MAKING EFFICIENT LEARNING ALGORITHMS WITH EXPONENTIALLY MANY FEATURES

by

Qingping Tao

A DISSERTATION

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfillment of Requirements

For the Degree of Doctor of Philosophy

Major: Computer Science

Under the Supervision of Professor Stephen D. Scott

Lincoln, Nebraska

December, 2004

MAKING EFFICIENT LEARNING ALGORITHMS WITH EXPONENTIALLY MANY FEATURES

Qingping Tao, Ph.D.

University of Nebraska, 2004

Advisor: Stephen D. Scott

Expanding the learning problems' input spaces to high-dimensional feature spaces can increase expressiveness of the hypothesis class and thus may improve the performance of linear threshold-based learning algorithms such as Perceptron and Winnow. However, since the number of features is dramatically increased, these algorithms will not run efficiently unless special techniques are used. Such techniques include Monte Carlo approaches, grouping strategies and kernels. We investigated these techniques and applied them to two problems: DNF learning and generalized multiple-instance learning (GMIL).

For DNF learning, we used a new approach to learn generalized (non-Boolean) DNF formulas, which uses all (exponentially many) possible terms over n attributes as inputs to Winnow. Then the weighted sums to Winnow are approximated using a Markov chain Monte Carlo (MCMC) method. We proposed an optimized version of this basic algorithm, which produces exactly the same classifications while often using fewer Markov chain simulations. We also empirically evaluated four MCMC sampling techniques in terms of accuracy of weighted sum estimates.

For the GMIL problem, our work is based on Scott et al.'s algorithm GMIL-1 for their GMIL model, which has exponential time and space complexity since it uses all possible boxes in a discretized space as features. We proposed a Winnow-based algorithm GMIL-2 with a new grouping strategy, which has the same generalization ability as GMIL-1 and can save more than 98% time and memory in practice. Then we proposed a kernel that

exactly corresponds to the feature mapping used by GMIL-1. We showed that this kernel is #P-complete to compute, and then gave a fully polynomial randomized approximation scheme for it. Our kernel showed improvements in both generalization error and time complexity over GMIL-1 and GMIL-2, and outperformed other MIL algorithms on benchmark data sets. We also proposed two extensions to further improve its generalization abilities. Both GMIL-2 and the kernels have been successfully applied to several applications: drug discovery, content-based image retrieval and biological sequence analysis.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor, Dr. Stephen D. Scott for his guidance and support during my stay at the University of Nebraska-Lincoln. His insight into the great potential of Monte Carlo approaches in machine learning opened the gate to this research. His boundless enthusiasm, infinite patience and constant encouragement have expedited the completion of this thesis. I also want to thank Dr. Vinodchandran N. Variyam for his invaluable discussion and especially for sharing his expertise in computational complexity theory and for his direct contributions to Chapters 4 and 5 in this thesis. Along with Dr. Scott and Dr. Variyam, I would like to thank Dr. Ashok Samal, Dr. Jitender Deogun and Dr. John Orr for serving on my supervising committee and carefully reading this thesis.

Second, I thank all former and present members of Dr Scott's group for all of their helps over years. I also want to thank my colleagues who ever cooperated with me and helped me during my studies at the University of Nebraska-Lincoln. I would like to thank all my friends for making my stay in Lincoln enjoyable and memorable for life. There have been so many people who have directly or indirectly influenced the successful completion of this research. I wish to express my gratitude to all of them.

Finally, I thank my parents Xinlian Li and Siguo Tao for their constant support and love throughout my life. I am greatly indebted to them. I would like to thank my dear sister Baiyun Tao for believing in me and hosting me for many vacations. Special thanks to my lovely wife Jie Bai, who is the most important person I met during my Ph.D. study.

This research was funded in part by NSF grants CCR-0092761 and EPS-0091900, and a grant from the NU 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.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Constructing computer programs that can *learn* from experience is the goal of machine learning. Much of learning involves inducing general concepts from specific training examples or experience. Usually training examples are described by a set of *attributes or features* and assigned with *labels*. Each concept can be thought as a general function defined over these features, for example, a boolean function. A learning algorithm is used to approximate a target function that can correctly identify the labels of unseen examples.

Many successful machine learning algorithms have been developed and applied to all kinds of applications such as information retrieval, natural language processing and bioinformatics. One class of widely used algorithms are linear threshold-based learning algorithms such as Perceptron [36] and Winnow [26]. A general hypothesis generated by these algorithms is a linear function $W(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$ where $\mathbf{x}$ is the feature vector and $\mathbf{w}$ is its weight vector. A learning algorithm takes an example $\mathbf{x}$, calculates $W(\mathbf{x})$, then predicts +1 if $W(\mathbf{x})$ is greater than a threshold θ and -1 otherwise.

Although linear functions have limited expressiveness, they have been applied successfully to real world classification problems by expanding their features. For example, a set of features $x_1, \cdots, x_n$ can be expended using conjunctions such as $\neg x_i \wedge x_j \wedge x_k$ if they are

boolean-valued features, or using polynomial terms such as $x_i^2 x_j x_k$ if they are real-valued features. I.e. the original training examples are expanded to higher-dimensional examples, in which each conjunction or term is a basic feature. Clearly these expanded features increase expressiveness and thus may improve the performance of linear threshold-based learning algorithms. However, the number of features is dramatically increased. For example, if all conjunctions are used, the number of expanded features is 3^n. Thus these algorithms can not be used in practice unless special techniques are used to compute the weighted sums of the inputs efficiently.

There are several ways to solve this problem. For learning algorithms that use additive weight updates such as Perceptron, one approach is to use kernel techniques [38], such as boolean kernels [23] and polynomial kernels, to implicitly expand features as long as kernels can be efficiently computed. For learning algorithms that use multiplicative weight updates such as Winnow, there is no known kernel that can be efficiently computed in a general case. So we look for other approaches to this problem. One way is to exploit commonalities among the inputs, partitioning them into a polynomial number of groups such that given a single member of each group, the total weight contribution of that group can be efficiently computed. For example, it has been shown that Winnow can be simulated efficiently over exponentially many features for learning some simple geometric concept classes [30]. Unfortunately many applications do not appear to exhibit such structure. Although it seems that a brute-force implementation is the only option to guarantee complete correctness, a fast approximation algorithm may exist.

In this dissertation, we study various techniques to efficiently compute weighted sums when exponentially many expanded features are used. The first problem we investigated is learning DNF formulas. We presented a new approach for learning generalized (non-Boolean) DNF formulas. It works by implicitly enumerating all possible terms over n attributes and using these terms as inputs to Winnow to learn a disjunction over them. We

applied an approximation algorithm to estimate weighted sums of Winnow with a Markov Chain Monte Carlo (MCMC) method. Based on this basic algorithm, We gave an optimized version [41] that produces exactly the same classifications while often using fewer Markov chain simulations. We also applied four MCMC sampling techniques and empirically compared them.

The second problem we have been working on is generalized multiple-instance learning (GMIL) [40, 50]. Since Dietterich et al. [14] introduced the multiple-instance learning (MIL) model, it has been successfully applied to several applications such as drug discovery and content-based image retrieval. Recently, Scott et al. [40] generalized the MIL model and presented an algorithm (GMIL-1) for GMIL, which uses all possible boxes in a discretized space as its features and thus has exponential time and space complexity. In our work, we first gave a new Winnow-based algorithm GMIL-2 [43] that uses a compact set of GMIL-1's features. We evaluated GMIL-2 on two conventional MIL application areas (drug discovery and content-based image retrieval). Our experimental results show that it has the same generalization ability as GMIL-1 and needs much less time and memory to train. Furthermore, we applied GMIL-2 to a new application area, biological sequence analysis, and got promising results [47].

Then we defined a kernel $k_\wedge$ [42] that exactly corresponds to the feature mapping used by GMIL-1 when Perceptron is used. We showed that $k_\wedge$ is #P-complete to compute and presented a fully polynomial randomized approximation scheme (FPRAS) for it. Our results showed improvements in both generalization error and time complexity over GMIL-1 and GMIL-2 on all three application areas. Further, $k_\wedge$ outperformed other MIL algorithms on benchmark data sets (Musk data sets). Based on $k_\wedge$, We proposed two extended kernels, $k_{\min}$ [44] that generalizes Weidmann et al's count-based MIL model [50], and a normalized kernel. Both of them generalized noticeably better than $k_\wedge$ in several applications.

The rest of the dissertation is as follows. In the next chapter, we introduce our work on

applying Winnow for DNF learning. In Chapter 3 we describe the generalized multiple-instance learning model and our fast Winnow-based algorithm GMIL-2. In Chapter 4 we present the kernel $k_\wedge$ for GMIL, its hardness results and a FPRAS for it. In Chapter 5 we propose two extensions to $k_\wedge$ and present the experimental results. Then we conclude in Chapter 6.

Chapter 2

Learning DNF Formulas with Winnow

2.1 Introduction

Chawla et al. [10] 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 Winnow algorithm [26] 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.

In this chapter, our aim is to explore possible ways that can speed up Chawla et al.'s algorithm. We also want to show that the algorithm is very useful in practice even without any theoretical guarantees.

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 the

lower bounds on how much computation time our algorithm can 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 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 when compared to brute force Winnow (where weighted sums are exactly computed), so generalization accuracy is not compromised by our 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 [43] (adapted from Goldman et al. [19] for learning concepts from a generalization of the multiple-instance model [14]) 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 this chapter is as follows. In Section 2.2 we discuss related work. Section 2.3 describes Chawla et al.'s MCMC approach for estimating weighted sums and presents our optimized version. Section 2.4 gives four different MCMC sampling techniques. Section 2.5 gives several extensions of the basic Winnow algorithm. We then report our experimental results in Section 2.6 and conclude in Section 2.7.

2.2 Related Work

2.2.1 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.

Example 2.1 (Monotone DNF) *A monotone DNF is a DNF 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: ϕ, 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 "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 [8, 9, 23]), but positive results exist only in restricted cases, and the general DNF problem remains open [1].

[1] It is unlikely that an efficient distribution-free (PAC) DNF-learning algorithm exists [6, 5].

The algorithm Winnow of Littlestone [26] 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 the update 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 [2]. 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 DNF.

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[3] in n. Thus another approach is needed to compute W_t. One possibility is to use kernels, as illustrated by Khardon et al. [23] 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. [10], who use

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

[3] 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.

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.

2.2.2 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. [33] and Geman et al.[17], 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 $T = S \cdot T_0$. 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 π 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 π after taking a polynomial-length random walk through Ω. More formally, we measure the difference between $\hat{\pi}_{P,t}$ and π by *variation distance:*

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

The number of simulation steps T required for $\Delta_P(T) \leq \epsilon$ for all $P \in \Omega$ 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 ϵ 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 ϵ 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. [20]). Chawla et al. [10] combined these two solutions and gave a Metropo-

lis sampler [33] to approximate the weighted sum in Winnow for learning DNF[4]. Then they evaluated this algorithm on simple simulated data sets. We extend their work by adding optimizations (Section 2.3.3) and applying three other sampling techniques: Gibbs, Metropolized Gibbs and parallel tempering.

2.3 Estimating Weighted Sums with MCMC

2.3.1 Chawla et al.'s MCMC Solution for Winnow

We now describe Chawla et al.'s MCMC solution [10] 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[5] 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 the Metropolis sampler (see Section 2.4.1). 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 $m_t = u_t(P_e) + v_t(P_e)$

[4]They also used a similar technique to approximate weighted sums in Weighted Majority [27] of classifiers created by boosting [37] for predicting nearly as well as the best pruning of an ensemble.

[5]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 $w_{t,P}(\alpha_i)/w_{t,P}(\alpha_j) = (\alpha_i/\alpha_j)^{\varpi_t(P)}$.

where $P_e = (0, 0, \ldots, 0)$. Then they get

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

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.

Theorem 2.1 *[10] Assume $a \leq f_{i,t} \leq b$ for all i. let the sample size $S_t = \lceil 130 \hat{r}_t b/(a\epsilon^2) \rceil$ 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 $\epsilon a/(5b\hat{r}_t)$. Then for any $\delta > 0$, $\hat{W}_t(\alpha)$ satisfies*

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

As stated in Theorem 2.1, Chawla et al.'s algorithm can yield ϵ-good approximations for W_t if enough samples are drawn after the chain mixes. But 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.

2.3.2 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 2.4. 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 $1 \leq i \leq r'_t$, r'_t is the smallest integer such that $\left(1 + \kappa/m'_t\right)^{r'_t - 1} \geq \alpha$, 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 2.3 below, we extend Theorem 2.1.

Lemma 2.2 *For any distribution π of Ω_t, if $m'_t \geq \max\{u_t(P_e), v_t(P_e)\}$, for all i and P, $e^{-\kappa} \leq f'_{i,t}(P) \leq e^{\kappa}$.*

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 $u_t(P_e), v_t(P_e) \geq 0$, $u_t(P_e) \geq \wp_t^{max} \geq \wp_t^{min} \geq -v_t(P_e)$. Therefore, $\hat{m}_t \geq \max\{|\wp_t^{max}|, |\wp_t^{min}|\}$.

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,

$$\left(1+\frac{\kappa}{m_t^{'}}\right)^{\varpi(P)} \leq \left(1+\frac{\kappa}{m_t^{'}}\right)^{\wp_t^{max}} \leq \left(1+\frac{\kappa}{m_t^{'}}\right)^{m_t^{'}} \leq e^{\kappa},$$

and

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

then we have,

$$\begin{aligned} e^{-\kappa} \leq \ & 1/f_{i,t}^{'}(P) \ \leq e^{\kappa} \\ e^{-\kappa} \leq \ & f_{i,t}^{'}(P) \ \leq e^{\kappa} \end{aligned}$$

□

By substituting Lemma 2.2's bounds into Theorem 2.1, we get Corollary 2.3.

Corollary 2.3 *If $m_t^{'} \geq \max\{u_t(P_e), v_t(P_e)\}$, 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 $\epsilon/(5e^{2\kappa}\hat{r}_t)$. 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 Corollary 2.3, 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 2.3.1). But Corollary 2.3 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 Corollary 2.3 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.

2.3.3 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:

$$\mathcal{C} \prod_{i \in \Psi'} \frac{W_t(\alpha_{i,t})}{W_t(\alpha_{i-1,t})} \left(\frac{\alpha_{i,t}}{\alpha_{i-1,t}}\right)^{-\wp_t^{min}} \geq \theta, \tag{2.1}$$

$$\mathcal{D} \prod_{i \in \Psi'} \frac{W_t(\alpha_{i,t})}{W_t(\alpha_{i-1,t})} \left(\frac{\alpha_{i,t}}{\alpha_{i-1,t}}\right)^{-\wp_t^{max}} < \theta, \tag{2.2}$$

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 2.4 *If $\exists \Psi' \subseteq \Psi_t$ that satisfies condition (2.1) , then $W_t(\alpha) \geq \theta$; If $\exists \Psi' \subseteq \Psi_t$ that satisfies condition (2.2), 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 2.4 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 (2.1), and $\Gamma_0(\Psi_t)$ be the set of all Ψ' that satisfy (2.2). 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 2.5.

Theorem 2.5 *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 2.5, 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 2.1.

In Table 2.1, 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 ($< r_t, r_t - 1, \cdots, 2 >$ or $< 2, 3, \cdots, r_t >$) 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 $W_1(\alpha) = 2^{n_t}$ for all examples. If $2^{n_t} \geq \theta$, $y_1' = 1$, 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 2.6 *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*

Table 2.1: 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

---

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 2.5, Ψ_1^k is a minimum 1-prediction set if k is the minimum value that makes Ψ_1^k satisfy (2.1).

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 (2.1) can be written

as

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

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

$$\sum_{P\in\Omega_t} \quad \alpha_{r_t-k,t}^{\varpi_t(P)-\wp_t^{min}} \leq (1+\epsilon)2^{n_t} \tag{2.3}$$

$$\sum_{P\in\Omega_t} \quad (\alpha_{r_t-k,t}^{\varpi_t(P)-\wp_t^{min}} - (1+\epsilon)) \leq 0, \tag{2.4}$$

(2.4) is equivalent to (2.1). So Ψ_1^k is a minimum 1-prediction set if k is the minimum solution of (2.4). 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

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

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 (2.4). Therefore the minimum solution of (2.4) 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 2.6, our optimized MCMC solution uses at least $\ln(1+\epsilon)$ fewer chains than that [10]'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.

2.4 Sampling from $\pi_{\alpha,t}$

All that remains is efficiently drawing samples from the chains (lines 5 and 15 in Table 2.1). [10] 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.

2.4.1 Metropolis Sampler for Winnow

The Metropolis sampler [33] 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] 2.2.

2.4.2 Gibbs Sampler for Winnow

The Gibbs sampler [17] is widely applicable to problems where the variables have conditional distributions of a parametric form that can easily be sampled from. In a single

[6]We compute the weights of P and Q' by evaluating them on the training set (see Footnote 3).

Table 2.2: The Metropolis sampler for Winnow

With probability $1/n_t$ let $Q = P$; otherwise,
Select i uniformly at random from $1, \ldots, n_t$ and let $Q^{'} = (p_1, \ldots, p_{i-1}, 1 - p_i, \ldots, p_{n_t})$;
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 $Q = P$.

---

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 2.3.

Table 2.3: The Gibbs sampler for Winnow

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

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

end for

The Gibbs sampler has a number of distinct features. The acceptance rate of the Gibbs sampler is uniformly equal to 1.Therefore, First, it accepts each new value generated from the conditional distribution. (Of course, the value might be the same as the old value.) In contrast, the Metropolis sampler accepts the new value with some probability. Second, 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.

2.4.3 Metropolized Gibbs Sampler for Winnow

The Metropolized Gibbs sampler [28] 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 2.4.2, 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 2.3 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

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

2.4.4 Parallel Tempering for Winnow

The idea of parallel tempering [18] 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 2.4. In Table 2.4, 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 2.3.3). 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 2.5 since we will only check the constraints after the simulation of a group of chains instead of

a single chains.

Table 2.4: 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**

2.5 Other Extensions

2.5.1 Discretizing Real-Valued Attributes

Since our algorithm can only handle discrete-valued attributes, we employ the method of Elomaa and Rousu [16] 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 [16] 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.

2.5.2 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 2.2.1). We also tried the popular approach of assigning to the missing attribute the most common value in the same class.

2.5.3 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. Our optimized solution 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.

2.6 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. [10] and Voting data from the UCI repository [4]. 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 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 ≥ 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 [34] 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.

2.6.1 Comparisons on Estimating Weighted Sums

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. [10], which is the average error of the estimates ($|\hat{W} - W|/W$).

Figure 2.1 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 2.1 represents averages over more than 70000 estimates.

[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 2.4).

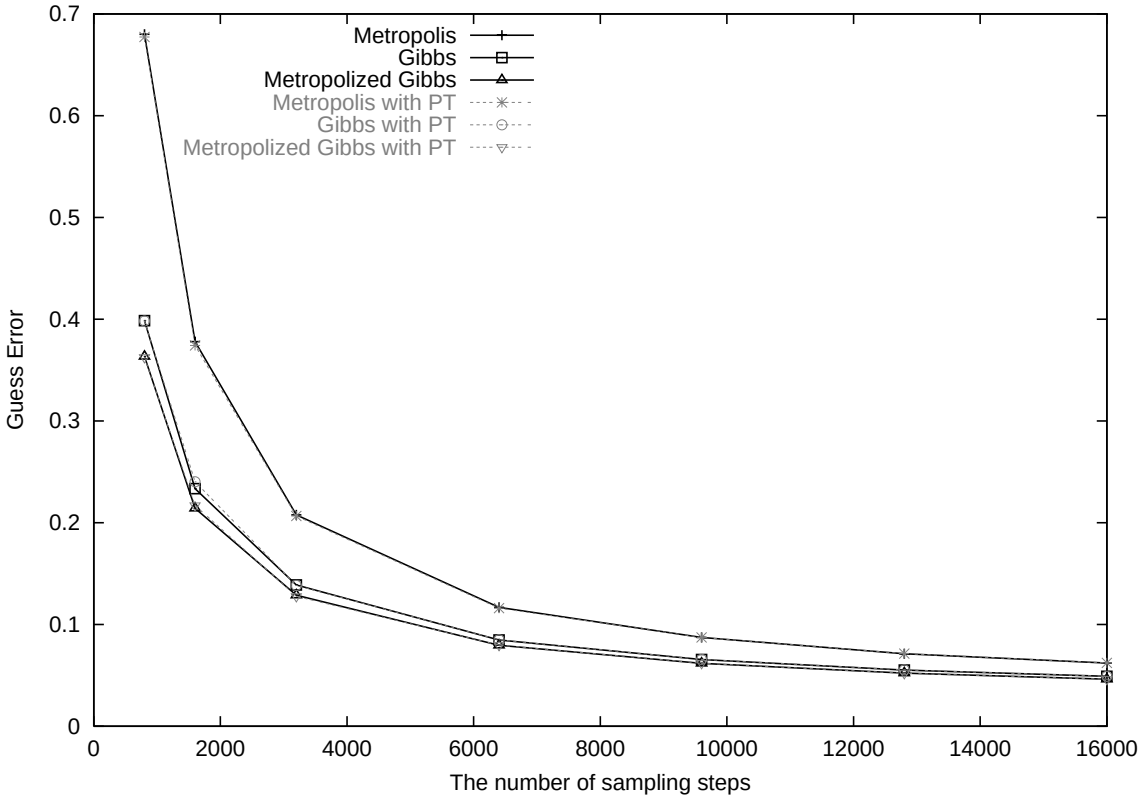


Figure 2.1: Guess Error vs. the number of sampling steps T_s on Voting.

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

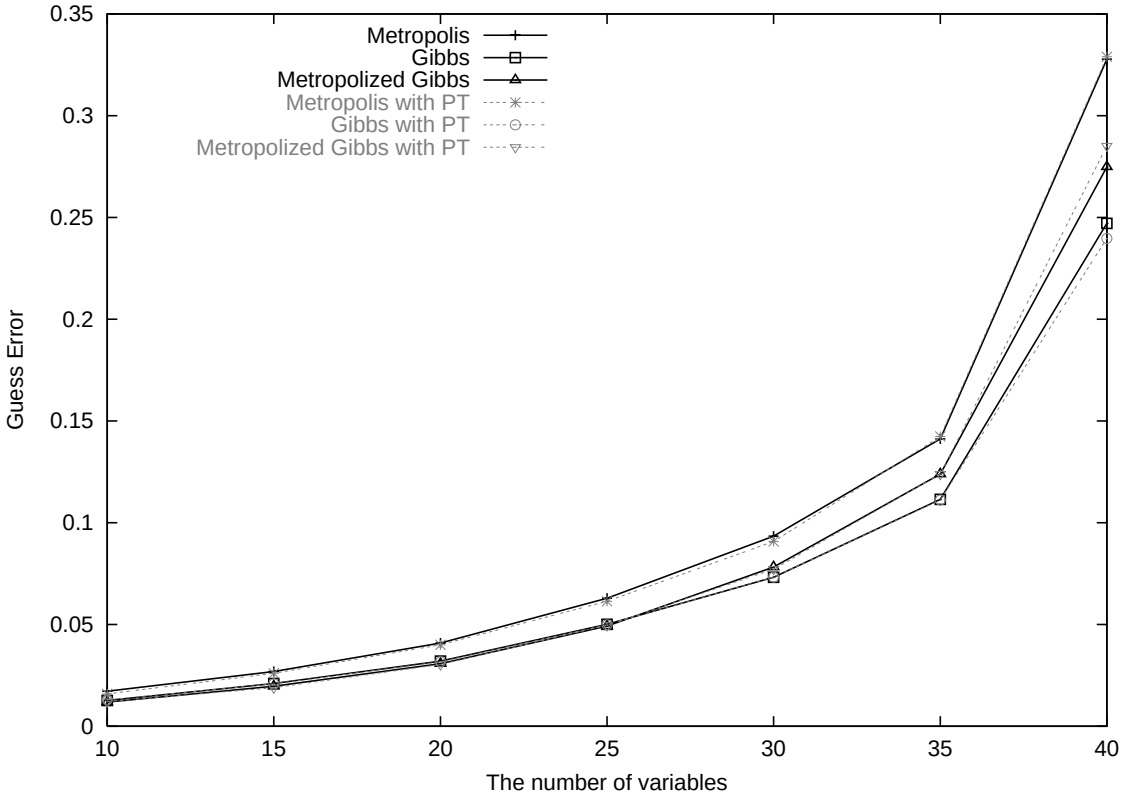


Figure 2.2: Guess Error vs. the number of variables n on simulated data.

of the six samplers on the simulated data [8]. In Figure 2.2, 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. As with Voting, parallel tempering only improved performance very slightly. Considering results in Figures 2.1 and 2.2, we can say that Gibbs sampler is the best choice on average in terms of the accuracy of estimating weighted sums.

Table 2.5: 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	-	-

2.6.2 Comparisons on 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 2.5 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 tested, there does not seem to be any disadvantage

[8] It seems it is impractical to run brute force Winnow when n is bigger than 20. But the simulated data is randomly generated from monotone DNF. Each variable only can be 1 or 0. Since we do not need to consider the variable with 0 value (see Section 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 $n = 40$.

Table 2.6: 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	$\mathbf{5.0 \pm 2.1}$	5.3 ± 1.8	5.0 ± 1.3	4.8 ± 1.6	5.5 ± 2.4
G	$\mathbf{4.6 \pm 2.5}$	$\mathbf{4.8 \pm 1.5}$	5.0 ± 2.4	4.8 ± 2.1	6.0 ± 2.5	$\mathbf{4.6 \pm 1.8}$	$\mathbf{4.3 \pm 2.0}$
MG	5.1 ± 1.1	5.0 ± 1.5	$\mathbf{5.0 \pm 2.1}$	$\mathbf{4.6 \pm 1.6}$	$\mathbf{4.8 \pm 1.8}$	5.0 ± 2.7	5.9 ± 2.5
M+PT	5.3 ± 1.9	4.1 ± 0.9	$\mathbf{5.0 \pm 1.6}$	5.5 ± 1.5	$\mathbf{5.0 \pm 1.8}$	$\mathbf{4.1 \pm 2.0}$	$\mathbf{3.6 \pm 1.9}$
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	$\mathbf{5.1 \pm 1.1}$	$\mathbf{3.2 \pm 0.9}$	5.4 ± 2.5	$\mathbf{4.8 \pm 2.7}$	5.0 ± 2.5	5.0 ± 2.2	5.5 ± 2.4
BF							5.0 ± 2.0

to using an MCMC-based approximation in place of brute force.

To further investigate these observations, we reran the experiments of Section 2.6.1. There were 7 experiments for simulated data and 7 experiments for Voting. For Voting, 10-fold cross-validation was done for each experiment. Table 2.6 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 2.4.4), 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 2.1 and Corollary 2.3. 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 2.7 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

Table 2.7: 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	$\mathbf{6.0 \pm 3.0}$	$\mathbf{7.2 \pm 5.9}$	7.0 ± 7.0
G	1.8 ± 1.6	3.0 ± 2.5	7.8 ± 4.3	$\mathbf{3.4 \pm 3.0}$	8.0 ± 3.5	7.4 ± 4.9	$\mathbf{6.2 \pm 3.8}$
MG	1.8 ± 1.4	$\mathbf{2.6 \pm 2.3}$	$\mathbf{7.2 \pm 3.5}$	3.8 ± 2.9	6.4 ± 3.9	8.8 ± 5.2	7.0 ± 3.9
M+PT	$\mathbf{1.4 \pm 1.5}$	4.4 ± 2.4	7.4 ± 3.7	3.2 ± 2.8	$\mathbf{6.4 \pm 4.1}$	$\mathbf{7.0 \pm 3.8}$	$\mathbf{5.8 \pm 4.5}$
G+PT	1.6 ± 2.0	3.0 ± 1.8	$\mathbf{5.6 \pm 3.1}$	4.4 ± 3.3	7.4 ± 4.5	7.2 ± 3.6	6.0 ± 3.4
MG+PT	1.6 ± 1.5	$\mathbf{2.6 \pm 2.6}$	7.0 ± 4.3	$\mathbf{3.2 \pm 2.3}$	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

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.

2.6.3 Comparisons of Computation Cost

Here we report speedups of our optimized algorithm (Section 2.3.3) over Chawla et al.'s solution in terms of the total numbers of Markov chains that are used. In Tables 2.8 and 2.9, "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

Table 2.8: 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

$Opt\ MCMC = \sum_{t=1}^{\tau} r'_t$, 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 2.6. To compute $Savings_{theory}$, we computed the number of chains saved for each trial and add them up over all trials. So $Savings_{theory} = \sum_{t=1}^{\tau} |\ln(Sav_\theta(W_t))|/MCMC$, 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 $Savings_{theory}$. 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 2.8 and 2.9 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 are more chances for our algorithm to stop early. As shown in Tables 2.8 and 2.9, when $Savings_{theory}$ is big, the true savings is often big.

Table 2.9: 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	-

2.7 Conclusions

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 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.

Chapter 3

A Faster Winnow-based Algorithm for Generalized Multiple-Instance Learning

3.1 Introduction

In multiple-instance learning (MIL), each example is a multiset (called a *bag*) of instances (points) rather than a single instance. Dietterich et al. [14] first introduced the MIL model motivated by the problem of predicting whether a molecule would bind at a particular site. Since shape of a molecule largely determines binding affinity, they represented each molecule by a high-dimensional vector that describes its shape, and labeled molecules that bind at a site as positive examples and those that do not bind as negative. Then they learned an axis-parallel box that distinguishes the positives from the negatives. The motivation for the MIL model is the fact that a single molecule can have multiple conformations (shapes), and only one conformation need bind at the site for the molecule to be considered positive. Thus when an example is negative, all conformations in it are negative, but if an example is positive, then it may be the case that only one conformation of the set is positive, and the learner does not know which one. Since its introduction, the MIL model has been applied

to content-based image retrieval [32, 53], where each instance in a multi-instance example (*bag*) represents a feature of an image, and it is not known which feature corresponds to the content the user wants to retrieve. As with binding prediction, the MIL model used for content-based image retrieval assumes that the label of an example is a disjunction of the labels of the instances in the example.

Recently Scott et al. [40] introduced a generalized MIL (GMIL) model where the target concept is a *set* of points and the label for a bag is determined by a more general (non-disjunctive) function over the attributes. They also gave an algorithm GMIL-1 that learns geometric concepts in the GMIL model and applied it to several application areas including content-based image retrieval, drug discovery, robot vision, and protein sequence analysis, comparing their results to those of DD [31] and EMDD [52], which are very successful algorithms in the conventional MIL model. Their algorithm's classification error rates were consistently competitive with those of DD and EMDD, and in several cases significantly better. Thus the GMIL model is better suited for some applications than the conventional MIL model.

However, GMIL-1 is inherently inefficient, requiring hours of computation time and hundreds of megabytes of memory to train. Here we present a much faster and more memory-efficient algorithm that can handle the requirements of real pattern recognition systems. Our experimental results show that our new algorithm has the same generalization ability as the basic algorithm, but needs much less computation time and memory.

The rest of this chapter is organized as follows. In the following section, we discuss related work in conventional MIL. Then we describe Scott et al.'s GMIL model and their algorithm GMIL-1 for learning in GMIL. After that, we describe our new, faster GMIL algorithm. In the experimental section, we evaluate our algorithm on three application areas (discovery of antagonist drugs, content-based image retrieval and identifying Trx-fold proteins) and compare it to GMIL-1. In the last section we give our conclusions and

future work.

3.2 The Conventional MIL Model

In the original MIL model [14], each example P is a *bag* (multiset) of instances, and P is given a label of positive if and only if at least one of the instances in P is labeled positive (it is unknown which instance(s) in P are labeled positive). Typically, the label of a point $p \in P$ is determined by its proximity to a target point c. They also argued empirically that axis-parallel boxes (APR) around target points are good hypotheses for this and other similar learning problems. An APR b can be defined by its lower bounds and upper bounds on all dimensions. A point p is contained by an APR b if its projection lies between the lower bound and upper bound of b on all dimensions. If point p is contained by a target APR, p's label is positive and negative otherwise. This MIL model has been extensively studied [2, 29, 31, 7, 48, 52, 1], along with extensions for real-valued labels [15, 35] and with applications focusing on molecular binding affinity (related to drug discovery) and content-based image retrieval.

In most MIL work, the label of a bag depends only on the label of a single point, and the label of each point typically is assumed to depend on a single target point. Exceptions include the algorithms DD [31] and EMDD [52] in which a target concept can be a disjunction over multiple points. In Maron et al.'s work, they mapped each pair of instances to a new instance and added spatial information about the instance pair, which defined a pairwise conjunctive type of learning model. However, they found that allowing more than 2 disjuncts in the target concept or taking more than 2 instances at a time proved computationally very difficult. However, such disjunctions have to be severely restricted for the sake of computational efficiency. In other work, De Raedt [13] generalizes MIL in the context of inductive logic programming and defines an interesting framework connecting

many forms of learning, allowing relations between instances. However, the transformations he gives between the models have exponential time and space complexity. Also, such generalizations must be explicitly tuned.

3.3 Scott et al.'s Generalized MIL Model

A naive extension of the conventional MIL model is that a target concept C is a set of $\leq k$ axis-parallel boxes and a bag B is positive if and only if (1) every point of B lies in some box of C, and (2) every box of C contains a point from B. Intuitively, this extension of concept class captures the notion of visual similarity quite effectively. However, in pattern recognition applications, this class is not robust against noise or occlusions. Instead, often what is employed is a target concept C is a set of at most k axis-parallel boxes and a bag B is positive if and only if strictly fewer than s boxes of C do not contain a point from B.

In Scott et al.'s GMIL model [40], the target concept is a set of points $C = \{c_1, \ldots, c_k\}$, and the label for a bag $B = \{p_1, \ldots, p_n\}$ is positive if and only if there is a subset of r target points $C' = \{c_{i_1}, \ldots, c_{i_r}\} \subseteq C$ such that each $c_{i_j} \in C'$ is near some point in B, where "near" is defined as within a certain distance under some weighted norm. Here r is a threshold indicating the minimum number of target points that must each be "hit" by some point from B (the same point in B could hit multiple target points). In other words, if we define a boolean attribute a_i for each target point c_i that is 1 if there exists a point $p_j \in B$ near it and 0 otherwise, then the bag's label is some r-of-k threshold function over the attributes (so there are k relevant attributes and B's label is 1 iff at least r of these attributes are 1). Note that if $r = 1$ then this model is the conventional multi-instance model, except that there are multiple target points and the final concept is a union of these points. If $r = k$ then a conjunctive model exists, where each target point must be hit by some instance in B.

Then the model can be further extended in the following way. If we let $\bar{C} = \{\bar{c}_1, \ldots, \bar{c}_{k'}\}$

be a set of "repulsion" points, then we can require a positive bag to *not* have any points near each point of $\bar{C}' = \{\bar{c}'_{i_1}, \ldots, \bar{c}'_{i_{r'}}\} \subseteq \bar{C}$ in addition to having a point near each point in C'. This means that to be positive, certain feature values have to be present in some points of bag B, but also certain feature values must be *absent*. This proved useful in some of our experiments as well as those of Scott et al..

3.4 GMIL-1: A Winnow-based Algorithm for Generalized Multiple-Instance Learning

Scott et al.'s algorithm (GMIL-1) [40] for learning geometric concepts in the GMIL model assumes that each bag is a multiset of at most n points from the finite, discretized space $\mathcal{X} = \{1, \ldots, s\}^d$. Their algorithm then enumerates all the possible $N = (s(s+1)/2)^d$ axis-parallel boxes in $\mathcal{X}$ and creates two attributes for each box b: a_b and $\bar{a}_b$. Given a bag $B \in \mathcal{X}^n$, the algorithm sets $a_b = 1$ if some point from B lies in b and $a_b = 0$ otherwise. Then[1] $\bar{a}_b = 1 - a_b$. These $2N$ attributes are then given to Winnow [26], which learns a linear threshold unit.

If each bag's label is given by some r-of-k threshold function over the attributes (so there are k relevant attributes and B's label is 1 iff at least r of these attributes are 1), then applying well-known results allows us to conclude that Winnow's generalization error when learning such functions is bounded by a polynomial in k, r, d, and $\log s$. However, its time complexity is $\Omega(s^{2d})$ per trial, which is exponential in both $\log s$ (the number of bits needed to describe each instance) and d. As an example of why this is a problem even for small d, consider the case where $s = 10$ in each dimension for a 4-dimensional space. Then the number of attributes to Winnow in this case is $2 \cdot (10(10+1)/2)^4 \approx 1.83 \times 10^7$.

[1] Both these complementary attributes are required since Winnow only learns monotone disjunctions (defined below).

To overcome the exponential dependence on $\log s$, GMIL-1 partitions the set of N boxes into *groups* such that it is guaranteed that for each box b in a group G, all attributes a_b have the same weight in Winnow, as do all attributes $\bar{a}_b$. Thus the algorithm maintains only one representative box per group G and exactly computes Winnow's weighted sum (i.e. the quantity compared to the threshold when making predictions) by summing the products of each group's weight and its size:

$$\sum_{i=1}^{N} a_i\, w_{a_i} + \bar{a}_i\, w_{\bar{a}_i} = \sum_{G \in \Gamma} |G|\, (a_G\, w_{a_G} + \bar{a}_G\, w_{\bar{a}_G}),$$

where $|G|$ is the number of boxes in group G, a_G is the attribute for group G's representative box, and Γ is the set of all groups. The set of groups is built by using the union of points from all bags[2] to rectilinearly partition $\mathcal{X}$ (see Figure 3.1). The result is a set of *regions* such that any pair of boxes b_i and b_j are in the same group if both have their "lower left" corners in region W and their "upper right" corners in region Z. It is easy to see that when the groups are constructed this way, for each pair of boxes b and b' in the same group G, b and b' contain the same subset of points from all bags. Thus $a_b = a_{b'}$ and $\bar{a}_b = \bar{a}_{b'}$ for all bags, and all the Winnow weight updates are the same for attributes a_b and $a_{b'}$ as well as $\bar{a}_b$ and $\bar{a}_{b'}$. Using this construction, at most $O(m^{2d+1})$ groups are built, where m is the number of points used to partition the space. Thus the exponential dependence on $\log s$ is removed, but the exponential dependence on d remains.

Scott et al. applied GMIL-1 on four application areas: robot vision, content-based image retrieval, protein sequence identification, and drug discovery. Their experiments showed that GMIL-1 is competitive with (and often better than) DD [31] and EMDD [52], which are two of the better algorithms in the conventional MIL model (some of these

[2]When applying GMIL-1 on some applications, Scott et al. also used clustering on the training set to preprocess the data to reduce the number of points m used to build the groups. This was done with data sets where the dimension $d > 2$.

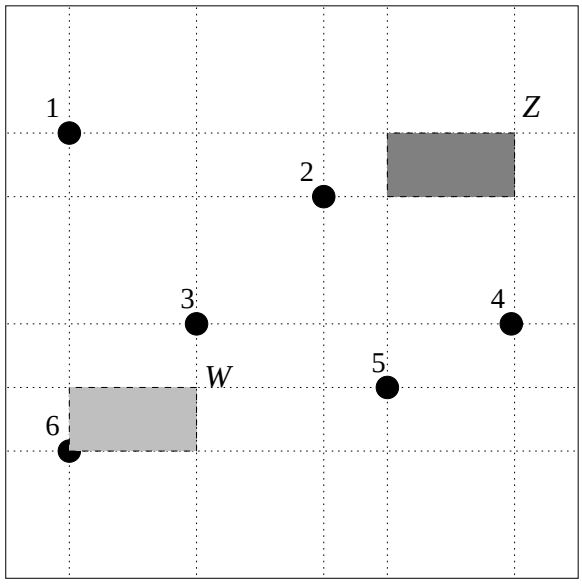


Figure 3.1: An example of how GMIL-1 constructs its groups.

previous results are summarized in the experimental section). But in these experiments GMIL-1 built around 4–7 million groups, requiring 500–700 Mb of memory and around 30 hours to train. However, they also found that after training was completed, most of the groups could be thrown out with no impact on prediction error. Indeed, after training they could always discard at least 80% (typically more than 97%) of the groups with the lowest weight in Winnow and have a hypothesis that predicts nearly as well as the original. This suggests that only a relatively very small subset of groups is needed to learn these concepts well, which motivates our design of GMIL-2.

3.5 The Algorithm GMIL-2

GMIL-2 is similar to GMIL-1 in that it groups boxes together, assigns boolean attributes to these groups, and gives these attributes to Winnow to learn an r-of-k threshold function over these attributes. The key difference from GMIL-1 is how GMIL-2 builds the groups. First, rather than using the union of points from all training bags, GMIL-2 uses a set Ψ of

representative points that capture the distribution of the training bags. Second, rather than basing the construction of the groups on a rectilinear partitioning of $\mathcal{X}$, GMIL-2 directly considers all subsets of Ψ.

3.5.1 The Basic Algorithm

Recall that the reason we can group boxes together in GMIL-1 is that the boxes in each group contain the same set of points. For example, in Figure 3.1, the group defined by regions W and Z is a set of boxes that contains and only contains points 2, 3 and 5. Instead of representing this group as a box, we can represent it as a set $S = \{2, 3, 5\}$. Any box that contains point 2, 3 and 5 and does not contain point 1, 4 and 6 belongs to this group, such as boxes $b1$ and $b2$ in Figure 3.2.

So in this representation, any subset of $\{1, 2, 3, 4, 5, 6\}$ is a potential group. But not all of these subsets are valid groups, e.g. $\{4, 6\}$ since any box containing points 4 and 6 must also contain points 3 and 5. We can easily check if S is valid by finding the smallest box containing S and comparing S with the point set contained by the box (described below).

Suppose a set of points Ψ is given as representatives of a distribution of points in a d-dimensional feature space $\mathcal{X}$. Let S be a subset of Ψ. We call the smallest box that contains S the minimum bounding box (MBX) of S. A subset $S \subseteq \Psi$ is *valid* if there is no point $p \in \Psi \setminus S$ that is contained by the MBX of S, otherwise *invalid*. Each valid subset S represents a group G_S of boxes that contain and only contain S in Ψ. GMIL-2 enumerates all such groups. For each group G_S, we define two attributes: l_S and $\bar{l}_S$. Given a bag $B \in \mathcal{X}^n$, the algorithm sets $l_S = 1$ and $\bar{l}_S = 0$ if for some point p from B, the MBX of $\{p\} \cup S$ does not contain any point from $\Psi \setminus S$. Otherwise it sets $l_S = 0$ and $\bar{l}_S = 1$. These attributes are then given to Winnow.

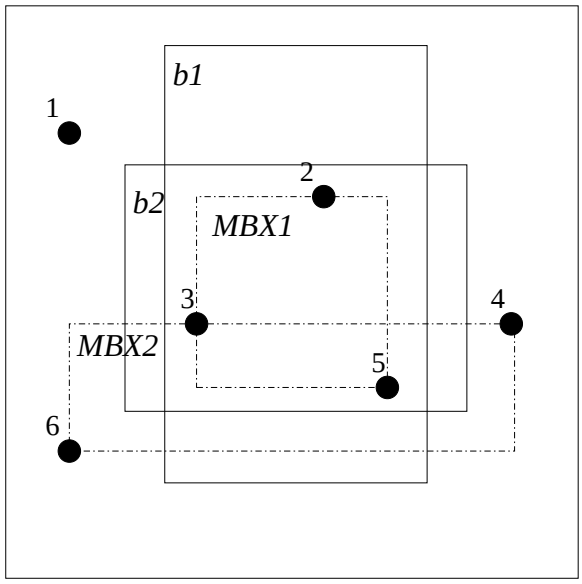


Figure 3.2: An example of how GMIL-2 constructs its groups.

3.5.2 Building the Group Set

GMIL-2 needs to build the group set $\Gamma = \{G_S \mid S \subseteq \Psi$ and S is valid.$\}$. A naive algorithm is enumerating all $2^{|\Psi|}$ subsets of Ψ and checking if they are valid. But it has a time complexity exponential in the number of representatives. A simple improvement can be achieved when $|\Psi| > 2d$, where d is the number of dimensions. Since each box is defined by its lower and upper bounds in each dimension, any MBX of S is determined by at most $2d$ points. Thus we can build a set Υ containing all possible MBXs by enumerating all subsets that have equal or less than $2d$ representatives. From Υ, we can build a group set $\Gamma' = \{S \mid S \subseteq \Psi$ and S is the largest set contained by $b \in \Upsilon\}$. It is easy to see that $\Gamma' = \Gamma$. The time complexity of this algorithm is $O(\sum_{i=0}^{2d} \binom{|\Psi|}{i})$.

We now describe our algorithm to build the set of groups Γ. Obviously, brute-force enumeration and testing of all $2^{|\Psi|}$ subsets of Ψ is impractical for even moderately sized Ψ. Thus our algorithm for building the set of groups exploits the geometric property that many of these $2^{|\Psi|}$ subsets will be invalid. Our algorithm is similar to breadth-first searching of

the set of all possible groups. This BFS algorithm also leads to a heuristic for choosing Ψ in such a way that allows $|\Psi|$ to be quite large while keeping $|\Gamma|$ small, e.g. in our experiments, we used $|\Psi| \in [5, 105]$ but had $|\Gamma| < 10^5$.

Our BFS algorithm first puts into Γ the empty set and all singleton subsets $\{p\}$ for each $p \in \Psi$ because all such subsets are valid. After that, it identifies all valid subsets with size 2 and adds them to Γ, then valid subsets with size 3, and so on up to size $|\Psi|$. To find all valid subsets with size $k+1$, BFS looks at all size-k subsets. For each size-k subset S', BFS considers each point $p \in \Psi \setminus S'$. Each point p is added to S' to form $S'_p = S' \cup \{p\}$. Then BFS builds the MBX b_p for each S'_p and adds to Ψ the largest set $S''_p \subseteq \Psi$ contained by b_p. (Note that we might have $|S''_p| > k+1$. That set is still added to Ψ.)[3] Since any size-$(k+1)$ valid subset can be built by adding one or more points to a valid subset with size less than $k+1$ according to Theorem 3.1, the algorithm is guaranteed to find all valid size-$(k+1)$ subsets.

Theorem 3.1 *For any valid non-empty subset $S \subseteq \Psi$, there exist a valid subset S' and a point p such that $p \in S \setminus S'$ and $MBX_S = MBX_{S' \cup \{p\}}$.*

Proof: Since S is not empty, there is a point $p_1 \in S$. Let $S_1 = \{p\}$. $S_1 \subset S$ and S is valid. Then we pick a point p_2 from $S \setminus S_1$ and let $S_2 = S_1 \cup \{p_2\}$. MBX_{S_2} must lie in MBX_S. Otherwise MBX_S is not the smallest box that contains S since $S_2 \subset S$. Then we build a new subset S'_2 that is the set of points from Ψ that is contained by MBX_{S_2}. Since MBX_S contains MBX_{S_2}, MBX_S must contain S'_2. So $S'_2 \subset S$ and S'_2 is bigger than $S'_1 = S_1$ and also valid.

By repeating the above procedure, we can get a series of S'_i. At each time S'_i gets bigger and $S \setminus S'_i$ gets smaller. So for at most $|S|$ times, we will get a S'_i that $S'_{i+1} = S$ is built

[3]For example, in Figure 3.2, adding 2 into $\{3, 5\}$ yields a valid subset $S'_2 = S''_2 = \{2, 3, 5\}$. But adding 4 into $\{3, 6\}$ yields $S'_4 = \{3, 4, 6\}$ and $S''_4 = \{3, 4, 5, 6\}$. Then S''_4 is added to Ψ.

by adding a p_{i+1} into S'_i unless S is not valid. Then S'_i and p_{i+1} are what we claimed in the theorem. □

The time complexity of our BFS group-building algorithm is obviously $O(|\Psi||\Gamma|)$, compared to the trivial lower bound of $\Omega(|\Gamma|)$ required to completely enumerate Γ. The time complexity of our BFS building algorithm is $O(\sum_{k=1}^{|\Psi|} \rho_i |\Psi|) = O(|\Psi||\Gamma|)$, where ρ is the number of valid size-k subsets. So the computation time is decided by the total number of groups. Since $|\Gamma| \leq \min\{2^{|\Psi|}, |\Psi|^{2d}\}$, at worst case, the time complexity of the BFS algorithm is $|\Psi|$ times of either that of the brute-force enumeration or GMIL-1's building algorithm. That means the speed of our BFS algorithm is relatively close to other two algorithms even in the worst case if Ψ is small. If Ψ is big, the brute-force enumeration and GMIL-1 both need a great amount of time. However usually not all subsets can be valid especially when Ψ is big . Thus $|\Gamma|$ is often much less than $\min\{2^{|\Psi|}, |\Psi|^{2d}\}$.

3.5.3 Selecting Representatives

Obviously the more representatives that are used, the more accurately they represent the true distribution of the points in the bags. This tempts one to let Ψ be the union of the points in all the training bags. However this could make $|\Gamma|$ prohibitively large. Thus we need to reduce $|\Psi|$. One way is clustering all the points in the training bags and setting Ψ to the set of point representatives of these clusters.

Intuitively, the larger Ψ is, the better our resolution will be of the instance space and the better our algorithm will learn. However, because point representatives of clusters will typically lie in general position (i.e. not lie on the same low-dimensional hyperplanes), increasing $|\Psi|$ using cluster representatives only will dramatically increase $|\Gamma|$. To see this, note that if most subsets of the points in Figure 3.2 were collinear instead of in general position, the number of distinct valid groups would decrease significantly. This motivates our method of building Ψ: first we set Ψ to be the point representatives of a small number

of clusters (e.g. 5). Then we add many random points (e.g. 100) to Ψ that are collinear with these point representatives.

We could get better representatives by increasing the number of clusters. But this also increase the computation time of building Γ. We want to find a way to increase the size of Ψ without dramatically increasing the size of Γ. Here we point a general way to make this possible, which uses the fact that S is invalid if the MBX of S contains a point $p \notin S$. Specifically, first we cluster all points in training bags into N clusters. Then we build a grid

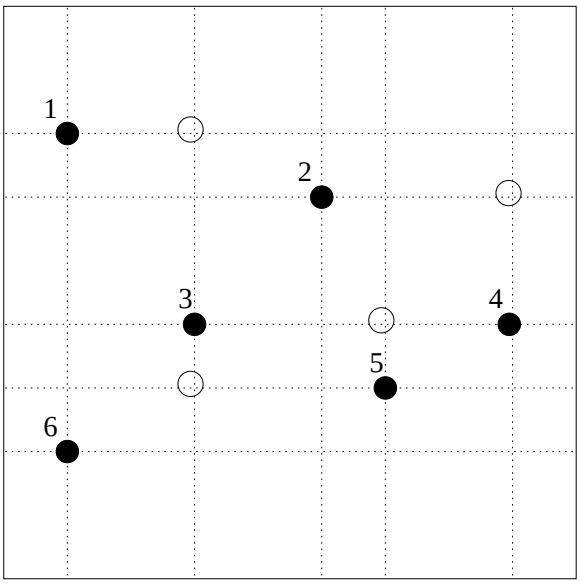


Figure 3.3: An example of how GMIL-2 adds additional points.

$\mathcal{G}$ such that the intervals on each dimension are defined by the projections of N clusters on this dimension. Let $\Omega_{\mathcal{G}}$ be the set of all crossing points on $\mathcal{G}$. We choose the N clusters and K additional random points from $\Omega_{\mathcal{G}}$ as our representatives. For example, in Figure 3.3, we add to the $N = 6$ cluster representatives (black points) $K = 4$ additional (white) points from $\Omega_{\mathcal{G}}$. A valid subset that contains point 3 and 5 must also contain another two white points. In addition, the values of these additional points come from N clusters instead of any other random values. The heuristic is that these values should have more valuable

information than any other arbitrary values if our clusters are meaningful.

3.5.4 Contrasting GMIL-1 with GMIL-2

Obviously the only difference in the running of both learning algorithms is the makeup of the group set Γ. Here we present some more detailed comparisons between the group sets under various special cases of the representative sets.

First consider when Ψ for GMIL-2 is the union of all points in all training bags. Let S be a valid subset of Ψ and let Γ_1^S be the union of all groups for GMIL-1 whose boxes contain exactly S. Then the GMIL-2 group represented by S is exactly equal to Γ_1^S, $l_S = a_G$ and $\bar{l}_S = \bar{a}_G$ for all training bags, and the two algorithms are equivalent. The only difference is that the set of groups (i.e. the set of attributes given to Winnow) is smaller for GMIL-2.

Now consider the case when GMIL-1 builds its grid on a set of representative points rather than the union of all points in all bags (see Footnote 2). Now since the points in Figure 3.2 are only representatives, it is possible that there is e.g. a point x from the training bags that lies in box $b1$ but not in $b2$. From its use of the grid to build its group set, GMIL-1's groups can recognize this fact, but GMIL-2's subset-based construction of Γ prevents this. Thus GMIL-2 operates in a space with lower resolution than GMIL-1, and the only way it can match GMIL-1's resolution is to set $\Psi = \Omega_G$. If this is the case, then the two algorithms are again equivalent.

Thus for special cases, the two algorithms build equivalent group sets. If only a subset of all training points or grid points is used as representative points, GMIL-2 can be treated as an approximation to GMIL-1. The time complexity of GMIL-2 is polynomial in d but in the worst case exponential in $|\Psi|$. Thus we can now scale up our algorithm to high dimensions without much of a time complexity increase. However, we obviously cannot use all points from the training bags, which means we learn on a reduced resolution. This may cause a significant increase in prediction error for large d (we are currently exploring

this). But whether or not GMIL-2 is suitable for high-dimensional data, our experiments on smaller dimensional data confirm that GMIL-2 is significantly faster than GMIL-1. Further, it turns out that the low resolution does not hurt in practice.

3.6 Experimental Results

We now describe our experiments comparing GMIL-2 with GMIL-1 in terms of generalization error and computation costs in two application areas: drug discovery and content-based image retrieval. Our results show that GMIL-2 is much more efficient than GMIL-1 with no significant change in prediction error. We also compare these results with prediction errors of DD and EMDD on the same data sets and a new application area, identifying Trx-fold proteins, where GMIL-1 cannot be applied due to its impractical space and time requirements.

3.6.1 The Application Areas

We applied GMIL-2 on several application areas. First we describe two of them where MIL have been successfully applied, drug discovery and content-based image retrieval. We also describe the potential tasks in these two applications and the corresponding data sets. Then we introduce a new application area, identifying Trx-fold Proteins.

Binding Affinity/Antagonist Drugs

Dietterich et al. [14] introduced the conventional MIL model motivated by predicting whether a conformation of a particular molecule would bind to a single site in another molecule. But an open problem is how to predict “antagonist drugs”, whose jobs are to bind at multiple sites in a single molecule by fitting in several of them simultaneously. This cannot be represented by a single axis-parallel box in the standard MIL model, but there

is a natural representation of this in our model by setting $r = k$. Such a target concept would typically be defined in a very high-dimensional space, since a representation of a conformation is very complex [14].

We used a generalization of the synthetic data of Dooly et al. [15] to reflect the notion of antagonist drugs [14] where a molecule must bind at multiple sites to be labeled positive. Dooly et al. created their data by first generating a single "artificial receptor" t. Then "artificial molecules" (denoted B_i) were created, each with 3–5 instances per bag. The label of bag B_i was determined as follows. For each $B_{ij} \in B_i$, they computed $E_{B_{ij}}$, which is the binding energy of B_{ij} to t. They then identified the instance $B_{ij} \in B_i$ that most strongly binds to t and set $E_{B_i} = E_{B_{ij}}$. They then normalized E_{B_i} to $[0, 1]$ and thresholded it at $1/2$ to get a binary label as to whether the molecule binds at t.

In our generalization, there are multiple target points ("subtargets"), each of which must bind to some instance in a bag for the bag to be positive. I.e. bag B_i's label is positive iff each subtarget induces a normalized binding energy of at least $1/2$ in some point $B_{ij} \in B_i$. That is, one thresholds

$$\min_{t \in T} \left\{ \max_{B_{ij} \in B_i} \left\{ \frac{\sum_{k=1}^{n} s_k V\left(t_k - B_{ijk}\right)}{E_{max}} \right\} \right\} .$$

We used the modified data generator to build ten 5-dimensional data sets with 4 sub-targets[4]. For each data set, we first randomly generated 4 sub-targets and then used the target to label a training set and test set, each of size 200. One bag was for training all algorithms, one bag was for testing, and the third was used by EMDD as a validation set for selecting its final hypothesis (so EMDD had a *de facto* training set of 400 bags). Each data set consisted consisted of approximately 47% positive bags.

[4]Each sub-target corresponds to a site that a molecule must bind to in order to be considered a positive bag.

Content-Based Image Retrieval

In content-based image retrieval (CBIR), the user presents examples of desired images, and the task is to determine commonalities among the query images. [32] [32] explored the use of conventional MIL for CBIR. They filtered and subsampled the images and then extracted "blobs" (groups of m adjacent pixels), which were mapped to one point in a bag. Then they used the MIL algorithm diverse density (DD) [31] to learn a hypothesis and find candidate images in the database. This work was extended by Zhang et al. [53], who observed that in their experiments, it was likely that not one but several target points (in a disjunctive form) were responsible for labeling the examples. In addition, it is arguable that if the target concept were conjunctive (e.g. the desired images contain a field *and* a sky), then the standard MIL model will not work and a more expressive hypothesis is needed. This is especially true if a criterion for being positive included the *absence* of certain features, e.g. the images contain a field and contain *no* sky. Such a notion cannot be represented by DD or EMDD.

We experimented with two CBIR tasks[5]. One is Zhang et al.'s "sunset" task: to distinguish images containing sunsets from those not containing sunsets. Like Zhang et al., we built 30 random testing sets of 720 examples (120 positives and 600 negatives): 150 negatives each from the waterfall, mountain, field, and flower sets. Each of 30 training sets consisted of 50 positives and 50 negatives.

Another task is to test a conjunctive CBIR concept, where the goal is to distinguish images containing a field with no sky from those containing a field and sky or containing no field. We relabeled Zhang et al.'s field images from positive to negative those that contained the sky. Each training set had 6 bags of each of flower, mountain, sunset, and waterfall for negatives, and had around 30 fields, 6 of them negative and the rest positive. Each negative test set had 150 bags of each of flower, mountain, sunset, and waterfall. Also, each test set

[5]Based on data from [49], the Corel Image Suite, and www.webshots.com.

had 120 fields, around 50 serving as positives and the remainder as negatives.

Identifying Trx-fold Proteins with Low Primary Sequence Conservation

The low conservation of primary sequence in protein superfamilies such as Thioredoxin-fold (Trx-fold) makes conventional methods difficult to use. We propose using multiple-instance learning as a tool for identification of new Trx-fold proteins as well as other proteins belonging to superfamilies with low primary sequence conservation [47]. We mapped a protein's primary sequence to a bag in the following way. First, we found in each positive and negative sequence the primary sequence motif (typically CxxC) that is known to exist in all Trx-fold proteins. We then extracted a window of size 214 around it (30 residues upstream, 180 downstream) and aligned these windows around the motif. We then mapped all sequences to 8-dimensional profiles based on the numeric properties of Kim et al. [24] and used them as inputs to the multiple-instance learning algorithm.

GMIL-1 is not suitable for the protein data set. As an example why this is a problem, consider the case where 5 clusters is used to built the grid on the 9-dimensional space (8-dimensional profiles and an additional dimension for identifying the positions of amid acids in protein sequences). Then the number of attributes used by GMIL-1 is $2 \cdot (5(5+1))^9 \approx 3.94 \times 10^{13}$. So it is not practical to use GMIL-1. Instead we can apply GMIL-2 to this data set. We used GMIL-2 to perform cross-validation tests: 20-fold CV on the 20 positives and 8-fold CV on the 160 negatives. So in each round, we trained GMIL-2 on 19 positive proteins plus one of 8 sets of negative proteins, and tested on the held-out positive protein plus the remaining 7 sets of negative proteins. We repeated this for each of the 8 sets of negative proteins.

3.6.2 Comparison of Prediction Error

For both GMIL-1 and GMIL-2, we k-means clustered all training sets and used the clusters' point representatives when building group sets (see Footnote 2). The drug discovery data was clustered into $m = 5$ clusters and $m = 6$ clusters for CBIR. For building groups for GMIL-2, we added to Ψ a randomly-chosen size-K subset of $\Omega_{\mathcal{G}}$, where K was varied to measure the effect of reduced resolution on generalization error. In all experiments, we trained both algorithms until they achieved zero training error or until they trained for 100 rounds.

In Figure 3.4, all plots show the same trend. When the number of additional points K is small, GMIL-2 has very high training error and testing error. But by increasing K, we see a dramatic decrease in the false negative error rates[6]. Once K exceeds some value (which depends on the data set), we see that GMIL-2's performance for both FP and FN error rates is statistically indistinguishable from those of GMIL-1. I.e. GMIL-2 learned a hypothesis that is statistically the same as GMIL-1's. In addition, the value of K required to do this is quite small (10–60), and much less than $|\Omega_{\mathcal{G}}|$ (as discussed earlier, $K = |\Omega_{\mathcal{G}}|$ would make the two algorithms exactly equivalent). For example, for the drug discovery data GMIL-2's prediction error was indistinguishable from GMIL-1's using $|\Psi| = 5 + K = 65$ representatives while GMIL-1 used 16807 grid points. Thus GMIL-2 can run on a relatively small number of representatives without a significant increase in prediction error[7].

3.6.3 Comparison of Computation Costs

The computation time of both GMIL-1 and GMIL-2 depends on the number of groups and how fast they converge. Their memory requirements also depend on the number of groups.

[6]For small K, false positive error is 0 because the initial weights in Winnow cause it to always predict negative with using low-resolution inputs.

[7]We also compared GMIL-1 and GMIL-2 on robot vision data, where all training bags (with no clustering) were used to build GMIL-1's grid on 185344 points and GMIL-2 used $|\Psi| = 300$ points. Our results were similar in that GMIL-2's FP and FN rates were statistically indistinguishable from those of GMIL-1.

In Table 3.1(a), we compare the number of groups built by GMIL-1 to the number built by GMIL-2 with $K = 100$. It is obvious that GMIL-2 built much smaller group sets than GMIL-1, which means GMIL-2 used much less memory and ran much faster than GMIL-1. We also compared the average number of trials required for each algorithm to converge to zero error on the training set. From Table 3.1(b), we see that GMIL-2 converged in about the same number or even fewer trials than GMIL-1. So generally speaking, they have the same rate of convergence. In all these experiments, GMIL-1 required 500-700 Mb of memory and ran for around 30 hours to train on a single data set. In contrast, GMIL-2 used less than 50 Mb of memory and finished training in less than one hour on average.

Table 3.1: Comparison of the computation costs of GMIL-1 and GMIL-2.

(a) The average number of groups generated.

Data Sets	GMIL-1	GMIL-2	Savings (%)
Drug discovery	4.08×10^6	6.86×10^4	98.3
CBIR: sunset	3.89×10^6	6.19×10^4	98.4
CBIR: conjunctive	5.75×10^6	8.46×10^4	98.5

(b) The average number of trials in which the algorithms converged.

Data Sets	GMIL-1	GMIL-2
Drug discovery	26.8	34.0
CBIR: sunset	41.7	29.0
CBIR: conjunctive	21.4	16.8

3.6.4 Comparison to DD and EMDD

We now compare generalization error of GMIL-1 and GMIL-2 (with $K = 100$) to those of EMDD and DD (that work in the conventional MIL model). EMDD and DD were run on the original, unclustered data, so they have full resolution, in contrast to the GMIL algorithms.

Results are presented in Table 3.2. For drug discovery, the FP errors of the two GMIL algorithms are less than DD and EMDD, but their FN errors are higher. We suspect that

some of the FN error comes from the data's low resolution due to clustering in forming the grid.

In the sunset task of CBIR, all four algorithms have similar FP and FN error; none of them shows a significant advantage over the others. However, for the conjunctive task, GMIL-1 and GMIL-2 achieved the lowest FP and FN errors. This occurred despite the potential loss of information due to clustering. Thus we see that conjunctive CBIR concepts benefit from generalized MIL. Also we notice that GMIL-2 has a little bit higher prediction error but again shows almost same generalization ability as GMIL-1 when compared to DD and EMDD.

For the protein data set, our results show that GMIL-2 performed well, with relatively high true positive and true negative rates. GMIL-2 can identify 75% of the Trx-fold proteins in this test with a 75% true negative rate, which are both better than EMDD. Although DD have around 87% true positive rate, their true negative rates are less that 44%, which is much lower than GMIL-2.

Table 3.2: Generalization error of GMIL-1, GMIL-2, DD, and EMDD.

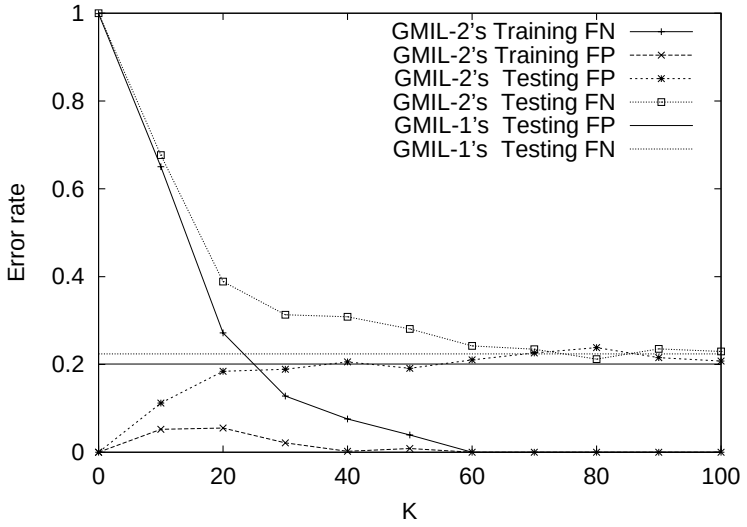
FP error.				
Data Sets	GMIL-2	GMIL-1	DD	EMDD
Drug discovery	0.208	0.201	0.274	0.263
CBIR: sunset	0.082	0.080	0.078	0.082
CBIR: conjunctive	0.140	0.128	0.173	0.213
Protein	0.250	N/A	0.668	0.365
FN error.				
Data Sets	GMIL-2	GMIL-1	DD	EMDD
Drug discovery	0.230	0.224	0.108	0.109
CBIR: sunset	0.160	0.157	0.168	0.166
CBIR: conjunctive	0.244	0.219	0.282	0.244
Protein	0.250	N/A	0.125	0.360

3.7 Conclusions

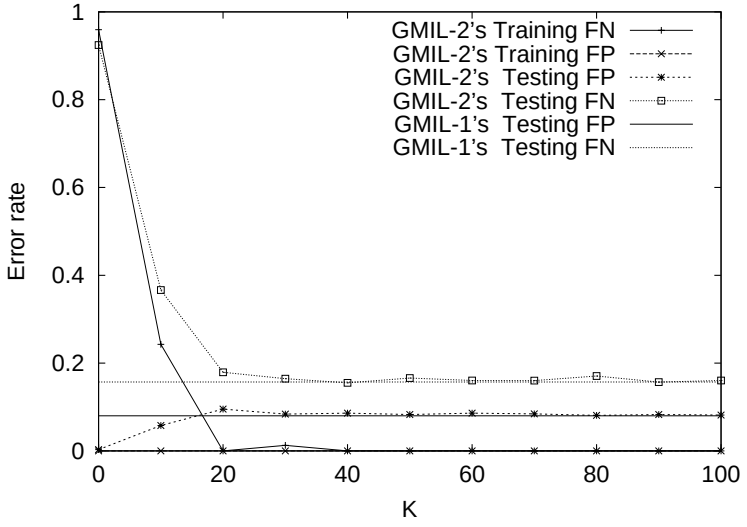
While the standard MIL model is powerful, there exist applications with natural target concepts that cannot be represented. This is what motivated the prior introduction of the GMIL model, along with the algorithm GMIL-1 to learn geometric concepts in it. Here we presented GMIL-2, a much faster algorithm than GMIL-1. GMIL-2 uses a compact group set based on a set of representative points. Our experimental results show that our strategy for selecting representatives not only generates a relatively small Ψ, but also has almost same generalization ability as GMIL-1 that uses N clusters to separate the space. An interesting fact we found is that in some applications, we do not even need a training set to generate $\mathcal{G}$.It seems a grid has more general meaning than clusters. One benefit is that we can find a "good" grid for an application and build and store Ψ. Then we can use this Ψ for any training set [8]. We can also train our algorithm on-line, which should be useful for applications with dynamically maintained databases.

Our experimental results show that it has the same generalization ability as GMIL-1 and needs much less time and memory to train. Thus we was able to apply our new algorithm to protein sequence data where GMIL-1 cannot be used in practice. Besides Trx-protein sequence data, GMIL-2 have also been applied to other protein data sets such as GPCR and achieved promising results [51, 25]. In next chapter, we will also look at applications with relatively high dimension, e.g. $d > 100$. We will explore kernel techniques (for use in a support vector machine) to quickly compute the weighted sum of the inputs, entirely eliminating the need to enumerate the groups and yielding a more scalable algorithm.

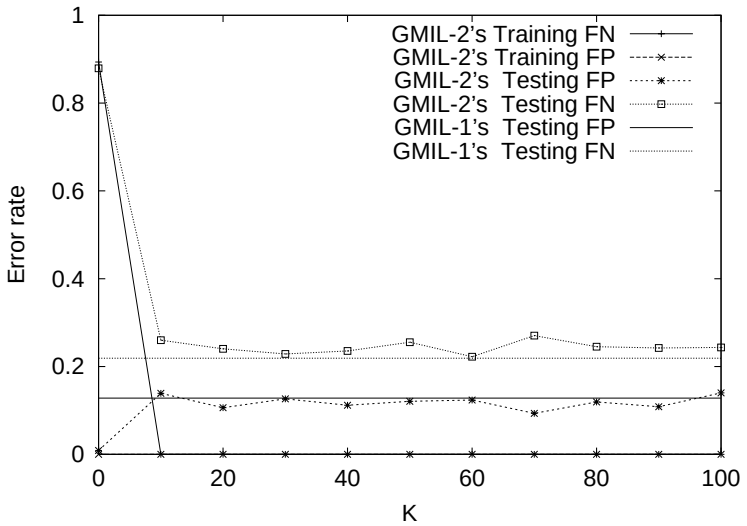
[8] For example, there are only 20 distinct amid acids in protein sequences. So we can use the 20 profiles as representatives to build a group set, which does not depend on any training set.



(a) Drug discovery.



(b) CBIR: sunset.



(c) CBIR: conjunctive.

Figure 3.4: Comparison of prediction error rates of GMIL-1 and GMIL-2 while varying K (the number of additional points used by GMIL-2).

Chapter 4

A Kernel for Generalized Multiple-Instance Learning

4.1 Introduction

Both GMIL-1 and GMIL-2 have exponential time complexity and are thus limited to small numbers of points n or small numbers of dimensions d. In seeking out an algorithm that scales polynomially in both n and d, we give a kernel that can be used with a support vector machine [46, 12] to efficiently learn geometric multiple-instance concepts. We show that this kernel exactly corresponds to the feature mapping used by GMIL-1. To compute the kernel, one takes two bags of points P and Q and counts the number of boxes defined on $\{0, \ldots, s\}^d$ that contain at least one point from P and at least one point from Q. We first show that this problem is #P-complete, present a fully polynomial randomized approximation scheme (FPRAS) for it, and then empirically evaluate our kernel.

The rest of this chapter is organized as follows. In the next section we introduce some notation. Then in Section 4.3 we present our kernel-based reformulation of GMIL-1. We show that computing this kernel is equivalent to counting the number of boxes that contain

at least one point from both sets P and Q, a problem that we formally define in Section 4.4 as #BOXAnd. We prove that this problem is #P-complete and give an FPRAS for it. In Section 4.5 we describe experimental results of our new kernel on applications such as content-based image retrieval, prediction of drug affinity to bind to multiple sites simultaneously, protein sequence identification, and the Musk data sets. Finally, we conclude in Section 4.6.

4.2 Notation and Definitions

Let $\mathcal{X}$ denote $\{0, \ldots, s\}^d$ (though our results trivially generalize to $\mathcal{X} = \prod_{i=1}^{d} \{0, \ldots, s_i\}$). Let $B_{\mathcal{X}}$ denote the set of all axis-parallel boxes (including degenerate boxes) from $\mathcal{X}$. We uniquely identify any box $b \in B_{\mathcal{X}}$ as a pair (b_ℓ, b_u), where b_ℓ is the "lower left" corner and b_u is the "upper right" corner. There are $s+1$ possible values for each corner, so the number of intervals in each dimension is $\binom{s+1}{2} + s + 1$ since we allow degenerate intervals. Thus $|B_{\mathcal{X}}| = \left(\binom{s+1}{2} + s + 1\right)^d = \binom{s+2}{2}^d$.

For multisets $P, Q \subseteq \mathcal{X}$, let $B(P)$ denote the set of boxes in $B_{\mathcal{X}}$ that contain a point from P and $B(P \wedge Q)$ denote the set of boxes in $B_{\mathcal{X}}$ that contain a point from P and a point from Q. When P and Q contain single points then we will omit set notation. For example $B(\{p\} \wedge \{q\})$ will be denoted as $B(p \wedge q)$.

We will use vector notation to refer to points in $\mathcal{X}$ only when it is necessary (e.g. in Section 4.4.1); otherwise we will just use lower case letters to refer to points in $\mathcal{X}$. The notion of approximation that we use is defined as follows.

Definition 4.1 *Let f be a counting problem. Then a randomized algorithm $\mathcal{A}$ is an* FPRAS (Fully Polynomial Randomized Approximation Scheme) *if for any instance x, and parameters $\epsilon, \delta > 0$,*

$$\Pr\left[|\mathcal{A}(x) - f(x)| \leq \epsilon f(x)\right] \geq 1 - \delta$$

and A's running time is polynomial in $|x|$, $1/\epsilon$, and $1/\delta$. Further, we call $\mathcal{A}(x)$ an ϵ-good approximation *of $f(x)$.*

The "kernel trick," as it is often called, defines a kernel $k(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x}) \cdot \phi(\mathbf{y})$ to compute the dot product between two vectors $\mathbf{x}$ and $\mathbf{y}$ after they are mapped to a new space by some function $\phi(\cdot)$. To gain intuition on why kernels are useful in training, consider the operation of the Perceptron algorithm, which learns a linear threshold unit like Winnow. If we let $\mathbf{w}_k \in \Re^N$ be the vector of weights maintained by Perceptron at update k, $\mathbf{x}_i \in \Re^N$ be the ith training example, and $y_i \in \{-1, +1\}$ be example $\mathbf{x}_i$'s label, then Perceptron's prediction on the label of $\mathbf{x}_i$ is $\hat{y}_{ik} = \mathbf{w}_k \cdot \mathbf{x}_i$ (the sign of $\hat{y}_{ik}$ the prediction and $|\hat{y}_{ik}|$ is its confidence in that prediction). If $\hat{y}_{ik} y_i < 0$, then a prediction mistake was made and the weight vector is updated as follows: $\mathbf{w}_{k+1} = \mathbf{w}_k + \eta\, y_i\, \mathbf{x}_i$, where η is a learning rate. The Perceptron algorithm also has a *dual* form, which is rooted in the observation that if $\mathbf{w}_0 = \mathbf{0}$, then after update k,

$$\hat{y}_i = \mathbf{w}_k \cdot \mathbf{x}_i = \eta \sum_{j=1}^{m} (\alpha_j\, y_j\, \mathbf{x}_j) \cdot \mathbf{x}_i = \eta \sum_{j=1}^{m} \alpha_j\, y_j\, (\mathbf{x}_j \cdot \mathbf{x}_i) \ ,$$

where α_j is the number of prediction mistakes that Perceptron has made so far on example $\mathbf{x}_j$, i.e. the number of weight updates made in response to mistakes on $\mathbf{x}_j$. Now since no weight vector is explicitly maintained, it is easy to see how one can add a mapping ϕ:

$$\hat{y}_i = \sum_{j=1}^{m} \alpha_j\, y_j\, \left(\phi\left(\mathbf{x}_j\right) \cdot \phi\left(\mathbf{x}_i\right)\right) \ , \tag{4.1}$$

where $\phi\left(\mathbf{x}_i\right) \cdot \phi\left(\mathbf{x}_j\right)$ comes from evaluating the kernel.

4.3 Kernel-Based Reformulation of GMIL-1

In seeking out an algorithm that scales polynomially in both n and d, we define a kernel that can be used with a support vector machine [46, 12, 38] (that simulates Perceptron above) to efficiently learn geometric multiple-instance concepts. We will show that computing such a kernel on two bags P and Q corresponds to counting the number of boxes that contain at least one point from each of P and Q. After we show that this problem is #P-complete, we develop an FPRAS for it.

First, We recall the mapping of GMIL-1. GMIL-1 enumerates the set $B_{\mathcal{X}}$ of all possible boxes (including degenerate ones) in $\mathcal{X}$ (so $|B_{\mathcal{X}}| = \binom{s+2}{2}^d$) and creates an attribute a_b for each box $b \in B_{\mathcal{X}}$. Given a bag $P \in \mathcal{X}^n$, the algorithm sets $a_b = 1$ if some point from P lies in b and $a_b = 0$ otherwise. To capture the notion of repulsion points, they also defined complementary attributes $\bar{a}_b = 1 - a_b$. This was done because Winnow in its standard form cannot represent negative weights. In our kernel formulation, we only use the N attributes a_b since Perceptron can represent negative weights. Then we have the following observation.

Observation 4.2 *Consider two bags $P, Q \subseteq \mathcal{X}$ and a mapping $\phi_\wedge(P) = (a_1, \ldots, a_N)$ where $a_i = 1$ if the corresponding box $b_i \in B_{\mathcal{X}}$ contains a point from P and 0 otherwise. Then when using an SVM for learning, the remapping used by GMIL-1 corresponds to using the kernel*

$$k_\wedge(P, Q) = \phi_\wedge(P) \cdot \phi_\wedge(Q) = |B(P \wedge Q)| \ ,$$

where $B(P \wedge Q)$ is the set of boxes that contain a point from P and contain a point from Q.

Proof: Since $\phi_\wedge(P)$ and $\phi_\wedge(Q)$ are binary vectors, their dot product is simply the number of 1s in corresponding positions. Since a bit from $\phi_\wedge(P)$ is 1 if and only if the correspond-

ing box contains a point from P, the value of $k_\wedge(P,Q)$ is obviously $|B(P \wedge Q)|$. Finally, $k_\wedge(P,Q)$ is a kernel by definition. □

4.4 The Box Counting Problem #BOXAnd

From Observation 4.2, it follows that the kernel $k_\wedge$ for GMIL-1 corresponds to the box counting problem that we call #BOXAnd, which we now define. The input to the problem is a triple $\langle \mathcal{X}, P, Q \rangle$. The problem #BOXAnd is to compute $|B(P \wedge Q)|$: the number of boxes in $B_\mathcal{X}$ that contain at least one point from each of P and Q. In this section we prove that #BOXAnd is #P-complete, and then we present an FPRAS for it.

4.4.1 Hardness Result for #BOXAnd

It is easy to see that #BOXAnd is in #P: given a particular box $b \in B_\mathcal{X}$, it is simple to verify that b contains a point from both P and Q. We now prove #P-completeness by reducing from the *monotone DNF counting problem* (#MDNF), shown to be #P-complete by Valiant [45]. An instance of #MDNF is a monotone boolean formula F (i.e. with no negated literals) in disjunctive normal form, and an algorithm for this problem is to output the number of satisfying assignments of F.

Let F be a monotone DNF formula in n variables with m monotone terms $t_1, t_2, \ldots, t_m$. Let $S(F)$ denote the set of all satisfying assignments of F. Then $S(F) = \bigcup_i S(t_i)$. Each monotone term t can be identified with an n-bit binary vector $\mathbf{v_t}$ as follows: $\mathbf{v_t} = v_1 v_2 \ldots v_n$ where $v_i = 1$ if $x_i \in t$ and $v_i = 0$ if $x_i \notin t$. Then, since t is monotone, the set of satisfying assignments for t, $S(t) = \{\mathbf{a} \mid \mathbf{a} \geq \mathbf{v_t}\}$. (For two n-bit vectors $\mathbf{u} = (u_1, u_2, \ldots, u_n)$ and $\mathbf{v} = (v_1, v_2, \ldots, v_n)$, $\mathbf{u} \geq \mathbf{v}$ iff $u_i \geq v_i$ for all $1 \leq i \leq n$.)

Theorem 4.3 #BOXAnd is #P-complete.

Proof: We have already established that #BOXAnd is in #P. We now show that #BOXAnd is #P-hard by reducing #MDNF to a special case of #BOXAnd where $\mathcal{X} = H_n = \{0,1\}^n$. The reduction f takes a formula $F = \bigvee_{1 \le i \le m} t_i$ and maps it to an instance $f(F) = \langle H_n, P, Q \rangle$ where $P = \{\mathbf{0}\}$ and $Q = \{\mathbf{v}_{t_1}, \mathbf{v}_{t_2}, \ldots, \mathbf{v}_{t_m}\}$.

We now argue that $|S(F)|$ equals the number of solutions to $\langle H_n, P, Q \rangle$ of #BOXAnd. Clearly, $B(\mathbf{0} \wedge \mathbf{v}) = \{(\mathbf{0}, \mathbf{u}) \mid \mathbf{u} \ge \mathbf{v}\}$. For any term t_i, $\mathbf{a} \in S(t_i) \Leftrightarrow \mathbf{a} \ge \mathbf{v}_{t_i} \Leftrightarrow (\mathbf{0}, \mathbf{a}) \in B(\mathbf{0} \wedge \mathbf{v}_{t_i})$. Thus the number of satisfying assignments of F = $|\bigcup_{1 \le i \le m} S(t_i)| = |\bigcup_{1 \le i \le m} B(\mathbf{0} \wedge \mathbf{v}_{\mathbf{t_i}})| = |B(\{\mathbf{0}\} \wedge Q)|$ = the number of solutions to $\langle H_n, P, Q \rangle$. □

4.4.2 An FPRAS for #BOXAnd

Our algorithm for estimating $|B(P \wedge Q)|$ is based on the general technique from Karp et al. [22] on the union of sets problem. In this problem, the goal is to take a description of m sets $B_1, \ldots, B_m$ and estimate the size of $B = \bigcup_{i=1}^{m} B_i$. In order to apply their technique, three criteria must be satisfied. First, for all $i \in \{1, \ldots, m\}$, $|B_i|$ must be easily computed. Second, for all $i \in \{1, \ldots, m\}$, we must be able to sample uniformly elements from B_i. Finally, given any $s \in B$ and any $i \in \{1, \ldots, m\}$, we must be able to easily determine if $s \in B_i$.

If the above criteria are satisfied, Karp et al.'s algorithm proceeds as follows. First define $U = \{(s, i) \mid s \in B_i \text{ and } 1 \le i \le m\}$ (so $|U| = \sum_{i=1}^{m} |B_i|$). Define another set $G = \{(s, i) \mid i \text{ is the smallest index such that } s \in B_i\}$. Then we have $G \subseteq U$ and $|G| = |B|$. Karp et al.'s algorithm runs in trials. For each trial, first a set B_i is chosen at random with probability $|B_i|/|U|$. Then an element $s \in B_i$ is chosen uniformly at random. These two steps together uniformly sample a pair (s, i) from U. Finally, if $(s, i) \in G$ we increment a counter γ, otherwise do nothing. Our final estimate of $|B|$ is $|U|\gamma/S$, where S is the number of samples drawn. The following theorem bounds the error of this approximation.

Theorem 4.4 *[22] If $S \geq 4(|U|/|G|)\ln(2/\delta)/\epsilon^2$, then*

$$\Pr\left[(1-\epsilon)|B| \leq |U|\gamma/S \leq (1+\epsilon)|B|\right] \geq 1-\delta.$$

We now apply Karp et al.'s result to #BOXAnd. Recall that for two points $p, q \in \mathcal{X}$, $B(p \wedge q)$ denotes the set of boxes that contain both p and q. Let $W = |B(P \wedge Q)|$. Then $W = |\bigcup_{p\in P, q\in Q} B(p \wedge q)|$. It is straightforward to compute $|B(p \wedge q)|$. Given points $p, q \in \mathcal{X}$, let $\ell = (\ell_1, \ldots, \ell_d)$ be the lower corner of the bounding box of p and q, i.e. $\ell_i = \min\{p_i, q_i\}$ for all i. Similarly define $u = (u_1, \ldots, u_d)$ as the upper corner. Then $|B(p \wedge q)| = \left(\prod_{1\leq i\leq d}(\ell_i + 1)\right)\left(\prod_{1\leq i\leq d}(s - u_i + 1)\right)$. Since we can exactly compute $|B(p \wedge q)|$ for all $(p, q) \in P \times Q$ and there are only n^2 such sets, we can easily choose a set $B(p \wedge q)$ with probability $|B(p \wedge q)|/\left(\sum_{p\in P, q\in Q} |B(p \wedge q)|\right)$. Further, since we can uniformly sample from $B(p \wedge q)$ by uniformly selecting lower and upper corners, we can uniformly sample from the set $U = \{(p, q, c) \mid p \in P, q \in Q, c \in B(p \wedge q)\}$.

Note that $|U| = \sum_{p\in P, q\in Q} |B(p \wedge q)|$. Now consider all the pairs (p, q) such that $p \in P$ and $q \in Q$. We define a total order $\prec$ on these pairs by sorting first by p's index in P, and then by q's index in Q. I.e. given points $p_i, p_{i'} \in P$ and $q_j, q_{j'} \in Q$, we define $(p_i, q_j) \prec (p_{i'}, q_{j'})$ iff $i < i'$ or $i = i'$ and $j < j'$.

Consider another set $G = \{(p, q, c) \in U \mid$ there are no pairs $(p', q') \prec (p, q)$ such that $c \in B(p' \wedge q')\}$. Then $|G| = |\bigcup_{p\in P, q\in Q} B(p \wedge q)| = W$. We check whether $(p, q, c) \in G$ in $O(dn^2)$ time by checking c against each set $B(p \wedge q)$ for all $p \in P$ and $q \in Q$. Finally, we note that

$$|U| = \sum_{p\in P, q\in Q} |B(p \wedge q)| \leq n^2 \max_{p,q} |B(p \wedge q)| \leq n^2|G| \ . \tag{4.2}$$

Thus by drawing a sufficient number of samples (p, q, c) uniformly from U and incrementing γ when $(p, q, c) \in G$, we know that $\hat{W} = |U|\gamma/S$ is an ϵ-good approximation of

W, as stated in the following theorem. Since S, the time to draw each sample, and the time to check each sample for membership in G are all polynomial in n, d, $\log s$, $1/\epsilon$, and $1/\delta$, our algorithm for #BOXAnd is a FPRAS.

Theorem 4.5 *If* $S \geq 4n^2 \ln(2/\delta)/\epsilon^2$, *then*

$$\Pr\left[(1-\epsilon)W \leq \hat{W} = |U|\gamma/S \leq (1+\epsilon)W\right] \geq 1-\delta.$$

Proof: Directly from application of Equation (4.2) to Theorem 4.4. □

Our algorithm as presented has running time $O(n^4 d(\log s)\ln(1/\delta)/\epsilon^2)$ since it takes $O(dn^2)$ steps to check each sample for membership in G. However, it is possible to check for membership in G in time $O(dn)$. Given a triple (p_i, q_j, c) sampled from U, first check all points $p_{i'} \in P$ that are contained in c. If $i' < i$ for some $p_{i'} \in c$, then $(p_{i'}, q_j) \prec (p_i, q_j)$ and $(p_i, q_j, c) \notin G$. If there does not exist such a $p_{i'}$, then check all points $q_{j'} \in Q$ that are contained in c. Again, if $j' < j$ for some $q_{j'} \in c$, then $(p_i, q_{j'}) \prec (p_i, q_j)$ and $(p_i, q_j, c) \notin G$. If no such $q_{j'}$ exists, then $(p_i, q_j, c) \in G$. This check requires time $O(dn)$, reducing the total running time to $O(n^3 d(\log s)\ln(1/\delta)/\epsilon^2)$.

To further reduce time complexity, we can adapt Karp et al.'s "self-adjusting coverage algorithm" as shown in Table 4.1, which is a more efficient algorithm for the union of sets problem. The following theorem bounds the error of this algorithm.

Theorem 4.6 *[22] If* $T \geq 8(1+\epsilon)m\ln(2/\delta)/\epsilon^2$, *then*

$$\Pr\left[(1-\epsilon)|B| \leq \hat{Y} \leq (1+\epsilon)|B|\right] \geq 1-\delta.$$

After making changes on Step 5, 11 and 12 in Table 4.1, we can get a self-adjusting coverage algorithm for #BOXAndas shown in Table 4.2. Since m, the number of all possible sets $B(p \wedge q)$, is less or equal to n^2, we have the following theorem directly from

Table 4.1: Self-adjusting coverage algorithm [22]

1: Given T.
2: $gtime \leftarrow 0$
3: $N_T \leftarrow 0$
4: **loop**
5: randomly choose $(s, i) \in U$ with probability $1/|U|$ as before
6: **repeat**
7: $gtime \leftarrow gtime + 1$
8: **if** $gtime > T$ **then**
9: **go to FINISH**
10: **end if**
11: randomly choose $j \in \{1, \cdots, m\}$ with probability $1/m$
12: **until** $s \in B_j$
13: $N_T \leftarrow N_T + 1$
14: **end loop**
15: **FINISH:** $\hat{Y} \leftarrow T \cdot |U|/(m \cdot N_T)$

Theorem 4.6.

Theorem 4.7 *If* $T \geq 8(1+\epsilon)n^2 \ln(2/\delta)/\epsilon^2$*, then*

$$\Pr\left[(1-\epsilon)|B| \leq \hat{Y} \leq (1+\epsilon)|B|\right] \geq 1-\delta.$$

Our self-adjusting coverage algorithm only needs to check a box for membership in $B(p \wedge q)$ instead of G like in the previous algorithm (see Step 13 in Table 4.2), which can be done in time $O(d)$. Then we get a FPRAS for #BOXAnd with running time $O(n^2 d(\log s)\ln(1/\delta)/\epsilon^2)$.

4.4.3 Discussion

According to Observation 1, $k_\wedge(P, Q)$ is a kernel since it is the dot product of two remapped vectors. But there is no guarantee that the Gram matrix computed by our approximation algorithm is positive semidefinite. However, it is reasonable to believe that if ϵ is small

Table 4.2: Self-adjusting coverage algorithm for #BOXAnd

Given T and bag P and Q
$gtime \leftarrow 0$
$N_T \leftarrow 0$
loop
  randomly choose $(p, q, c) \in U$ with probability $1/|U|$
  repeat
    $gtime \leftarrow gtime + 1$
    if $gtime > T$ **then**
      go to FINISH
    end if
    randomly choose $p' \in P$ with probability $1/|P|$
    randomly choose $q' \in Q$ with probability $1/|Q|$
  until $c \in B(p' \wedge q')$
  $N_T \leftarrow N_T + 1$
end loop
FINISH: $\hat{Y} \leftarrow T \cdot |U|/(m \cdot N_T)$

---

and the original Gram matrix has no zero eigenvalues, the approximated matrix would not adversely affect the SVM optimization engine. In fact, as shown in our experiments, our approximate kernel works very well when we set $\epsilon = 0.1$.

Another observation about our kernel is that its Gram matrix potentially can have large diagonal elements relative to the off-diagonal elements. $k_\wedge(P, Q)$ is the number of boxes that contain a point from P and a point from Q. If few points from P and Q are close to each other, $k_\wedge(P, Q)$ will be much smaller than $k_\wedge(P, P)$ and $k_\wedge(Q, Q)$. This worsens when d is large. For example, in our Musk experiments, the ratio of diagonal entries in the kernel matrix to the off-diagonal entries was often around 10^{50}. In practice, SVMs do not work well with diagonally dominated Gram matrices. To solve this problem, Schölkopf et al. [39] propose first using a nonlinear function to reduce the value of each matrix element, such as a sub-polynomial function $\varphi(x) = \text{sign}(x) \cdot |x|^\rho$ with

$0 < \rho < 1$. To then get a positive definite Gram matrix, they then use the empirical kernel map $\phi_n(x) = (k'(x, x_1), k'(x, x_2), \cdots, k'(x, x_n))$, where $k'(x, x_i) = \varphi(k(x, x_i))$. Finally they apply the kernel $k_{emp}(x, y) = \phi_n(x) \cdot \phi_n(y)$. In the empirical kernel, the set $\{x_1, \cdots, x_n\}$ can consist of all training and testing bags (referred to as *transduction*) or of only the training bags. We applied this method with $k_\wedge$ to address our diagonal dominance problem (see Section 4.5.4). Also, since k_{emp} is always a kernel no matter what k is, with this method we do not need to worry about whether our ϵ-approximation of $k_\wedge$ is really a kernel.

4.5 Experimental Results

To compare our kernel to the algorithm GMIL-1 [40] and GMIL-2 [43], we tested our kernel with SVM^{light} [21] on the data sets we used in Chapter 3: content-based image retrieval (real data) and predicting when drugs would bind at multiple sites of a molecule (simulated data), and the protein data that are also used by Wang et al. [47]. All these data sets have dimension at most 8, since GMIL-1 and GMIL-2 cannot scale well to higher dimensions. To evaluate our algorithm on high-dimensional data, we also tested on simulated multi-site binding data with higher dimension and the Musk data sets from the UCI repository [4] that have 168 dimensions. In all our tests, we computed our kernel using our self-adjusting approximation algorithm with $\epsilon = 0.1$ and $\delta = 0.01$.

4.5.1 Content-Based Image Retrieval

In content-based image retrieval (CBIR), the user presents examples of desired images, and the task is to determine commonalities among the query images and retrieve similar ones from the database. Maron and Ratan [32] explored the use of conventional MIL for CBIR. They filtered and subsampled the images and then extracted "blobs" (groups of m adjacent

pixels), which were mapped to one point in a bag. Then they used the algorithm diverse density (DD) [31] to learn a hypothesis and find candidate images in the database. This work was later extended by Zhang et al. [53].

We experimented with the two CBIR tasks used in Chapter 3. One is the "sunset" task: to distinguish images containing sunsets from those not containing sunsets. We built 30 random testing sets of 720 examples (120 positives and 600 negatives): 150 negatives each from the waterfall, mountain, field, and flower sets. Each of 30 training sets consisted of 50 positives and 50 negatives.

Another CBIR task is to test a conjunctive CBIR concept, where the goal was to distinguish images containing a field with no sky from those containing a field and sky or containing no field. Each training set had 6 bags of each of flower, mountain, sunset, and waterfall for negatives, and had around 30 fields, 6 of them negative and the rest positive. Each negative test set had 150 bags of each of flower, mountain, sunset, and waterfall. Also, each test set had 120 fields, around 50 serving as positives and the remainder as negatives.

The top two rows of Table 4.3 summarize the prediction error of our algorithm ("$k_\wedge$"), GMIL-1, and GMIL-2. For comparison purposes, we also give results for the algorithms Diverse Density [31] and EMDD [52] that operate in the conventional MIL model. The sunset task fits well into the conventional MIL model; hence there is little difference in performance between any of the algorithms on this data set. But since the conjunctive task requires identifying images that have a field and have no sky, we see that the generalized model is required[1]. We also note that the use of an SVM improved prediction accuracy when compared to all algorithms, including GMIL-1 and GMIL-2, which use the same hypothesis space as our algorithm.

[1] [40] showed that repulsion points are required for this learning task.

Table 4.3: Generalization error for CBIR (top), protein (middle), and drug affinity (bottom) learning tasks.

Task	$k_\wedge$	GMIL-1	GMIL-2	EMDD	DD
sunset	0.088	0.095	0.098	0.096	0.099
conj.	0.108	0.134	0.147	0.215	0.181
protein	0.213	N/A	0.251	0.338	0.608
5-dim	0.205	0.212	0.218	0.191	0.196
10-dim	0.175	N/A	N/A	0.223	0.216
20-dim	0.207	N/A	N/A	0.268	0.255

4.5.2 Identifying Trx-fold Proteins

The low conservation of primary sequence in protein superfamilies such as Thioredoxin-fold (Trx-fold) makes conventional methods difficult to use. We have proposed using multiple-instance learning as a tool for identification of new Trx-fold proteins as well as other proteins belonging to superfamilies with low primary sequence conservation [47]. Here, we applied our kernel and performed cross-validation tests: 20-fold CV on the 20 positives and 8-fold CV on the 160 negatives. As shown in Table 4.3, our kernel has the lowest prediction error compared to all algorithms including GMIL-2, DD and EMDD.

4.5.3 Multi-Site Drug Binding Affinity

Dietterich et al. [14] introduced the conventional MIL model motivated by predicting whether a conformation of a particular molecule would bind to a single site in another molecule. They also described an open problem of how to predict drugs that bind at multiple sites in a single molecule by fitting in several of them simultaneously.

We used Dooly et al.'s modified data generator to build ten 5-dimensional data sets (200 training bags and 200 testing bags), each with 4 subtargets. To test how well our kernel handles higher-dimensional data, we also generated data with dimension 10 and 20, each with 5 subtargets. As with the 5-dimensional data, ten sets were generated, each with 200

training bags and 200 testing bags. Results are at the bottom of Table 4.3.

4.5.4 Musk Data Sets

To evaluate our algorithm on even higher-dimensional data, we tested on the Musk data sets from the UCI repository [4], which represent different conformations of various molecules, labeled according to whether they exhibit a "musk-like" odor when smelled by a human expert. Most results reported by others on this data set are based on 10-fold cross-validation on the 92 bags. We performed 10-fold cross-validation experiments on the same 10 partitions used by Dietterich [14].

For the Musk experiments, the ratio of diagonal entries in the kernel matrix to the off-diagonal entries was often around 10^{50}. So we applied the method of Shölkopf et al. [39] to solve this problem. We used the sub-polynomial function $x^{1/50}$ to reduce the range of each entry in the Gram matrices and then let SVM^{light} work with the empirical kernels described in Section 4.4.3.

For comparison purposes, Table 4.4 summarizes results from Andrews et al. [1] on their multiple-instance (from the conventional MIL model) SVMs, mi-SVM and MI-SVM as well as their results with EMDD [2]. Results for DD come from Maron and Lozano-Pérez [31], and "IAPR" is the iterative axis-parallel rectangle algorithm from Dietterich et al. [14].

There are significant improvements after our empirical kernels are used. The empirical kernel with transduction has the second lowest error rate on Musk1 and the lowest error rate on Musk2. Its performance is close or even better than IAPR, which is specially tuned for the Musk data sets. Although without transduction, our kernel has slightly higher error

[2] Andrews et al. [1] point out that the EMDD experiments of Zhang and Goldman [52] were optimistically biased since they used the test set to choose the final hypotheses. Thus Andrews et al. reran those experiments. We also reran EMDD on the same 10-fold partitioning used by us and Dietterich et al. [14]. Without tuning, we got even worse error rates, 0.152/Musk1 and 0.206/Musk2 for EMDD.

rates since no testing examples are used to build the empirical kernels. But it has the best performance compared to those algorithms from the conventional MIL model (except for IAPR).

Another interesting point to make is that EMDD and DD were about 60% faster than our algorithm[3] for the 10-dimensional binding affinity data, but only 44% faster for the 20-dimensional data. For Musk 2, EMDD and DD were approximately 24.4% slower than our algorithm per experiment, and for Musk 1 they were 675% slower. This occurred despite the fact that we computed the entire Gram matrix a priori rather than simply computing the entries as needed during SVM optimization, i.e. we had significant wasted computation. Further, since each learning task in applications such as CBIR and drug binding affinity can be treated as a database query (i.e. the data set stays fixed but each learning task involves a user specifying a different labeling of the training set), one could build the kernel matrix once for the entire database and reuse it for each query. This would amortize the cost of building the matrix over many queries. This is in fact what we did in our Musk experiments, so the total time spent by our algorithm for all 10 folds was 2 hours for Musk 1 on a 750 MHz Sun Blade (compared to 135 hours for EMDD and DD) and 40 hours for Musk 2 (compared to 485 hours for EMDD and DD).

4.6 Conclusions

The conventional MIL model has proven to be a very powerful one with many applications and efficient algorithms. Algorithms GMIL-1 [40] and our faster variant GMIL-2 [43] in Scott et al.'s generalization of the conventional MIL model have enjoyed success in applications that cannot be represented in the conventional MIL model. However, both algorithms are inherently inefficient, preventing scaling to higher-dimensional data. We

[3]Training with SVM^{light} took less than ten seconds once the kernel matrix was computed, while kernel computation took on the order of minutes.

Table 4.4: Classification error on the Musk data sets. EMDD, mi-SVM, and MI-SVM are from [1], DD is from [31], and IAPR is from [14].

Algorithms	Musk 1	Musk 2
$k_\wedge$	0.176	0.227
$k_{\wedge\ emp}$ non-transduction	0.120	0.118
$k_{\wedge\ emp}$ transduction	**0.088**	**0.097**
EMDD	0.152	0.151
DD	0.120	0.160
mi-SVM	0.126	0.164
MI-SVM	0.221	0.157
IAPR	**0.076**	**0.108**

formulated their algorithms as a kernel that can be used with a support vector machine to learn concepts in their model. We showed that such a kernel is hard to compute in general, and then we presented an FPRAS for it. Finally, we evaluated our kernel empirically.

Our results not only showed competitiveness with other algorithms in terms of generalization error, but also in speed (e.g. versus DD and EMDD), especially for data of more than 20 dimensions. We also note that it is trivial to parallelize the computation of our kernel to get an almost linear speedup. Further, since each learning task in applications such as CBIR and drug binding affinity can be treated as a database query, one could build the kernel matrix once for the entire database and reuse it for each query. This would amortize the cost of building the matrix over many queries.

Chapter 5

Extended Kernels for Generalized Multiple-Instance Learning

While the kernel $k_\wedge$ enjoyed empirical success, it has some limitations. In this chapter, we propose two extensions to $k_\wedge$, a count-based kernel and a normalized kernel. Our experiments showed that both of them improve the performance of $k_\wedge$.

5.1 A Count-based Kernel for GMIL

5.1.1 Introduction

The GMIL model we used in the previous chapters could be further generalized along the lines of Weidmann et al. [50]. In this section, one of our contributions is a new remapping that generalizes Weidmann et al.'s "count-based" MIL model and a kernel $k_{\min}$ that corresponds to that mapping. We then show that, as with $k_\wedge$, $k_{\min}$ is #P-complete to compute, so we give an FPRAS for it. Our final contribution is an empirical evaluation of our new kernel on several MIL data sets. We found that $k_{\min}$ can generalize better than $k_\wedge$ for a learning task in content-based image retrieval and in biological sequence analysis, but there is little

room for improvement in the other learning tasks. However, we note that despite $k_{\min}$'s richer representation over $k_\wedge$, it does not overfit.

The rest of this section is as follows. Section 5.1.2 describes the difference between Scott et al.'s GMIL model and Weidmann et al.'s. In Section 5.1.3 we present our extension to $k_\wedge$, which we call $k_{\min}$. Section 5.1.4 shows that computing $k_{\min}$ is #P-complete. Then we give an FPRAS for it in Section 5.1.5. In Section 5.1.6 we describe experimental results of our new kernel on the applications content-based image retrieval, biological sequence analysis, and the Musk data sets.

5.1.2 More on Generalized Multiple-Instance Learning

Scott et al. [40] generalized the MIL model such that rather than P's label being a disjunction of the labels of P's instances, it is a threshold function. Each target concept is defined by a set of k "attraction" points $C = \{c_1, \ldots, c_k\}$ and a set of k' "repulsion" points $\bar{C} = \{\bar{c}_1, \ldots, \bar{c}_{k'}\}$. The label for a bag $P = \{p_1, \ldots, p_n\}$ is positive if and only if there is a subset of r points $C' \subseteq C \cup \bar{C}$ such that each attraction point $c_i \in C'$ is near some point in P (where "near" is defined as within a certain distance under some weighted norm) and each repulsion point $\bar{c}_j \in C'$ is not near any point in P. In other words, if one defines a boolean attribute a_i for each attraction point $c_i \in C$ that is 1 if there exists a point $p \in P$ near it and 0 otherwise and another boolean attribute $\bar{a}_i$ for each repulsion point $\bar{c}_j \in \bar{C}$ that is 1 if there is no point from P near it, then P's label is an r-of-$(k + k')$ threshold function over the attributes (so there are $k + k'$ relevant attributes and P's label is 1 iff at least r of these attributes are 1). Note that if $r = 1$ and if there are no repulsion points, then this model is the conventional multi-instance model, except that there are multiple target points and the final concept is a union of these points.

Independently of Scott et al., Weidmann et al. [50] defined their own generalizations of the MIL model. Their first (referred to as *presence-based* MIL in [50]) is the same as Scott

et al.'s with $r = k$ and no repulsion points. Their second (*threshold-based* MIL) generalizes presence-based MIL by requiring each $c_i \in C$ to be near at least $t_i \geq 0$ distinct points from P for P to be labeled positive. Their third (*count-based* MIL) generalizes threshold-based by requiring the number of distinct points from P that are near c_i to be at least t_i and at most z_i, which cannot be represented by Scott et al.'s model.

Count-based MIL can represent the idea of repulsion points by setting $z_i = 0$ for each repulsion point. Thus this model generalizes the one of Scott et al. when $r = k + k'$. However, the ability of Scott et al.'s model to represent r-of-$(k + k')$ threshold concepts for $r < k + k'$ expands its representational ability beyond that of Weidmann et al.'s. As an example of why this is useful, consider the representation of a human face with shape-based features. For the eyes, the target concept might require two regions with elongation near $3/2$ and an Euler number (number of connected regions minus number of holes) of 0. In addition, one wants (for the mouth) one region with elongation near 8 and 0 or 1 holes in the region. Then there are other constraints (in terms of the above features or based on other shape descriptors) for the shape of the face. There exists a target concept for this case in presence-based MIL so long as all features are visible. But if some parts of the face are occluded (e.g. due to the subject wearing sunglasses) and the set of parts that are occluded can vary, then it is difficult to represent the target concept with even count-based MIL. In contrast, an r-of-k threshold function like that from Scott et al. is a natural way to represent the target concept. Thus we give a new kernel that generalizes both models in the following section.

5.1.3 Extending Kernel $k_\wedge$

We now extend $k_\wedge$ to work in a GMIL model that generalizes count-based MIL model of Weidmann et al. [50]. Recall that their count-based MIL model stipulates that a bag P is positive if and only if each concept point $c_i \in C$ is near at least t_i, and at most z_i, distinct

points from P. Note that count-based MIL can represent the idea of repulsion points by setting $z_i = 0$ for each repulsion point. Thus this model generalizes the one of Scott et al. (used by us in Chapters 3 and 4) when $r = k+k'$. However, the ability of Scott et al.'s model to represent r-of-$(k + k')$ threshold concepts for $r < k + k'$ expands its representational ability beyond the scope of the generalizations of Weidmann et al.

We define a remapping and a kernel to capture the notion of count-based MIL, but using r-of-$(k + k')$ threshold concepts (expanding its representational ability beyond that of Weidmann et al.). Recall the old mapping of $k_\wedge$ (see Section 4.3), where $\phi_\wedge(P)$ is a vector of $|B_\mathcal{X}|$ bits, and for each box $b \in B_\mathcal{X}$, attribute $a_b = 1$ if box b contains a point from bag P and 0 otherwise. In our new mapping $\phi_{\min}(P)$, each box $b \in B_\mathcal{X}$ has n bits associated with it, and $a_{bi} = 1$ if box b contains at least i points from P and 0 otherwise. (Thus if b contains exactly j points from P, we have $a_{bi} = 1$ for $i \leq j$ and $a_{bi} = 0$ for $i > j$.) To see how this captures count-based MIL, imagine that there is exactly one target box b, and all positive bags have at least x and at most $y - 1$ points in b. A weight vector capturing this target concept has $w_{bx} = +1$, $w_{by} = -1$, all other weights 0, and a bias term of $-1/2$. Adjusting the bias term and adding other nonzero components to the weight vector allows us to represent multiple target boxes in an r-of-k threshold function.

Let $P_b \subseteq P$ be the set of points from P that are contained in b. Then it is straightforward to see that the dot product $\phi_{\min}(P) \cdot \phi_{\min}(Q)$ is equivalent to the kernel $k_{\min}(P, Q)$, which can be reformulated as follows.

$$
\begin{aligned}
k_{\min}(P,Q) &= \sum_{b\in B_{\mathcal{X}}} \min(|P_b|,|Q_b|) \\
&= \sum_{b\in B(P\wedge Q)} \min(|P_b|,|Q_b|) \qquad (5.1)\\
&= \sum_{b\in B(P\wedge Q)} \frac{|P_b||Q_b|}{\max(|P_b|,|Q_b|)} \\
&= \sum_{b\in B(P\wedge Q)} \sum_{p\in P_b, q\in Q_b} \frac{1}{\max(|P_b|,|Q_b|)} \\
&= \sum_{b\in B(P\wedge Q)} \sum_{p\in P, q\in Q} \frac{I(p\in P_b)I(q\in Q_b)}{\max(|P_b|,|Q_b|)} \\
&= \sum_{p\in P, q\in Q} \sum_{b\in B(p\wedge q)} \frac{1}{\max(|P_b|,|Q_b|)} \ . \qquad (5.2)
\end{aligned}
$$

where $I(\cdot) = 1$ if its argument is true and 0 otherwise.

5.1.4 A Hardness Result for $k_{\min}$

Consider the corresponding counting problem #BOXMin which is defined as follows: Given a triple $\langle \mathcal{X}, P, Q\rangle$, compute $k_{\min}(P,Q)$. We need another related problem for showing the hardness of #BOXMin, which we now define. The problem #BOXAnd defined in Chapter 4 is, given input the triple $\langle \mathcal{X}, P, Q\rangle$, compute $k_{\wedge}(P,Q) = |B(P\wedge Q)|$. In our proof showing that #BOXAnd is #P-complete, we also showed that a restricted version where $|P| = 1$ is #P-complete. We call this problem #RestrictedBOXAnd.

Theorem 5.1 *#RestrictedBOXAnd is #P-complete.*

Theorem 5.2 *#BOXMin is #P-complete.*

Proof: #BOXMin is in #P: Given a triple $\langle \mathcal{X}, P, Q\rangle$, a nondeterministic machine first guesses a $b \in \mathcal{X}$ and then computes $\min(|P_b|,|Q_b|)$. If the minimum is 0, it rejects. Oth-

erwise it branches into $\min(|P_b|, |Q_b|)$ paths and accepts. It is clear that the number of accepting paths = $k_{\min}(P, Q)$.

We now show that in fact computing $k_{\min}(P, Q)$ where P contains only one point is #P-complete by reducing #RestrictedBOXAnd to the restricted version of #BOXMin. The reduction is the identity map: an instance $\langle \mathcal{X}, \{p\}, Q\rangle$ of #RestrictedBOXAnd is mapped to the instance $\langle \mathcal{X}, \{p\}, Q\rangle$ of $k_{\min}(P, Q)$. Then we get

$$\begin{aligned} k_{\min}(\{p\}, Q) &= \sum_{b \in B_{\mathcal{X}}} \min(|P_b|, |Q_b|) \\ &= \sum_{b \in B(p \wedge Q)} \min(|P_b|, |Q_b|) + \sum_{b \notin B(p \wedge Q)} \min(|P_b|, |Q_b|) \\ &= \sum_{b \in B(p \wedge Q)} 1 = |B(p \wedge Q)| = k_{\wedge}(\{p\}, Q) \ . \end{aligned}$$

The third equality is because of the following. For all $b \in B(p \wedge Q)$, $p \in b$ and $|Q \cap b| \geq 1$. Hence the minimum is exactly 1. For all $b \notin B(p \wedge Q)$, $p \notin b$ or $Q \cap b = \phi$. Hence the minimum is 0. Therefore computing $k_{\min}(\{p\}, Q)$ is the same as computing $|B(\{p\} \wedge Q)|$, which is #P-complete. □

5.1.5 Approximating $k_{\min}$

One way to approximate $k_{\min}$ is to approximate (5.1) via a simple change to our algorithm for $k_{\wedge}$. When a sampled triple $(p, q, b) \in G$, we increment γ by $\min(|P_b|, |Q_b|)$ instead of by 1. Unfortunately, the best sample size bound we can get for this technique (via Lemma 5.3 below) is $S = n^6 \ln(2/\delta)/(2\epsilon^2)$, yielding a time complexity of $\Theta(n^7 d(\log s) \log(1/\delta)/\epsilon^2)$. Thus we instead approximate (5.2). We fix each (p, q) pair and approximate that term of the summation by uniformly sampling boxes from $B(p \wedge q)$ and taking the average of $1/\max(|P_b|, |Q_b|)$ for each box b in the sample. Multiplying this average by $|B(p \wedge q)|$ gives us an approximation of that term of the sum.

To bound the sample sizes required to estimate the required quantities, we employ the Hoeffding bound.

Lemma 5.3 *(Hoeffding) Let X_i be independent random variables all with mean μ such that for all i, $a \leq X_i \leq b$. Then for any $\lambda > 0$,*

$$\Pr\left[\left|\frac{1}{S}\sum_{i=1}^{S} X_i - \mu\right| \geq \lambda\right] \leq 2e^{-2\lambda^2 S/(b-a)^2}.$$

Since we're interested in ϵ-good approximations, we'll use $\lambda = \epsilon\mu$. Then we have the following theorem.

Theorem 5.4 *Let $\hat{k}_{\min}(P,Q)$ be our approximation of $k_{\min}(P,Q)$ via approximating each term of (5.2) individually as described above. Then after using $n^2(n-1)^2\ln(2n^2/\delta)/(2\epsilon^2)$ total samples and $O(n^5 d\ln(n/\delta)\log s/\epsilon^2)$ total time,*

$$\Pr\left[(1-\epsilon)k_{\min}(P,Q) \leq \hat{k}_{\min}(P,Q) \leq (1+\epsilon)k_{\min}(P,Q)\right] \geq 1-\delta.$$

Proof: First note that an ϵ-good approximation of each (p,q) term of the summation yields an ϵ-good approximation of $k_{\min}(P,Q)$. Thus we focus on a single (p,q) pair. Given $b \in B(p \wedge q)$, let $X(b) = 1/\max(|P_b|,|Q_b|)$. Then

$$\mu = \mathrm{E}[X] = \frac{1}{|B(p \wedge q)|}\sum_{b \in B(p \wedge q)} 1/\max(|P_b|,|Q_b|) \ .$$

Thus $X, \mu \in [1/n, 1]$. Lemma 5.3 says that our approximation (using a sample of size S) is not ϵ-good with probability at most

$$2e^{-2\epsilon^2\mu^2 S n^2/(n-1)^2} \leq 2e^{-2\epsilon^2 S/(n-1)^2} \ ,$$

since $\mu \geq 1/n$. Setting this to be at most δ/n^2 (so we can apply the union bound over

all n^2 failure probabilities) and solving for S, we get $S \geq (n-1)^2 \ln(2n^2/\delta)/(2\epsilon^2)$ as sufficient for an ϵ-good approximation of each term. Repeat this n^2 times (once per (p,q) pair) to approximate (5.2). The time complexity is $O(n^5 d \ln(n/\delta) \log s/\epsilon^2)$ since it takes time linear in n, d, and $\log s$ to compute each max. □

There is no guarantee that the Gram matrix computed by our approximation algorithm is positive semidefinite. However, it is reasonable to believe that if ϵ is small and $k_{\min}$'s Gram matrix (which is positive semidefinite) has no zero eigenvalues, the approximated matrix would not adversely affect SVM optimization. In our experiments, our approximate kernel works well with $\epsilon = 0.1$. Another issue that our kernel shares with $k_\wedge$ is that $k_{\min}$'s Gram matrix potentially can have large diagonal elements relative to the off-diagonal elements. We also applied Schölkopf et al.'s method [39] as in Section 4.4.3 to address our diagonal dominance problem.

5.1.6 Experimental Results

To evaluate our new kernel, we tested it with SVM^{light} [21] on the following learning tasks: content-based image retrieval, biological sequence analysis, and the Musk data.

Content-Based Image Retrieval

We experimented with the two CBIR tasks used in Chapter 3 and 4. One is the "sunset" task: to distinguish images containing sunsets from those not containing sunsets. Another CBIR task is to test a conjunctive CBIR concept, where the goal was to distinguish images containing a field with no sky from those containing a field and sky or containing no field.

The top two rows of each table in Table 5.1 summarize the prediction error of $k_{\min}$, Tao et al.'s kernel $k_\wedge$, and GMIL-2 [43], a faster version of GMIL-1 with comparable accuracy (Since the CBIR data had $d = 5$ and $n \in [2, 15]$ and typically $n < 10$, we were able to also exactly compute the kernel values.). For comparison purposes, we also give results for

Table 5.1: Generalization errors for CBIR and protein learning tasks. $k_\wedge$ and $k_{\min}$ are from exact computation of the kernel, $\hat{k}_\wedge$ and $\hat{k}_{\min}$ are based on approximations of the kernel with $\epsilon = 0.1$ and $\delta = 0.01$.

Total Error

TASK	$k_{\min}$	$\hat{k}_{\min}$	$k_\wedge$	$\hat{k}_\wedge$	**GMIL-1**	**GMIL-2**	**EMDD**	**DD**
sunset	0.084	0.084	0.086	0.088	0.095	0.098	0.096	0.099
conj.	0.084	0.084	0.106	0.108	0.134	0.147	0.215	0.181
protein	N/A	0.215	N/A	0.218	N/A	0.250	0.365	0.664

False Positive Error

TASK	$k_{\min}$	$\hat{k}_{\min}$	$k_\wedge$	$\hat{k}_\wedge$	**GMIL-1**	**GMIL-2**	**EMDD**	**DD**
sunset	0.112	0.112	0.121	0.120	0.080	0.082	0.082	0.078
conj.	0.198	0.198	0.184	0.192	0.128	0.140	0.213	0.173
protein	N/A	0.215	N/A	0.218	N/A	0.250	0.365	0.668

False Negative Error

TASK	$k_{\min}$	$\hat{k}_{\min}$	$k_\wedge$	$\hat{k}_\wedge$	**GMIL-1**	**GMIL-2**	**EMDD**	**DD**
sunset	0.078	0.078	0.078	0.080	0.160	0.157	0.166	0.168
conj.	0.076	0.075	0.100	0.102	0.219	0.244	0.244	0.282
protein	N/A	0.144	N/A	0.169	N/A	0.250	0.360	0.125

the algorithms Diverse Density [31] and EMDD [52] that operate in the conventional MIL model. The sunset task fits well into the conventional MIL model; hence error rates for EMDD and DD are only about 1% higher than ours, and there is little improvement of $\hat{k}_{\min}$ over $\hat{k}_\wedge$. But since the conjunctive task is more complex, we see that the generalized model helps significantly over the conventional model. Further, there is much improvement of $k_{\min}$ over $k_\wedge$ on the positives and a slight degradation on the negatives, yielding an overall improvement in generalization error.

Identifying Trx-fold Proteins

We have used GMIL-2 and $k_\wedge$ to perform cross-validation tests on a Trx-fold Protein data set (see Chapter 3 and 4) To compare with these results, we performed the same tests with $k_{\min}$, comparing to $k_\wedge$, EMDD and DD (bottom row of each table in Table 5.1). Here we

see that applying $k_{\min}$ instead of $k_\wedge$ yields little improvement in false positive error, but there is a noticeable improvement in false negative error.

Musk Data Sets

Finally, we tested on the Musk data sets from the UCI repository[1]. For the Musk experiments, the ratio of diagonal entries in the kernel matrix to the off-diagonal entries was often around 10^{50}. So we applied the method of Schölkopf et al. [39] to solve this problem (see Section 5.1.5). We used the sub-polynomial function $x^{1/50}$ to reduce the range of each entry in the Gram matrices. We then let SVM^{light} work with the original Gram matrices as well as transduction empirical kernels and non-transduction empirical kernels.

Table 5.2 summarizes our results and those from Andrews et al. [1] with mi-SVM and MI-SVM and their results with EMDD. Results for DD come from [31], TLC results come from the generalized MIL algorithm of Weidmann et al. [50], DD-SVM is from [11], and "IAPR" is the iterative axis-parallel rectangle algorithm from Dietterich et al.

While we see that the empirical kernels based on $k_\wedge$ and $k_{\min}$ provide some of the best results on both Musk sets, there is no improvement in $k_{\min}$ over $k_\wedge$. In fact, the results exactly match except for false positive error on Musk 1 for the transduction case (not shown), in which $k_\wedge$ is better. One possible explanation for this is that there is no room for improvement via a richer hypothesis class for these data sets. Another explanation for this phenomenon is that since[2] $k_{\min}(P,Q)/k_\wedge(P,Q) \in [1,n]$ for all P, Q and the kernel is so diagonally dominant for such high-dimensional input data, the kernel values are too similar to each other to make a difference in training and testing. Specifically, when using the original (non-empirical) kernels, diagonal dominance is so severe that an extra factor of n (which is at most 1000 for either Musk set) makes no difference to the optimizer; it will perform relatively poorly anyway. Further, when we apply the empirical kernel, we

[1] `http://www.ics.uci.edu/~mlearn/MLRepository.html`

[2] This follows immediately from Eqn 5.1. Recall that n is the size of each bag.

get $k_{\min\ emp}(P,Q)/k_{\wedge\ emp}(P,Q) \leq n^{1/50} \leq 1.15$. Thus there is little difference in the empirical versions of these kernels, especially considering that $k_{\min}(P,Q)/k_{\wedge}(P,Q)$ rarely equals n. Thus in cases like Musk when there is diagonal dominance, there is probably little reason to choose $k_{\min}$ over $k_{\wedge}$.

5.2 A Normalized "Kernel" for GMIL

The Gram matrices of $k_{\wedge}$ and $k_{\min}$ usually have entries with huge values. For example, for musk data sets, each entry is larger than 10^{600}. These big entries can easily cause overflow and other numerical problems. So we have to reduce the magnitude of our kernels. One way to do this is to divide each entry by some large constant, which is we did in Chapter 4 and Section 5.1. In this section, we introduce another method, that is, normalizing $k_{\wedge}(P,Q)$ with $k_{\vee}(P,Q)$ instead of a constant, where $k_{\vee}(P,Q)$ is the number of boxes that contain a point from P or a point from Q. Intuition behind this is that for a big value of $k_{\vee}$, dividing by $k_{\vee}$ can reduce the impact of an accidental match due to lots of 1s in two re-mapped feature vectors, which can be caused by bags with a large number of points.

To compute $k_{\vee}(P,Q)$, we first consider the more basic problem #BOX defined as follows. An instance of the problem is a tuple $\langle \mathcal{X}, P \rangle$, where P is a set of n points from $\mathcal{X}$. An algorithm should output the number of boxes in $B_{\mathcal{X}}$ that contain a point from P. That is, an algorithm for #BOX on input $\langle \mathcal{X}, P \rangle$ should output $|B(P)|$.

Theorem 5.5 #BOX is #P-complete.

Proof: First, it is clear that #BOX is in #P. Recall that for sets P and Q, $B(P \wedge Q)$ denotes the set of boxes that contains a point from P and a point from Q, and $B(P)$ denotes the set of boxes that contain a point from P. From Theorem 4.3 we have that computing

Table 5.2: Classification error on the Musk data sets. EMDD, mi-SVM, and MI-SVM are from [1], DD is from [31], TLC is from [50], DD-SVM is from [11], and IAPR is from [14].

Total Error

ALGORITHM	**MUSK1**	**MUSK2**
$\hat{k}_{\min}$	0.176	0.227
$\hat{k}_{\min\ emp}$ non-transduction	0.120	0.118
$\hat{k}_{\min\ emp}$ transduction	0.098	0.097
$\hat{k}_{\wedge}$	0.176	0.227
$\hat{k}_{\wedge\ emp}$ non-transduction	0.120	0.118
$\hat{k}_{\wedge\ emp}$ transduction	0.088	0.097
TLC	0.113	0.169
EMDD	0.152	0.151
DD	0.120	0.160
mi-SVM	0.126	0.164
MI-SVM	0.221	0.157
DD-SVM	0.142	0.087
IAPR	0.076	0.108

False Positive Error

ALGORITHM	**MUSK1**	**MUSK2**
$\hat{k}_{\min}$	0.205	0.310
$\hat{k}_{\min\ emp}$ non-transduction	0.115	0.079
$\hat{k}_{\min\ emp}$ transduction	0.140	0.062
$\hat{k}_{\wedge}$	0.205	0.310
$\hat{k}_{\wedge\ emp}$ non-transduction	0.115	0.079
$\hat{k}_{\wedge\ emp}$ transduction	0.115	0.062

False Negative Error

ALGORITHM	**MUSK1**	**MUSK2**
$\hat{k}_{\min}$	0.145	0.100
$\hat{k}_{\min\ emp}$ non-transduction	0.125	0.183
$\hat{k}_{\min\ emp}$ transduction	0.065	0.158
$\hat{k}_{\wedge}$	0.145	0.100
$\hat{k}_{\wedge\ emp}$ non-transduction	0.125	0.183
$\hat{k}_{\wedge\ emp}$ transduction	0.065	0.158

$|B(P \wedge Q)|$ is #P-complete. Since $B(P \wedge Q) = B(P) \cap B(Q)$, we have:

$$\begin{aligned} |B(P \wedge Q)| &= |B(P) \cap B(Q)| \\ &= |B(P)| + |B(Q)| - |B(P) \cup B(Q)| \\ &= |B(P)| + |B(Q)| - |B(P \cup Q)| \ . \end{aligned}$$

That is, #BOXAnd can be computed using three queries to #BOX. Since computing #BOXAnd is #P-complete, #BOX is #P-hard. □

Since $k_\vee(P, Q) = |B(P \cup Q)|$, computing $k_\vee(P, Q)$ is #P-hard. We also notice that computing $k_\vee(P, Q)$ is a special case of #BOXAnd because $|B(P \cup Q)| = |B((P \cup Q) \cap (P \cup Q))|$. A direct, but less efficient method for approximating $k_\vee(P, Q)$ is to run our algorithm from Section 4.4.2 to compute $k_\wedge(P \cup Q, P \cup Q)$ by drawing $4n^2 \ln(2/\delta)/\epsilon^2$ samples.

We now describe a more efficient way to approximate $Y = |B(P \cup Q)|$. Note that $|B(P \cup Q)| = |\bigcup_{p \in P \cup Q} B(p)|$ and the number of all possible sets $B(p)$ is at most $2n$, where $B(p)$ denotes the set of boxes that contain the point p. It is easy to verify that the three criteria for Karp et al.'s algorithm (see Section 4.4.2) are satisfied in this case: (1) for any p, $|B(p)|$ can be computed easily; (2) we can efficiently sample from $B(p)$; and (3) given $c \in B_\mathcal{X}$ and $p \in P$, we can efficiently check whether $c \in B(p)$. Therefore by drawing $8n \ln(2/\delta)/\epsilon^2$ samples, we will get an ϵ-good approximation $\hat{Y}$ of Y.

There is no guarantee that $k_\wedge/k_\vee$ is a kernel. However our experimental results show that $k_\wedge/k_\vee$ has better performance compared to $k_\wedge$. We tested $k_\wedge/k_\vee$ with SVM^{light} on the following learning tasks: content-based image retrieval, biological sequence analysis and the Musk data sets. As shown in Table 5.3, the prediction errors of $k_\wedge/k_\vee$ are lower than those of $k_\wedge$ for all data sets except for Musk 2.

Table 5.3: Generalization errors of $k_\wedge$ and $k_\wedge/k_\vee$ for CBIR, protein and Musk learning tasks.

TASK	$\hat{k}_\wedge$	$\hat{k}_\wedge/\hat{k}_\vee$
CBIR sunset	0.088	0.086
CBIR conj.	0.108	0.098
Protein	0.218	0.183
Musk1	0.176	0.153
Musk2	0.227	0.275

5.3 Conclusions

We have shown that the kernel $k_\wedge$ is very powerful and well-suited for MIL learning tasks. We extended this kernel by increasing its representational power to generalize "count-based" MIL of Weidmann et al. Empirical results show noticeable improvements in generalization error for $k_{\min}$ over $k_\wedge$ for the conjunctive CBIR task and protein classification. In particular, our new kernel reduced false negative error over $k_\wedge$ while not increasing false positive error. For the sunset and Musk learning tasks, there was little improvement in any error rate, but there was little degradation either. This implies that our new kernel does not overfit despite its richer representation. We also presented a normalized version of $k_\wedge$. Our experimental results showed that it improves the performance of $k_\wedge$ on most tasks.

Chapter 6

Conclusions and Future Work

In this dissertation, we addressed the computation problems of linear threshold-based learning algorithms with exponentially many features and different methods for making these algorithms efficient. We presented our related research work and results on two applications: DNF learning and generalized multiple-instance learning.

First we applied Winnow to DNF learning and 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 original solution without any loss of classification accuracy. 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 all approximation algorithms had prediction error that was no worse (and often better than) a brute force version that exactly computed the weighted sums. A possible future work is applying our algorithm to other algorithms and exploring other MCMC techniques such as blocking and over-relaxation [34]. Also, Khardon et al. [23] give a negative result for exactly simulating Winnow for learning DNF. Does a similar result exist for randomized algorithms simulating Winnow with high probability?

Second, we introduced a generalization of the multiple-instance learning (MIL) model that is much more expressive than the conventional multiple instance model. However, the learning algorithm GMIL-1 for this model required significant time and memory to run. We presented and empirically evaluated a new algorithm GMIL-2 that uses a new strategy to group features together by exploiting their geometry property. Our experimental results showed that it has the same generalization ability as our previous algorithm, but requires much less computation time and memory.

Furthermore, to yield a more scalable algorithm, We reformulate GMIL-1 using a kernel $k_\wedge$ for a support vector machine, reducing its time complexity from exponential to polynomial. Computing $k_\wedge$ is equivalent to counting the number of axis-parallel boxes in a discrete, bounded space that contain at least one point from each of two multisets P and Q. We showed that this problem is #P-complete, but then gave a fully polynomial randomized approximation scheme (FPRAS) for it. Our results showed competitiveness with other algorithms in terms of generalization error. And most significantly $k_\wedge$ achieved some of the best results on the benchmark data sets.

While $k_\wedge$ enjoyed empirical success, it has other limitations such as its representation. We first extended $k_\wedge$ by enriching its representation and empirically evaluate our new kernel $k_{\min}$ on data from content-based image retrieval, biological sequence analysis, and drug discovery. We found that $k_{\min}$ generalized noticeably better than $k_\wedge$ in content-based image retrieval and biological sequence analysis and was slightly better or even with $k_\wedge$ in the other applications, showing that an SVM using $k_{\min}$ does not overfit despite its richer representation. Then we proposed a normalized version of $k_\wedge$ (which divides by $k_\vee$) and empirically compared it with $k_\wedge$. The experimental results showed the improvement on generalization errors in most tests. Since the computation problems in both extensions are #P-complete, we gave two FPRASs for them. An open question that remains is: there are many results bounding the generalization error of SVMs, but they assume that the kernel is

exactly computed. What effect does an ϵ-approximate kernel (assuming it is a kernel) have on generalization error?

Bibliography

[1] S. Andrews, I. Tsochantaridis, and T. Hofmann. Support vector machines for multiple-instance learning. In *Advances in Neural Information Processing Systems 15*, 2002.

[2] P. Auer. On learning from multi-instance examples: Empirical evaluation of a theoretical approach. In *Proc. 14th International Conference on Machine Learning*, pages 21–29. Morgan Kaufmann, 1997.

[3] P. Auer and M. Warmuth. Tracking the best disjunction. *Machine Learning*, 32(2):127–150, August 1998.

[4] C. Blake, E. Keogh, and C. J. Merz. UCI repository of machine learning databases, 2004. `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

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

[6] A. Blum, M. Furst, J. Jackson, M. Kearns, Y. Mansour, and S. Rudich. Weakly learning DNF and characterizing statistical query learning using Fourier analysis. In *Proceedings of Twenty-sixth ACM Symposium on Theory of Computing*, pages 253–262, 1994.

[7] A. Blum and A. Kalai. A note on learning from multiple-instance examples. *Machine Learning*, 30:23–29, 1998.

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

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

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

[11] Y. Chen and J. Z. Wang. Image categorization by learning and reasoning with regions. *Journal of Machine Learning Machine Learning*, 5:913–939, 2004.

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

[13] L. De Raedt. Attribute-value learning versus inductive logic programming: The missing links. In *Proc. 8th International Conference on Inductive Logic Programming*, pages 1–8. Springer Verlag, 1998.

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

[15] D. R. Dooly, Q. Zhang, S. A. Goldman, and R. A. Amar. Multiple-instance learning of real-valued data. *Journal of Machine Learning Research*, 3(Dec):651–678, 2002.

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

[17] S. Geman and D. Geman. Stochastic relaxation, gibbs distributions and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:721–741, 1984.

[18] C. Geyer. Markov chain monte carlo maximum likelihood. In *Computing Science and Statistics: Proc. of the 23rd Symposium on the Interface*, pages 156–163, 1991.

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

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

[21] T. Joachims. Making large-scale SVM learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods: Support Vector Learning*, chapter 11, pages 169–184. MIT Press, 1999.

[22] R. Karp, M. Luby, and N. Madras. Monte-Carlo approximation algorithms for enumeration problems. *Journal of Algorithms*, 10:429–448, 1989.

[23] R. Khardon, D. Roth, and R. Servedio. Efficiency versus convergence of boolean kernels for online learning algorithms. In *Advances in Neural Information Processing Systems 14*, pages 423–430, 2001.

[24] J. Kim, E. N. Moriyama, C. G. Warr, P. J. Clyne, and J. R. Carlson. Identification of novel multi-transmembrane proteins from genomic databases using quasi-periodic structural properties. *Bioinformatics*, 16(9):767–775, 2000.

[25] Jason Lee. Protein function classification with generalized multiple-instance learning. Master's thesis, Dept. of Computer Science and Engineering, University of Nebraska, Summer 2004.

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

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

[28] J. Liu. Peskun's theorem and a modified discrete-state gibbs sampler. *Biometrika*, 83:1087–1092, 1996.

[29] P. M. Long and L. Tan. PAC learning axis-aligned rectangles with respect to product distributions from multiple-instance examples. *Machine Learning*, 30:7–21, 1998.

[30] Wolfgang Maass and Manfred K. Warmuth. Efficient learning with virtual threshold gates. In *Proc. 12th International Conference on Machine Learning*, pages 378–386. Morgan Kaufmann, 1995.

[31] O. Maron and T. Lozano-Pérez. A framework for multiple-instance learning. In *Advances in Neural Information Processing Systems 10*, 1998.

[32] O. Maron and A. L. Ratan. Multiple-instance learning for natural scene classification. In *Proc. 15th International Conf. on Machine Learning*, pages 341–349. Morgan Kaufmann, San Francisco, CA, 1998.

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

[34] R. M. Neal. Suppressing random walks in markov chain monte carlo using ordered overrelaxation. Technical Report 9508, Dept. of Statistics, University of Toronto, 1995.

[35] S. Ray and D. Page. Multiple instance regression. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 425–432, 2001.

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

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

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

[39] B. Schölkopf, J. Weston, E. Eskin, C. Leslie, and W. S. Noble. A kernel approach for learning from almost orthogonal patterns. In *Proceedings of the 13th European Conference on Machine Learning*, pages 511–528, 2002.

[40] S. D. Scott, J. Zhang, and J. Brown. On generalized multiple-instance learning. Technical Report UNL-CSE-2003-5, Dept. of Computer Science, University of Nebraska, 2003.

[41] Q. Tao and S. Scott. An analysis of MCMC sampling methods for estimating weighted sums in Winnow. In *Artificial Neural Networks in Engineering*, volume 13, pages 15–20, 2003.

[42] Q. Tao, S. Scott, N. V. Vinodchandran, and T. Osugi. SVM-based generalized multiple-instance learning via approximate box counting. In *Proceedings of the Twenty-First International Conference on Machine Learning*, pages 799–806, 2004.

[43] Qingping Tao and Stephen Scott. A faster algorithm for generalized multiple-instance learning. In *Proceedings of the Seventeenth Annual FLAIRS Conference*, pages 550–555. Miami Beach, Florida, 2004.

[44] Qingping Tao, Stephen Scott, N. V. Vinodchandran, Thomas Takeo Osugi, and Brandon Mueller. An extended kernel for generalized multiple-instance learning. In *Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence*, page To appear, 2004.

[45] L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal of Computing*, 8:410–421, 1979.

[46] V. Vapnik. *Statistical Learning Theory*. Adaptive and Learning Systems for Signal Processing, Communications, and Control. John Wiley and Sons, New York, 1998.

[47] Chang Wang, Stephen D. Scott, Jun Zhang, Qingping Tao, Dmitri E. Fomenko, and Vadim N. Gladyshev. A study in modeling low-conservation protein superfamilies. Technical Report UNL-CSE-2004-0003, Dept. of Computer Science, University of Nebraska, 2004.

[48] J. Wang and J. D. Zucker. Solving the multiple-instance problem: A lazy learning approach. In *Proc. 17th International Conf. on Machine Learning*, pages 1119–1125, 2000.

[49] J. Z. Wang, J. Li, and G. Wiederhold. SIMPLIcity: semantics-sensitive integrated matching for picture libraries. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(9):947–963, 2001.

[50] N. Weidmann, E. Frank, and B. Pfahringer. A two-level learning method for generalized multi-instance problems. In *Proceedings of the European Conference on Machine Learning*, pages 468–479, 2003.

[51] Jun Zhang. A fast algorithm for generalized multiple-instance learning and its application to protein superfamily identification. Master's thesis, Dept. of Computer Science and Engineering, University of Nebraska, Fall 2003.

[52] Q. Zhang and S. A. Goldman. EM-DD: An improved multiple-instance learning technique. In *Neural Information Processing Systems 14*, pages 1073–1080, 2001.

[53] Q. Zhang, S. A. Goldman, W. Yu, and J. E. Fritts. Content-based image retrieval using multiple-instance learning. In *Proc. 19th International Conf. on Machine Learning*, pages 682–689. Morgan Kaufmann, San Francisco, CA, 2002.