acceptance as there is with Metropolis; all changes are automatically accepted.

In our application, for any state $P \in \Omega_t$, each variable $p_i$ only has two possible values: 0 and 1. If $P_0 = (p_1, \ldots, p_i = 0, \ldots, p_{n_t})$ and $P_1 = (p_1, \ldots, p_i = 1, \ldots, p_{n_t})$, 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 the Gibbs sampler for $M_t(\alpha)$ makes transitions as in Table 4.

The Gibbs sampler has a number of distinct features. The acceptance rate of the Gibbs sampler is equal to 1. Therefore, unlike Metropolis sampling, Gibbs sampling accepts each new value generated from the conditional distribution. In contrast, the Metropolis sampler probabilistically accepts the new value. Second, the conditional distributions of the Gibbs sampler are constructed based on prior knowledge of $\pi$. 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.

[7] We compute the weights of $P$ and $Q'$ by evaluating them on the training set (see Sect. 2.1).

**Table 4** The Gibbs sampler for Winnow for transition from state $P$ to state $Q$

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

$$\pi_{\alpha,t}(1 - q_i \mid Q \setminus \{q_i\}) = w_{t,Q'}/(w_{t,Q'} + w_{t,Q}) = 1/(1 + \alpha^{\varpi_t(Q) - \varpi_t(Q')});$$

5: **end for**

### 4.3 Metropolized Gibbs sampler for Winnow

The Metropolized Gibbs sampler (Liu 1996) is a modification of the Gibbs sampler that uses an acceptance probability. 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 Sect. 4.2, each component in $\Omega_t$ has only two possible values. So in this case 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 probability in line 4 of Table 4 with

$$\min\left\{1, \frac{\pi_{\alpha,t}(1 - q_i \mid Q \setminus \{q_i\})}{1 - \pi_{\alpha,t}(1 - q_i \mid Q \setminus \{q_i\})}\right\},$$

where

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

### 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 a Markov chain, using e.g. a Metropolis sampler or Gibbs sampler. In each swap step, two chains exchange their current states. For example, if chain $i$ is in state $P_i$ and chain $j$ is in state $P_j$, then after a swap step, chain $i$ will be in state $P_j$ and chain $j$ will be in state $P_i$. Swap steps are introduced to allow greater mobility and faster mixing. After each chain makes 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 $\min\{1, \frac{\pi_i(P_j)\pi_j(P_i)}{\pi_i(P_i)\pi_j(P_j)}\}$.

**Table 5** Parallel tempering for Winnow

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

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

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**

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 5. In Table 5, 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 Sect. 3.2). In order to apply our optimized solution, we partition the sequence into small groups, use parallel tempering in each group and run these groups sequentially. Then we apply our optimized solution on these groups. However, in this case we might not find the best subset of chains as indicated in Theorem 3 since we will only check the constraints after the simulation of a group of chains instead of a single chain.

## 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[8] according to an 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 $\biguplus_i S_i$ be a partition of $S$. The average class entropy of the partition is:

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

where the entropy function $H(S) = -\sum_{j=1}^{m} \mathcal{P}(C_j, S)\log_2 \mathcal{P}(C_j, S)$, $m$ is the number of classes and $\mathcal{P}(C_j, S)$ 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

[8] In our experiments, we set the number of intervals to be 10, which gave us a good ability to distinguish values without becoming computationally difficult.

be obtained by a weighted summation over the impurities of its subsets. Cumulativity facilitates incremental evaluation of impurity values. So we can apply the dynamic programming scheme suggested by Elomaa and Rousu.

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 $\Omega$ 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.

### 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 Sect. 2.1). We used this approach in our experiments of Sect. 6 to handle missing attributes in the Vote and Annealing data sets.

### 5.3 Multi-class classification

Since Winnow 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. Our optimized solution of Sect. 3.2 can be directly applied to the training procedure without any change. But for testing, since we need to compare weighted sums to make predictions, we can first use our optimized algorithm to make predictions for all Winnows. If only a single Winnow predicts 1, we predict the class with that Winnow. If more than one Winnow predicts 1 or all of them predict 0, we then estimate the weighted sums for these Winnows and compare them. 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 Vote data from the UCI repository (Blake et al. 2004). We then tested our optimized algorithm on the simulated data and five UCI data sets (see Table 6) 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. Finally, we explored the effect of significantly reducing the mixing times and sample sizes.

**Table 6** UCI data sets used in our experiments

| Data set | No. of attributes ($n$) | No. of classes | No. of examples |
|---|---|---|---|
| Iris | 4 | 3 | 150 |
| Car | 6 | 4 | 1728 |
| Breast Cancer | 9 | 2 | 286 |
| Vote | 16 | 2 | 435 |
| Auto | 25 | 7 | 205 |
| Annealing | 38 | 6 | 798 |

We started our 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 $\geq 300$. 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 $\Omega_t$ is $n$, the number of variables. Thus for most of our experiments, we set the burn-in time $T_0 = n^2$. Our experiments showed that this burn-in time worked very well, though we also found that it often sufficed to choose a much smaller value of $T_0$. 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[9] $T_s$ for all six samplers.

### 6.1 Comparisons of computation cost

First we report speedups of our optimized algorithm (Sect. 3.2) over Chawla et al.'s solution in terms of the total numbers of Markov chains that are used. In Tables 7 and 8, "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 $\tau$ is the number of trials. "Savings" is the percentage of chains our algorithm saved, that is, Savings $= 1 -$ (Opt MCMC/MCMC). "Savings$_{\text{theory}}$" is our lower bound of the percentage of chains our algorithm can save according to the worst-case analysis of Theorem 4. To compute Savings$_{\text{theory}}$, we computed the theoretical number of chains saved for each trial and summed them over all trials. So Savings$_{\text{theory}} = \sum_{t=1}^{\tau} |\ln(Sav_\theta(W_t))|/\text{MCMC}$, where $Sav_\theta(W_t) = 1 + \epsilon_t$ if $W_t \geq \theta$, and $Sav_\theta(W_t) = 1 - \epsilon_t$ if $W_t < \theta$. Since it is not practical to compute $W_t$ for Auto and Annealing, we cannot compute Savings$_{\text{theory}}$ for these data sets. In the tables, $\bar{\varepsilon}$ is the average over all trials $t$ of $\varepsilon_t = |(W_t - \theta)/\theta|$, which is the normalized distance between the true weighted sum $W_t$ and threshold $\theta$.

Results in Tables 7 and 8 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 for this is

[9]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 Sect. 4).

**Table 7** Comparisons of the total number (in thousands) of Markov chains on simulated data

| $n$ | $\bar{\varepsilon}$ | MCMC | Opt MCMC | Savings (%) | $\text{Savings}_{\text{theory}}$ (%) |
|---|---|---|---|---|---|
| 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 8** Comparisons of the total number (in thousands) of Markov chains on UCI data sets

| Data Set | $\bar{\varepsilon}$ | MCMC | Opt MCMC | Savings (%) | $\text{Savings}_{\text{theory}}$ (%) |
|---|---|---|---|---|---|
| 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 |
| Vote | 0.587 | 1185.5 | 1132.0 | 4.5 | 0.48 |
| Auto | – | 13822.8 | 13751.7 | 0.5 | – |
| Annealing | – | 4855.2 | 3781.6 | 22.1 | – |

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 are more chances for our algorithm to stop early. As shown in Tables 7 and 8, when $\text{Savings}_{\text{theory}}$ is large, the true savings is also large.

### 6.2 Comparisons on estimating weighted sums

Our second set of experiments was 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 predictions in these experiments, we did not test 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). Thus 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 relative error of the estimates: $|\hat{W} - W|/W$.

Figure 1 shows the results on Vote,[10] 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 each point in Fig. 1 represents averages over more than $435 \cdot 200 = 87000$ estimates.

In Fig. 1, 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 the Markov chains may be close to the stationary distribution $\pi_t$. The parallel tempering

[10]The curves for parallel tempering were nearly exactly coincident with those of their non-PT counterparts, so the PT curves are omitted for clarity's sake.

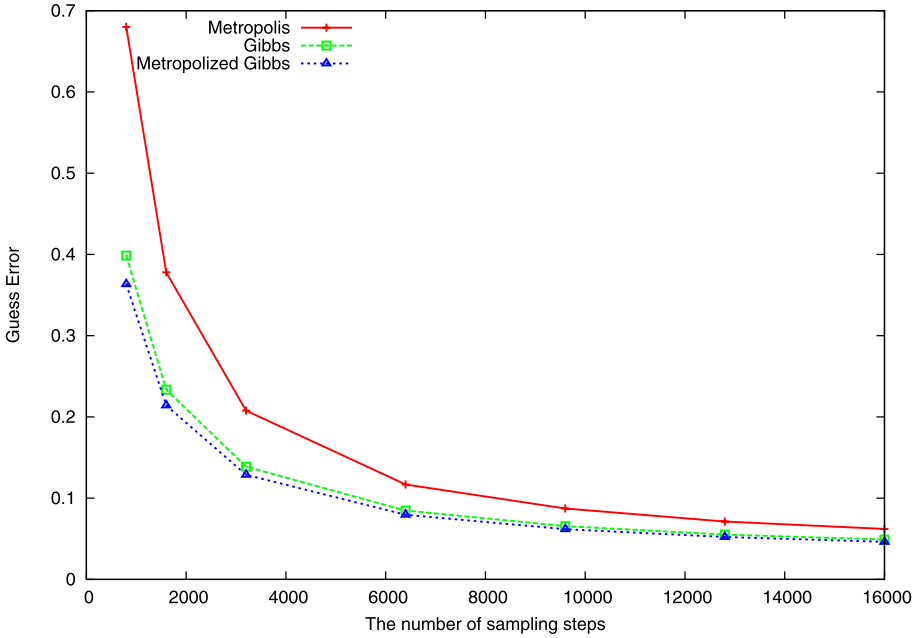


**Fig. 1** Guess error vs. the number of sampling steps $T_s$ on Vote

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 an improvement. For all $T_s$, Guess Errors of Gibbs and Metropolized Gibbs are consistently lower than those of Metropolis. This is especially true for smaller values of $T_s$. Although Metropolized Gibbs has a lower Guess Error than Gibbs, the difference between these two samplers is very small, especially when $T_s \in \{9600, 12800, 16000\}$.

To evaluate the effect of varying the number of attributes $n$, we measured Guess Error of the six samplers on the simulated data.[11] In Fig. 2, $T_s$ is fixed at 10000 and $T_0 = n^2$. Gibbs and Metropolized Gibbs are still consistently 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. As with Vote, parallel tempering made a negligible difference, so its curves are omitted. Considering the results in Figs. 1 and 2, we submit that Gibbs is (by a small margin) the best choice on average in terms of the accuracy of estimating weighted sums.

### 6.3 Comparisons on prediction error

Now we describe experiments that examine the prediction error of MCMC-based Winnow. In contrast with the experiments of Sect. 6.2, we did not update Winnow's weights with the brute force version. Instead, we had one instance of Winnow per algorithm, each updating

[11] It seems impractical to run brute force Winnow when $n$ is larger than 20, but the simulated data is randomly generated from monotone DNF. Thus each variable only can be 1 or 0. Since we do not need to consider the variables with 0 value (see Sect. 3.1), the expectation of the number of variables needed by brute force Winnow is $n/2$. So we can run brute force Winnow even when $n = 40$.

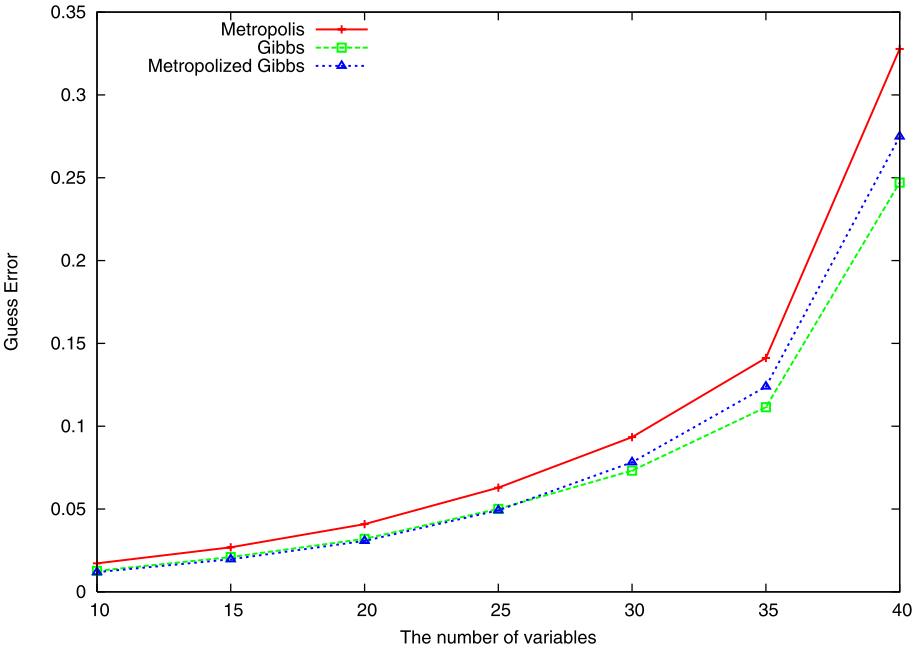


**Fig. 2** Guess error vs. the number of variables $n$ on simulated data

**Table 9** Comparisons of prediction errors with 95% confidence intervals on UCI data sets ($T_0 = n^2$ and $T_s = 10n^2$). Best results are in **bold**

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

M—Metropolis, G—Gibbs, MG—Metropolized Gibbs, PT—Parallel tempering, BF—Brute force

its weights independently of the others. We performed $k$-fold cross-validation on six data sets from the UCI repository and computed 95% confidence intervals (we did not run brute force Winnow on Auto and Annealing). Since Winnow can only make binary predictions, we trained one Winnow DNF learner for each class. After training, we took an unlabeled example, estimated the weighted sums of all Winnows on that example, and predicted the class with the Winnow that had the maximum weighted sum. Table 9 summarizes the results. For each data set, all six samplers have similar performance. All confidence intervals of

**Table 10** Comparisons of prediction errors with 95% confidence intervals on Vote ($n = 16$ and $T_0 = 256$). Best results are in **bold**

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

M—Metropolis, G—Gibbs, MG—Metropolized Gibbs, PT—Parallel tempering, BF—Brute force

each data set overlap. Although no sampler showed a significant advantage over the others, we note that prediction errors of all three samplers are often lower than their PT versions. Another interesting fact is that brute force Winnow lacks a statistically significant advantage over any MCMC-based Winnow. Thus for the data sets we tested, there does not seem to be any disadvantage to using an MCMC-based approximation in place of brute force.

We then reran some experiments from Sect. 6.2, on both simulated data and Vote. For Vote, 10-fold cross-validation was done for each experiment. Table 10 reports the results. Again, all confidence intervals overlap, implying that there's little difference in performance when varying the sampler (including brute force) or $T_s$. That said, non-PT samplers often had a slight edge over their PT counterparts for $n < 9600$, with the slight advantage switching to PT for $n \geq 9600$. It is possible that this happens because when $T_s$ is small, Guess Error of all samplers is high, inducing high variance in the estimates $\hat{W}$. Since parallel tempering allows greater mobility (see Sect. 4.4), PT would introduce more variation, which may make it hard for Winnow to learn. Finally, note that brute force does not have any significant advantage over the approximations.

Table 11 reports results for simulated data. Here we fixed $T_s$ and varied $n$. (When $n \geq 100$, we set $T_0 = 100$ rather than $n^2$ since the results were nearly identical for the two settings and the former setting greatly speeds execution.) Each experiment involved 10 training/testing sets. No differences were significant at the 95% level. In particular, we found no significant advantage to using parallel tempering, nor to using brute force.

From all the above results, we found that in cases where $n$ is small enough to run brute force, MCMC-based Winnow's performance at least matches that of brute force. We also found that parallel tempering introduces more random behavior in MCMC, making its performance harder 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. We also found that as $n$ grows, we can keep $T_0$ relatively much smaller than $n^2$ and still perform comparably to setting $T_0 = n^2$. We explore this phenomenon further in Sect. 6.4.

### 6.4 Experiments on significantly reduced $S$ and $T$

Chawla et al. (2004) ran experiments on simulated data with very good results using surprisingly small values of $S$ and $T$. We further investigate this by testing on the Auto data set, using values of $S$ and $T$ that are significantly reduced from those used by Chawla et al.

**Table 11** Comparisons of prediction errors with 95% confidence intervals on simulated data ($T_0 = n^2$ and $T_s = 10000$ for $n \leq 40$, $T_0 = 100$ and $T_s = 10000$ for $n \geq 100$). Best results are in **bold**

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

| $n$ | 100 | 200 | 400 |
|---|---|---|---|
| M | $18.8 \pm 5.1$ | $33.8 \pm 9.7$ | $37.2 \pm 9.1$ |
| G | $20.0 \pm 4.9$ | $33.4 \pm 8.0$ | $36.4 \pm 8.7$ |
| MG | $\mathbf{18.2 \pm 4.4}$ | $\mathbf{32.4 \pm 8.8}$ | $36.8 \pm 9.2$ |
| M+PT | $19.6 \pm 5.3$ | $33.8 \pm 8.6$ | $36.6 \pm 8.5$ |
| G+PT | $19.0 \pm 4.9$ | $33.0 \pm 8.9$ | $\mathbf{35.2 \pm 9.2}$ |
| MG+PT | $18.4 \pm 5.3$ | $34.2 \pm 8.5$ | $36.6 \pm 8.4$ |

M—Metropolis, G—Gibbs, MG—Metropolized Gibbs, PT—Parallel Tempering, BF—Brute force

Since we used such small values of $S$ and $T$, we used multiple runs to draw samples instead of a single run (see Sect. 2.3). One reason for multiple runs is that samples are independent even with small $S$ (for a single run, the samples are highly dependent with small $S$). Another reason for multiple runs is that our small $T$ means the MCMC algorithms can only explore a small state space close to the origin. Since each state or term in that space has fewer specified attributes than those farther from the origin, Winnow in a sense learns from a restricted hypothesis space by limiting the number of attributes appearing in terms. This serves as an implicit regularizer, which may reduce the risk of overfitting.

To evaluate the effect of varying $S$ and $T$, we measured prediction accuracy of our algorithm on the Auto data set via 6-fold cross validation. The first thing we noticed is that in Figs. 3 and 4, training and testing errors level off, suggesting that Winnow can converge to a hypothesis even with very small values of $S$ and $T$. Further, we note that even though the final test error rates are higher in Figs. 3 and 4 than they are for Auto in Table 9, their 95% confidence intervals overlap.[12]

In Fig. 3, $T$ is fixed at 100 and $S \in \{10, 50, 100\}$. We see that for this value of $T$, changing $S$ has little effect on test or train error. In Fig. 4, a more pronounced difference is visible. For a fixed value of $S = 50$, varying $T \in \{10, 50, 100\}$ changed both training and testing error. This suggests that for relatively small $S$ and $T$, varying $T$ has a greater impact on performance than varying $S$. What is unusual about Fig. 4 is that while error varies with varying $T$, increasing $T$ does not necessarily reduce error. In fact, it appears that the opposite may be true (for most training iterations, the differences between the $T = 10$ and $T = 50$ test error

[12]To simplify the plots, error bars are omitted. For training iteration 80, 95% confidence intervals ranged from $\pm 4$ to $\pm 7.4$.

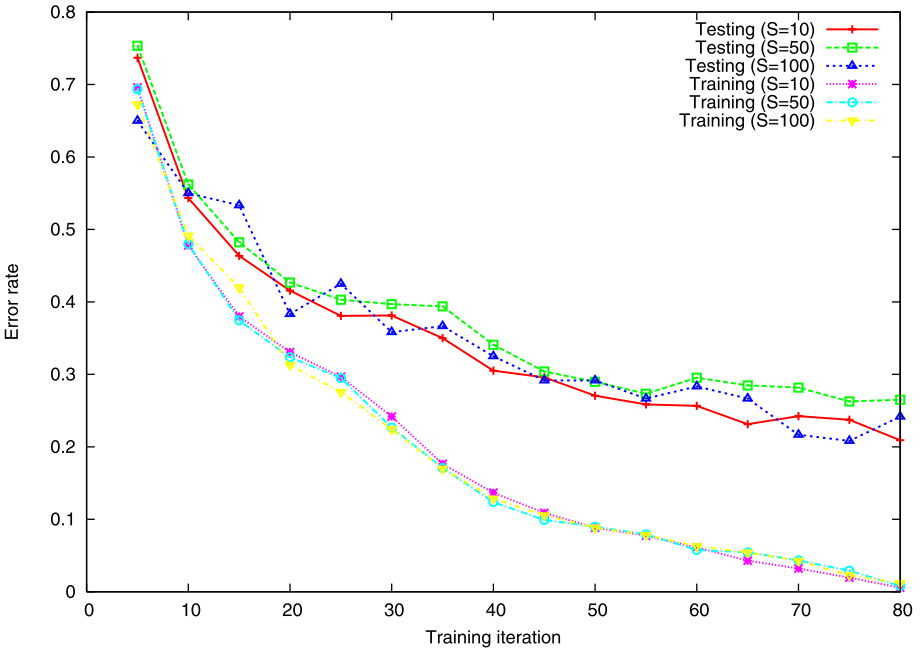


**Fig. 3** Prediction errors vs. training iterations on Auto data set for $T = 100$

**Table 12** Prediction errors with 95% confidence intervals of MCMC-based Winnow, brute-force Winnow and C4.5. Best results are in **bold**

| Data Set | MCMC | Brute force | C4.5 |
|---|---|---|---|
| Iris | $\mathbf{5.3 \pm 3.1}$ | $7.3 \pm 3.2$ | $6.7 \pm 7.7$ |
| Car | $\mathbf{2.7 \pm 1.3}$ | $3.3 \pm 1.1$ | $6.4 \pm 1.1$ |
| Breast cancer | $\mathbf{30.0 \pm 5.1}$ | $33.3 \pm 4.8$ | $34.7 \pm 5.1$ |
| Vote | $\mathbf{4.4 \pm 1.8}$ | $5.0 \pm 2.0$ | $5.7 \pm 2.0$ |
| Auto | $\mathbf{19.8 \pm 5.3}$ | – | $22.9 \pm 9.8$ |
| Annealing | $\mathbf{0.6 \pm 0.6}$ | – | $6.3 \pm 1.5$ |

curves are significant at the 90% level). This may be a result of the implicit regularization mentioned earlier.

In Table 12, we compare our algorithm with C4.5 (Quinlan 1996). We performed $k$-fold cross-validation on all data sets. All algorithms used the same partitions. We set $S = 20$ and $T = 50$ for MCMC-based Winnow. We noticed that Winnow has lower error rates than C4.5, some significant at the 95% confidence level. We also note that the results for such small values of $S$ and $T$ are very similar to those from Table 9. Finally, we see that yet again, brute-force Winnow has higher error rates than MCMC-based Winnow, though confidence intervals overlap.

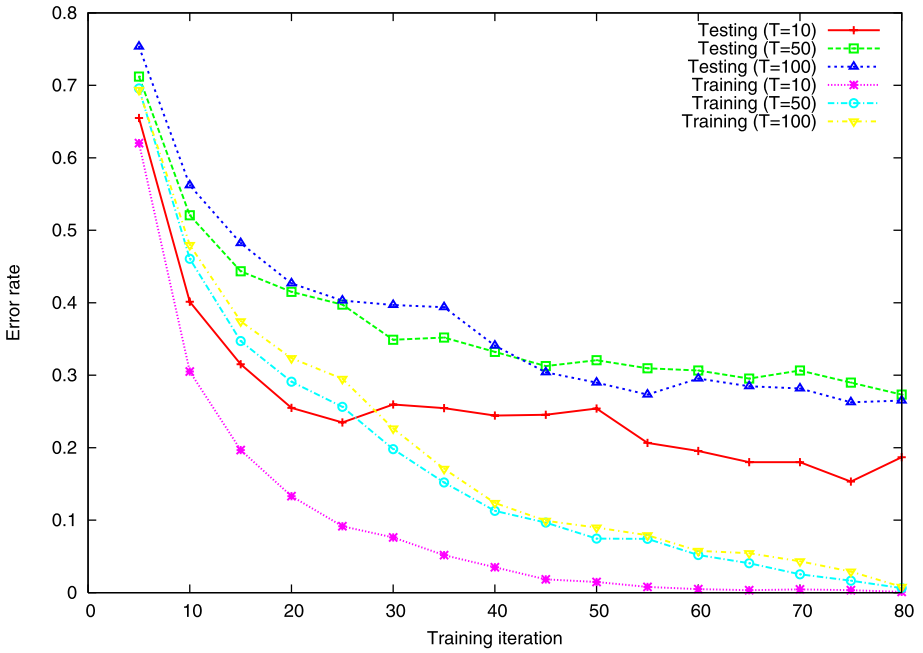


**Fig. 4** Prediction errors vs. training iterations on Auto data set for $S = 50$

## 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 solution without 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 of the smallest size and gave lower bounds on how many chains our solution can save. We empirically compared three new MCMC sampling techniques: Gibbs, Metropolized Gibbs and parallel tempering. They often 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 has good performance on average. Further, we found that the approximation algorithms had prediction error that was comparable or superior to C4.5 and a brute force version of Winnow that exactly computed the weighted sums. This advantage held even when very small sample sizes and mixing times were used.

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, one could investigate the effect of pruning terms (i.e. using only terms with high weight) on prediction accuracy, found via both the Markov chain and a genetic algorithm. Other interesting work is to develop 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. (2005) discuss this idea in a slightly different context. Also, Khardon et al. give a negative result for exactly simulating Winnow for learning DNF

(Sect. 2.1). Does a similar result exist for randomized algorithms approximately 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. The authors thank the anonymous referees for their comments. Qingping Tao completed this work at the University of Nebraska.

## Appendix: 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 $S$. Thus one possible way of reducing the computation time is to reduce $r_t$. However, as we show below, reducing $r_t$ does not gain us anything, since for our bounds to apply, it requires a larger sample size and a smaller variation distance between the empirical distribution and the chain's stationary distribution (i.e. longer simulations of the chain to draw samples).

Define $f'_{i,t}(P) = w_{t,P}(\alpha'_{i-1,t})/w_{t,P}(\alpha'_{i,t})$, where $\alpha'_{i,t} = (1+\kappa/m'_t)^{i-1}$ for $1 \le i \le r'_t$, $r'_t$ is the smallest integer such that $(1+\kappa/m'_t)^{r'_t-1} \ge \alpha$, and $\kappa$ 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 $\kappa$ or use a smaller $m'_t$, we would decrease $r'_t$. In Corollary 6 below, we extend Theorem 1. We start with a lemma used to prove Corollary 6.

**Lemma 5** *For any distribution $\pi$ of $\Omega_t$, if $m'_t \ge \max\{u_t(P_e), v_t(P_e)\}$, then 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 of any Winnow input. 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, $m'_t \ge \max\{|\wp_t^{max}|, |\wp_t^{min}|\}$.

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 $\varpi(P) = u_t(P) - v_t(P)$. Since

$$\left(1+\frac{\kappa}{m'_t}\right)^{\varpi(P)} \le \left(1+\frac{\kappa}{m'_t}\right)^{\wp_t^{max}} \le \left(1+\frac{\kappa}{m'_t}\right)^{m'_t} \le e^{\kappa}$$

and

$$\left(1+\frac{\kappa}{m'_t}\right)^{\varpi(P)} \ge \left(1+\frac{\kappa}{m'_t}\right)^{\wp_t^{min}} \ge \left(1+\frac{\kappa}{m'_t}\right)^{-m'_t} \ge e^{-\kappa},$$

we have $e^{-\kappa} \le 1/f'_{i,t}(P) \le e^{\kappa}$ and $e^{-\kappa} \le f'_{i,t}(P) \le e^{\kappa}$. □

By substituting Lemma 5's bounds into Theorem 1, we get Corollary 6.

**Corollary 6** *Let the sample size* $S_t = \lceil 130e^{2\kappa}\hat{r}_t/\epsilon^2 \rceil$ *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* $\epsilon/(5e^{2\kappa}\hat{r}_t)$. *If* $m'_t \geq \max\{u_t(P_e), v_t(P_e)\}$, *then for any* $\delta > 0$, $\hat{W}_t(\alpha)$ *satisfies*

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

According to Corollary 6, 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 Sect. 3.1). But Corollary 6 tells us that if we increase $\kappa$ by 1, we potentially 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 $\kappa$.

## References


Auer, P., & Warmuth, M. (1998). Tracking the best disjunction. *Machine Learning*, *32*(2), 127–150.

Blake, C., Keogh, E., & Merz, C. J. (2004). *UCI repository of machine learning databases*. http://www.ics.uci.edu/~mlearn/MLRepository.html.

Blum, A., Chalasani, P., & Jackson, J. (1993). On learning embedded symmetric concepts. In *Proceedings of the sixth annual workshop on computational learning theory* (pp. 337–346). New York: ACM Press.

Blum, A., Furst, M., Jackson, J., Kearns, M., Mansour, Y., & Rudich, S. (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., Jackson, J., & Tamon, C. (2004). More efficient PAC-learning of DNF with membership queries under the uniform distribution. *Journal of Computer and System Sciences*, *68*, 205–234.

Chawla, D., Li, L., & Scott, S. D. (2004). On approximating weighted sums with exponentially many terms. *Journal of Computer and System Sciences*, *69*(2), 196–234.

Dietterich, T. G., Lathrop, R. H., & Lozano-Perez, T. (1997). Solving the multiple-instance problem with axis-parallel rectangles. *Artificial Intelligence*, *89*(1–2), 31–71.

Elomaa, T., & Rousu, F. (1999). General and efficient multisplitting of numerical attributes. *Machine Learning*, *36*(3), 201–244.

Geman, S., & Geman, D. (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: proceedings of the 23rd symposium on the interface* (pp. 156–163).

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

Jerrum, M., & Sinclair, A. (1996). The Markov chain Monte Carlo method: An approach to approximate counting and integration. In D. Hochbaum (Ed.), *Approximation algorithms for NP-hard problems* (pp. 482–520). Kent: PWS.

Khardon, R., & Servedio, R. (2005). Maximum margin algorithms with boolean kernels. *Journal of Machine Learning Research*, *6*, 1405–1429.

Khardon, R., Roth, D., & Servedio, R. (2005). Efficiency versus convergence of boolean kernels for online learning algorithms. *Journal of Artificial Intelligence Research*, *24*, 341–356.

Klivans, A., & Servedio, R. A. (2004). Learning DNF in time $2^{\tilde{O}(n^{1/3})}$. *Journal of Computer and System Sciences*, *68*(2), 303–318.

Littlestone, N. (1988). Learning quickly when irrelevant attributes abound: a new linear-threshold algorithm. *Machine Learning*, *2*(4), 285–318.

Littlestone, N., & Warmuth, M. K. (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., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., & Teller, E. (1953). Equation of state calculation by fast computing machines. *Journal 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.

Quinlan, J. R. (1996). Improved use of continuous attributes in C4.5. *Journal of Artificial Intelligence Research*, *4*, 77–90.

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

Tao, Q., & Scott, S. (2003). An analysis of MCMC sampling methods for estimating weighted sums in Winnow. In *Intelligent engineering systems through artificial neural networks (ANNIE 2003)* (Vol. 13, pp. 15–20), St. Louis, MO, November 2003.

Tao, Q., & Scott, S. (2004). A faster algorithm for generalized multiple-instance learning. In *Proceedings of the 17th international Florida artificial intelligence research society conference (FLAIRS)* (pp. 550–555).