Copyright
by
Hsiang-Fu Yu
2016

The Dissertation Committee for Hsiang-Fu Yu
certifies that this is the approved version of the following dissertation:

Scalable Algorithms for Latent Variable Models in Machine Learning

Committee:

Inderjit S. Dhillon, Supervisor

Keshav Pingali

Lili Qiu

S.V.N. Vishwananthan

Chih-Jen Lin

Scalable Algorithms for Latent Variable Models in Machine Learning

by

Hsiang-Fu Yu, B.S., M.S.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

August 2016

Acknowledgments

I thank my supervisor Prof. Inderjit Dhillon for sharing his constructive advice, providing immense support and inspiring guidance, and giving me enough freedom to explore a diverse set of projects. I also gratefully acknowledge the members of my Ph.D. committee for their time and valuable feedback on this thesis. I also want to express my gratitude to Prof. Chih-Jen Lin, who led me on the path of research and has been my mentor for more than ten years in both research and life. I am also grateful to my earliest mentor Prof. Sen-Peng Eu, who guided me on my first research project.

I would like to thank Cho-Jui Hsieh, who I have been actively collaborated with for a decade. It has been a wonderful experience to work with him. I also want to thank David Inouye and Nikhil Rao for their help in improving my writing and presentation skills. I am also grateful to everyone who has helped and taught me in the past few years.

I like to acknowledge my other collaborators for their help in letting me be more productive than I would have been able to on my own: colleagues from the center for big data analytics at UT Austin (Cho-Jui Hsieh, Nikhil Rao, Si Si, David Inouye, Kai-Yang Chiang, Kai Zhong, Lei Qi, Jiong Zhang, and Pradeep Ravikumar), members from Prof. S.V.N Vishwanathan's group (Hyokun Yun and Parameswaran), and researchers from Microsoft Research

(Purushottom Kar, Prateek Jain and Mikhail Bilenko), and members from Prof. Chih-Jen Lin's group (Chia-Hua Ho, Hsing-Yi Huang, and Yu-Chin Juan). I also acknowledge the National Science Foundation and Intel for providing fundings for parts of this work.

Finally, and most importantly, I dedicate this thesis to my dear parents Chien-Ming Yu and Hsiu-Mei Liu. I could not have finished this work without their love and warm support.

Scalable Algorithms for Latent Variable Models in Machine Learning

Publication No. _____

Hsiang-Fu Yu, Ph.D.
The University of Texas at Austin, 2016

Supervisor: Inderjit S. Dhillon

Latent variable modeling (LVM) is a popular approach in many machine learning applications, such as recommender systems and topic modeling, due to its ability to succinctly represent data, even in the presence of several missing entries. Existing learning methods for LVMs, while attractive, are infeasible for the large-scale datasets required in modern big data applications. In addition, such applications often come with various types of side information such as the text description of items and the social network among users in a recommender system. In this thesis, we present scalable learning algorithms for a wide range of latent variable models such as low-rank matrix factorization and latent Dirichlet allocation. We also develop simple but effective techniques to extend existing LVMs to exploit various types of side information and make better predictions in many machine learning applications such as recommender systems, multi-label learning, and high-dimensional time-series prediction. In addition, we also propose a novel approach for the maximum

inner product search problem to accelerate the prediction phase of many latent variable models.

Table of Contents

List of Tables

List of Figures

Chapter 1

Overview

Latent variable modeling is a popular approach in many machine learning applications due to its ability to succinctly represent data, even in the presence of several missing entries. We briefly describe two applications below: recommender systems and topic modeling.

In a recommender system, the goal is to learn a model from past incomplete rating data such that each user's preference over all items can be estimated with the model. See the left plot of Figure 1.1 for an example of a partially observed rating matrix between users and movies. Low-rank matrix factorization (MF), as a latent variable modeling approach, has empirically been show to be a superior model than traditional nearest-neighbor based approaches in the Netflix Prize and KDD Cup 2011 [34] competitions. The idea is to find latent embeddings for users and items in a low-dimensional space such that the preference or the interaction between a user and an item can be estimated by the inner product between their latent embeddings. For example, based on the observed rating in the left plot of Figure 1.1, we demonstrate a possible two-dimensional latent embedding for each user and each movie in the right plot of Figure 1.1. After the success of MF approaches in the Netflix

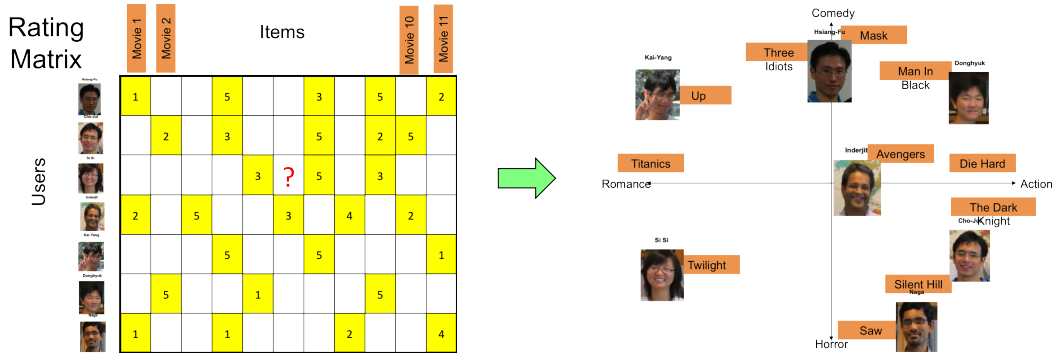


Figure 1.1: Latent Variable Modeling for Recommender Systems. The goal is to find a low dimensional embedding for each user and each item such that the interaction between a user and an item can be estimated by the inner product between the two embeddings.

Prize and KDD Cup 2011 competitions, there has been a great deal of work dedicated to the design of *fast and scalable* methods to learn *effective* latent embeddings for recommender systems [65, 116, 149].

Topic modeling provides a way to aggregate terms in a document corpus into *latent topics* and associate each document with a mixture weight vector over these latent topics based on the tokens of this document. Figure 1.2 demonstrates the idea behind topic modeling: each document in the left plot is mapped onto an embedding over a simplex of the three latent topics. Latent Dirichlet Allocation (LDA) [15] is among the most popular topic modeling approaches. The task of topic inference for LDA is challenging as the number of unknown variables is very large. By integrating out the natural parameters due to the property of conjugate prior, [42] proposes an inference method, called collapsed Gibbs sampling (CGS), for LDA [42]. However, due to its sequential property, CGS is not scalable to very *large document collections* and

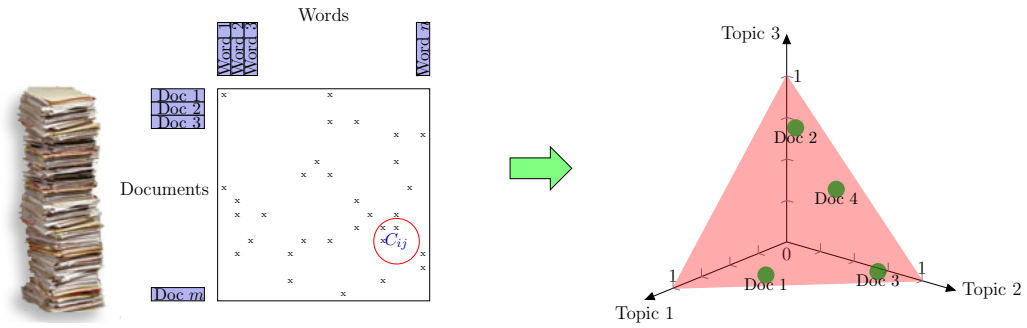


Figure 1.2: Latent Variable Modeling for Topic Modeling. The task is to find a latent embedding on a simplex formed by latent topics for each document in a text corpus.

a *large number of topics*. Unsurprisingly, there have been significant attempts at developing scalable inference algorithms for LDA [55, 68, 84, 110, 133, 134]. In this thesis, we propose a novel alternative one which outperforms recent state-of-the-art topic modeling approaches on massive problems which involve millions of documents, billions of words, and thousands of topics.

While we investigate scaling up standard LVMs like recommender systems and topic modeling, we also consider another aspect of LVMs, namely using side information to improve prediction performance. For example, the text description for items and the social network among users can be beneficial in recommender systems. The goal of this thesis is to address the two challenging questions:

- *How can we design efficient and scalable learning algorithms for massive data?*

- *How can we exploit various types of information in an efficient and effective way?*

This thesis is organized as follows: In Chapters 2-4, we focus on efficient learning algorithms for latent variable models, while in Chapters 5-7, we focus on exploiting various types of side information to make better predictions in large scale ML applications. In Chapter 8, we focus on the acceleration of the prediction phase in latent variable models.

In Chapter 2, we begin with matrix factorization when the missing values are present. In the context of web-scale datasets with millions of users and billions of ratings, scalability becomes an important issue. We will first review two existing algorithms for matrix factorization: alternating least squares (ALS) and stochastic gradient descent (SGD). Due to the cubic time complexity in the target rank, ALS is not scalable to large-scale datasets. On the other hand, SGD conducts efficient updates but usually suffers from slow convergence that is sensitive to the parameters. Thus, we propose a new coordinate descent based algorithm called CCD++ which is fast and easy to parallelize on both multi-core and distributed systems.

In Chapter 3, we focus on improving the efficiency of the collapsed Gibbs sampling (CGS) algorithms for topic modeling with massive document collections which contain millions of documents and billions of word tokens. This task is challenging for two reasons. First, one needs to deal with a large number of topics (typically on the order of thousands). Second, one needs a

scalable and efficient way to distribute the computation across multiple machines. In this work, we use an appropriately modified Fenwick tree to perform CGS (called F+LDA), which reduces the per-step time complexity from $O(T)$ to $O(\log T)$, where T is the number of topics. We further design a novel asynchronous framework called Nomad-LDA to distribute the computation. The combination of the two algorithms is called F+Nomad-LDA, which outperforms recent state-of-the-art topic modeling approaches on massive problems that involve millions of documents, billions of words, and thousands of topics.

In Chapter 4, we study the problem for recommender systems that have only *implicit user feedback*, such as Xbox movie recommendation [90] where we only know movies that have been watched by users rather than having explicit ratings from the users. The two possible ratings are positive and negative but only *some/a few positive* entries are observed. One-class matrix factorization is a popular approach for such scenarios that treats some missing entries as negative. Two major ways to select negative entries are by sub-sampling a set with similar size to that of observed positive entries or by including all missing entries as negative. They are referred to as "Subsampled" and "Full" approaches in this thesis, respectively. Currently, detailed comparisons between these two selection schemes on large-scale data are still lacking. One important reason is that the Full approach leads to a hard optimization problem after treating all missing entries as negative. In this chapter, we successfully develop efficient optimization techniques to solve this challenging problem so that the Full approach becomes practically viable. We then compare in detail

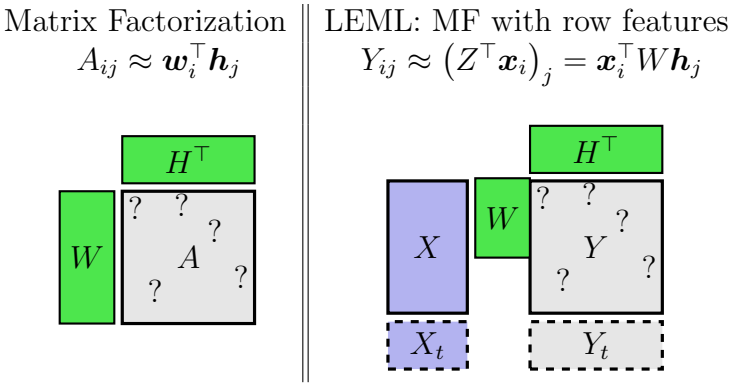


Figure 1.3: LEML: a low-rank structured multi-label learning, which can be regarded as matrix factorization with row features.

the two approaches Subsampled and Full for selecting negative entries. Our results show that the Full approach of including many more missing entries as negative yields better results.

In Chapter 5, we present LEML, a novel low-rank structured algorithm for large-scale multi-label learning with missing values. Each entry of the label matrix Y_{ij} is approximated by $Y_{ij} \approx \boldsymbol{x}_i^\top W \boldsymbol{h}_j$. If we treat the label matrix as the rating matrix and $\boldsymbol{x}_i$ as the feature vector for the i-th row, LEML can be regarded as an extension of matrix factorization with numerical side information (See Figure 1.3). In this work, we develop techniques that exploit the structure of specific loss functions — such as the squared loss function — to obtain efficient algorithms. We present extensive empirical results on a variety of benchmark datasets and show that our methods perform significantly better than existing methods based on label compression and can scale up to very large datasets such as a Wikipedia dataset that has more than 200,000 labels.

In Chapter 6, we tackle the problem of recommender systems when pairwise relationships among users and/or items are given as a graph type of side information. We formulate and derive a highly efficient alternating minimization scheme based on conjugate gradient called GRALS that solves optimizations with over 55 million observations up to two orders of magnitude faster than state-of-the-art methods based on (stochastic) gradient descent. On the theoretical front, we show that such methods generalize weighted nuclear norm formulations and derive statistical consistency guarantees. We validate our results on both real and synthetic datasets.

In Chapter 7, we design a novel temporal regularized matrix factorization framework called TRMF for high-dimensional time series prediction. TRMF not only supports data-driven dependency learning but also enables the forecasting power of matrix factorization. Furthermore, it is easy to handle missing values and high dimensional time series data. Time series prediction problems are becoming increasingly high-dimensional in modern applications, such as climatology and demand forecasting. For example, in the latter problem, the number of items for which demand needs to be forecast might be as large as 50,000. In addition, the data is generally noisy and full of missing values. Thus, modern applications require methods that are highly scalable and can deal with noisy data in terms of corruptions or missing values. However, classical time series methods usually fall short of handling these issues. Our proposed TRMF supports data-driven temporal learning and forecasting, and adapts scalable matrix factorization methods that are eminently suited for

high-dimensional time series data that has many missing values. Our proposed TRMF is very general and subsumes many existing approaches for time series analysis. We make interesting connections to graph regularization methods in the context of learning dependencies in an autoregressive framework. Experimental results show the superiority of TRMF in terms of scalability and prediction quality. In particular, TRMF is two orders of magnitude faster than other methods on a problem of dimension 50,000, and generates better forecasts on real-world datasets such as a Wal-mart E-commerce dataset.

In Chapter 8, we study the computational issue in the prediction phase for many MF-based models: because of the large number of items, the prediction usually becomes a maximum inner product search problem (MIPS) with a very large number of candidate embeddings. For example, computing the preference for all m users over n items with latent embeddings of dimension k in a recommender system costs $O(mnk)$ operations, which is extremely time consuming when both m and n are large. There has been some work on how to perform MIPS with a time complexity sub-linear in n, recently [9, 64, 85, 104]. However, most of them do not have the flexibility to control the trade-off between search efficiency and search quality in the prediction phase. In this chapter, we study the MIPS problem with a computational budget. By carefully studying the problem structure of MIPS, we develop a novel Greedy-MIPS algorithm, which can handle budgeted MIPS by design. While simple and intuitive, Greedy-MIPS yields surprisingly superior performance compared to state-of-the-art approaches. As a specific example, on a candidate set con-

taining half a million vectors of dimension 200, Greedy-MIPS runs 200x faster than the naive approach while yielding search results with the top-5 precision greater than 75%.

Chapter 2

CCD++: Scalable Coordinate Descent for Matrix Factorization

In a recommender system, we want to learn a model from past incomplete rating data such that each user's preference over all items can be estimated with the model. Matrix factorization was empirically shown to be a better model than traditional nearest-neighbor based approaches in the Netflix Prize competition and KDD Cup 2011 [34]. Since then there has been a great deal of work dedicated to the design of fast and scalable methods for large-scale matrix factorization problems [65, 116, 149].

Let $A \in \mathbb{R}^{m \times n}$ be the rating matrix in a recommender system, where m and n are the number of users and items, respectively. The matrix factorization problem for recommender systems is

$$\min_{\substack{W \in \mathbb{R}^{m \times k} \\ H \in \mathbb{R}^{n \times k}}} \sum_{(i,j) \in \Omega} (A_{ij} - \boldsymbol{w}_i^\top \boldsymbol{h}_j)^2 + \lambda \big(\|W\|_F^2 + \|H\|_F^2 \big), \tag{2.1}$$

where Ω is the set of indices for observed ratings; λ is the regularization parameter; $\|\cdot\|_F$ denotes the Frobenius norm; $\boldsymbol{w}_i^\top$ and $\boldsymbol{h}_j^\top$ are the i^{th} and the j^{th}

The materials in this Chapter have been published in [136, 138]. I developed the algorithms and conducted the experiments.

row vectors of the matrices W and H, respectively. The goal of problem (2.1) is to approximate the incomplete matrix A by $WH^\top$, where W and H are rank-k matrices. Note that the well-known rank-k approximation by Singular Value Decomposition (SVD) cannot be directly applied to (2.1) as A is not fully observed.

Regarding problem (2.1), we can interpret $\boldsymbol{w}_i$ and $\boldsymbol{h}_j$ as the length-k feature vectors for user i and item j. The interaction/similarity between the i^{th} user and the j^{th} item is measured by $\boldsymbol{w}_i^\top \boldsymbol{h}_j$. As a result, solving problem (2.1) can be regarded as a procedure to find a "good" representation for each user and item such that the interaction between them can well approximate the real rating scores.

In recent recommender system competitions, we observe that alternating least squares (ALS) and stochastic gradient descent (SGD) have attracted much attention and are widely used for matrix factorization [24, 149]. ALS alternatively switches between updating W and updating H while fixing the other factor. Although the time complexity per iteration is $O(|\Omega|k^2 + (m + n)k^3)$, [149] shows that ALS is well suited for parallelization. It is then not a coincidence that ALS is the only parallel matrix factorization implementation for collaborative filtering in Apache Mahout.[1]

As mentioned in [65], SGD has become one of the most popular methods for matrix factorization in recommender systems due to its efficiency and

[1] http://mahout.apache.org/

simple implementation. The time complexity per iteration of SGD is $O(|\Omega|k)$, which is lower than ALS. However, as compared to ALS, SGD needs more iterations to obtain a good enough model, and its performance is sensitive to the choice of the learning rate. Furthermore, unlike ALS, parallelization of SGD is challenging, and a variety of schemes have been proposed to parallelize it [37, 66, 87, 100, 150].

In this chapter, we aim to design an efficient and easily parallelizable method for matrix factorization in large-scale recommender systems. Recently, [31] and [48] have showed that coordinate descent methods are effective for nonnegative matrix factorization (NMF). This motivates us to investigate coordinate descent approaches for (2.1). In this paper, we propose a coordinate descent based method, CCD++, which has fast running time and can be easily parallelized to handle data of various scales. Table 2.1 shows a comparison between the state-of-the-art approaches and our proposed algorithm CCD++. The main contributions of this work are:

- We propose a scalable and efficient coordinate descent based matrix factorization method CCD++. The time complexity per iteration of CCD++ is lower than that of ALS, and it achieves faster convergence than SGD.
- We show that CCD++ can be easily applied to problems of various scales on both shared-memory multi-core and distributed systems.

Notation. The following notation is used throughout this chapter. We denote matrices by uppercase letters and vectors by bold-faced lowercase

Table 2.1: Comparison between CCD++ and other state-of-the-art methods for matrix factorization.

	ALS	SGD	CCD++
Time complexity per iteration	$O(\|\Omega\|k^2 + (m+n)k^3)$	$O(\|\Omega\|k)$	$O(\|\Omega\|k)$
Convergence behavior	Stable	Sensitive to parameters	Stable
Scalability on distributed systems	Not scalable	Scalable	Scalable

letters. A_{ij} denotes the (i,j) entry of the matrix A. We use Ω_i to denote the column indices of observed ratings in the i^{th} row, and $\bar{\Omega}_j$ to denote the row indices of observed ratings in the j^{th} column. We denote the i^{th} row of W by $\boldsymbol{w}_i^\top$, and the t^{th} column of W by $\bar{\boldsymbol{w}}_t \in \mathbb{R}^m$:

$$W = \begin{bmatrix} \vdots \\ \boldsymbol{w}_i^\top \\ \vdots \end{bmatrix} = \begin{bmatrix} \cdots & \bar{\boldsymbol{w}}_t & \cdots \end{bmatrix}.$$

Thus, both w_{it} (i.e., the t^{th} element of $\boldsymbol{w}_i$) and $\bar{w}_{ti}$ (i.e., the i^{th} element of $\bar{\boldsymbol{w}}_t$) denote the same entry, W_{it}. For H, we use similar notation $\boldsymbol{h}_j$ and $\bar{\boldsymbol{h}}_t$.

The rest of this chapter is organized as follows. An introduction to ALS and SGD is given in Section 2.1. We then present our coordinate descent approaches in Section 2.2. In Section 2.3, we present strategies to parallelize CCD++ and conduct scalability analysis under different parallel computing environments. We then present experimental results in Section 2.4.

2.1 Related Work

As mentioned in [65], the two standard approaches to approximate the solution of problem (2.1) are ALS and SGD. In this section we briefly introduce these methods and discuss recent parallelization approaches.

2.1.1 Alternating Least Squares

Problem (2.1) is intrinsically a non-convex problem; however, when fixing either W or H, (2.1) becomes a quadratic problem with a globally optimal solution. Based on this idea, ALS alternately switches between optimizing W while keeping H fixed, and optimizing H while keeping W fixed. Thus, ALS monotonically decreases the objective function value in (2.1) until convergence.

Under this alternating optimization scheme, (2.1) can be further separated into many independent least squares subproblems. Specifically, if we fix H and minimize over W, the optimal $\boldsymbol{w}_i^*$ can be obtained independently of other rows of W by solving the regularized least squares subproblem:

$$\min_{\boldsymbol{w}_i} \quad \sum_{j \in \Omega_i} (A_{ij} - \boldsymbol{w}_i^\top \boldsymbol{h}_j)^2 + \lambda \|\boldsymbol{w}_i\|^2, \tag{2.2}$$

which leads to the closed form solution

$$\boldsymbol{w}_i^* = (H_{\Omega_i}^\top H_{\Omega_i} + \lambda I)^{-1} H^\top \boldsymbol{a}_i, \tag{2.3}$$

where $H_{\Omega_i}^\top$ is the sub-matrix with columns $\{\boldsymbol{h}_j : j \in \Omega_i\}$, and $\boldsymbol{a}_i^\top$ is the i^{th} row of A with missing entries filled by zeros. To compute each $\boldsymbol{w}_i^*$, ALS needs $O(|\Omega_i|k^2)$ time to form the $k \times k$ matrix $H_{\Omega_i}^\top H_{\Omega_i}$ and additional $O(k^3)$ time

to solve the least squares problem. Thus, the time complexity of a full ALS iteration (i.e., updating W and H once) is $O(|\Omega|k^2 + (m+n)k^3)$.

In terms of parallelization, [149] points out that ALS can be easily parallelized in a row-by-row manner as each row of W or H can be updated independently. However, in a distributed system, when W or H exceeds the memory capacity of a computation node, the parallelization of ALS becomes more challenging. More details are discussed in Section 2.3.3.

2.1.2 Stochastic Gradient Descent

Stochastic gradient descent (SGD) is widely used in many machine learning problems [16], and it has also been shown to be effective for matrix factorization [65]. In SGD, for each update, a rating (i, j) is randomly selected from Ω, and the corresponding variables $\boldsymbol{w}_i$ and $\boldsymbol{h}_j$ are updated by

$$\boldsymbol{w}_i \leftarrow \boldsymbol{w}_i - \eta\left(\frac{\lambda}{|\Omega_i|}\boldsymbol{w}_i - R_{ij}\boldsymbol{h}_j\right),$$
$$\boldsymbol{h}_j \leftarrow \boldsymbol{h}_j - \eta\left(\frac{\lambda}{|\bar{\Omega}_j|}\boldsymbol{h}_j - R_{ij}\boldsymbol{w}_i\right),$$

where $R_{ij} = A_{ij} - \boldsymbol{w}_i^\top \boldsymbol{h}_j$, and η is the learning rate. For each rating A_{ij}, SGD needs $O(k)$ operations to update $\boldsymbol{w}_i$ and $\boldsymbol{h}_j$. If we define $|\Omega|$ consecutive updates as one iteration of SGD, the time complexity per SGD iteration is thus only $O(|\Omega|k)$. As compared to ALS, SGD appears to be faster in terms of the time complexity for one iteration, but typically it needs more iterations than ALS to achieve a good enough model.

However, conducting several SGD updates in parallel directly might

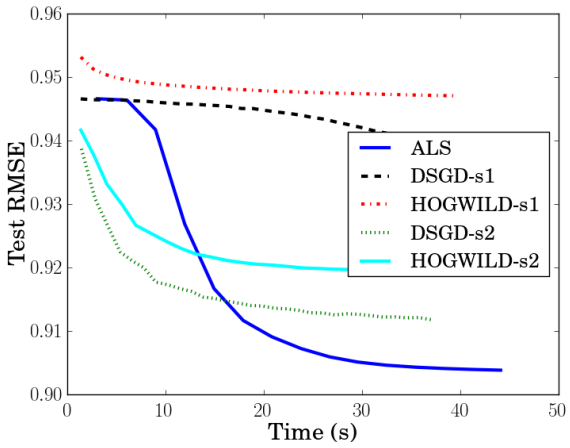


Figure 2.1: Comparison between ALS, DSGD, and HogWild on the movielens10m dataset with $k = 40$ on a 8-core machine (-s1 and -s2 stand for different initial learning rates).

raise an overwriting issue as the updates for the ratings in the same row or the same column of A involve the same variables. Moreover, traditional convergence analysis of standard SGD mainly depends on its sequential update property. These issues make parallelization of SGD a challenging task. Recently, several update schemes to parallelize SGD have been proposed. For example, "delayed updates" are proposed in [66] and [2], while [150] uses a bootstrap aggregation scheme. A lock-free approach called HogWild is investigated in [87], in which the overwriting issue is ignored based on the intuition that the probability of updating the same row of W or H is small when A is sparse. The authors of [87] also show that HogWild is more efficient than the "delayed update" approach in [66]. For matrix factorization, [37] and [100]

propose Distributed SGD (DSGD)[2] which partitions A into blocks and updates a set of independent blocks in parallel at the same time. Thus, DSGD can be regarded as an exact SGD implementation with a specific ordering of updates.

Another issue with SGD is that the convergence is highly sensitive to the learning rate η. In practice, the initial choice and adaptation strategy for η are crucial issues when applying SGD to matrix factorization problems. As the learning rate issue is beyond the scope of this paper, here we only briefly discuss how the learning rate is adjusted in HogWild and DSGD. In HogWild [87], η is reduced by multiplying a constant $\beta \in (0, 1)$ at each iteration. In DSGD, [37] proposes using the "bold driver" scheme, in which, at each iteration, η is increased by a small proportion (5% is used in [37]) when the function value decreases; when the value increases, η is drastically decreased by a large proportion (50% is used in [37]).

2.1.3 Experimental Comparison

Next, we compare various parallel matrix factorization approaches: ALS,[3] DSGD,[4] and HogWild[5] on the movielens10m dataset with $k = 40$ and $\lambda = 0.1$ (more details on the dataset are given later in Table 2.2 of Section 2.4). Here we conduct the comparison on an 8-core machine (see Section 2.4.2

[2] In [100], the name "Jellyfish" is used

[3] Intel MKL is used in our implementation of ALS.

[4] We implement a multi-core version of DSGD according to [37].

[5] HogWild is downloaded from http://research.cs.wisc.edu/hazy/victor/Hogwild/ and modified to start from the same initial point as ALS and DSGD.

for the detailed description of the experimental environment). All 8 cores are utilized for each method.[6] Figure 2.1 shows the comparison; "-s1" and "-s2" denote two choices of the initial η.[7] The reader might notice that the performance difference between ALS and DSGD is not as large as in [37]. The reason is that the parallel platform used in our comparison is different from that used in [37], which is a modified Hadoop distributed system.

In Figure 2.1, we first observe that the performance of both DSGD and HogWild is sensitive to the choice of η. In contrast, ALS, a parameter-free approach, is more stable, albeit it has higher time complexity per iteration than SGD. Next, we can see that DSGD converges slightly faster than HogWild with both initial η's. Given the fact that the computation time per iteration of DSGD is similar to that of HogWild (as DSGD is also a lock-free scheme), we believe that there are two possible explanations: 1) the "bold driver" approach used in DSGD is more stable than the exponential decay approach used in HogWild; 2) the variable overwriting might slow down convergence of HogWild.

2.2 Coordinate Descent Approaches

Coordinate descent is a classic and well-studied optimization technique [14, Section 2.7]. Recently it has been successfully applied to various large-

[6] In HogWild, seven cores are used for SGD updates, and one core is used for random shuffle.

[7] for -s1, initial $\eta = 0.001$; for -s2, initial $\eta = 0.05$.

scale problems such as linear SVMs [49], maximum entropy models [135], NMF problems [31, 48], and sparse inverse covariance estimation [50]. The basic idea of coordinate descent is to update a single variable at a time while keeping others fixed. There are two key components in coordinate descent methods: one is the update rule used to solve each one-variable subproblem, and the other is the update sequence of variables.

In this section, we apply coordinate descent to attempt to solve (2.1). We first form the one-variable subproblem and derive the update rule. Based on the rule, we investigate two sequences to update variables: item/user-wise and feature-wise.

2.2.1 The Update Rule

If only one variable w_{it} is allowed to change to z while fixing all other variables, we need to solve the following one-variable subproblem:

$$\min_{z} \; f(z) = \sum_{j \in \Omega_i} \left(A_{ij} - (\boldsymbol{w}_i^\top \boldsymbol{h}_j - w_{it} h_{jt}) - z h_{jt} \right)^2 + \lambda z^2. \tag{2.4}$$

As $f(z)$ is a univariate quadratic function, the unique solution z^* to (2.4) can be easily found:

$$z^* = \frac{\sum_{j \in \Omega_i} (A_{ij} - \boldsymbol{w}_i^\top \boldsymbol{h}_j + w_{it} h_{jt}) h_{jt}}{\lambda + \sum_{j \in \Omega_i} h_{jt}^2}. \tag{2.5}$$

Direct computation of z^* via (2.5) from scratch takes $O(|\Omega_i| k)$ time. For large k, we can accelerate the computation by maintaining the residual matrix R,

$$R_{ij} \equiv A_{ij} - \boldsymbol{w}_i^\top \boldsymbol{h}_j, \; \forall (i,j) \in \Omega.$$

In terms of R_{ij}, the optimal z^* can be computed by:

$$z^* = \frac{\sum_{j\in\Omega_i}(R_{ij} + w_{it}h_{jt})h_{jt}}{\lambda + \sum_{j\in\Omega_i} h_{jt}^2}. \tag{2.6}$$

When R is available, computing z^* by (2.6) only costs $O(|\Omega_i|)$ time. After z^* is obtained, w_{it} and $R_{ij},\ \forall j \in \Omega_i$, can also be updated in $O(|\Omega_i|)$ time via

$$R_{ij} \leftarrow R_{ij} - (z^* - w_{it})h_{jt},\ \forall j \in \Omega_i, \tag{2.7}$$

$$w_{it} \leftarrow z^*. \tag{2.8}$$

Note that (2.7) requires $O(|\Omega_i|)$ operations. Therefore, if we maintain the residual matrix R, the time complexity of each single variable update is reduced from $O(|\Omega_i|k)$ to $O(|\Omega_i|)$. Similarly, the update rules for each variable in H, h_{jt} for instance, can be derived as

$$R_{ij} \leftarrow R_{ij} - (s^* - h_{jt})w_{it},\ \forall i \in \bar{\Omega}_j, \tag{2.9}$$

$$h_{jt} \leftarrow s^*, \tag{2.10}$$

where s^* can be computed by either:

$$s^* = \frac{\sum_{i\in\bar{\Omega}_j}(A_{ij} - \boldsymbol{w}_i^\top \boldsymbol{h}_j + w_{it}h_{jt})w_{it}}{\lambda + \sum_{i\in\bar{\Omega}_j} w_{it}^2}, \tag{2.11}$$

or

$$s^* = \frac{\sum_{i\in\bar{\Omega}_j}(R_{ij} + w_{it}h_{jt})w_{it}}{\lambda + \sum_{i\in\bar{\Omega}_j} w_{it}^2}. \tag{2.12}$$

With update rules (2.7)-(2.10), we are able to apply any update sequence over variables in W and H. We now investigate two main sequences: item/user-wise and feature-wise update sequences.

2.2.2 Item/User-wise Update: CCD

First, we consider the item/user-wise update sequence, which updates the variables corresponding to either an item or a user at a time.

ALS can be viewed as a method which adopts this update sequence. As mentioned in Section 2.1.1, ALS switches the updating between W and H. To update W when fixing H or vice versa, ALS solves many k-variable least squares subproblems. Each subproblem corresponds to either an item or a user. That is, ALS cyclically updates variables with the following sequence:

$$\overbrace{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_m}^{W}, \overbrace{\boldsymbol{h}_1, \ldots, \boldsymbol{h}_n}^{H}.$$

In ALS, the update rule in (2.3) involves forming a $k \times k$ Hessian matrix and solving a least squares problem which takes $O(k^3)$ time. However, it is not necessary to solve all subproblems (2.2) exactly in the early stages of the algorithm. Thus, [93] proposed a cyclic coordinate descent method (CCD), which is similar to ALS with respect to the update sequence. The only difference lies in the update rules. In CCD, $\boldsymbol{w}_i$ is updated by applying (2.8) over all elements of $\boldsymbol{w}_i$ (i.e., $w_{i1}, \ldots, w_{ik}$) once. The entire update sequence of one iteration in CCD is

$$\overbrace{\underbrace{w_{11}, \ldots, w_{1k}}_{\boldsymbol{w}_1}, \ldots, \underbrace{w_{m1}, \ldots, w_{mk}}_{\boldsymbol{w}_m}}^{W}, \overbrace{\underbrace{h_{11}, \ldots, h_{1k}}_{\boldsymbol{h}_1}, \ldots, \underbrace{h_{n1}, \ldots, h_{nk}}_{\boldsymbol{h}_n}}^{H}. \tag{2.13}$$

Algorithm 2.1 describes the CCD procedure with T iterations. Note that if we set the initial W to 0, then the initial residual matrix R is exactly equal to A, so no extra effort is needed to initialize R.

As mentioned in Section 2.2.1, the update cost for each variable in W and H, taking w_{it} and h_{jt} for instance, is just $O(|\Omega_i|)$ or $O(|\bar{\Omega}_j|)$. If we define one iteration in CCD as updating all variables in W and H once, the time complexity per iteration for CCD is thus

$$O\left(\left(\sum_i |\Omega_i| + \sum_j |\bar{\Omega}_j|\right)k\right) = O(|\Omega|k).$$

We can see that an iteration of CCD is faster than an iteration of ALS when $k > 1$, because ALS requires $O(|\Omega|k^2 + (m+n)k^3)$ time at each iteration. Of course, each iteration of ALS makes more progress; however, at early stages of this algorithm, it is not clear that this extra progress helps.

Instead of cyclically updating through $w_{i1}, \ldots, w_{ik}$, one may think of a greedy update sequence that sequentially updates the variable that decreases the objective function the most. In [48], a greedy update sequence is applied to solve the NMF problem in an efficient manner by utilizing the property that all subproblems in NMF share the same Hessian. However, unlike NMF, each subproblem (2.2) of problem (2.1) has a potentially different Hessian as $\Omega_{i_1} \neq \Omega_{i_2}$ for $i_1 \neq i_2$ in general. Thus, if the greedy coordinate descent (GCD) method proposed in [48] is applied to solve (2.1), m different Hessians are required to update W, and n Hessians are required to update H. The computation of Hessian for $\boldsymbol{w}_i$ and $\boldsymbol{h}_j$ needs $O(|\Omega_i|k^2)$ and $O(|\bar{\Omega}_j|k^2)$ to compute, respectively. The total time complexity of GCD to update W and H once is thus $O(|\Omega|k^2)$ operations per iteration, which is the same complexity as ALS.

Algorithm 2.1 CCD Algorithm [93]

Input: A, W, H, λ, k, T

Initialize $W = 0$ and $R = A$.
for $iter = 1, 2, \ldots, T$ **do**
  for $i = 1, 2, \ldots, m$ **do** ▷ Update W.
    for $t = 1, 2, \ldots, k$ **do**
      Obtain z^* using (2.6).
      Update R and w_{it} using (2.7) and (2.8).
  for $j = 1, 2, \ldots, n$ **do** ▷ Update H.
    for $t = 1, 2, \ldots, k$ **do**
      Obtain s^* using (2.12).
      Update R and h_{jt} using (2.9) and (2.10).

Algorithm 2.2 CCD++ Algorithm

Input: A, W, H, λ, k, T

Initialize $W = 0$ and $R = A$.
for $iter = 1, 2, \ldots$ **do**
  for $t = 1, 2, \ldots, k$ **do**
    Construct $\hat{R}$ by (2.16).
    for $inneriter = 1, 2, \ldots, T$ **do** ▷ T CCD iterations for (2.17).
      Update $\boldsymbol{u}$ by (2.18).
      Update $\boldsymbol{v}$ by (2.19).
    Update $(\bar{\boldsymbol{w}}_t, \bar{\boldsymbol{h}}_t)$ and R by (2.20) and (2.21).

2.2.3 Feature-wise Update: CCD++

The factorization $WH^\top$ can be represented as a summation of k outer products:

$$A \approx WH^\top = \sum_{t=1}^{k} \bar{\boldsymbol{w}}_t \bar{\boldsymbol{h}}_t^\top, \tag{2.14}$$

where $\bar{\boldsymbol{w}}_t \in \mathbb{R}^m$ is the t^{th} column of W, and $\bar{\boldsymbol{h}}_t \in \mathbb{R}^n$ is the t^{th} column of H. From the perspective of the latent feature space, $\bar{\boldsymbol{w}}_t$ and $\bar{\boldsymbol{h}}_t$ correspond to the

t^{th} latent feature.

This leads us to our next coordinate descent method, CCD++. At each time, we select a specific feature t and conduct the update

$$(\bar{\boldsymbol{w}}_t, \bar{\boldsymbol{h}}_t) \leftarrow (\boldsymbol{u}^*, \boldsymbol{v}^*),$$

where $(\boldsymbol{u}^*, \boldsymbol{v}^*)$ is obtained by solving the following subproblem:

$$\min_{\boldsymbol{u}\in\mathbb{R}^m, \boldsymbol{v}\in\mathbb{R}^n} \sum_{(i,j)\in\Omega} \left(R_{ij} + \bar{w}_{ti}\bar{h}_{tj} - u_i v_j\right)^2 + \lambda(\|\boldsymbol{u}\|^2 + \|\boldsymbol{v}\|^2), \tag{2.15}$$

where $R_{ij} = A_{ij} - \boldsymbol{w}_i^\top \boldsymbol{h}_j$ is the residual entry for (i, j). If we define

$$\hat{R}_{ij} = R_{ij} + \bar{w}_{ti}\bar{h}_{tj}, \ \forall (i,j) \in \Omega, \tag{2.16}$$

(2.15) can be rewritten as:

$$\min_{\boldsymbol{u}\in\mathbb{R}^m, \boldsymbol{v}\in\mathbb{R}^n} \sum_{(i,j)\in\Omega} (\hat{R}_{ij} - u_i v_j)^2 + \lambda(\|\boldsymbol{u}\|^2 + \|\boldsymbol{v}\|^2), \tag{2.17}$$

which is exactly the rank-one matrix factorization problem (2.1) for the matrix $\hat{R}$. Thus we can apply CCD to (2.17) to obtain an approximation by alternatively updating $\boldsymbol{u}$ and updating $\boldsymbol{v}$. When the current model (W, H) is close to an optimal solution to (2.1), $(\bar{\boldsymbol{w}}_t, \bar{\boldsymbol{h}}_t)$ should be also very close to an optimal solution to (2.17). Thus, the current $(\bar{\boldsymbol{w}}_t, \bar{\boldsymbol{h}}_t)$ can be a good initialization for $(\boldsymbol{u}, \boldsymbol{v})$. The update sequence for $\boldsymbol{u}$ and $\boldsymbol{v}$ is

$$u_1, u_2, \ldots, u_m, v_1, v_2, \ldots, v_n.$$

When the rank is equal to one, (2.5) and (2.6) have the same complexity. Thus, during the CCD iterations to update u_i and v_j, z^* and s^* can be directly

obtained by (2.5) and (2.11) without additional residual maintenance. The update rules for $\boldsymbol{u}$ and $\boldsymbol{v}$ at each CCD iteration become as follows:

$$u_i \leftarrow \frac{\sum_{j\in\Omega_i} \hat{R}_{ij} v_j}{\lambda + \sum_{j\in\Omega_i} v_j^2}, \ i = 1, \ldots, m, \tag{2.18}$$

$$v_j \leftarrow \frac{\sum_{i\in\bar{\Omega}_j} \hat{R}_{ij} u_j}{\lambda + \sum_{i\in\bar{\Omega}_j} u_i^2}, \ j = 1, \ldots, n. \tag{2.19}$$

After obtaining $(\boldsymbol{u}^*, \boldsymbol{v}^*)$, we can update $(\bar{\boldsymbol{w}}_t, \bar{\boldsymbol{h}}_t)$ and R by

$$(\bar{\boldsymbol{w}}_t, \bar{\boldsymbol{h}}_t) \leftarrow (\boldsymbol{u}^*, \boldsymbol{v}^*). \tag{2.20}$$

$$R_{ij} \leftarrow \hat{R}_{ij} - u_i^* v_j^*, \ \forall (i,j) \in \Omega, \tag{2.21}$$

The update sequence for each outer iteration of CCD++ is

$$\bar{\boldsymbol{w}}_1, \bar{\boldsymbol{h}}_1, \ldots, \bar{\boldsymbol{w}}_t, \bar{\boldsymbol{h}}_t, \ldots, \bar{\boldsymbol{w}}_k, \bar{\boldsymbol{h}}_k. \tag{2.22}$$

We summarize CCD++ in Algorithm 2.2. A similar procedure with the feature-wise update sequence is also used in [13] to avoid the over-fitting issue in recommender systems.

Each time when the t^{th} feature is selected, CCD++ consists of the following steps to update $(\bar{\boldsymbol{w}}_t, \bar{\boldsymbol{h}}_t)$: constructing $O(|\Omega|)$ entries of $\hat{R}$, conducting T CCD iterations to solve (2.17), updating $(\bar{\boldsymbol{w}}_t, \bar{\boldsymbol{h}}_t)$ by (2.20), and maintaining $|\Omega|$ residual entries by (2.21). Since each CCD iteration in Algorithm 2.2 costs only $O(|\Omega|)$ operations, the time complexity per iteration for CCD++, where all k features are updated by T CCD iterations, is $O(|\Omega|kT)$.

At first glance, the only difference between CCD++ and CCD appears to be their different update sequence. However, such difference might affect

the convergence. A similar update sequence has also been considered for NMF problems, and [47] observes that such a feature-wise update sequence leads to faster convergence than other sequences on moderate-scale matrices. However, for large-scale sparse NMF problems, when all entries are known, the residual matrix becomes a $m \times n$ dense matrix, which is too large to maintain. Thus [31, 48] utilize the property that all subproblems share a single Hessian, where there are no missing values, to develop techniques that allow efficient variable updates without maintenance of the residual.

Due to the large number of missing entries in A, problem (2.1) does not share the above favorable property. However, as a result of the sparsity of observed entries, the residual maintenance is affordable for problem (2.1) with a large-scale A. Furthermore, the feature-wise update sequence might even bring faster convergence as it does for NMF problems.

2.3 Parallelization of CCD++

With the exponential growth of dyadic data on the web, scalability becomes an issue when applying state-of-the-art matrix factorization approaches to large-scale recommender systems. Recently, there has been growing interest in addressing the scalability problem by using parallel and distributed computing. Both CCD and CCD++ can be easily parallelized. Due to the similarity with ALS, CCD can be parallelized in the same way as ALS in [37]. For CCD++, we propose two versions: one version for multi-core shared memory systems and the other for distributed systems.

It is important to select an appropriate parallel environment based on the scale of the recommender system. Specifically, when the matrices A, W, and H can be loaded in the main memory of a single machine, and we consider a distributed system as the parallel environment, the communication among machines might dominate the entire procedure. In this case, a multi-core shared memory system is a better parallel environment. However, when the data/variables exceed the memory capacity of a single machine, a distributed system, in which data/variables are distributed across different machines, is required to handle problems of this scale. In the following sections, we demonstrate how to parallelize CCD++ under both these parallel environments.

2.3.1 CCD++ on Multi-core Systems

In this section we discuss the parallelization of CCD++ under a multi-core shared memory setting. If the matrices A, W, and H fit in a single machine, CCD++ can achieve significant speedup by utilizing all cores available on the machine.

The key component in CCD++ that requires parallelization is the computation to solve subproblem (2.17). In CCD++, the approximate solution to the subproblem is obtained by updating $\boldsymbol{u}$ and $\boldsymbol{v}$ alternately. When $\boldsymbol{v}$ is fixed, from (2.18), each variable u_i can be updated independently. Therefore, the update to $\boldsymbol{u}$ can be divided into m independent jobs which can be handled by different cores in parallel.

Given a machine with p cores, we define $S = \{S_1, \ldots, S_p\}$ as a partition

of row indices of W, $\{1, \ldots, m\}$. We decompose $\boldsymbol{u}$ into p vectors $\boldsymbol{u}^1, \boldsymbol{u}^2, \ldots, \boldsymbol{u}^p$, where $\boldsymbol{u}^r$ is the sub-vector of $\boldsymbol{u}$ corresponding to S_r. A simple strategy is to make equal-sized partitions (i.e., $|S_1| = |S_2| = \cdots = |S_p| = m/p$). The workload on the r^{th} core to update $\boldsymbol{u}^r$ equals $\sum_{i \in S_r} 4|\Omega_i|$, which is not the same for all cores. As a result, this strategy leads to load imbalance, which reduces core utilization. An ideal partition can be obtained by solving

$$\min_{S} \left(\max_{r=1}^{p} \sum_{i \in S_r} |\Omega_i| \right) - \left(\min_{r=1}^{p} \sum_{i \in S_r} |\Omega_i| \right),$$

which is a known NP-hard problem. Hence, for multi-core parallelization, instead of being assigned to a fixed core, we assign jobs dynamically based on the availability of each core. When a core finishes a small job, it can always start a new job without waiting for other cores. Such dynamic assignment usually achieves good load balance on multi-core machines. Most multi-core libraries (e.g., OpenMP[8] and Intel TBB[9]) provide a simple interface to conduct this dynamic job assignment. Thus, from now, partition S_r will refer to the indices assigned to the r^{th} core as a result of this dynamic assignment. Such an approach can be also applied to update $\boldsymbol{v}$ and the residual R.

We now provide the details. At the beginning for each subproblem, each core c constructs $\hat{R}$ by

$$\hat{R}_{ij} \leftarrow R_{ij} + \bar{w}_{ti}\bar{h}_{tj}, \ \forall (i,j) \in \Omega_{S_r}, \tag{2.23}$$

[8] http://openmp.org/

[9] http://threadingbuildingblocks.org/

where $\Omega_{S_r} = \bigcup_{i \in S_r} \{(i,j) : j \in \Omega_i\}$. Each core r then

$$updates\ u_i \leftarrow \frac{\sum_{j \in \Omega_i} \hat{R}_{ij} v_j}{\lambda + \sum_{j \in \Omega_i} v_j^2}\ \forall i \in S_r. \tag{2.24}$$

Updating H can be parallelized in the same way with $G = \{G^1, \ldots, G^p\}$, which is a partition of row indices of H, $\{1, \ldots, n\}$. Similarly, each core r

$$updates\ v_j \leftarrow \frac{\sum_{i \in \bar{\Omega}_j} \hat{R}_{ij} u_i}{\lambda + \sum_{i \in \Omega_i} u_i^2}\ \forall j \in G_r. \tag{2.25}$$

As all cores on the machine share a common memory space, no communication is required for each core to access the latest $\boldsymbol{u}$ and $\boldsymbol{v}$. After obtaining $(\boldsymbol{u}^*, \boldsymbol{v}^*)$, we can also update the residual R and $(\bar{\boldsymbol{w}}_t^r, \bar{\boldsymbol{h}}_t^r)$ in parallel by assigning core r to perform the update:

$$(\bar{\boldsymbol{w}}_t^r, \bar{\boldsymbol{h}}_t^r) \leftarrow (\boldsymbol{u}^r, \boldsymbol{v}^r). \tag{2.26}$$

$$R_{ij} \leftarrow \hat{R}_{ij} - \bar{w}_{ti} \bar{h}_{tj},\ \forall (i,j) \in \Omega_{S_r}, \tag{2.27}$$

We summarize our parallel CCD++ approach in Algorithm 2.3.

2.3.2 CCD++ on Distributed Systems

In this section, we investigate the parallelization of CCD++ when the matrices A, W, and H exceed the memory capacity of a singe machine. To avoid frequent access from disk, we consider handling these matrices with a distributed system, which connects several machines with their own computing resources (e.g., CPUs and memory) via a network. The algorithm to parallelize CCD++ on a distributed system is similar to the multi-core version of parallel

Algorithm 2.3 Parallel CCD++ on multi-core systems

Input: A, W, H, λ, k, T

Initialize $W = 0$ and $R = A$.
for $iter = 1, 2, \ldots,$ **do**
  for $t = 1, 2, \ldots, k$ **do**
    Parallel: core r constructs $\hat{R}$ using (2.23).
    for $inneriter = 1, 2, \ldots, T$ **do**
      Parallel: core r updates $\boldsymbol{u}^r$ using (2.24).
      Parallel: core r updates $\boldsymbol{v}^r$ using (2.25).
    Parallel: core r updates $\bar{\boldsymbol{w}}_t^r$ and $\bar{\boldsymbol{h}}_t^r$ using (2.26).
    Parallel: core r updates R using (2.27).

CCD++ introduced in Algorithm 2.3. The common idea is to enable each machine/core to solve subproblem (2.17) and update a subset of variables and residual in parallel.

When W and H are too large to fit in memory of a single machine, we have to divide them into smaller components and distribute them to different machines. There are many ways to divide W and H. In the distributed version of parallel CCD++, assuming that the distributed system is composed of p machines, we consider p-way row partitions for W and H: $S = \{S_1, \ldots, S_p\}$ is a partition of the row indices of W; $G = \{G_1, \ldots, G_p\}$ is a partition of the row indices of H. We further denote the sub-matrices corresponding to S_r and G_r by W^r and H^r, respectively. In the distributed version of CCD++, machine r is responsible for the storage and the update of W^r and H^r. Note that the dynamic approach to assign jobs in Section 2.3.1 cannot be applied here because not all variables and ratings are available on all machines. Partitions S and G should be determined prior to any computation.

Algorithm 2.4 Parallel CCD++ on distributed systems

Input: A, W, H, λ, k, T

Initialize $W = 0$ and $R = A$.
for $iter = 1, 2, \ldots$ **do**
  for $t = 1, 2, \ldots, k$ **do**
    Broadcast: machine r broadcasts $\bar{\boldsymbol{w}}_t^r$ and $\bar{\boldsymbol{h}}_t^r$.
    Parallel: machine r constructs $\hat{R}$ using (2.28).
    for $inneriter = 1, 2, \ldots T$ **do**
      Parallel: machine r updates $\boldsymbol{u}^r$ using (2.24).
      Broadcast: machine r broadcasts $\boldsymbol{u}^r$.
      Parallel: machine r updates $\boldsymbol{v}^r$ using (2.25).
      Broadcast: machine r broadcasts $\boldsymbol{v}^r$.
    Parallel: machine r updates $\bar{\boldsymbol{w}}_t^r$, $\bar{\boldsymbol{h}}_t^r$ using (2.26).
    Parallel: machine r updates R using (2.29).

Typically, the memory required to store the residual R is much larger than for W and H, thus we should avoid communication of R. Here we describe an arrangement of R on a distributed system such that all updates in CCD++ can be done without any communication of the residual. As mentioned above, machine r is in charge of updating variables in W^r and H^r. From the update rules of CCD++, we can see that values R_{ij}, $\forall (i,j) \in \Omega_{S_r}$, are required to update variables in W^r, while values R_{ij}, $\forall (i,j) \in \bar{\Omega}_{G_r}$, are required to update H^r, where $\Omega_{S_r} = \bigcup_{i \in S_r} \{(i,j) : j \in \Omega_i\}$, and $\bar{\Omega}_{G_r} = \bigcup_{j \in G_r} \{(i,j) : i \in \bar{\Omega}_j\}$. Thus, the following entries of R should be easily accessible from machine r:

$$\Omega^r = \Omega_{S_r} \cup \bar{\Omega}_{G_r} = \{(i,j) : i \in S_r \text{ or } j \in G_r\}.$$

Thus, only entries R_{ij}, $\forall (i,j) \in \Omega^r$, are stored in machine r. Specifically, entries corresponding to Ω_{S_r} are stored in CRS format, and entries corresponding to $\bar{\Omega}_{G_r}$ are stored in CCS format. Thus, the entire R has two copies stored

on the distributed system. Assuming that the latest R_{ij}'s corresponding to Ω^r are available on machine r, the entire $\bar{\boldsymbol{w}}_t$ and $\bar{\boldsymbol{h}}_t$ are still required to construct the $\hat{R}$ in subproblem (2.17). As a result, we need to broadcast $\bar{\boldsymbol{w}}_t$ and $\bar{\boldsymbol{h}}_t$ in the distributed version of CCD++ such that a complete copy of the latest $\bar{\boldsymbol{w}}_t$ and $\bar{\boldsymbol{h}}_t$ is locally available on each machine to compute $\hat{R}$:

$$\hat{R}_{ij} \leftarrow R_{ij} + \bar{w}_{ti}\bar{h}_{tj} \ \forall (i,j) \in \Omega^r. \tag{2.28}$$

During T CCD iterations, machine r needs to broadcast the latest copy of $\boldsymbol{u}^r$ to other machines before updating $\boldsymbol{v}^r$ and broadcast the latest $\boldsymbol{v}^r$ before updating $\boldsymbol{u}^r$.

After T alternating iterations, each machine r has a complete copy of $(\boldsymbol{u}^*, \boldsymbol{v}^*)$, which can be used to update $(\bar{\boldsymbol{w}}_t^r, \bar{\boldsymbol{h}}_t^t)$ by (2.26). The residual R can also be updated without extra communication by

$$R_{ij} \leftarrow \hat{R}_{ij} + \bar{w}_{ti}\bar{h}_{tj} \ \forall (i,j) \in \Omega^r, \tag{2.29}$$

as $(\bar{\boldsymbol{w}}_t^r, \bar{\boldsymbol{h}}_t^r)$ is also locally available on each machine r.

The distributed version of CCD++ is described in Algorithm 2.4. In summary, in distributed CCD++, each machine r only stores W^r and H^r and residual matrices $R_{S_r:}$ and $R_{:G_r}$. In an ideal case, where $|S_r| = m/p, |G_r| = n/p, \sum_{i \in S_r} |\Omega_i| = |\Omega|/p$, and $\sum_{j \in G_r} |\bar{\Omega}_j| = |\Omega|/p$, the memory consumption on each machine is mk/p variables of W, nk/p variables of H, and $2|\Omega|/p$ entries of R. As all communication in Algorithm follows the same scenario: each machine r broadcasts the $|S_r|$ (or $|G_r|$) local variables to other machines and gathers

the remaining $m - |S_r|$ (or $n - |G_r|$) latest variables from other machines. Such communication can be achieved efficiently by an AllGather operation, which is a collective operation defined in the Message Passing Interface (MPI) standard.[10] With a recursive-doubling algorithm, AllGather operations can be done in

$$\alpha \log p + \frac{p-1}{p} M\beta, \tag{2.30}$$

where M is the message size in bytes, α is the startup time per message, independent of the message size, and β is transfer time per byte [118]. Based on Eq. (2.30), the total communication time of Algorithm 2.4 per iteration is

$$\left(\alpha \log p + \frac{8(m+n)(p-1)\beta}{p}\right) k(T+1),$$

where we assume that each entry of W and H is a double-precision floating-point number.

2.3.3 Scalability Analysis of Other Methods

As mentioned in Section 2.1.1, ALS can be easily parallelized when entire W and H can fit in the main memory of one computer. However, it is hard to be scaled up to very large-scale recommender systems when W or H cannot fit in the memory of a single machine. When ALS updates $\boldsymbol{w}_i$, H_{Ω_i} is required to compute the Hessian matrix $(H_{\Omega_i}^\top H_{\Omega_i} + \lambda I)$ in Eq. (2.3). In parallel ALS, even though each machine only updates a subset of rows of W or H at a time, [149] proposes that each machine should gather the entire

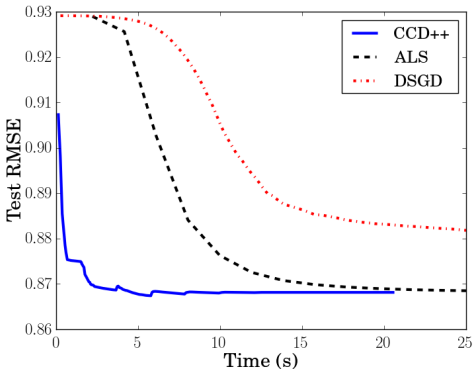
[10] http://www.mcs.anl.gov/research/projects/mpi/

latest H or W before the updates. However, when W or H is beyond the memory capacity of a single machine, it is not feasible to gather entire W or H and store them in the memory before the updates. Thus, each time when some rows of H or W are not available locally but are required to form the Hessian, the machine has to initiate communication with other machines to fetch those rows from them. Such complicated communication could severely reduce the efficiency of ALS. Furthermore, the higher time complexity per iteration of ALS is unfavorable when dealing with large W and H. Thus, ALS is not scalable to handle recommender systems with very large W and H.

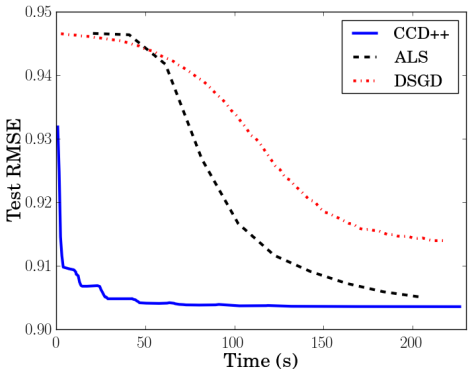
Recently, [37] proposed a distributed SGD approach, DSGD, which partitions A into blocks and conducts SGD updates with a particular ordering. Similar to our approach, DSGD stores W, H, and A in a distributed manner such that each machine only needs to store $(n+m)k/p$ variables and $|\Omega|/p$ rating entries. Each communication scenario in DSGD is that each machine sends m/p (or n/p) variables to a particular machine, which can be done by a SendReceive operation. As a result, the communication time per iteration of DSGD is $\alpha p + 8mk\beta$. Thus, both DSGD and CCD++ can handle recommender systems with very large W and H.

2.4 Experimental Results

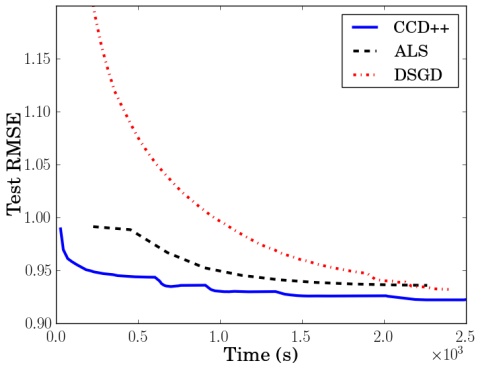
In this section, we compare CCD++, ALS, and SGD in large-scale datasets under serial, multi-core and distributed platforms. For CCD++, we use the implementation with our adaptive technique based on function value



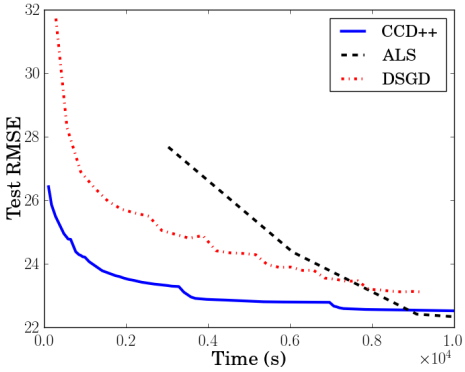
(a) movielens1m: Time versus RMSE



(b) movielens10m: Time versus RMSE

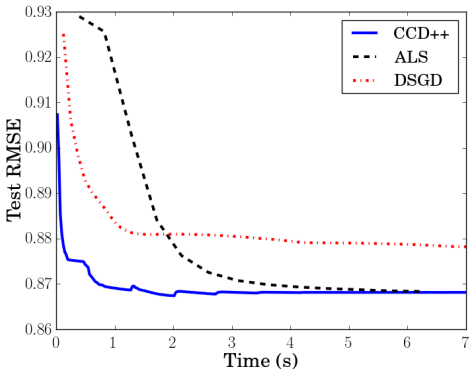


(c) netflix: Time versus RMSE

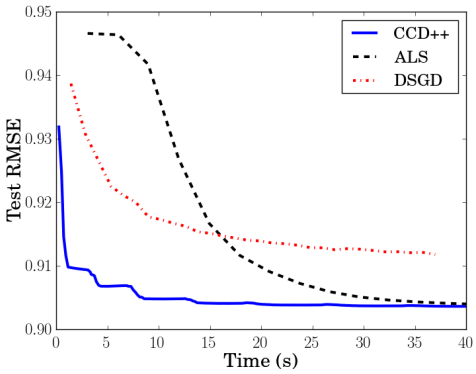


(d) yahoo-music: Time versus RMSE

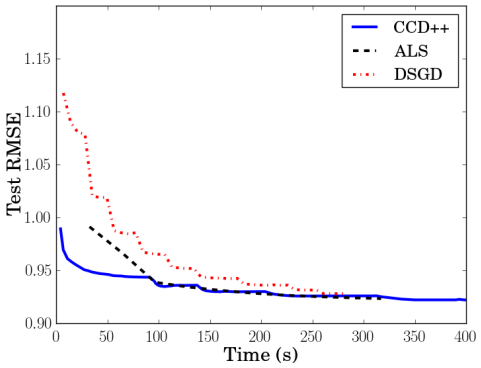
Figure 2.2: RMSE versus computation time on a serial setting for different methods (time is in seconds). Due to non-convexity of the problem, different methods may converge to different values.



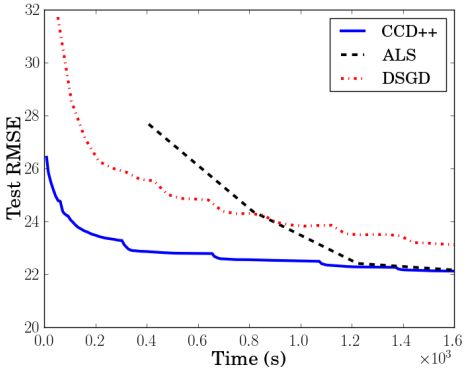
(a) movielens1m: Time versus RMSE



(b) movielens10m: Time versus RMSE



(c) netflix: Time versus RMSE



(d) yahoo-music: Time versus RMSE

Figure 2.3: RMSE versus computation time on an 8-core system for different methods (time is in seconds). Due to non-convexity of the problem, different methods may converge to different values.

Table 2.2: The statistics and parameters for each dataset

	m	n	$\lvert\Omega\rvert$	$\lvert\Omega^{\text{test}}\rvert$	k	λ
movielens1m	6,040	3,952	900,189	100,020	40	0.1
movielens10m	71,567	65,133	9,301,274	698,780	40	0.1
netflix	2,649,429	17,770	99,072,112	1,408,395	40	0.05
yahoo-music	1,000,990	624,961	252,800,275	4,003,960	100	1
synthetic-u	3,000,000	3,000,000	8,999,991,830	90,001,535	10	0.001
synthetic-p	20,000,000	1,000,000	14,661,239,286	105,754,418	30	0.001

reduction. We implement ALS with the Intel Math Kernel Library.[11] Based on the observation in Section 2.1, we choose DSGD as an example of the parallel SGD methods because of its faster and more stable convergence than other variants. In this paper, all algorithms are implemented in C++ to make a fair comparison. Similar to [149], all of our implementations use the weighted λ-regularization.[12]

Datasets. We consider four public datasets for the experiment: movielens1m, movielens10m, netflix, and yahoo-music. These datasets are extensively used in the literature to test the performance of matrix factorization algorithms [37, 65, 75]. The original training/test split is used for reproducibility.

To conduct experiments in a distributed environment, we follow the procedure used to create the Jumbo dataset in [87] to generate the synthetic-u dataset, a 3M by 3M sparse matrix with rank 10. We first build the ground truth W and H with each variable uniformly distributed over the interval $[0, 1)$.

[11] Our C implementation is 6x faster than the MATLAB version provided by [149].

[12] $\lambda\left(\sum_i |\Omega_i| \|\boldsymbol{w}_i\|^2 + \sum_j |\bar{\Omega}_j| \|\boldsymbol{h}_j\|^2\right)$ is used to replace the regularization term in (2.1).

We then sample about 9 billion entries uniformly at random from $WH^\top$ and add a small amount of noise to obtain our training set. We sample about 90 million other entries without noise as the test set.

Since the observed entries in real-world datasets usually follow power-law distributions, we further construct a dataset synthetic-p with the unbalanced size 20M by 1M and rank 30. The power-law distributed observed set Ω is generated using the Chung-Lu-Vu (CLV) model proposed in [30]. More specifically, we first sample the degree sequence $a_1, \cdots, a_m$ for all the rows following the power-law distribution $p(x) \propto x^{-c}$ with $c = -1.316$ (the parameter c is selected to control the number of nonzeros). We then generate another degree sequence $b_1, \cdots, b_n$ for all the columns by the same power law distribution and normalize it to ensure $\sum_{j=1}^{n} b_j = \sum_{i=1}^{m} a_i$. Finally, each edge (i, j) is sampled with probability $\frac{a_i b_j}{\sum_k b_k}$. The values of the observed entries are generated in the same way as in synthetic-u. For training/test split, we randomly select about 1% observed entries as test set and the rest observed entries as the training set.

For each dataset, the regularization parameter λ is chosen from $\{1, 0.5, 0.1, 0.05, 0.01, 0.005, 0.001\}$ with the lowest test RMSE. The parameter k of both synthetic datasets are set according to the ground truth, and for real datasets we choose k from $\{20, 40, 60, 80, 100\}$ with the lowest test RMSE. See Table 2.2 for more information about the statistics and parameters used for each dataset.

2.4.1 Experiments on a Single Machine Serial Setting

We first compare CCD++ with ALS and DSGD in a serial setting.

Experimental platform. As mentioned in Section 2.1.3, we use an 8-core Intel Xeon X5570 processor with 32KB L1-cache, 256KB L2-cache, 8MB L3-cache, and enough memory for the comparison. We only use 1 core for the serial setting in this section, while we will use multiple cores in the multi-core experiments (Section 2.4.2).

Results on training time. Figure 2.2 shows the comparison of the running time versus RMSE for the four real-world datasets in a serial setting, and we observe that CCD++ is faster than ALS and DSGD.

2.4.2 Experiments on a Multi-core Environment

In this section, we compare the multi-core version of CCD++ with other methods on a multi-core shared-memory environment.

Experimental platform. We use the same environment as in Section 2.4.1. The processor has 8 cores, and the OpenMP library is used for multi-core parallelization.

Results on training time. We ensure that eight cores are fully utilized for each method. Figure 2.3 shows the comparison of the running time versus RMSE for the four real-world datasets. We observe that the performance of CCD++ is generally better than parallel ALS and DSGD for each dataset.

Results on speedup. Another important measurement in parallel computing is the speedup – how much faster a parallel algorithm is when we increase the number of cores. To test the speedup, we run each parallel method on yahoo-music with various numbers of cores, from 1 to 8, and measure the running time for one iteration. Although we have shown in Section 2.1.3 that with regard to convergence DSGD has better performance than HogWild, it remains interesting to see how HogWild performs in terms of speedup. Thus, we also include HogWild into the comparison. The results are shown in Figure 2.4. Based on the slope of the curves, we observe that CCD++ and ALS have better speedup than both SGD approaches (DSGD and HogWild). This can be explained by the cache-miss rate for each method. Due to the fact that CCD++ and ALS access variables in contiguous memory spaces, both of them enjoy better locality. In contrast, due to the randomness, two consecutive updates in SGD usually access non-contiguous variables in W and H, which increases the cache-miss rate. Given the fixed size of the cache, time spent in loading data from memory to cache becomes the bottleneck for DSGD and HogWild to achieve better speedup when the number of cores increases.

2.4.3 Experiments on a Distributed Environment

In this section, we conduct experiments to show that distributed CCD++ is faster than DSGD and ALS for handling large-scale data on a distributed system.

Experimental platform. The following experiments are conducted on a large-scale parallel platform at the Texas Advanced Computing Center (TACC), Stampede[13]. Each computing node in Stampede is an Intel Xeon E5-2680 2.7GHz CPU machine with 32 GB memory and communicates by FDR 56 Gbit/s cable. For a fair comparison, we implement a distributed version with MPI in C++ for all the methods. The reason we do not use Hadoop is that almost all operations in Hadoop need to access data and variables from disks, which is quite slow and thus not suitable for iterative methods. It is reported in [76] that ALS implemented with MPI is 40 to 60 times faster than its Hadoop implementation in the Mahout project. We also tried to run the ALS code provided as part of the GraphLab library[14] but in our experiments the GraphLab code (which has an asynchronous implementation of ALS) did not converge. Hence, we developed our own implementation of ALS, using which we report all ALS results.

Results on yahoo-music. First we show comparisons on the yahoo-music dataset, which is the largest real-world dataset we used in this paper. Figure 2.5 shows the result with 4 computing nodes – we can make similar observations as in Figure 2.3.

Results on synthetic datasets. When data is large enough, the benefit of distributed environments is obvious.

[13] http://www.tacc.utexas.edu/user-services/user-guides/stampede-user-guide#compenv

[14] We downloaded version 2.1.4679 from https://code.google.com/p/graphlabapi/

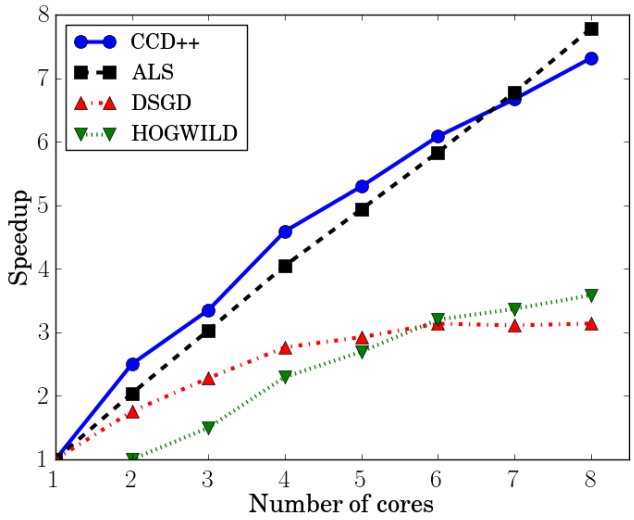


Figure 2.4: Speedup comparison among four algorithms with the yahoo-music dataset on a shared-memory multi-core machine. CCD++ and ALS have better speedups than DSGD and HogWild because of better locality.

For the scalability comparison, we vary the number of computing nodes, ranging from 32 to 256, and compare the time and speedup for three algorithms on the synthetic-u and synthetic-p datasets. As discussed in Section 2.3, ALS requires larger memory on each machine. In our setting it requires more than 32GB memory when using 32 nodes on synthetic-p dataset, so we run each algorithm with at least 64 nodes for this dataset. Here we calculate the training time as the time taken to achieve 0.01 test RMSE on synthetic-u and 0.02 test RMSE on synthetic-p respectively. The results are shown in Figure 2.6(a) and 2.7(a). We can see clearly that CCD++ is more than 8 times faster than both DSGD and ALS on synthetic-u and synthetic-p datasets with the number of computing nodes varying from 32 to 256. We also show the speedup of ALS,

DSGD, and CCD++ on both datasets in Figure 2.6(b) and 2.7(b). Note that since the data cannot be loaded in memory of a single machine, the speedup using p machines is T_p/T_{32} on synthetic-u and T_p/T_{64} on synthetic-p respectively, where T_p is the time taken on p machines. We observe that DSGD achieves super linear speedup on both datasets. For example, on synthetic-u dataset, the training time for DSGD is 2768 seconds using 32 machines and 218 seconds using 256 machines, so it achieves $2768/218 \approx 12.7$ times speedup with only 8 times the number of machines. This super linear speedup is due to the caching effect. In DSGD, each machine stores one block of W and one block of H. When the number of machines is large enough, these blocks can fit into the L2-cache, which leads to dramatic reduction in the memory access time. On the other hand, when the number of machines is not large enough, these blocks cannot fit into cache. Thus DSGD, which accesses entries in the block at random, suffers from frequent cache misses. In contrast, for CCD++ and ALS, the cache miss is not that severe even when the block of W and H cannot fit into cache since the memory is accessed sequentially in both methods.

Though the speedups are smaller than in a multi-core setting, CCD++ takes the least time to achieve the desired RMSE. This shows that CCD++ is not only fast but also scalable for large-scale matrix factorization on distributed systems.

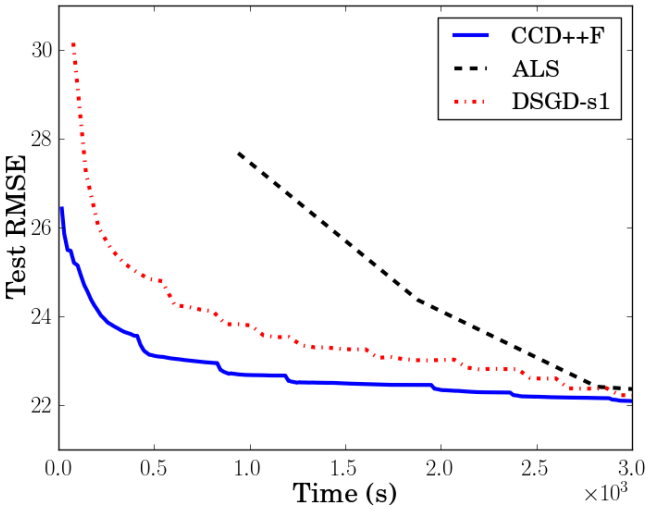
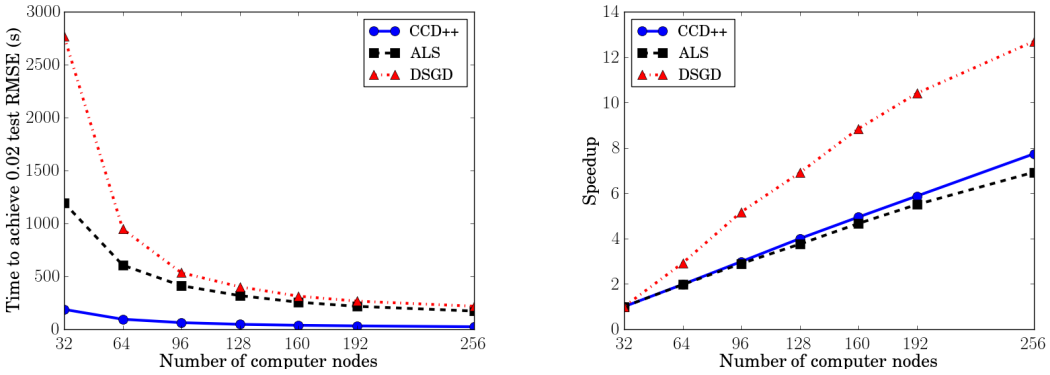


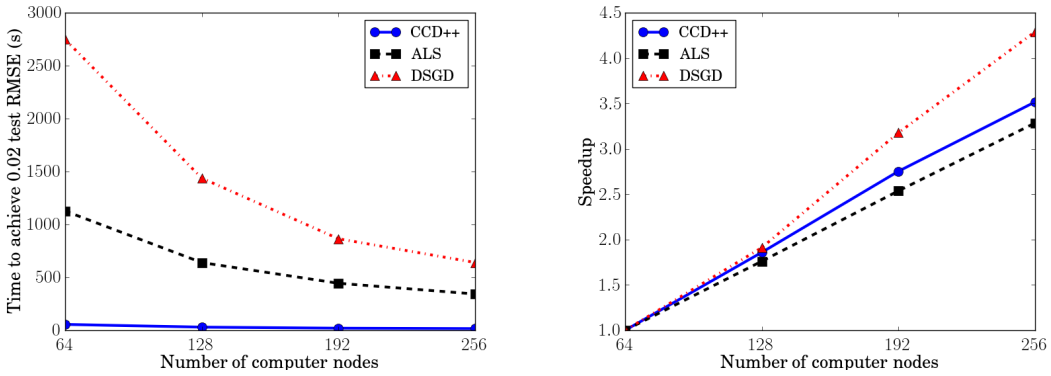
Figure 2.5: Comparison among CCD++, ALS, and DSGD with the yahoo-music dataset on a MPI distributed system with 4 computing nodes.



(a) Number of computation nodes versus training time

(b) Number of computation nodes versus speedup

Figure 2.6: Comparison among CCD++, ALS and DSGD on the synthetic-u dataset (9 billion ratings) on a MPI distributed system with varying number of computing nodes. The vertical axis in the left panel is the time for each method to achieve 0.01 test RMSE, while the right panel shows the speedup for each method. Note that, as discussed in Section 2.4.3, speedup is T_p/T_{32}, where T_p is the time taken on p machines.



(a) Number of computation nodes versus training time

(b) Number of computation nodes versus speedup

Figure 2.7: Comparison among CCD++, ALS and DSGD on the synthetic-p dataset (14.6 billion ratings) on a MPI distributed system with varying number of computing nodes. The vertical axis in the left panel is the time for each method to achieve 0.02 test RMSE, while the right panel shows the speedup for each method. Note that, as discussed in Section 2.4.3, speedup is T_p/T_{64}, where T_p is the time taken on p machines.

2.5 Summary of the Contributions

In this chapter, we have shown that the coordinate descent method is efficient and scalable for solving large-scale matrix factorization problems in recommender systems. The proposed method CCD++ not only has lower time complexity per iteration than ALS, but also achieves faster and more stable convergence than SGD in practice. We also explore different update sequences and show that the feature-wise update sequence (CCD++) gives better performance. Moreover, we show that CCD++ can be easily parallelized in both multi-core and distributed environments and thus can handle large-scale datasets where both ratings and variables cannot fit in the memory of a single machine. Empirical results demonstrate the superiority of CCD++ under both parallel environments. For instance, running with a large-scale synthetic dataset (14.6 billion ratings) on a distributed memory cluster, CCD++ is 49 times faster to achieve the desired test accuracy than DSGD when we use 64 processors, and when we use 256 processors, CCD++ is 40 times faster than DSGD and 20 times faster than ALS.

Chapter 3

F+Nomad-LDA: An Asynchronous Distributed Framework for Topic Modeling

Topic models provide a way to aggregate vocabulary from a document corpus to form latent "topics." In particular, Latent Dirichlet Allocation (LDA) [15] is one of the most popular topic modeling approaches. Learning meaningful topic models with massive document collections which contain millions of documents and billions of tokens is challenging because of two reasons. First, one needs to deal with a large number of topics (typically on the order of thousands). Second, one needs a scalable and efficient way of distributing the computation across multiple machines.

Unsurprisingly, there have been significant attempts at developing scalable inference algorithms for LDA. To tackle large number of topics, [134] proposed a clever sparse sampling trick that is widely used in packages like MALLET and Yahoo! LDA. More recently, [68] proposed using the Alias table method to speed up sampling from the multinomial distribution. At the same

The materials in this Chapter have been published in [140]. I developed the algorithms and conducted the experiments.

time, there has also been significant effort towards distributing the computation across multiple processors. Some early efforts in this direction include [133] and [55], where the basic idea is to partition the documents across processors. During each inner iteration the words in the vocabulary are partitioned across processors and each processor only updates the latent variables associated with the subset of documents and words that it owns. After each inner iteration, a synchronization step is used to update global counts and to re-partition the words across processors. In fact, a very similar idea was independently discovered in the context of matrix completion by [37] and [100]. However, in the case of LDA we need to keep a global count synchronized across processors which significantly complicates matters as compared to matrix completion. Arguably, most of the recent efforts towards scalable LDA such as [84, 110] have been focused on the global count issue either implicitly or explicitly. Recently there is also a growing trend in machine learning towards asynchronous algorithms which avoid bulk synchronization after every iteration. For example, in the context of LDA see the work of [5], and in the more general machine learning context see *e.g.*, [39, 71].

In this chapter, we propose a new asynchronous distributed topic modeling algorithm called F+Nomad LDA which simultaneously tackles the twin problems of large number of documents and large number of topics. In order to handle large number of topics we use an appropriately modified Fenwick tree. This data structure allows us to sample from a multinomial distribution over T items in $O(\log T)$ time. Moreover, when topic counts change, the data

structure can be updated in $O(\log T)$ time. In order to distribute the computation across multiple processors, we present a novel asynchronous framework inspired by the Nomad algorithm of [145]. While we believe that our framework can handle variable update schedules of many different methods, in this chapter we will primarily focus on Collapsed Gibbs Sampling (CGS). Our technical contributions can be summarized as follows:

- We identify the following key property of various inference methods for topic modeling: only a single vector of size k needs to be synchronized across multiple processors.
- We present a variant of the Fenwick tree which allows us to efficiently encode a multinomial distribution using $O(T)$ space. Sampling can be performed in $O(\log T)$ time and maintaining the data structure requires only $O(\log T)$ work.
- F+Nomad LDA: we propose a novel parallel framework for various types of inference methods for topic modeling. Our framework utilizes the concept of nomadic tokens to avoid locking and conflict at the same time. Our parallel approach is fully asynchronous with non-blocking communication, thus leading to good

  speedups. Moreover, our approach minimizes the staleness of the variables (at most k variables can be stale) for distributed parallel computation.
- We demonstrate the scalability of our methods by performing extensive empirical evaluation on large datasets which contain millions of docu-

ments and **billions** of words.

3.1 Notation and Background

We begin by very briefly reviewing Latent Dirichlet Allocation (LDA) [15]. Suppose we are given I documents denoted as $d_1, d_2, \ldots, d_I$, and let J denote the number of words in the vocabulary. Moreover, let n_i denote the number of words in document d_i. Let w_j denote the j-th word in the vocabulary and $w_{i,j}$ denote the j-th word in the i-th document. Assume that the documents are generated by sampling from T topics denoted as $\boldsymbol{\phi}_1, \boldsymbol{\phi}_2, \ldots, \boldsymbol{\phi}_T$; a topic is simply a J dimensional multinomial distribution over words. Each document includes some proportion of the topics. These proportions are latent, and we use the T dimensional probability vector $\boldsymbol{\theta}_i$ to denote the topic distribution for a document d_i. Moreover, let $z_{i,j}$ denote the latent topic from which $w_{i,j}$ was drawn. Let α and β be hyper parameters of the Dirichlet distribution. The generative process for LDA can be described as follows:

1 Draw T topics $\boldsymbol{\phi}_k \sim \texttt{Dirichlet}(\beta)$, $k = 1, \ldots, T$.

2 For each document $d_i \in \{d_1, d_2, \ldots, d_I\}$:

- Draw $\boldsymbol{\theta}_i \sim \texttt{Dirichlet}(\alpha)$.
- For $j = 1, \ldots, n_i$
  - Draw $z_{i,j} \sim \texttt{discrete}(\boldsymbol{\theta}_i)$.
  - Draw $w_{i,j} \sim \texttt{discrete}(\boldsymbol{\phi}_{z_{i,j}})$.

3.1.1 Inference

The inference task for LDA is to characterize the posterior distribution $Pr(\boldsymbol{\phi}_i, \boldsymbol{\theta}_i, z_{i,j} \mid w_{i,j})$. In the Bayesian setting, we want an efficient way to draw samples from this posterior distribution. Collapsed Gibbs Sampling (CGS) [42] is a popular inference scheme for LDA. Define

$$n_{z,i,w} := \sum_{j=1}^{n_i} I(z_{i,j} = z \text{ and } w_{i,j} = w), \tag{3.1}$$

$n_{z,i,*} = \sum_w n_{z,i,w}$, $n_{z,*,w} = \sum_i n_{z,i,w}$, and $n_{z,*,*} = \sum_{i,w} n_{z,i,w}$, where $I(\cdot)$ is the indicator function. The update rule for CGS can be written as follows

1 Decrease $n_{z_{i,j},i,*}$, $n_{z_{i,j},*,w_{i,j}}$, and $n_{z_{i,j},*,*}$ by 1.

2 Resample $z_{i,j}$ according to

$$\mathbb{P}(z_{i,j}|w_{i,j}, \alpha, \beta) \propto \frac{\left(n_{z_{i,j},i,*} + \alpha_{z_{i,j}}\right)\left(n_{z_{i,j},*,w_{i,j}} + \beta_{w_{i,j}}\right)}{n_{z_{i,j},*,*} + \sum_{j=1}^{J} \beta_j}. \tag{3.2}$$

3 Increase $n_{z_{i,j},i,*}$, $n_{z_{i,j},*,w_{i,j}}$, and $n_{z_{i,j},*,*}$ by 1.

Although in this chapter we will focus on CGS, note that there are many other inference techniques for LDA such as collapsed variational Bayes, stochastic variational Bayes, or expectation maximization which essentially follow a very similar update pattern [6]. We believe that the parallel framework proposed in this chapter will apply to this wider class of inference techniques as well.

3.1.2 Review of Multinomial Sampling

Given a T-dimensional discrete distribution characterized by unnormalized parameters $\boldsymbol{p}$ with $p_t \geq 0$ such as in (3.2), many sampling algorithms can be applied to draw a sample z such that $\mathbb{P}(z = t) \propto p_t$.

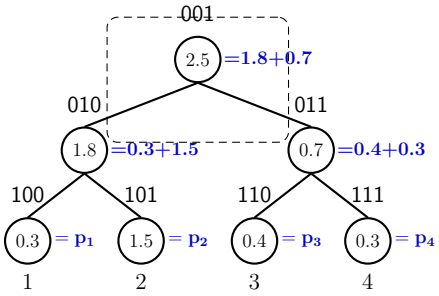
Table 3.1: Comparison of samplers for a T-dimensional multinomial distribution $\boldsymbol{p}$ described by unnormalized parameters $\{p_t : t = 1, \ldots, T\}$.

	Data Structure Space	Initialization Time	Initialization Space	Generation Time	Parameter Update Time
LSearch	$c_T = \boldsymbol{p}^\top \mathbf{1}$: $O(1)$	$O(T)$	$O(1)$	$O(T)$	$O(1)$
BSearch	$\boldsymbol{c} = \texttt{cumsum}(\boldsymbol{p})$: $O(T)$	$O(T)$	$O(1)$	$O(\log T)$	$O(T)$
Alias Method	$prob, alias$: $O(T)$	$O(T)$	$O(T)$	$O(1)$	$O(T)$
F+tree Sampling	$\texttt{F.initialize}(\boldsymbol{p})$: $O(T)$	$O(T)$	$O(1)$	$O(\log T)$	$O(\log T)$

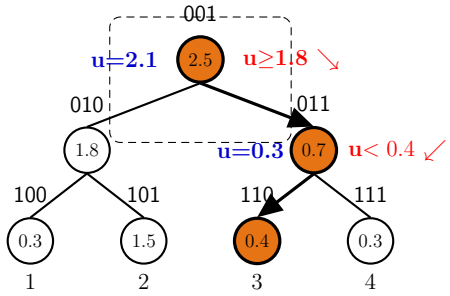
- LSearch: Linear search on $\boldsymbol{p}$. **Initialization:** Compute the normalization constant $c_T = \sum_t p_t$. **Generation:** First generate $u = \texttt{uniform}(c_T)$, a uniform random number in $[0, c_T)$, and perform a linear search to find $z = \min\{t : (\sum_{s \le t} p_s) > u\}$.
- BSearch: Binary search on $\boldsymbol{c} = \texttt{cumsum}(\boldsymbol{p})$. **Initialization:** Compute $\boldsymbol{c} = \texttt{cumsum}(\boldsymbol{p})$ such that $c_t = \sum_{s:s \le t} p_s$. **Generation:** First generate the cumulated sum $u = \texttt{uniform}(c_T)$ and perform a binary search on $\boldsymbol{c}$ to find $z = \min\{t : c_t > u\}$.
- Alias method. **Initialization:** Construct an Alias table [121] for $\boldsymbol{p}$, which contains two arrays of length T: $alias$ and $prob$. See [120] for a linear time construction scheme. **Generation:** First generate $u = \texttt{uniform}(T)$, $j = \lfloor u \rfloor$, and

$$z = \begin{cases} j + 1 & \text{if } (u - j) \le prob[j + 1] \\ alias[j + 1] & \text{otherwise} \end{cases}.$$

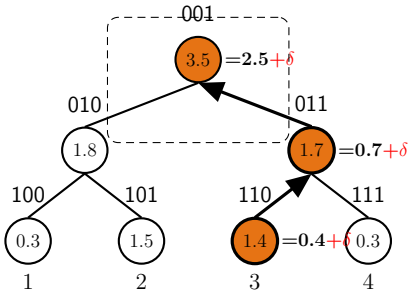
See Table 3.1 for a comparison of the time/space requirements of each of the above sampling methods.



(a) F+tree for $\boldsymbol{p} = [0.3, 1.5, 0.4, 0.3]^\top$



(b) Sampling



(c) Updating (with $\delta = 1.0$)

Figure 3.1: Illustration of sampling and updating using F+tree in logarithmic time.

3.2 F+LDA: A Logarithmic-Time Fenwick Tree Sampling

In this section, we first describe a binary tree structure F+tree for fast T-dimensional multinomial sampling. The initialization of an F+tree is linear in T and the cost to generate a sample is logarithmic in T. Furthermore, F+tree can also be maintained in logarithmic time for a single parameter update of p_t. We will explain how such properties of F+tree can be explored to significantly accelerate LDA sampling.

3.2.1 F+tree Sampling

F+tree, first introduced for weighted sampling without replacement [129], is a simplified and generalized version of Fenwick tree [36], which supports both efficient sampling and update procedures. In fact, Fenwick tree can be regarded as a compressed version of the F+tree studied in this chapter. For simplicity, we assume T is a power of 2. F+tree is a complete binary tree with $2T - 1$ nodes for a given $\boldsymbol{p}$, where

- each leaf node corresponds to a dimension t and stores p_t as its value, and
- each internal node stores the sum of the values of all of its leaf descendants, or equivalently the sum of values of its two children due to binary tree structure.

See Figure 3.1(a) for an example with $\boldsymbol{p} = [0.3, 1.5, 0.4, 0.3]$ and $T = 4$. Nodes in the dotted rectangle are internal nodes. Similar to the representation used

in a heap [33], an array $\texttt{F}$ of length $2T$ can be used to represent the F+tree structure. Let i be the index of each node, and $\texttt{F}[i]$ be the value stored in the i-th node. The index of the left child, right child, and parent of the i-th node is $2i$, $2i+1$, and $\lfloor i/2 \rfloor$, respectively. The 0/1 string along each node in Figure 3.1 is the binary number representation of the node index.

Initialization. By the definition of F+tree, given $\boldsymbol{p}$, the values of $\texttt{F}$ be defined as follows:

$$\texttt{F}[i] = \begin{cases} p_{i-T+1} & \text{if } i \geq T, \\ \texttt{F}[2i] + \texttt{F}[2i+1] & \text{if } i < T. \end{cases} \tag{3.3}$$

Thus, $\texttt{F}$ can be constructed in $O(T)$ by initializing elements using (3.3) in reverse. Unlike the Alias method, there is no extra space required in the F+tree initialization in addition to $\texttt{F}$.

Sample Generation. Sampling using a F+tree can be carried out as a simple top-down traversal procedure to locate $z = \min\{t : (\sum_{s:s\leq t} p_s) > u\}$ for a number uniformly sampled between $[0, \sum_t p_t)$. Note that $\sum_t p_t$ is stored in $\texttt{F}[1]$, which can be directly used to generate $u = \texttt{uniform}(\texttt{F}[1])$. Let $\texttt{leaves}(i)$ be the set of all leaf descendant of the i-th node. We can consider a general recursive step in the traversal with the current node i and $u \in [0, \texttt{F}[i])$. The definition of F+tree guarantees that

$$u \geq \texttt{F}[i.\texttt{left}] \Rightarrow z \in \texttt{leaves}(i.\texttt{right}),$$

$$u < \texttt{F}[i.\texttt{left}] \Rightarrow z \in \texttt{leaves}(i.\texttt{left}),$$

This provides a guideline to determine which child to go next. If right child is chosen, $\texttt{F}[i.\texttt{left}]$ should be subtracted from u to ensure $u \in [0, \texttt{F}[i.\texttt{right}])$.

Note that as half of the possible t values are removed from the set of candidates, it is clear that this sampling procedure costs only $O(\log T)$ time. The detailed procedure, denoted by $\texttt{F.sample}(u)$, is described in Algorithm 3.1. A toy example with $u = 2.1$ is illustrated in Figure 3.1(b).

Algorithm 3.1 Logarithmic time sampling: $\texttt{F.sample}(u)$.

Input: $\texttt{F}$: an F+tree for $\boldsymbol{p}$, $u = \texttt{uniform}(\texttt{F}[1])$.
Output: $z = \min\{t : (\sum_{s \le t} p_s) > u\}$

- $i \leftarrow 1$
- While i is not a leaf
  - If $u \ge \texttt{F}[i.\texttt{left}]$,
    - $u \leftarrow u - \texttt{F}[i.\texttt{left}]$
    - $i \leftarrow i.\texttt{right}$
  - Else
    - $i \leftarrow i.\texttt{left}$
- $z \leftarrow i - T + 1$

Algorithm 3.2 Logarithmic time F+tree maintenance for a single parameter update: $\texttt{F.update}(t, \delta)$

Input: a F+tree $\texttt{F}$ for $\boldsymbol{p}$, t, δ.
Output: F+tree $\texttt{F}$ is updated for $\bar{\boldsymbol{p}} \equiv \boldsymbol{p} + \delta \boldsymbol{e}_t$

- $i \leftarrow \texttt{leaf}[t]$
- While i is a valid node
  - $\texttt{F}[i] = \texttt{F}[i] + \delta$
  - $i \leftarrow i.\texttt{parent}$

Maintenance for Parameter Updates. A simple and efficient maintenance routine to deal with slight changes on the multinomial parameters $\boldsymbol{p}$ can be very useful in CGS for LDA (See details in Section 3.2.2). F+tree structure supports a logarithmic time maintenance routine for a single ele-

ment change on $\boldsymbol{p}$. Assume the t-th component is updated by δ:

$$\bar{\boldsymbol{p}} \leftarrow \boldsymbol{p} + \delta \boldsymbol{e}_t,$$

where $\boldsymbol{e}_t$ is the t-th column of the identity matrix of order T. A simple bottom-up update procedure to modify a F+tree $\texttt{F}$ for the current $\boldsymbol{p}$ to a F+tree for $\bar{\boldsymbol{p}}$ can be carried out as follows. Let $\texttt{leaf}[t]$ be the leaf node corresponding to t. For all the ancestors i of $\texttt{leaf}[t]$ (self included), perform the following delta update:

$$\texttt{F}[i] = \texttt{F}[i] + \delta.$$

See Figure 3.1(c) for an illustration with $t = 3$ and $\delta = 1.0$. The detailed procedure, denoted by $\texttt{F.update}(t, \delta)$, is described in Algorithm 3.2. The maintenance cost is linear to the depth of the F+tree, which is $O(\log T)$. Note that to deal with a similar change in $\boldsymbol{p}$, LSearch can update its normalization constant $c_T \leftarrow c_T + \delta$ in a constant time, while both BSearch and Alias method require to re-construct the entire data structure (either $\boldsymbol{c} = \texttt{cumsum}(\boldsymbol{p})$ or the Alias table: *alias* and *prob*), which costs $O(T)$ time in general.

See Table 3.1 for a summary of the complexity analysis for each multinomial sampling approach. Clearly, LSearch has the smallest update cost but the largest generation cost, and Alias method has the best generation cost but the worst maintenance cost. In contrast, F+tree sampling has a logarithmic time procedure for both operations.

Table 3.2: Comparison of various sampling methods for LDA. We use $\#MH$ to denote number of Metropolis-Hasting steps for Alias LDA. Note that in this table only the time complexity is presented—there are some hidden constants that also play important roles in practice. For example, the initialization cost of the Alias table is much more than linear search although they have the same time complexity.

	F+LDA		F+LDA		Sparse-LDA			Alias-LDA	
Sample Sequence	Word-by-Word		Doc-by-Doc		Doc-by-Doc			Doc-by-Doc	
Exact Sampling	Yes		Yes		Yes			No	
Decomposition	$\alpha\left(\frac{n_{tw}+\beta}{n_t+\bar{\beta}}\right)$	$+n_{td}\left(\frac{n_{tw}+\beta}{n_t+\bar{\beta}}\right)$	$\beta\left(\frac{n_{td}+\alpha}{n_t+\bar{\beta}}\right)$	$+n_{tw}\left(\frac{n_{td}+\alpha}{n_t+\bar{\beta}}\right)$	$\frac{\alpha\beta}{n_t+\bar{\beta}}$	$+\beta\left(\frac{n_{td}}{n_t+\bar{\beta}}\right)$	$+n_{tw}\left(\frac{n_{td}+\alpha}{n_t+\bar{\beta}}\right)$	$\alpha\left(\frac{n_{tw}+\beta}{n_t+\bar{\beta}}\right)$	$+n_{td}\left(\frac{n_{tw}+\beta}{n_t+\bar{\beta}}\right)$
Sampling method	F+tree	BSearch	F+tree	BSearch	LSearch	LSearch	LSearch	Alias	Alias
Fresh samples	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes
Initialization	$O(\log T)$	$O(\lvert T_d\rvert)$	$O(\log T)$	$O(\lvert T_w\rvert)$	$O(1)$	$O(1)$	$O(\lvert T_w\rvert)$	$O(1)$	$O(\lvert T_d\rvert)$
Sampling	$O(\log T)$	$O(\log\lvert T_d\rvert)$	$O(\log T)$	$O(\log\lvert T_w\rvert)$	$O(T)$	$O(\lvert T_d\rvert)$	$O(\lvert T_w\rvert)$	$O(\#MH)$	$O(\#MH)$

Algorithm 3.3 F+LDA with word-by-word sampling

- `F.initialize`$(\boldsymbol{q})$, with $q_t = \frac{\beta}{n_t+\bar{\beta}}$
- For each word w
  - `F.update`$(t, n_{tw}/(n_t+\bar{\beta}))$ $\quad \forall t \in T_w$
  - For each occurrence of w, say $w_{i,j} = w$ in d_i
    - $t \leftarrow z_{i,j}$
    - Decrease n_t, n_{td_i}, n_{tw} by one
    - `F.update`(t, δ) with $\delta = \frac{n_{tw}+\beta}{n_t+\bar{\beta}} - \texttt{F}[\texttt{leaf}(t)]$
    - $\boldsymbol{c} \leftarrow$ `cumsum`$(\boldsymbol{r})$ (on T_w only)
    - $t \leftarrow$ `discrete`$(\boldsymbol{p}, \texttt{uniform}(\alpha\texttt{F}[1] + \boldsymbol{r}^\top \mathbf{1}))$ by (3.6)
    - Increase n_t, n_{td_i}, n_{tw} by one
    - `F.update`(t, δ) with $\delta = \frac{n_{tw}+\beta}{n_t+\bar{\beta}} - \texttt{F}[\texttt{leaf}(t)]$
    - $z_{i,j} \leftarrow t$
  - `F.update`$(t, -n_{tw}/(n_t+\bar{\beta}))$ $\quad \forall t \in T_w$

3.2.2 F+LDA = LDA with F+tree Sampling

In this section, we give details on applying F+tree sampling to CGS for LDA. Let us focus on a single CGS step in LDA with the current document id d_i, the current word w, and the current topic assignment t_{cur}. For simplicity of presentation, we further denote $n_{td} = n_{t,d_i,*}$, $n_{tw} = n_{t,*,w}$, and $n_t = n_{t,*,*}$ and assume $\alpha_t = \alpha, \forall t$, $\beta_j = \beta, \forall j$, and $\bar{\beta} = J \times \beta$. The multinomial parameter $\boldsymbol{p}$ of the CGS step in (3.2) can be decomposed into two terms as follows.

$$\begin{aligned} p_t &= \frac{(n_{td}+\alpha)(n_{tw}+\beta)}{n_t+\bar{\beta}}, \quad \forall t = 1, \ldots, T. \\ &= \beta\left(\frac{n_{td}+\alpha}{n_t+\bar{\beta}}\right) + n_{tw}\left(\frac{n_{td}+\alpha}{n_t+\bar{\beta}}\right). \end{aligned} \tag{3.4}$$

Let $\boldsymbol{q}$ and $\boldsymbol{r}$ be two vectors with $q_t = \frac{n_{td}+\alpha}{n_t+\bar{\beta}}$ and $r_t = n_{tw}q_t$. Some facts and implications about this decomposition:

(a) $\boldsymbol{p} = \beta\boldsymbol{q} + \boldsymbol{r}$. This leads to a simple two-level sampling for $\boldsymbol{p}$

$$\texttt{discrete}(\boldsymbol{p}, u) = \begin{cases} \texttt{discrete}(\boldsymbol{r}, u) & \text{if } u \leq \boldsymbol{r}^\top \mathbf{1}, \\ \texttt{discrete}(\boldsymbol{q}, \frac{u - \boldsymbol{r}^\top \mathbf{1}}{\beta}) & \text{otherwise,} \end{cases}$$

where $\mathbf{1}$ is the all-ones vector and $\boldsymbol{p}^\top \mathbf{1}$ denotes the normalization constant for $\boldsymbol{p}$, and $u = \texttt{uniform}(\boldsymbol{p}^\top \mathbf{1})$. This means that sampling for $\boldsymbol{p}$ can be very fast if $\boldsymbol{q}$ and $\boldsymbol{r}$ can be sampled efficiently.

(b) $\boldsymbol{q}$ is always dense but only two elements will be changed at each CGS step if we follow a document-by-document sampling sequence. Note $\boldsymbol{q}$ only depends on n_{td}. Decrement or increment of a single n_{td} only changes a single element of $\boldsymbol{q}$. We propose to apply F+tree sampling for $\boldsymbol{q}$ for its logarithmic time sampling and maintenance. At the beginning of CGS for LDA, a F+tree $\texttt{F}$ for $\boldsymbol{q}$ with $q_t = \frac{\alpha}{n_t + \beta}$ is constructed in $O(T)$ time. When CGS switches to a new document d_i, perform the following updates

$$\texttt{F.update}(t, \frac{n_{td}}{n_t + \bar{\beta}}) \quad \forall t \in T_d := \{t : n_{td} \neq 0\}.$$

When CGS finishes sampling for this document, we can perform

$$\texttt{F.update}(t, \frac{-n_{td}}{n_t + \bar{\beta}}) \; \forall t \in T_d.$$

Both updates can be done in $O(|T_d| \log T)$. As $|T_d|$ is upper bounded by the number of words in this document, the amortized sampling cost for each word in the document remains $O(\log T)$.

(c) $\boldsymbol{r}$ is T_w sparse, where $T_w := \{t : n_{tw} \neq 0\}$. Unlike $\boldsymbol{q}$, all the elements of $\boldsymbol{r}$ change when we switch from one word to another word in the same document. Moreover, $\boldsymbol{r}$ is only used once to compute $\boldsymbol{r}^\top \mathbf{1}$ and to generate at

most one sample. Thus, we propose to use BSearch approach to perform the sampling for $\boldsymbol{r}$. In particular, we only calculate the cumulative sum on nonzero elements in T_w. Thus, the initialization cost of BSearch is $O(|T_w|)$ and the sampling cost is $O(\log|T_w|)$.

Word-by-word CGS for LDA. Other than the traditional document-by-document CGS for LDA, we can also consider CGS using a word-by-word sampling sequence. For this sequence, we consider another decomposition of (3.4) as follows.

$$p_t = \alpha\left(\frac{n_{tw}+\beta}{n_t+\bar{\beta}}\right) + n_{td}\left(\frac{n_{tw}+\beta}{n_t+\bar{\beta}}\right), \quad \forall t. \tag{3.5}$$

For this decomposition (3.5), $\boldsymbol{q}$ and $\boldsymbol{r}$ have analogous definitions such that $q_t = \frac{n_{tw}+\beta}{n_t+\bar{\beta}}$ and $r_t = n_{td}q_t$, respectively. The corresponding three facts for (3.5) are as follows.

(a) $\boldsymbol{p} = \alpha\boldsymbol{q} + \boldsymbol{r}$. The two-level sampling for $\boldsymbol{p}$ is

$$\texttt{discrete}(\boldsymbol{p}, u) = \begin{cases} \texttt{discrete}(\boldsymbol{r}, u) & \text{if } u \le \boldsymbol{r}^\top \mathbf{1}, \\ \texttt{discrete}(\boldsymbol{q}, \frac{u-\boldsymbol{r}^\top\mathbf{1}}{\alpha}) & \text{otherwise,} \end{cases} \tag{3.6}$$

(b) $\boldsymbol{q}$ is always dense but only very few elements will be changed at each CGS step using word-by-word sampling sequence. A F+tree structure `F` is maintained for $\boldsymbol{q}$. The amortized update time for each occurrence of a word is $O(\log T)$ and the sampling generation for $\boldsymbol{q}$ using `F` also costs $O(\log T)$. Thus, $\texttt{discrete}(\boldsymbol{q}, u) := \texttt{F.sample}(u)$.

(c) $\boldsymbol{r}$ is a sparse vector with $|T_d|$ non-zeros. BSearch is used to construct $\boldsymbol{c} = \texttt{cumsum}(\boldsymbol{r})$ in $O(T_d)$ space and time. $\boldsymbol{c}$ is used to perform binary

search to generate a sample required by CGS for the occurrence of the current word. Thus, $\texttt{discrete}(\boldsymbol{r}, u) := \texttt{binary\_search}(\boldsymbol{c}, u)$.

The detailed procedure of using word-by-word sampling sequence is described in Algorithm 3.3. Let us analyse the performance difference of F+LDA between two sampling sequences of a large number of documents. The amortized cost for each CGS step is $O(|T_d| + \log T)$ for the word-by-word sequence and $O(|T_w| + \log T)$ for the document-by-document sequence. Note that $|T_d|$ is always bounded by the number of words in a document, which is usually a much smaller number than a large T (say 1024). In contrast, $|T_w|$ approaches to T when the number of documents increases. Thus, we can expect that F+LDA with the word-by-word sequence is faster than the document-by-document sequence. Empirical results in Section 3.4.1 also confirm our analysis.

3.2.3 Related Work

SparseLDA [134] is the first sampling method which considered decomposing $\boldsymbol{p}$ into a sum of sparse vectors and a dense vector. In particular, it considers a three-term decomposition of p_t as follows.

$$p_t = \frac{\alpha\beta}{n_t + \bar{\beta}} + \beta\left(\frac{n_{td}}{n_t + \bar{\beta}}\right) + n_{tw}\left(\frac{n_{td} + \alpha}{n_t + \bar{\beta}}\right),$$

where the first term is dense, the second term is sparse with $|T_d|$ non-zeros, and the third term is sparse with $|T_w|$ non-zeros. In both SparseLDA implementations (Yahoo! LDA [110] and Mallet LDA [134]), LSearch is applied to all of these three terms. As SparseLDA follows the document-by-document

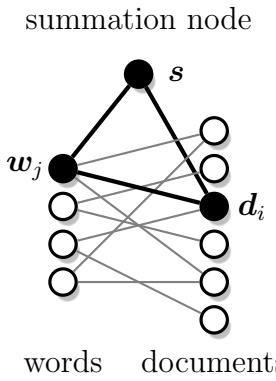
sequence, very few elements will be changed for the first two terms at each CGS step. Sampling procedures for the first two terms have very low chance to be performed due to the observation that most mass of p_t is contributed from the third term. The choice of LSearch, whose normalization constant c_T can be updated in $O(1)$ time, for the first two terms is reasonable. Note that $O(T)$ and $O(|T_d|)$ initialization costs for the first two terms can be amortized. The overall amortized cost for each CGS step is $O(|T_w| + |T_d| + |T|)$.

AliasLDA [68] is a recently proposed approach which reduces the amortized cost of each step to $O(|T_d|)$. AliasLDA considers the following decomposition of $\boldsymbol{p}$:

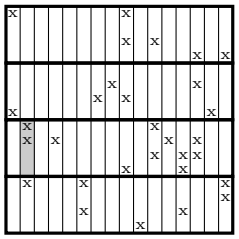
$$p_t = \alpha\left(\frac{n_{tw} + \beta}{n_t + \bar{\beta}}\right) + n_{td}\left(\frac{n_{tw} + \beta}{n_t + \bar{\beta}}\right).$$

Instead of the "exact" multinomial sampling for $\boldsymbol{p}$, AliasLDA considers a proposal distribution $\boldsymbol{q}$ with a very efficient generation routine and performs a series of Metropolis-Hasting (MH) steps using this proposal to simulate the true distribution $\boldsymbol{p}$. In particular, the proposal distribution is constructed using the latest second term and a stale version of the first term. For both terms, Alias method is applied to perform the sampling. $\#MH$ steps decides the quality of the sampling results. The overall amortized cost for each CGS step is $O(|T_d| + \#MH)$. Note the initialization cost $O(|T|)$ for the first term can be amortized as long as the same Alias table can be used to generate T samples.

See Table 3.2 for a detailed summary for LDA using various sampling



(a) Access graph for LDA



(b) Task and Data Partition

Figure 3.2: Abstract access graph for LDA.

methods. Note that the hidden coefficient ρ_A in the $O(|T_d|)$ notation for the construction of the Alias table is larger than the coefficient ρ_B for the construction of BSearch and the coefficient ρ_F for the maintenance and sampling of F+tree. Thus as long as $T < 2^{\frac{\rho_A - \rho_B}{\rho_F}|T_d|}$, F+LDA using the word-by-word sampling sequence is faster than AliasLDA. Empirical results in Section 3.4.1 also show the superiority of F+LDA over AliasLDA for real-world datasets even using $T = 50,000$.

3.3 Nomad-LDA: A Novel Nomadic Distributed Approach

In this section we present our second innovation—a novel parallel framework for CGS. Note that the same technique can also be used for other inference techniques for LDA such as collapsed variational Bayes and stochastic variational Bayes [6] since they follow similar update patterns.

To explain our proposed approach, we find it instructive to consider a hypergraph G. Let $G = (V, E)$ be a hypergraph with $(I + J + 1)$ nodes:

$$V = \{\boldsymbol{d}_i : i = 1, \ldots, I\} \cup \{\boldsymbol{w}_j : j = 1, \ldots, J\} \cup \{\boldsymbol{s}\},$$

and hyperedges:

$$E = \{e_{ij} = \{\boldsymbol{d}_i, \boldsymbol{w}_j, \boldsymbol{s}\}\},$$

where $|E| = \sum_i n_i$. Note that G contains multi-edges, which means that the same hyperedge can appear more than once in E just as a single word can appear multiple times in a document. Clearly, G is equivalent to a bag-of-words representation of the corpus $\{d_1, \ldots, d_I\}$; each $\boldsymbol{d}_i$ is associated with the i-th document, each $\boldsymbol{w}_j$ is associated with the j-th vocabulary, and each hyperedge e_{ij} corresponds to one occurrence of the vocabulary w_j in the i-th document d_i. See Figure 3.2(a) for a visual illustration; here, each gray edge corresponds to an occurrence of a word and the black triangle highlights a particular hyperedge $e_{ij} = \{\boldsymbol{d}_i, \boldsymbol{w}_j, \boldsymbol{s}\}$.

To further connect G to the update rule of CGS, we associate each node of G with a T-dimensional vector. In many inference methods, an update based on a single occurrence w_{ij} can be realized as a graph operation on G which accesses values of nodes in a single hyperedge e_{ij}. More concretely, let us define the t-th coordinate of each vector as follows:

$$(\boldsymbol{d}_i)_t := n_{t,i,*}, \quad (\boldsymbol{w}_j)_t := n_{t,*,w_j}, \text{ and } \quad (\boldsymbol{s})_t := n_{t,*,*}.$$

Based on the update rule of CGS, we can see that the update for the occurrence of w_{ij} only reads from and writes to the values stored in $\boldsymbol{d}_i$, $\boldsymbol{w}_{w_{ij}}$, and $\boldsymbol{s}$.

Interestingly, this property of the updates is reminiscent of that of the stochastic gradient descent (SGD) algorithm for matrix completion model. Similarly to LDA, matrix completion model has two sets of parameters $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_J$ and $\boldsymbol{d}_1, \ldots, \boldsymbol{d}_I$, and each SGD update requires only one of $\boldsymbol{w}_j$ and one of $\boldsymbol{d}_i$ to be read and modified. Since each update is highly localized, there is considerable parallelism available; [145] exploits this property to propose an efficient asynchronous parallel SGD algorithm for matrix completion.

The crucial difference in the case of LDA, however, is that there is an additional variable $\boldsymbol{s}$ which participates in every hyperedge of the graph. Thus, if we change the update sequence from $(e_{ij}, e_{i'j'})$ to $(e_{i'j'}, e_{ij})$, then even if $i \neq i'$ and $j \neq j'$ the result of updates will not be the same since the value of $\boldsymbol{s}$ changes in the first update. Fortunately, this dependency is very weak; each element of $\boldsymbol{s}$ is a large number because it is a summation over the whole corpus and each update changes its value at most by one, therefore the relative change of $\boldsymbol{s}$ made in a short period of time is often negligible.

While existing approaches such as Yahoo! LDA [110] exploit this observation by introducing a parameter server and let each machine query the server to retrieve recent updates, it is certainly not desirable in a large scale system that every machine has to query the same central server. Motivated by the "nomadic" algorithm introduced by [145] for matrix completion, we propose a new parallel framework for LDA that is decentralized, asynchronous and lock-free.

Algorithm 3.4 The basic Nomad LDA algorithm

Given: initialized $\boldsymbol{s}_l$, $\bar{\boldsymbol{s}}$, and local queue $\boldsymbol{q}_l$

- **While** stop signal has not been received
  - If receive a token τ, $push(\boldsymbol{q}_l, \tau)$
  - $\tau \leftarrow pop(\boldsymbol{q}_l)$
  - If $\tau = \tau_s$
    - $\boldsymbol{s} \leftarrow \boldsymbol{s} + (\boldsymbol{s}_l - \bar{\boldsymbol{s}})$
    - $\boldsymbol{s}_l \leftarrow \boldsymbol{s}$
    - $\bar{\boldsymbol{s}} \leftarrow \boldsymbol{s}$
    - Send τ_s to another worker
  - Else if $\tau = \tau_j := (j, \boldsymbol{w}_k)$
    - Perform the j-th subtask
    - Send τ_s to another worker

3.3.1 Nomadic Framework for Parallel LDA

Let p be the number of parallel workers, which can be a thread in a shared-memory multi-core machine or a processor in a distributed memory multi-machine system.

Data Partition and Subtask Split. The given document corpus is split into p portions such that the l-th worker owns the l-th partition of the data, $D_l \subset \{1, \ldots, J\}$. Unlike the other parallel approach where each unit subtask is a document owned by the worker, our approach uses a fine-grained split for tasks. Note that in the inference for LDA, each word occurrence corresponds to an update. Thus, we consider a unit subtask $\boldsymbol{t}_j$ as all occurrences of word w_j in all documents owned by the worker. See Figure 3.2(b) for an illustration in the data partition and task split. Each "x" denotes an occurrence of a word. Each block row (bigger rectangle) represents a data partition

owned by a worker, while each smaller rectangle stands for a unit subtask for the worker.

Asynchronous Computation. It is known that synchronous computation would suffer from *the curse of last reducer* when the load-balance is poor. In this work, we aim to develop an asynchronous parallel framework where each worker maintains a local job queue $\boldsymbol{q}_l$ such that the worker can keep performing the subtask popped from the queue without worrying about data conflict and synchronization. To achieve this goal, we first study the characteristics of subtasks. The subtask $\boldsymbol{t}_j$ for the l-th worker involves updates on all occurrences of $\boldsymbol{w}_j$ in D_l, which means that to perform $\boldsymbol{t}_j$, the l-th worker must acquire permission to access $\{\boldsymbol{d}_i : i \in D_l\}$, $\boldsymbol{w}_j$, and $\boldsymbol{s}$. Our data partition scheme has guaranteed that two workers will never need to access a same $\boldsymbol{d}_i$ simultaneously. Thus we can always keep the ownership of $\boldsymbol{d}_i, \forall i \in D_l$ to l-th worker. The difficulty for parallel execution comes from the access to $\boldsymbol{w}_j$ and $\boldsymbol{s}$ which can be accessed by different workers at the same time. To overcome this difficulty, we propose to use a *nomadic token passing* scheme to avoid access conflicts. Token passing is a standard technique used in telecommunications to avoid conflicting access to a resource shared by many members. The idea is "owner computes": only the member with the ownership of the token has the permission to access the resource. Here we borrow the same idea to avoid the situation where two workers require access to the same $\boldsymbol{w}_j$ and $\boldsymbol{s}$.

Nomadic Tokens for $\boldsymbol{w}_j$. We have a token τ_j dedicated for the ownership of each word $\boldsymbol{w}_j$. These J tokens are nomadically passed among

p workers. The ownership of a token τ_j means the worker can perform the subtask $\boldsymbol{t}_j$. Each token τ_j is a tuple $(j, \boldsymbol{w}_j)$, where the first entry is the index for the token, and the second entry is the latest value of $\boldsymbol{w}_j$. For a worker, a token τ means the activation of the corresponding inference subtask. Thus, we can guarantee that 1) the values of $\boldsymbol{w}_j$ used in each subtask is always up-to-date; 2) no two workers require access to a same $\boldsymbol{w}_j$.

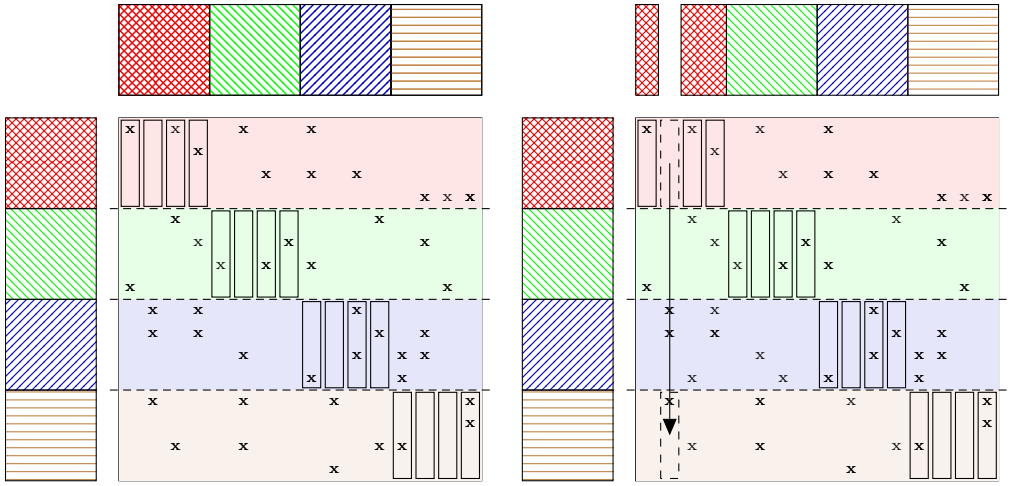
Nomadic Token for $\boldsymbol{s}$. So far we have successfully kept the values of $\boldsymbol{d}_i$ and $\boldsymbol{w}_j$ used in each subtask latest, and avoid access conflicts by nomadic token passing. However, all subtasks depend on each other due to the need to access $\boldsymbol{s}$. Based on the summation property, we propose to deal with this issue by creating a special nomadic token $\tau_s = (0, \boldsymbol{s})$ for $\boldsymbol{s}$, where 0 is the token index for τ_s, and have two copies of $\boldsymbol{s}$ in each worker: $\boldsymbol{s}_l$ and $\bar{\boldsymbol{s}}$. $\boldsymbol{s}_l$ is a local shadow node for $\boldsymbol{s}$. The l-th worker always uses the values of $\boldsymbol{s}_l$ to perform updates and makes the modification to $\boldsymbol{s}_l$. $\bar{\boldsymbol{s}}$ was the snapshot of $\boldsymbol{s}$ from the last arrival of τ_s. Due to the additivity of $\boldsymbol{s}$, the delta $\boldsymbol{s}_l - \boldsymbol{s}$ can be regarded as the effort that has been made since the last arrival of τ_s. Thus, each time that τ_s arrives, the worker can perform the following operations to accumulate its local effort to the global $\boldsymbol{s}$ and update its local $\boldsymbol{s}_l$.

1 $\boldsymbol{s} \leftarrow \boldsymbol{s} + (\boldsymbol{s}_l - \bar{\boldsymbol{s}})$

2 $\bar{\boldsymbol{s}} \leftarrow \boldsymbol{s}$

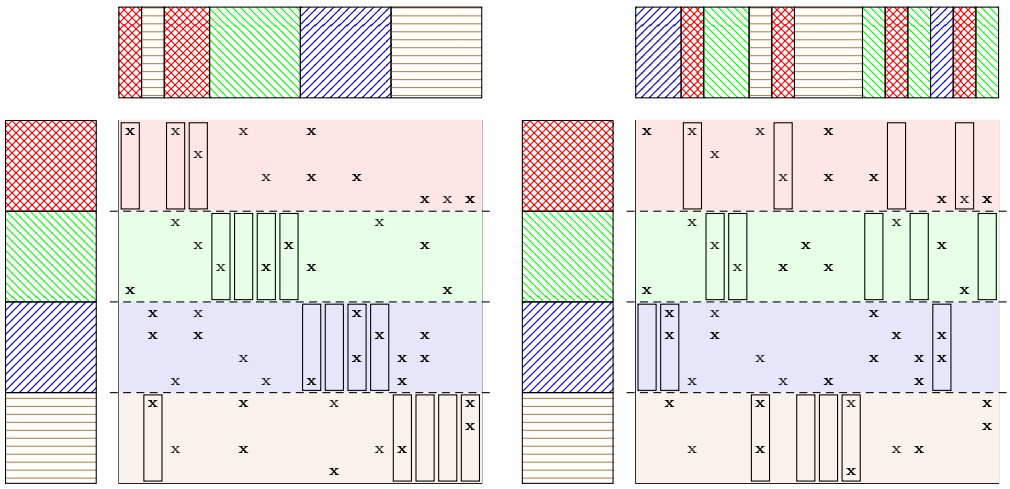
3 $\boldsymbol{s}_l \leftarrow \boldsymbol{s}$

We present the general idea of Nomad LDA in Algorithm 3.4 and its illustration in Figure 3.3.



(a) Initial assignment of $\boldsymbol{w}_j$. Each worker works only on the diagonal active area in the beginning.

(b) After a worker finishes processing j, it sends the corresponding $\boldsymbol{w}_j$ to another worker. Here, $\boldsymbol{w}_2$ is sent from worker 1 to 4.



(c) Upon receipt, the $\boldsymbol{w}_j$ is processed by the new worker. Here, worker 4 can now process $\boldsymbol{w}_2$ since it owns it.

(d) During the execution of the algorithm, the ownership of the $\boldsymbol{w}_j$ changes.

Figure 3.3: Illustration of the Nomad LDA algorithm

3.3.2 Related Work

Unlike the situation in the serial case, the latest values of $n_{z,*,w}$ and $n_{z,*,*}$ can be distributed among different machines in the distributed setting. The existing parallel approaches focus on developing mechanisms to communicate these values. We briefly review two approaches for parallelizing CGS in distributed setting: AdLDA [84] and Yahoo! LDA [110]. In both approaches, each machine has a local copy of the *entire* $n_{z,*,w}$ and $n_{z,*,*}$. AdLDA uses bulk synchronization to update its local copy after each iteration. At each iteration, each machine just uses the snapshot from last synchronization point to conduct Gibbs sampling. On the other hand, Yahoo! LDA creates a central parameter server to maintain the latest values for $n_{z,*,w}$ and $n_{z,*,*}$. Every machine asynchronously communicates with this machine to send the local update to the server and get new values to update its local copy. Note that the communication is done asynchronously in Yahoo! LDA to avoid expensive network locking. The central idea of Yahoo! LDA is that modest stale values would not affect the sampler significantly. Thus, there is no need to spend too much effort to synchronize these values. Note that for these two approaches, both values of $n_{z,*,w}$ and $n_{z,*,*}$ used in Gibbs sampling could be stale. In contrast, our proposed Nomad LDA has the following advantages:

- No copy of the entire $n_{z,*,w}$ is required in each machine.
- The value of $n_{z,*,w}$ used in the Gibbs sampling is always up-to-date in each machine.
- The computation is both asynchronous and decentralized.

Our Nomad LDA is close to a parallel approach for matrix completion [145], which also utilized the concept of nomadic variables. However, the application is completely different. [145] concentrates on parallelizing stochastic gradient descent for matrix completion. The access graph for this problem is a bipartite graph, and there is no variable that needs to be synchronized across processors.

3.4 Experimental Results

In this section we investigate the performance and scaling of our proposed algorithms. We demonstrate that our proposed F+tree sampling method is very efficient in handling large number of topics compared to the other approaches in Section 3.4.1. When the number of documents is also large, in Section 3.4.2 we show our parallel framework is very efficient in multi-core and distributed systems.

Datasets. We work with five real-world large datasets—Enron, NyTimes, PubMed, Amazon, and UMBC. The detailed data set statistics are listed in Table 3.3. Among them, Enron, NyTimes and PubMed are bag-of-word datasets in the UCI repository[1]. These three datasets have been used to demonstrate the scaling behavior of topic modeling algorithms in many recent papers [6, 68, 110]. In fact, the PubMed dataset stretches the capabilities of many implementations. For instance, we tried to use LDA code from

[1] https://archive.ics.uci.edu/ml/datasets/Bag+of+Words

http://www.ics.uci.edu/~asuncion/software/fast.htm,
but it could not handle PubMed.

To demonstrate the scalability of our algorithm, we use two more large-scale datasets—Amazon and UMBC. The Amazon dataset consists of approximately 35 million product reviews from Amazon.com, and was downloaded from the Stanford Network Analysis Project (SNAP) home page. Since reviews are typically short, we split the text into words, removed stop words, and using Porter stemming [94]. After this pre-processing we discarded words that appear fewer than 5 times or in 5 reviews. Finally, any reviews that were left with no words after this pre-processing were discarded. This resulted in a corpus of approximately 30 million documents and approximately **1.5 billion** words.

The UMBC WebBase corpus is downloaded from http://ebiquity.umbc.edu/blogger/2013/05/01/. It contains a collection of pre-processed paragraphs from the Stanford WebBase[2] crawl on February 2007. The original dataset has approximately 40 million paragraphs and 3 billion words. We further processed the data by stemming and removing stop words following the same procedure in LibShortText [137]. This resulted in a corpus of approximately **1.5 billion** words.

Hardware. The experiments are conducted on a parallel platform

[2]Stanford WebBase project: http://dbpubs.stanford.edu:8091/~testbed/doc2/WebBase/

Table 3.3: Data statistics for topic modeling experiments.

	# documents (I)	# vocabulary (J)	# words
Enron	37,861	28,102	6,238,796
NyTimes	298,000	102,660	98,793,316
PubMed	8,200,000	141,043	737,869,083
Amazon	29,907,995	1,682,527	1,499,602,431
UMBC	40,599,164	2,881,476	1,483,145,192

at the Texas Advanced Computing Center (TACC), called Maverick[3]. Each node contains 20 Intel Xeon E5-2680 CPUs and 256 GB memory. Each job can run on at most 32 nodes (640 cores) for at most 12 hours.

Parameter Setting. Throughout the experiments we set the hyper parameters $\alpha = 50/T$ and $\beta = 0.01$, where T is the number of topics. Previous papers showed that this parameter setting gives good model qualities [46], and many widely-used software such as Yahoo! LDA and Mallet-LDA also use this as the default parameters. To test the performance with a large number of topics, we set $T = 1024$ in all the experiments except the ones in Figure 3.5.

Evaluation. Our main competitor is Yahoo! LDA in large-scale distributed setting. To have a fair comparison, we use the same training likelihood routine to evaluate the quality of model (see Eq. (2) in [110] for details).

[3] https://portal.tacc.utexas.edu/user-guides/maverick

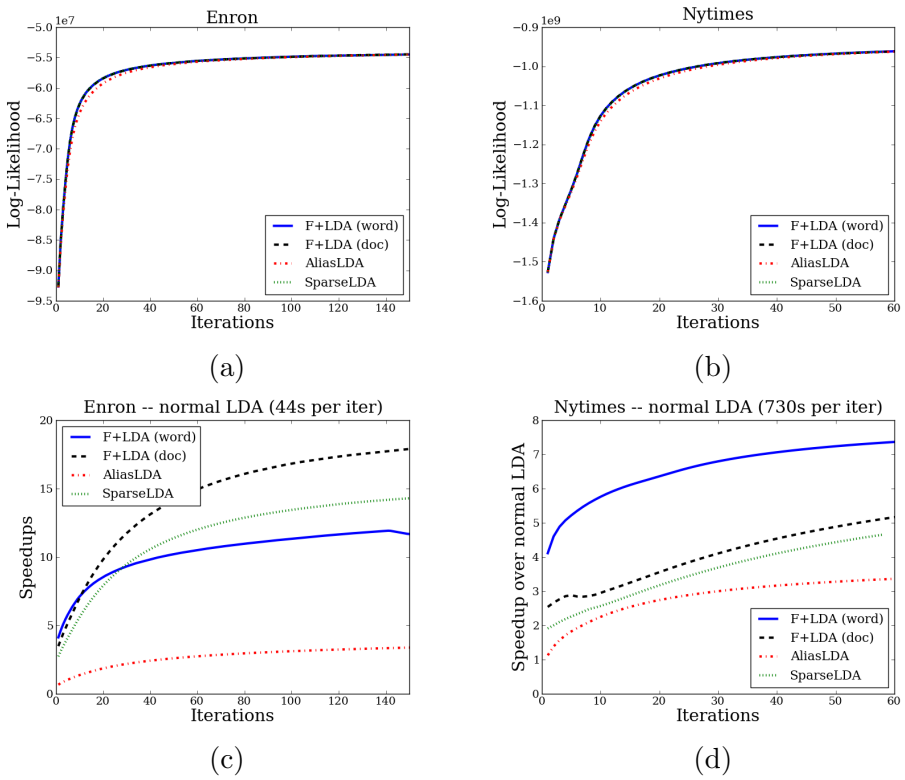
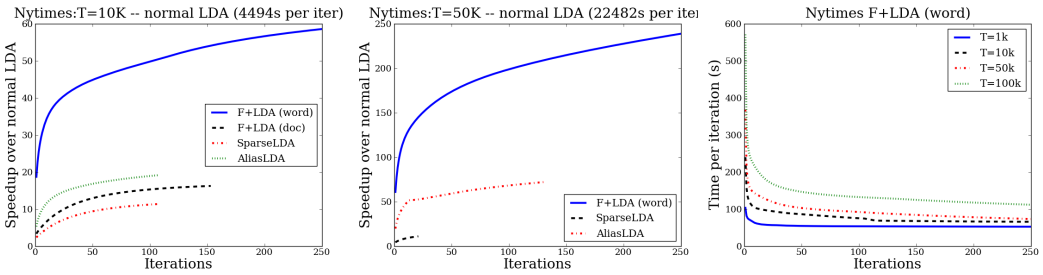


Figure 3.4: (a) and (b) present the convergence speed in terms of number of iterations. (c) and (d) present the sampling speed of each iteration—the y-axis is the speedup over the normal LDA implementation which takes $O(T)$ time to generate one sample. We observe all the sampling algorithms have similar convergence speed, while F+LDA(doc) is the fastest compared to other document-wise sampling approaches. Also, F+LDA(word) is faster than F+LDA(doc) for larger datasets, which confirms our analysis in Section 3.2.2.

3.4.1 Comparison of sampling methods: handling large number of topics

In this section, we compare various sampling strategies used for LDA in the serial setting. We include the following sampling strategies into the comparison (see Section 3.2 for details):



(a) (b) (c)

Figure 3.5: (a) and (b) present the sampling speed for $T = 10,000$ and $T = 50,000$. The superiority of F+LDA(word) is more significant when T is large. (c) shows the sampling speed for various T. The increase in sampling time is much smaller than the increase in the number of topics T. Note that the sampling time per iteration for $T = 100,000$ is only about twice as much as the time required for $T = 1,000$.

1 F+LDA: our proposed sampling scheme. Document/word-wise sampling order are denoted by F+LDA(doc) and F+LDA(word), respectively.

2 SparseLDA: the approach that uses linear search on PDF to conduct document-wise sampling. This approach is used in Yahoo! LDA and Mallet-LDA.

3 AliasLDA: the approach that uses Alias method to do the sampling with document-wise sampling order. This approach is proposed very recently in [68].

To have a fair comparison focusing on different sampling strategies, we implemented the above three approaches to use the same data structures. We use two smaller datasets—Enron and NyTimes to conduct the experiments. Note that [68] also conducts the comparison of different sampling approaches using

these two datasets after further preprocessing. Figure 3.4 presents the comparison results using $T = 1,024$, while in Figure 3.5 we show the results by varying T from $1,000$ to $100,000$.

We first compare F+LDA(doc), Sparse LDA, and Alias LDA, where all of the three approaches have the same document-wise sampling ordering. F+LDA(doc) and Sparse LDA follow the exact sampling distribution of the normal Gibbs sampling; as a result, we can observe in Figure 3.4(a) and 3.4(b) that they have the same convergence speed in terms of number of iterations. On the other hand, Alias LDA converges slightly slower than other approaches because it does not sample from the exact same distribution. Note that we found that this phenomenon becomes more clear when T is large. In terms of efficiency, Figures 3.4(c) and 3.4(d) indicate that F+LDA(doc) is faster than Sparse-LDA and Alias-LDA, which confirms our analysis in Section 3.2.

Next we compare the performance of document-wise and word-wise sampling for F+LDA. Figure 3.4(a) and 3.4(b) indicate that both orderings give similar convergence speed. As discussed in Section 3.2.2, using the F+tree sampling approach, the word-wise ordering is expected to be faster than document-wise ordering as the number of documents increases. This phenomenon is confirmed by our experimental results in Figures 3.4(c), 3.4(d), and 3.5(a) as F+LDA(word) is faster than F+LDA(doc) on the NyTimes dataset, which has a larger number of documents comparing to Enron. The experimental results also justify our use of word-wise sampling when applying the Nomad approach in multi-core and distributed systems.

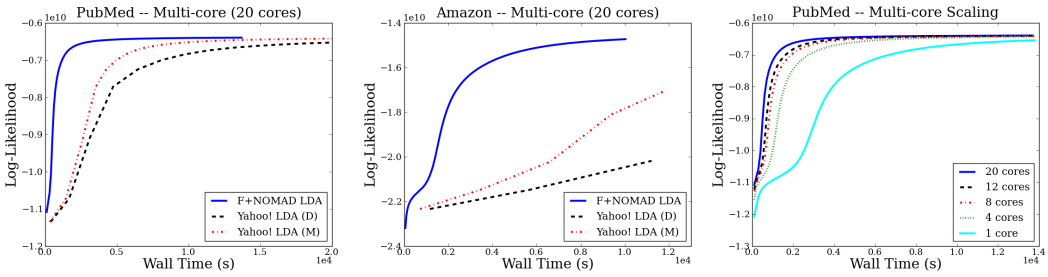
Figure 3.5 shows the results for even larger T. As the length of the computing time is limited to 12 hours by the Maverick system, the number of iterations is different for all methods. We first observe that when $T \geq 10,000$, AliasLDA starts to outperform SparseLDA in Figures 3.5(a) and 3.5(b). However, F+LDA(word) still outperforms these two methods significantly. In 3.5(c), we can see that when T is increased from $1,000$ to $100,000$, the sampling time required by F+LDA(word) increases only by a factor of two. This can be explained by the logarithmic time complexity of F+LDA(word).

3.4.2 Multi-core and Distributed Experiments

Now we combine our proposed F+tree sampling strategy with the nomadic parallelization framework. This leads to a new F+Nomad LDA sampler that can handle huge problems in multi-core and distributed systems.

3.4.2.1 Competing Implementations

We compare our algorithm against Yahoo! LDA for three reasons: a) It is one of the most efficient open source implementations of CGS for LDA, which scales to large datasets. b) [110] claims that Yahoo! LDA outperforms other open source implementation such as AD-LDA [84] and PLDA [124]. c) Yahoo! LDA uses a parameter server, which has become a generic approach for distributing large-scale learning problems. It is therefore interesting to see if a different asynchronous approach can outperform the parameter server on this specific problem. Yahoo! LDA is a disk-based implementation that assumes the



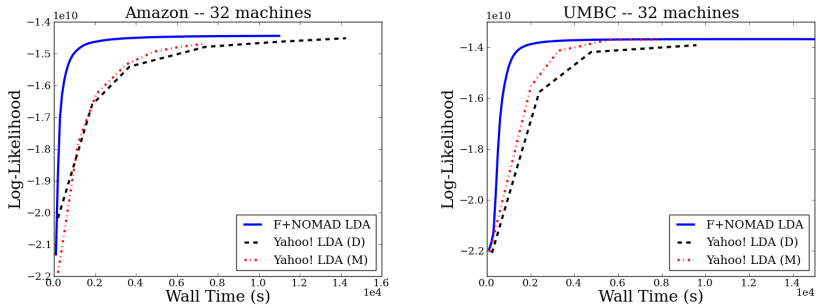
(a) (b) (c)

Figure 3.6: (a) and (b) show the comparison between Nomad LDA and Yahoo! LDA using 20 cores on a single machine. (c) shows the scaling performance of Nomad LDA as a function of number of cores.

latent variables associated with tokens in the documents are streamed from disk at each iteration. To have a fair comparison, in addition to running the disk-based Yahoo! LDA (denoted by Yahoo! LDA(D)), we further ran it on the `tmpfs` file system [111] which resides on RAM for the intermediate storage used by Yahoo! LDA. This way we eliminate the cost of disk I/O, and can make a fair comparison with our own code which does not stream data from disk; we use Yahoo! LDA(M) to denote this version.

3.4.2.2 Multi-core Experiments

Both F+Nomad LDA and Yahoo! LDA support parallel computation on a single machine with multiple cores. Here we conduct experiments on two datasets, Pubmed and Amazon, and the comparisons are presented in Figure 3.6. As can be seen from Figures 3.6(a) and 3.6(b), F+Nomad LDA handsomely outperforms both memory and disk version of Yahoo! LDA, and gets to a better quality solution within the same time budget. Given a desired log-



(a) (b)

Figure 3.7: The comparison between F+Nomad LDA and Yahoo! LDA on 32 machines with 20 cores per machine.

likelihood level, F+Nomad LDA is approximately 4 times faster than Yahoo! LDA.

Next we turn out attention to the scaling of F+Nomad LDA as a function of the number of cores. In Figure 3.6(c) we plot the convergence of F+Nomad LDA as the number of cores is varied. Clearly, as the number of cores increases the convergence speed is faster.

3.4.2.3 Distributed Memory Experiments

We compare the performance of F+Nomad LDA and Yahoo! LDA on two huge datasets, Amazon and UMBC, in a distributed memory setting. The number of machines is set to 32, and the number of cores per machine is 20. As can be seen from Figure 3.7, F+Nomad LDA dramatically outperforms both memory and disk version of Yahoo! LDA and obtains significantly better quality solution (in terms of log-likelihood) within the same wall clock time.

3.5 Summary of the Contributions

In this chapter, we present a novel F+Nomad LDA algorithm that can handle large number of topics as well as large number of documents. In order to handle large number of topics we use an appropriately modified Fenwick tree. This data structure allows us to sample from and update a T-dimensional multinomial distribution in $O(\log T)$ time. In order to handle large number of documents, we propose a novel asynchronous and non-locking parallel framework, which leads to impressive speedups in multi-core and distributed systems. The resulting algorithm is faster than Yahoo! LDA and is able to handle datasets with billions of words. The work of this chapter is published in [140].

Chapter 4

Efficient Algorithms for One-class Matrix Factorization

Matrix factorization (MF) is a popular technique for collaborative filtering. With observed ratings given by m users to n items, MF solves the following optimization problem.

$$\min_{W,H} \sum_{(i,j)\in\Omega} (A_{ij} - \boldsymbol{w}_i^\top \boldsymbol{h}_j)^2 + \sum_i \lambda_i \|\boldsymbol{w}_i\|^2 + \sum_j \bar{\lambda}_j \|\boldsymbol{h}_j\|^2. \tag{4.1}$$

Each entry $A_{ij} \in \mathbb{R}$ (e.g., score of 1 to 5) is the rating given by user i to item j and

$$\Omega = \{(i,j) : A_{ij} \text{ is observed}\}$$

is the set of observed ratings. With the regularization parameters $\lambda_i, \bar{\lambda}_j$, the goal is to find two low-rank latent matrices

$$W = \begin{bmatrix} \boldsymbol{w}_1^\top \\ \vdots \\ \boldsymbol{w}_m^\top \end{bmatrix} \in \mathbb{R}^{m\times k} \text{ and } H = \begin{bmatrix} \boldsymbol{h}_1^\top \\ \vdots \\ \boldsymbol{h}_n^\top \end{bmatrix} \in \mathbb{R}^{n\times k}$$

The materials in this Chapter have been presented in [142]. I developed the algorithms and conducted the experiments.

so that $\boldsymbol{w}_i^\top \boldsymbol{h}_j$ is a good approximation of A_{ij}. Note that k is a pre-specified latent factor satisfying

$$k \ll m \text{ and } k \ll n.$$

In contrast to $A_{ij} \in \mathbb{R}$ for rating-based MF, in some applications, the two possible A_{ij} values are 1 (positive) and 0 (negative), but only part of positive entries are observed. For example, in [90] for Xbox movies recommendation, we only know movies that have been watched by users. By assuming that users do not watch movies they do not like, we have some partial positive data but lack any negative information. To handle such a one-class scenario, one popular approach [54, 75, 88–90] is to treat some missing entries as negative and the sum of losses in (4.1) becomes

$$\sum_{(i,j)\in\Omega^+} (1 - \boldsymbol{w}_i^\top \boldsymbol{h}_j)^2 + \sum_{(i,j)\in\Omega^-} (0 - \boldsymbol{w}_i^\top \boldsymbol{h}_j)^2, \tag{4.2}$$

where Ω^+ is the set of observed positive entries, and Ω^- includes negative entries sampled from missing entries in A. The rationale is that among the large number of items, a user likes only a small subset of them. Therefore, most of the missing entries in A are negative. Currently two major approaches to select the set Ω^- are

1. Subsampled: the size of Ω^- is roughly similar to that of Ω^+.

$$|\Omega^-| = O(|\Omega^+|) \ll mn. \tag{4.3}$$

   Some reasons support this setting. First, given the so few observed positive entries, a large Ω^- may cause serious imbalance between positive

and negative entries. Second, using a too large Ω^- causes difficulties to solve the optimization problem. Studies that have considered this setting include [89, 90].

2. Full: all missing entries are considered as negative, so

$$\Omega^- = \{(i,j) \mid (i,j) \notin \Omega^+\}. \tag{4.4}$$

One apparent reason of using this setting is that all missing entries are considered. From this viewpoint, the Subsampled approach is just an approximation of the Full approach.

Handling the huge number of $|\Omega^-| = O(mn)$ elements causes the Full approach to be practically infeasible. However, under some circumstances the alternating least squares (ALS) optimization method given in [54, 88] can have similar complexity to that for Subsampled.

At the first glance it is unclear if the two approaches give different performances. Surprisingly, few works have compared them. The main existing study is [89], which reports the following observations on two rather small data sets (thousands of users and items).

- The Subsampled approach is worse than the Full.
- By a bagging approach to select 20 different Ω^- sets and average the resulting predictions, the performance is as good as the Full.

We feel there is a need to detailedly study the two approaches because of the following concerns.

- Experiments in [89] are for small data. Observations may be very different for large-scale data sets.
- The lack of studies caused that in some papers incapable baselines may be used to compare with newly proposed techniques. For example, in [63] to propose a one-class MF approach incorporating meta-features, they compare with a Subsampled setting. It is possible that Full is a better baseline.
- Because of the huge set Ω^- used by the Full approach, traditional optimization methods for MF fail to handle large data. Therefore, the Full approach may not be useful even if it can give better models.

In this chapter, we make the following major contributions.

- Beyond the work of [54, 88] that developed optimization methods for the Full approach, we create some more efficient ones for large-scale data sets.
- We conduct thorough comparisons between Full and Subsampled after considering their best settings.

Our conclusion is that Full yields much better results than Subsampled. With our proposed optimization techniques for training large problems, the Full approach by treating all missing entries as negative becomes practically viable.

One-class MF is a case of PU (positive-unlabeled) learning [51], which includes other important applications such as link prediction [79]. Our proposed optimization methods can be easily applied to them.

Table 4.1: Notation

m, n, k	numbers of users, items, and latent variables
A	$m \times n$ rating matrix
W, H	$m \times k$ and $n \times k$ latent matrices
$\boldsymbol{w}_i, \boldsymbol{h}_j$	$k \times 1$ vector; ith row of W and jth row of H
$\lambda_i, \bar{\lambda}_j$	regularization parameters
Ω	set of observed entries for standard MF
Ω^+	set of observed positive entries for one-class MF
Ω^-	set of selected negative entries for one-class MF
Ω_i^+	set of user i's observed entries (one-class MF)
$\bar{\Omega}_j^+$	set of item j's observed entries (one-class MF)
C	$m \times n$ matrix for weights of entries

Some studies extend the positive versus negative setting to other optimization problems. For example, instead of a squared loss, various ranking loss functions are considered in [73, 102]. We will include one ranking-loss approach in our empirical comparison. In [108], a variable is simultaneously optimized to decide if a missing entry is negative or not. To be focused, in the current work we do not discuss this approach.

This chapter is organized as follows. Section 4.1 reviews past studies on both Subsampled and Full approaches. In Section 4.2, we propose methods to solve large optimization problems for the Full approach. Section 4.3 gives detailed comparisons, while discussions and conclusions are in Section 4.4. Table 4.1 gives main notation in this chapter. Code for experiments is available at http://www.csie.ntu.edu.tw/~cjlin/papers/one-class-mf.

4.1 Existing Studies on Subsampled and Full Approaches

In this section, we briefly discuss existing studies of using the Full and the Subsampled settings. This discussion is important for the investigation in Section 4.2 to propose efficient optimization algorithms. To begin, we extend (4.1) to have the following general weighted MF formulation.

$$\min_{W,H} \sum_{i,j\in\Omega} C_{ij}(A_{ij} - \boldsymbol{w}_i^\top \boldsymbol{h}_j)^2 + \sum_i \lambda_i \|\boldsymbol{w}_i\|^2 + \sum_j \bar{\lambda}_j \|\boldsymbol{h}_j\|^2, \tag{4.5}$$

where C_{ij} is a cost associated with the loss. For one-class MF, the set Ω includes both positive and negative entries:

$$\Omega = \Omega^+ \cup \Omega^-.$$

For each user i we define

$$\Omega_i^+ \equiv \{j \mid (i,j) \text{ is an observed entry}\},$$

$$\Omega_i^- \equiv \{j \mid (i,j) \text{ is a missing entry selected as negative}\}, \text{ and}$$

$$\Omega_i \equiv \Omega_i^+ \cup \Omega_i^-.$$

Similarly, for each item j, we let

$$\bar{\Omega}_j^+ \equiv \text{ set of observed entries of item } j.$$

Usually in (4.5) we set

$$\lambda_i = \lambda|\Omega_i^+|, \ \forall i \text{ and } \bar{\lambda}_j = \lambda|\bar{\Omega}_j^+|, \ \forall j. \tag{4.6}$$

4.1.1 The Full Approach

Because Ω^- is a fixed set, weights C_{ij} are the main concern. In [88, 89], they consider $C_{ij} = 1, \forall (i,j) \in \Omega^+$ and the following settings for $C_{ij}, \forall (i,j) \in \Omega^-$.

- C_{ij} is a constant.
- $C_{ij} \propto |\Omega_i^+|$. That is, under the same j, $C_{ij} = |\Omega_i^+|\Delta$, $\forall i$, where Δ is a constant. The reason is that users associated with few items provide less reliable information.
- $C_{ij} \propto n - |\bar{\Omega}_j^+|$. Note that $|\bar{\Omega}_j^+|$ indicates the popularity of item j. Because more popular items are less likely to be negative, weights for them should be smaller.

In addition, C_{ij} can be decided by some user and item features. For example, [75] consider

$$C_{ij} = \begin{cases} 1 - \text{sim}(i,j) & \text{if } (i,j) \in \Omega^- \\ 1 & \text{otherwise} \end{cases},$$

where $\text{sim}(i,j)$ is the similarity between user i and item j.

4.1.2 The Subsampled Approach

Because Ω^- is no longer a fixed set, we have more options as follows:

- Size of $|\Omega^-|$.
- Sampling strategies for selecting Ω^-.
- Weights C_{ij}.

The number of combined choices is huge. We may further consider a user- or an item-oriented setting. For example, instead of $|\Omega^-| \propto |\Omega^+|$, we can consider $|\Omega_i^-| \propto |\Omega_i^+|$, $\forall i$. Here we show details of two studies that have used the Subsampled approach. [89] consider

$$|\Omega^-| \propto |\Omega^+| \quad \text{and} \quad P_{ij} \propto 1, |\Omega_i^+|, \text{ or } 1/|\bar{\Omega}_j^+|,$$

where P_{ij} is the probability to select $(i,j) \notin \Omega^+$ as negative. Their settings of P_{ij} follow from the same explanation in Section 4.1.1 for choosing C_{ij}. In another study [90], for a baseline setting they have

$$|\Omega_i^-| = |\Omega_i^+| \quad \text{and} \quad P_{ij} \propto |\bar{\Omega}_j^+|.$$

Neither papers specifically says their C_{ij} values. However, given that weight information has been used for selecting Ω^-, some simple settings such as $C_{ij} = 1$ may suffice. The discussion shows that deciding a suitable sampling scheme is not easy. We see that these two studies use opposite strategies: one has $P_{ij} \propto 1/|\bar{\Omega}_j^+|$, while the other has $P_{ij} \propto |\bar{\Omega}_j^+|$.

4.2 Efficient Optimization Algorithms for the Full Approach

In this section, we propose methods to solve the large optimization problem for the Full approach. To handle the difficulty caused by $|\Omega| = mn$, [88] assume that weights C_{ij} are under some conditions.[1] We consider a similar setting by assuming that

$$C_{ij} = \begin{cases} 1 & \text{if } (i,j) \in \Omega^+ \\ p_i q_j & \text{otherwise} \end{cases} \quad \text{and} \quad A_{ij} = \begin{cases} 1 & \text{if } (i,j) \in \Omega^+ \\ \bar{a} & \text{otherwise,} \end{cases} \tag{4.7}$$

where $\boldsymbol{p} \in \mathbb{R}^m$ and $\boldsymbol{q} \in \mathbb{R}^n$ are vectors. All existing settings discussed in Section 4.1.1 except [75] satisfy this assumption. For example, if $C_{ij} \propto$

[1] The condition in [54] is a special case of [88], where they assume $C_{ij} \geq 1,\ \forall (i,j) \in \Omega^+$ and $C_{ij} = 1$ otherwise.

Table 4.2: A summary of results in Section 4.2 by showing the complexity per iteration of optimization methods. For ALS and CD, an iteration means to update W and H once, while for SG it means to conduct $|\Omega|$ SG updates. *: It remains a challenge to apply SG to one-class MF; see Section 4.2.3.

	ALS	CD	SG
General MF			
	$O(\|\Omega\|k^2 + (m+n)k^3)$	$O(\|\Omega\|k)$	$O(\|\Omega\|k)$
One-class MF (Full approach)			
Direct	$O(mnk^2 + (m+n)k^3)$	$O(mnk)$	$O(mnk)$
New	$O(\|\Omega^+\|k^2 + (m+n)k^3)$	$O(\|\Omega^+\|k + (m+n)k^2)$	NA*

$|\Omega_i^+|, \forall (i,j) \notin \Omega^+$, then we have $C_{ij} = p_i q_j$ with $p_i = |\Omega_i^+|$. Note that for negative entries we consider a slightly more general setting of $A_{ij} = \bar{a}$, although in practice $\bar{a} = 0$ is most used.

We discuss three optimization approaches in this section. Table 4.2 lists the complexity per iteration, where details are in subsections. Clearly an mn term causes prohibitive running time if these methods are directly applied to the optimization problem of the Full approach. For two of the three methods we successfully reduce the mn term to $|\Omega^+|$, so the time complexity becomes similar to that for the Subsampled approach.

4.2.1 Alternating Least Squares (ALS)

Alternating Least Squares (ALS) has been a popular optimization method since the beginning of matrix factorization. It iteratively updates W and H by the following loop

1: **while** not optimal **do**

2: Solve (4.5) by fixing H

3: Solve (4.5) by fixing W

Consider the situation when H is fixed. We would like to solve for $i = 1, \ldots, m$,

$$\min_{\boldsymbol{w}_i} \quad \sum_{j \in \Omega_i} C_{ij} \big(A_{ij} - \boldsymbol{w}_i^\top \boldsymbol{h}_j\big)^2 + \lambda_i \|\boldsymbol{w}_i\|^2. \tag{4.8}$$

By defining $f(\boldsymbol{w}_i)$ as the function to be minimized, we can calculate

$$\begin{aligned} \nabla f(\boldsymbol{w}_i) &= -\sum_{j \in \Omega_i} C_{ij}(A_{ij} - \boldsymbol{w}_i^\top \boldsymbol{h}_j)\boldsymbol{h}_j + 2\lambda_i \boldsymbol{w}_i, \\ \nabla^2 f(\boldsymbol{w}_i) &= \sum_{j \in \Omega_i} C_{ij} \boldsymbol{h}_j \boldsymbol{h}_j^\top + 2\lambda_i I_k. \end{aligned}$$

For a quadratic function like $f(\boldsymbol{w}_i)$, by

$$f(\boldsymbol{w}_i) = f(\boldsymbol{0}) + \nabla f(\boldsymbol{0})^\top \boldsymbol{w}_i + \frac{1}{2} \boldsymbol{w}_i^\top \nabla^2 f(\boldsymbol{0}) \boldsymbol{w}_i,$$

we have that

$$\text{optimal } \boldsymbol{w}_i = -\big[\nabla^2 f(\boldsymbol{0})\big]^{-1} \nabla f(\boldsymbol{0}).$$

Thus, the optimal solution is

$$\boldsymbol{w}_i = \left(\sum_{j \in \Omega_i} C_{ij} \boldsymbol{h}_j \boldsymbol{h}_j^\top + \lambda_i I_k \right)^{-1} \sum_{j \in \Omega_i} C_{ij} A_{ij} \boldsymbol{h}_j, \tag{4.9}$$

where $I_k \in \mathbb{R}^{k \times k}$ is an identity matrix. Thus the cost of one ALS iteration to update W and H is

$$O(|\Omega| \times k^2 + (m+n)k^3), \tag{4.10}$$

where $|\Omega| \times k^2$ is for $\sum_{j \in \Omega_i} C_{ij} \boldsymbol{h}_j \boldsymbol{h}_j^\top, \forall i$ and mk^3 is for m matrix inversions. From (4.10), (4.3), and (4.4), for Subsampled and Full approaches the cost is

respectively

$$O(|\Omega^+|k^2 + (m+n)k^3) \quad \text{and} \quad O(mnk^2 + (m+n)k^3).$$

The mn term is huge, but [88] have successfully reduced Full's complexity under the assumption in (4.7).[2] Here we derive there results using our notation. Specifically we rewrite a summation in (4.9) as follows.

$$\sum_{j\in\Omega_i} C_{ij}\boldsymbol{h}_j\boldsymbol{h}_j^\top = \sum_{j\in\Omega_i^+} (1-p_iq_j)\boldsymbol{h}_j\boldsymbol{h}_j^\top + p_i\sum_{j=1}^n q_j\boldsymbol{h}_j\boldsymbol{h}_j^\top.$$

The crucial point of the above reformulation is that $\sum_{j=1}^n q_j\boldsymbol{h}_j\boldsymbol{h}_j^\top$ can be pre-computed with $O(nk^2)$ operations, so the cost is reduced to be the same as that for the Subsampled approach. The second term in (4.9) involves $O(mnk)$ operations, which can be reduced to $O(|\Omega^+|k)$ because from (4.7),

$$\sum_{j\in\Omega_i} C_{ij}A_{ij}\boldsymbol{h}_j = \sum_{j\in\Omega_i^+} A_{ij}\boldsymbol{h}_j - p_i\bar{a}\sum_{j\in\Omega_i^+} q_j\boldsymbol{h}_j + p_i\bar{a}\sum_{j=1}^n q_j\boldsymbol{h}_j,$$

where $\sum_{j=1}^n q_j\boldsymbol{h}_j$ can be pre-computed. Then the mn term does not appear at all in the calculation of (4.9).

4.2.2 Coordinate Descent (CD)

Coordinate descent has been a successful optimization method. Its early use for MF was in [31], but here we consider the efficient implementation in [136]. The idea of CD is to update one column of W and H at a time.

[2] [88] consider a more general setting so that C can be a low-rank matrix but in most cases the rank-one situation in (4.7) is considered.

Specifically, if the tth column is chosen, we let two vector variables

$$\boldsymbol{u} \in \mathbb{R}^m \text{ and } \boldsymbol{v} \in \mathbb{R}^n$$

denote the corresponding columns in W and H, respectively and form the following optimization problem.

$$\min_{\boldsymbol{u},\boldsymbol{v}} \quad \sum_{(i,j)\in\Omega} C_{ij}(\hat{R}_{ij} - u_i v_j)^2 + \sum_i \lambda_i u_i^2 + \sum_j \bar{\lambda}_j v_j^2, \tag{4.11}$$

where

$$\hat{R}_{ij} \equiv A_{ij} - \boldsymbol{w}_i^\top \boldsymbol{h}_j + W_{it} H_{jt}. \tag{4.12}$$

Note that we add the constant $W_{it}H_{jt}$ to $\hat{R}_{ij}$ so that $\hat{R}_{ij} - u_i v_j$ is the error for the (i, j) entry. Problem (4.11) is like an MF problem of using only one latent variable (i.e., $k = 1$), so we can solve it by ALS. Our procedure is described in Algorithm 4.1. Essentially we have a *two-level* CD procedure. At the outer

Algorithm 4.1 Coodinate descent for one-class MF.

```
while not optimal do
    for t = 1, ..., k do
        Let W, H's tth columns be initial u, v
        Approximately solve (4.11) by ALS:
        for s = 1, ..., S do
            Solve (4.11) by fixing v
            Solve (4.11) by fixing u
        Let u, v be W, H's tth columns
```

level, sequentially a pair of columns from W and H are selected, while at the inner level, these two columns are alternatively updated several times; see the S inner iterations in the above algorithm. For rating-based MF, [138] have

shown that taking some inner iterations leads to shorter running time than conducting only one update. In general S is a small constant; here we use 5.

Next we discuss details for solving (4.11) when $\boldsymbol{v}$ is fixed. By a similar derivation for (4.9), the optimal solution is

$$u_i = (\sum_{j \in \Omega_i} C_{ij} v_j^2 + \lambda_i)^{-1} (\sum_{j \in \Omega_i} C_{ij} \hat{R}_{ij} v_j). \tag{4.13}$$

If $\hat{R}_{ij}, \forall j \in \Omega_i$ are available (see implementation details in Section 4.2.2.1), the number of operations for (4.13) is $O(|\Omega_i|)$. Therefore, the cost of going through all columns in W and H is

$$O(|\Omega|) \times k. \tag{4.14}$$

A comparison with (4.10) shows that CD is more efficient than ALS.

For the two approaches (Subsampled and Full) for one-class MF, $|\Omega|$ in (4.14) becomes $|\Omega^+|$ and mn, respectively. Because mn is too large, we investigate if it can be reduced to $O(|\Omega^+|)$ under the assumption in (4.7). The first term of (4.13) can be rewritten as

$$\sum_{j \in \Omega_i} C_{ij} v_j^2 = \sum_{j \in \Omega_i^+} (1 - p_i q_j) v_j^2 + p_i \sum_{j=1}^{n} q_j v_j^2, \tag{4.15}$$

where $\sum_{j=1}^{n} q_j v_j^2$ is independent of i and can be pre-computed in $O(n)$. Then the cost of (4.15) is $O(|\Omega_i^+|)$. For the second term in (4.13), using (4.7) and

(4.12) we have after the following calculation:

$$\sum_{j\in\Omega_i} C_{ij}\hat{R}_{ij}v_j \tag{4.16}$$

$$\begin{aligned}
&= \sum_{j\in\Omega_i^+} \hat{R}_{ij}v_j - p_i \sum_{j\in\Omega_i^+} q_j(\bar{a} - \boldsymbol{w}_i^\top \boldsymbol{h}_j + W_{it}H_{jt})v_j + p_i \sum_{j=1}^{n} q_j(\bar{a} - \boldsymbol{w}_i^\top \boldsymbol{h}_j + W_{it}H_{jt})v_j \\
&= \sum_{j\in\Omega_i^+} \hat{R}_{ij}v_j - p_i \sum_{j\in\Omega_i^+} q_j(-A_{ij} + \bar{a} + A_{ij} - \boldsymbol{w}_i^\top \boldsymbol{h}_j + W_{it}H_{jt})v_j \\
&\qquad + p_i \sum_{j=1}^{n} q_j(\bar{a} - \boldsymbol{w}_i^\top \boldsymbol{h}_j + W_{it}H_{jt})v_j \\
&= \sum_{j\in\Omega_i^+} ((1 - p_i q_j)\hat{R}_{ij} + p_i q_j(A_{ij} - \bar{a}))v_j + p_i \sum_{j=1}^{n} q_j(\bar{a} - \boldsymbol{w}_i^\top \boldsymbol{h}_j + W_{it}H_{jt})v_j.
\end{aligned} \tag{4.17}$$

If $\hat{R}_{ij}, \forall j \in \Omega_i^+$ are available, the first term can be calculated in $O(|\Omega_i^+|)$ time. The summation in the second term can be written as

$$\bar{a}\sum_{j=1}^{n} q_j v_j - \boldsymbol{w}_i^\top \sum_{j=1}^{n} q_j \boldsymbol{h}_j v_j + W_{it}\sum_{j=1}^{n} q_j H_{jt} v_j, \tag{4.18}$$

in which each summation is independent of i. The main computational task is $\sum_{j=1}^{n} q_j \boldsymbol{h}_j v_j$ that can be pre-computed in $O(nk)$. Therefore, the cost for updating $\boldsymbol{u}$ is

$$O(|\Omega^+| + nk).$$

Then the cost to go through all W and H's k columns is

$$O(|\Omega^+|k + (m+n)k^2). \tag{4.19}$$

If k is not large, in general $(m+n)k^2$ is no more than $|\Omega^+|k$. Further, if a small number S of inner iterations are taken, the $O((m+n)k^2)$ operations

needed before them becomes a smaller portion of the total cost. Therefore, our procedure for the Full approach has comparable complexity to that for the Subsampled.

In the above analysis we assume that $\hat{R}_{ij}, j \in \Omega_i^+$ are available. In Section 4.2.2.1, we show that they can be obtained in $O(|\Omega_i^+|)$ cost, the same as other operations for the first term in (4.17). We also discuss other implementation details such as column or row access of W and H.

4.2.2.1 Implementation Details

The discussion so far assumes that $\hat{R}_{ij}$ defined in (4.12) are available. Here we investigate how they can be cheaply maintained. We begin with discussing the situation for rating-based MF and then extend the result to one-class MF. We note that for $(i, j) \in \Omega$,

$$\hat{R}_{ij} = A_{ij} - \boldsymbol{w}_i^\top \boldsymbol{h}_j + W_{it} H_{jt}.$$

Directly computing $\hat{R}_{ij}$ requires $O(k)$ operations for the dot product $\boldsymbol{w}_i^\top \boldsymbol{h}_j$. Thus, the construction of (4.11) costs $O(|\Omega|k)$ operations. In [136], the time complexity can be reduced to $O(|\Omega|)$ by maintaining the following residual R_{ij}

$$R_{ij} = A_{ij} - \boldsymbol{w}_i^\top \boldsymbol{h}_j, \ \forall (i, j) \in \Omega.$$

If R_{ij} is available, then $\hat{R}_{ij}$ can be computed in $O(1)$ without the dot product:

$$\hat{R}_{ij} \leftarrow R_{ij} + W_{it} H_{jt}, \forall (i, j) \in \Omega. \tag{4.20}$$

The maintenance of R_{ij} after $\boldsymbol{u}, \boldsymbol{v}$ are obtained can also be done in $O(|\Omega|)$ as follows:

$$R_{ij} \leftarrow \hat{R}_{ij} - u_i v_j, \forall (i,j) \in \Omega.$$

Therefore, in one iteration of going through W and H's all columns, the maintenance cost of R and $\hat{R}$ is the same as that in (4.14) for updating W and H.

Now consider the one-class scenario. For the Full approach, $O(|\Omega|) = O(mn)$ is too high to maintain all R_{ij}. However, based on the observation that only $\hat{R}_{ij}$ $\forall (i,j) \in \Omega^+$ are involved in (4.17), we can store and maintain only R_{ij} $\forall (i,j) \in \Omega^+$. As a result, the total time/space cost to maintain the residual is $O(|\Omega^+|)$.

Next we discuss an issue of memory usage. In (4.17), for

$$-\boldsymbol{w}_i^\top \sum_{j=1}^{n} q_j \boldsymbol{h}_j v_j + W_{it} \sum_{j=1}^{n} q_j H_{jt} v_j, i = 1, \ldots, m, \tag{4.21}$$

both W and H's rows ($\boldsymbol{w}_i$, $\boldsymbol{h}_j$) and columns ($H_{jt}, j = 1, \ldots, n$) are needed, but in practice one does not want to double the storage by storing W (or H) in both row-oriented and column-oriented formats. Here we demonstrate that all operations can be conducted by using the column-oriented format. A careful check shows that the m values in (4.21) can be calculated by

$$-\sum_{s:s\neq t} \bar{\boldsymbol{w}}_s \bar{\boldsymbol{h}}_s^\top \begin{bmatrix} q_1 v_1 \\ \vdots \\ q_n v_n \end{bmatrix}, \tag{4.22}$$

where $\bar{\boldsymbol{w}}_s, \bar{\boldsymbol{h}}_s, s = 1, \ldots, k$ are column vectors in W and H. That is,

$$W = \begin{bmatrix} \bar{\boldsymbol{w}}_1 & \cdots & \bar{\boldsymbol{w}}_k \end{bmatrix} \text{ and } H = \begin{bmatrix} \bar{\boldsymbol{h}}_1 & \cdots & \bar{\boldsymbol{h}}_k \end{bmatrix}.$$

Note that (4.22) can be pre-calculated in $O(nk)$ before $u_i, i = 1, \ldots, m$ are updated. Further, in (4.22) all we need is to access W and H column-wisely. Regarding the calculation of $\hat{R}_{ij}$ in (4.20), we can extract the t-th column of W and H, and then go through all $(i, j) \in \Omega^+$.

4.2.2.2 Related Works

We discuss some related works. An earlier study that has reduced the complexity of ALS to (4.19) is [93]. It is indeed a combination of ALS and CD. Under fixed H, instead of calculating the closed-form solution (4.9), they solve (4.8) by a fixed number of CD iterations. It has been shown in [138, Section 3.4] that for rating-based MF, the CD procedure considered here is much faster than the approach by [93]. Besides, their procedure is more complicated for needing the eigen-decomposition of a k by k matrix.

The recent work on PU (positive-unlabeled) learning [51] has mentioned that the CD framework in [136] can be modified to have the complexity (4.19), but detailed derivations are not given.

4.2.3 Stochastic Gradient (SG)

SG has been extensively used for MF [e.g., 65]. It reformulates (4.5) to the following objective function.

$$\min_{W,H} \sum_{(i,j)\in\Omega} \ell_{ij}(A_{ij}, \boldsymbol{w}_i^\top \boldsymbol{h}_j), \tag{4.23}$$

$$\text{where} \quad \ell_{ij}(A_{ij}, \boldsymbol{w}_i^\top \boldsymbol{h}_j) = C_{ij}(A_{ij} - \boldsymbol{w}_i^\top \boldsymbol{h}_j)^2 + \frac{\lambda_i}{|\Omega_i|}\|\boldsymbol{w}_i\|^2 + \frac{\bar{\lambda}_j}{|\bar{\Omega}_j|}\|\boldsymbol{h}_j\|^2.$$

Note that the regularization term is averaged in each l_{ij} because SG would like the expectation on one single instance to be the same as the whole. Taking $\|\boldsymbol{w}_i\|^2$ as an example, we can see the equivalence as follows in (4.23):

$$\sum_{j\in\Omega_i} \frac{\lambda_i}{|\Omega_i|}\|\boldsymbol{w}_i\|^2 = \lambda_i\|\boldsymbol{w}_i\|^2.$$

Recall that in (4.6) $\lambda_i = \lambda|\Omega_i^+|$ and $\lambda_j = \lambda|\bar{\Omega}_j^+|$; therefore, in the rating-based MF, where $\Omega = \Omega^+$, we have $\lambda_i/|\Omega_i| = \bar{\lambda}_j/|\bar{\Omega}_j| = \lambda$, which is exactly the choice in the common SG update rule used in [65]. At each step SG randomly selects an entry $(i, j) \in \Omega$ uniformly and updates $\boldsymbol{w}_i$ and $\boldsymbol{h}_j$ using the partial gradient.

$$\begin{aligned}
\boldsymbol{w}_i &\leftarrow \boldsymbol{w}_i - \eta\nabla_{\boldsymbol{w}_i}\ell_{ij}(A_{ij}, \boldsymbol{w}_i^\top\boldsymbol{h}_j),\\
\boldsymbol{h}_j &\leftarrow \boldsymbol{h}_j - \eta\nabla_{\boldsymbol{h}_j}\ell_{ij}(A_{ij}, \boldsymbol{w}_i^\top\boldsymbol{h}_j),
\end{aligned}$$

where η is the learning rate and

$$\begin{aligned}
\nabla_{\boldsymbol{w}_i}\ell_{ij}(A_{ij}, \boldsymbol{w}_i^\top\boldsymbol{h}_j) =& 2C_{ij}(\boldsymbol{w}_i^\top\boldsymbol{h}_j - A_{ij})\boldsymbol{h}_j + \frac{2\lambda_i}{|\Omega_i|}\boldsymbol{w}_i,\\
\nabla_{\boldsymbol{h}_j}\ell_{ij}(A_{ij}, \boldsymbol{w}_i^\top\boldsymbol{h}_j) =& 2C_{ij}(\boldsymbol{w}_i^\top\boldsymbol{h}_j - A_{ij})\boldsymbol{w}_i + \frac{2\bar{\lambda}_j}{|\bar{\Omega}_j|}\boldsymbol{h}_j.
\end{aligned} \tag{4.24}$$

Because learning rates may significantly affect the convergence speed of SG, we adjust them by the advanced setting in [28]. The calculation in (4.24) shows that each SG update costs $O(k)$ operations. For SG usually an outer iteration refers to $|\Omega|$ updates, so the cost per iteration is

$$O(|\Omega| \times k).$$

A huge difference between Subsampled and Full occurs because $|\Omega| = |\Omega^+|$ and mn, respectively. Implementing the Full approach faces the following two challenges.

1. Because we store only $A_{ij}, (i,j) \in \Omega^+$ rather than $\bar{a}$ of Ω^-, for any picked (i,j), an efficient mechanism is needed to check if it is in Ω^+. However, the implementation is not easy. For example, it takes $O(\log |\Omega^+|)$ using binary search and $O(1)$ using hash. Neither is very efficient.
2. The $O(mnk)$ computational cost to go through the entire Ω is prohibitive.

To address the first issue, we reformulate (4.23) to

$$\sum_{(i,j)\in\Omega^+} \ell_{ij}^+(A_{ij}, \boldsymbol{w}_i^\top \boldsymbol{h}_j) + \sum_{i=1}^{m}\sum_{j=1}^{n} \ell_{ij}^-(\bar{a}, \boldsymbol{w}_i^\top \boldsymbol{h}_j), \tag{4.25}$$

where

$$\ell_{ij}^+ = C_{ij}(A_{ij} - \boldsymbol{w}_i^\top \boldsymbol{h}_j)^2 - p_i q_j (\bar{a} - \boldsymbol{w}_i^\top \boldsymbol{h}_j)^2 + \lambda \frac{|\Omega_i^+| \|\boldsymbol{w}_i\|^2}{(n + |\Omega_i^+|)} + \frac{\lambda |\bar{\Omega}_j^+| \|\boldsymbol{h}_j\|^2}{(m + |\bar{\Omega}_j^+|)},$$
$$\ell_{ij}^- = p_i q_j (\bar{a} - \boldsymbol{w}_i^\top \boldsymbol{h}_j)^2 + \frac{\lambda |\Omega_i^+| \|\boldsymbol{w}_i\|^2}{n + |\Omega_i^+|} + \frac{\lambda |\bar{\Omega}_j^+| \|\boldsymbol{h}_j\|^2}{m + |\bar{\Omega}_j^+|}.$$

Note that the regularization term is re-distributed because $mn + |\Omega^+|$ terms are now involved. We design the following procedure to perform updates on ℓ_{ij}^+ or ℓ_{ij}^-.

1. Randomly choose $u \in (0,1)$.
2. If
$$u < \frac{|\Omega^+|}{mn + |\Omega^+|},$$

randomly select $(i,j) \in \Omega^+$, and use $\nabla_{\boldsymbol{w}_i} \ell_{ij}^+$ and $\nabla_{\boldsymbol{h}_j} \ell_{ij}^+$ to update $\boldsymbol{w}_i$ and $\boldsymbol{h}_j$.

Otherwise, randomly select $(i,j) \in \{1,\ldots,m\} \times \{1,\ldots,n\}$, and use $\nabla_{\boldsymbol{w}_i} \ell_{ij}^-$ and $\nabla_{\boldsymbol{h}_j} \ell_{ij}^-$ to update $\boldsymbol{w}_i$ and $\boldsymbol{h}_j$.

The procedure effectively alleviates the issue of checking if $(i,j) \in \Omega^+$ or not. It also generates an un-biased gradient estimate to the original optimization problem (4.5) Because of the property that one of the $mn + |\Omega^+|$ terms in (4.25) is uniformly sampled at each time, the expected gradient with respect to $\boldsymbol{w}_i$ is

$$\frac{\left(\sum_{(i,j)\in\Omega^+} \nabla_{\boldsymbol{w}_i} \ell_{ij}^+(A_{ij}, \boldsymbol{w}_i^\top \boldsymbol{h}_j) + \sum_{i=1}^{m} \sum_{j=1}^{n} \nabla_{\boldsymbol{w}_i} \ell^-(\bar{a}, \boldsymbol{w}_i^\top \boldsymbol{h}_j) \right)}{mn + |\Omega|},$$

which is proportional to the gradient of the original objective function in (4.23). Further, we explain in Section 4.4.3 that ℓ_{ij}^- can be interpreted as a regularization function.

For the second challenge, unfortunately we have not devised a good strategy to reduce the mn term to $O(|\Omega^+|)$. From the investigation of ALS and CD, the key to remove the $O(mn)$ calculation is that for computing $\sum_{j=1}^{n}(\cdots)$, $\forall i$, we can reformulate it to

$$(\text{terms related to } i) \times \sum_{j} (\text{terms related to } j), \forall i;$$

see, for example, the calculation in (4.18). Then the summation over j can be pre-computed. Therefore, we must be able to aggregate things related to i (or

j) in the algorithm. This is very different from the design of SG, in which an individual (i, j) is chosen at a time. We may modify SG in various ways. For example, in (4.25), the gradient of the second summation can be calculated without the $O(mn)$ cost, so we can conduct SG updates for terms in the first summation over Ω^+, while apply regular gradient descent for the second. Another possibility is to run SG in the ALS framework. That is, when H is fixed, we apply SG to update W. Unfortunately, these modifications move SG toward other methods such as ALS. Such modifications may be unnecessary as we can directly apply ALS or CD.

One past study that has observed the difficulty of sampling from the huge number of entries is [101]. In a ranking setting they show that SG converges slowly by uniform sampling. They then propose a context-dependent setting to oversample informative entries. However, such settings do not guarantee the complexity reduction like what we achieved for ALS and CD. Further, existing methods to parallelize SG for MF such as [27, 37] may become not applicable because A is split into blocks. In contrast, ALS and CD under our new settings for one-class MF can be easily parallelized.

Based on the discussion, SG may be less suitable for the Full approach. We experimentally confirm this result in Section 4.3.

4.3 Experimental Results

Our experiments include two parts. The first part is the comparison of the proposed optimization methods for the Full approach (Section 4.3.3).

Table 4.3: Data statistics for training and test sets. Zero columns/rows in A are removed, so m and n may be different from those of the original data. Data sets movielens1m and movielens10m are movielens with 1m and 10m (user, item) pairs.

	delicious	movielens1m	movielens10m	netflix	yahoo-music
m	2,000	6,040	69,878	480,198	1,000,990
n	3,000	3,952	10,677	17,770	624,961
$\|\Omega^+\|$	197,130	517,770	4,505,820	51,228,351	82,627,856
$\|\Omega^+_{\text{test}}\|$	49,306	57,511	499,864	5,690,839	9,178,962

In the second part we investigate in Section 4.3.4 the performances of the following approaches for one-class MF:

- Full: in the experiments for Full approaches, we consider a simplified setting of (4.7) as follows.

$$C_{ij} = \begin{cases} 1 & \forall (i,j) \in \Omega^+, \\ \alpha, & \forall (i,j) \in \Omega^-. \end{cases} \tag{4.26}$$

  The selection of the parameter α will be discussed in Section 4.4.2.

- Subsampled
- Ensemble: the ensemble of models from the Subsampled approach. See details in Section 4.3.1.1
- BPR: The approach in [102] by considering an AUC (i.e., rank-based) loss; see details in Section 4.3.1.2. We use the implementation in [29].

A focus is to compare the two techniques Full and Subsampled for selecting negative entries.

The only publicly available one-class MF data that we are aware of is

delicious, which is from [89].[3] We use the same 4-to-1 training/test split by [89] in our experiment.[4] Besides this data, following past works [63], we modify some rating-based MF data listed in Table 4.3 for experiments. We consider observed entries with ratings ≥ 4 as positive.[5] For some problems, training and test sets are available, but the test sets are too small. Therefore, other than delicious, we merge training and test sets of every problem first, and then do a 9-to-1 split to obtain training/test sets for our experiments.

Before showing experimental results we present implementation details and evaluation criteria in Sections 4.3.1 and 4.3.2, respectively.

4.3.1 Implementation Details

We tried the best to have efficient implementations for each optimization method. Here we show some places of applying highly-tuned linear algebra packages. When $q_j = \alpha,\ \forall j$, $\sum_{j=1}^{n} q_j \boldsymbol{h}_j \boldsymbol{h}_j^\top$ in ALS is essentially the product $\alpha H H^\top$ so we employ fast matrix-matrix operations in optimized BLAS (Basic Linear Algebra Subprograms). We use efficient `posv()` subroutine in the optimized LAPACK (Linear Algebra PACKage) to solve the system of linear equations in (4.9).[6] All three methods in Section 4.2 can be parallelized, but to focus on algorithmic differences, we use single-core implementations in our

[3] A tag recommendation data set from ECML'09 challenge is used in [101], but it is in fact a multi-label data set, where the tagging behavior does not reveal user preference. Thus, this data set is not suitable in our experiments. Furthermore, the data set is relatively small.

[4] The set delicious can be found at `http://www.rongpan.net/data/delicious.tar.bz2`. We use the first 4-to-1 training/test split in our experiments.

[5] For yahoo-music, scores ≥ 80 are considered as positive.

[6] In all timing experiments, ATLAS [128] is used as the optimized BLAS/LAPACK.

experiments.

4.3.1.1 Subsampled and Ensemble: Sampling Schemes

Let $\texttt{multinomial}(\{p_i\})$ be the distribution such that the i-th user index is selected with probability p_i and $\texttt{multinomial}(\{q_j\})$ be the distribution such that the j-th item index is selected with probability q_j. Most sampling schemes considered for the Subsampled approach in the past [89, 90] can be described by the following procedure:

- Sample $i' \sim \texttt{multinomial}(\{p_i\})$
- Sample $j' \sim \texttt{multinomial}(\{q_j\})$
- Add (i', j') into Ω^-

Following the discussion in Section 4.1.2, we give the detailed specification for the five Subsampled variants and the Ensemble approach compared in the experiments:

Sampling scheme	$\{p_i\}$	$\{q_j\}$
user	$p_i \propto \lvert\Omega_i^+\rvert$	$q_j = 1/n$
item-f	$p_i = 1/m$	$q_j \propto \lvert\bar{\Omega}_j^+\rvert$
item-w	$p_i = 1/m$	$q_j \propto m - \lvert\bar{\Omega}_j^+\rvert$
item-s	$p_i = 1/m$	$q_j \propto 1/\lvert\bar{\Omega}_j^+\rvert$
uniform	$p_i = 1/m$	$q_j = 1/n$

For the Ensemble approach [89], 20 Subsampled models with the uniform sampling scheme are aggregated to generate the ensemble model as follows:

$$A^{\mathsf{Ensemble}} = \frac{1}{20} \sum_{s=1}^{20} W_s H_s^\top,$$

where (W_s, H_s) is the s-th Subsampled model. The result for Ensemble is derived from the ranking induced by A^{Ensemble}.

4.3.1.2 Some Details of BPR

BPR is an approach for one-class collaborative filtering by solving the following optimization problem:

$$\min_{W,H} \sum_{i=1}^{m} \sum_{(j_+,j_-)\in\Omega_i^+\times\Omega_i^-} \Big\{ \log\big(1+\exp\big(-\boldsymbol{w}_i^\top(\boldsymbol{h}_{j_+}-\boldsymbol{h}_{j_-})\big)\big) + \frac{\lambda}{2}\big(\|\boldsymbol{w}_i\|^2+\|\boldsymbol{h}_{j_+}\|^2+\|\boldsymbol{h}_{j_-}\|^2\big)\Big\}. \tag{4.27}$$

In (4.27), all the pairs between observed items and unknown items are included into the formulation by a logistic loss. It can be seen as an approach to minimize the empirical AUC [102]. As the number of pairs is very large, [102] propose to apply SG to solve (4.27). At each SG step for BPR, a pair is selected randomly for the update. In our experiments, we use the BPR implementation available in LIBMF [29].

4.3.2 Evaluation Criteria

For the i-th user, let $\Omega_{\text{test}}(i)$ be the set of candidate items whose preference is predicted and evaluated in the testing phase. We exclude Ω_i^+ because the model has been trained to fit data in Ω^+. Thus $\Omega_{\text{test}}(i) = [n] \setminus \Omega_i^+$. We further denote $\Omega_{\text{test}}^+(i) \subset \Omega_{\text{test}}(i)$ as the subset of items which receive a positive response from the i-th user. Let

$$\pi_i : \Omega_{\text{test}}(i) \to \{1, \ldots, |\Omega_{\text{test}}(i)|\}$$

be the ranking predicted by the recommender system (i.e., $\pi_i(j)$ = rank of the j-th item) and $\pi_i^{-1}(r)$ be the item with rank r; that is, $\pi_i(\pi_i^{-1}) = r$. We consider the following evaluation criteria.

- nDCG@p: normalized discounted cumulated gain. The following criterion is widely used to evaluate the performance of the top p-ranked items.

$$\texttt{DCG}_i@p(\pi_i) = \sum_{r=1}^{p} \frac{[\pi_i^{-1}(r) \in \Omega^+_{\text{test}}(i)]}{\log_2(1+r)}.$$

  This value might be in a different range depending on p and $|\Omega^+_{\text{test}}(i)|$, so people usually consider the normalized DCG as follows:

$$\texttt{nDCG}_i@p = 100 \times \frac{\texttt{DCG}_i@p(\pi_i)}{\texttt{DCG}_i@p^{\max}},$$

  where $\texttt{DCG}_i@p^{\max} = \max_\pi \texttt{DCG}_i@p(\pi)$. In this chapter, we report the average nDCG@p, i.e., $\sum_i \texttt{nDCG}_i@p/m$, among all users.

- nHLU: normalized half life utility. HLU was first proposed in [17, Eq. 5] to evaluate the performance of recommender systems. For one-class recommender systems (i.e., the ground truth rating is either 1 or 0), the HLU for a user, $\texttt{HLU}_i(\pi_i)$, can be defined as follows

$$\texttt{HLU}_i(\pi_i) = \sum_{j \in \Omega^+_{\text{test}}(i)} \frac{1}{2^{(\pi_i(j)-1)/(\beta-1)}},$$

  where β is the "half life" parameter denoting the rank of the item on the list such that there is a 50-50 chance the i-th user will review that item.[7]

[7]Note that there is a typo in the description of HLU in [88, 89], where "/" is missing.

Following the usage in [17, 88, 89], $\beta = 5$ is used in our experiments. Similar to nDCG, the normalized half life utility can be defined as follows:

$$\texttt{nHLU}_i = 100 \times \frac{\texttt{HLU}_i(\pi_i)}{\texttt{HLU}_i^{\max}},$$

where $\texttt{HLU}_i^{\max} \equiv \max_\pi \texttt{HLU}_i(\pi)$. In this chapter, we report the average nHLU, $\sum_i \texttt{nHLU}_i/m$, among all users.[8]

- MAP: mean average precision. For the i-th user, the average precision $\texttt{AP}_i$ is defined as follows:

$$\texttt{AP}_i = \frac{\sum_{j \in \Omega^+_{\text{test}}(i)} 100 \times \frac{|\{j' \in \Omega^+_{\text{test}}(i) : \pi_i(j') \le \pi_i(j)\}|}{\pi_i(j)}}{|\Omega^+_{\text{test}}(i)|}.$$

  In this chapter, we report MAP, which is $\sum_i \texttt{AP}_i/m$.

- AUC: area under the ROC curve. For the i-th user, this is equivalent to the ratio of violating pairs among all the pairs from $\Omega^+_{\text{test}}(i) \times \Omega^-_{\text{test}}(i)$, and can be computed as follows [102]:

$$\texttt{AUC}_i = \frac{|\{(j, j') \in \Omega^+_{\text{test}}(i) \times \Omega^-_{\text{test}}(i) : \pi_i(j) > \pi_i(j')\}|}{|\Omega^+_{\text{test}}(i)| \times |\Omega^-_{\text{test}}(i)|}.$$

  In this chapter, we report the average AUC, $\sum_i \texttt{AUC}_i/m$, among all users.

Among these four evaluation criteria, nDCG and nHLU are more appropriate to evaluate the ranking performance of recommendation systems, as they put more weights on the top-ranked items. On the other hand, AUC is less appropriate because of its insensitivity to the position of the violating pairs.

Table 4.4: Range of parameters considered for each one-class MF approach. For each individual Subsampled model in the Ensemble approach, the best combination of parameters selected for the Subsampled approach is used.

Parameter	Subsampled	Full	BPR
λ	$\{10^{-4}, ..., 10^{-1}\}$	$\{10^{-4}, ..., 10^{-1}\}$	$\{10^{-8}, ..., 10^{-1}\}$
k	$\{2^4, 2^5, 2^6\}$	$\{2^4, 2^5, 2^6\}$	2^6
$C_{ij} = \alpha, \forall (i,j) \notin \Omega^+$	1	$\{2^{-5}, 2^{-3}, 2^{-1}, 2^0\}$	1
$\lvert\Omega^-\rvert / \lvert\Omega^+\rvert$	$\{1, 2\}$	-	-
sampling schemes	user/item-f/item-w/item-s uniform/uniform-ens	-	-

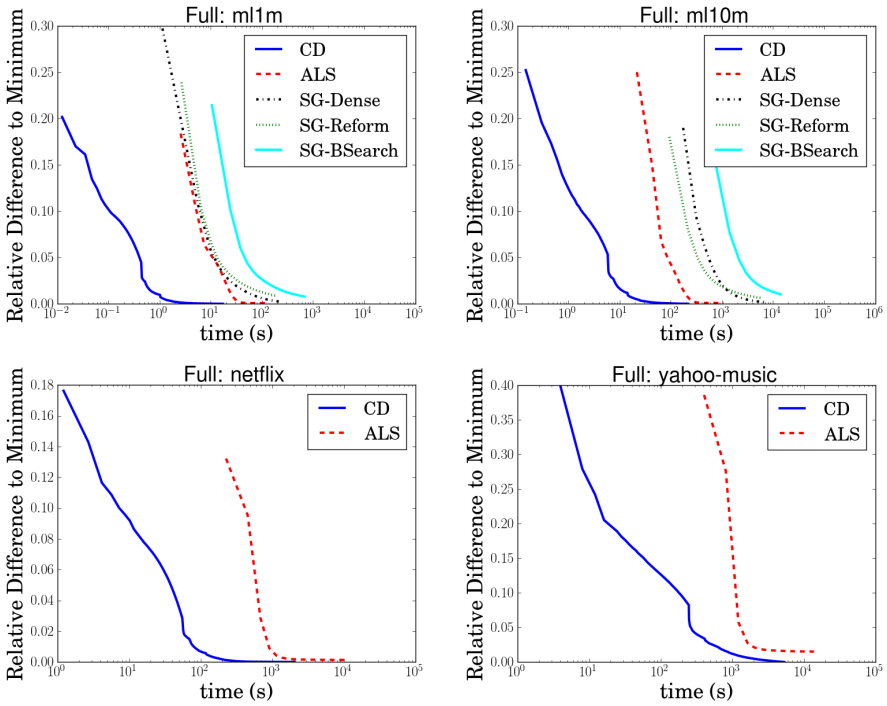
4.3.3 Comparison: Optimization Methods for Full

We compare the proposed optimization methods in Section 4.2 for the Full approach. For ALS and CD, only the proposed procedures are presented because from the complexity analysis they are clearly superior to the direct implementation on all mn values. For SG, we consider some variants for checking the effectiveness of the reformulation (4.25).

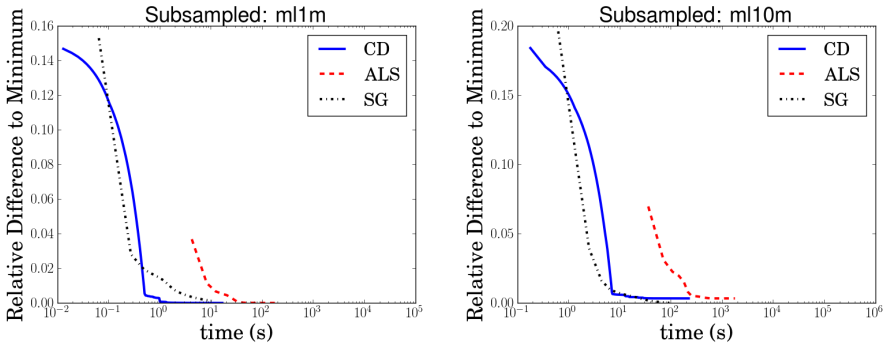
1. SG-Dense: the original formulation (4.23) is used and the whole Ω set (mn elements) is stored for easily checking if $(i,j) \in \Omega^+$ or not.
2. SG-BSearch: we still use (4.23), but do not store Ω. A binary search is conducted to check if (i,j) is in Ω^+.
3. SG-Reform: the reformulation (4.25) is used.

Because the goal is to compare methods for the same optimization problem, we simply set $\alpha = 0.1$ in (4.26). That is, $C_{ij} = 0.1$, $\forall (i,j) \notin \Omega^+$ and 1 otherwise.

[8]This is slightly different from the original formulation in [17], where $100 \times \frac{\sum_i \texttt{HLU}_i(\pi_i)}{\sum_i \texttt{HLU}_i^{\max}}$ is reported.



(a) Full (SG may be too slow to be shown)



(b) Subsampled

Figure 4.1: Comparison of optimization methods. The y-axis is the relative difference to the minimal value obtained among all the methods, while the x-axis is the running time in log-scale. See Section 4.3.3 for details of SG variants.

Other settings include $k = 64$ and $\lambda = 0.1$.

In Figure 4.1(a), we present the relation between running time (log-scaled) and the relative difference to the minimum objective function value obtained among all the algorithms. Results show that CD is much faster than ALS. This observation is expected because past studies for general MF has shown the superiority of CD. Now the mn terms in the complexity analysis of both methods are reduced to $O(|\Omega^+|)$, so the relationship still holds.

For smaller data sets movielens1m and movielens10m, SG-Dense is applicable by storing all mn elements of Ω. We observe that SG-Reform is close to SG-Dense, so our reformulation in (4.25) is effective. SG-BSearch is the slowest because of the binary search on checking if $(i, j) \in \Omega^+$. Unfortunately, all the SG variants are significant slower than CD and ALS for larger problems like netflix or yahoo-music, for which curves cannot even be generated in Figure 4.1(a).

Although we have explained in Section 4.2 that SG may be less suitable for Full, the very poor results are still somewhat surprising. Therefore, we investigate the running speed when these methods are applied to the Subsampled approach. We consider $|\Omega^-| = |\Omega^+|$, uniform sampling strategy, and the optimization problem (4.2). The comparison results are in Figure 4.1(b). Clearly SG becomes as fast as CD and much better than ALS. This result is consistent with past studies on rating-based MF. We thus conclude that SG fails for the Full approach because of the large mn elements. Another result by comparing Figures 4.1(a) and 4.1(b) is that CD and ALS take similar time

for Full and Subsampled. Therefore, our study enables the Full approach to be computationally feasible for one-class collaborative filtering.

4.3.4 Comparison: One-class MF Approaches

We compare one-class MF approaches listed in the beginning of this section. After finding that Full is better in initial experiments, we consider the following settings to check if the difference between Full and others is significant.

- For Full, we conduct parameter selection for each evaluation criterion by splitting the training set to two parts for training (90%) and evaluation (10%). The parameters achieving the best validation results are used to train the final model. We then report the performance on the test set.
- For Subsampled and BPR, we omit the validation procedure. Instead, for each evaluation criterion, we consider all parameter combinations (including several *sampling strategies* for Subsampled) and report the best test result. Therefore, we overfit the test set to get an optimistic estimate of the performance.
- For Ensemble, we combine 20 Subsampled models [89]. We use the same best parameter selected for Subsampled for each individual model.

To see if Full is significantly better, we deliberately overestimate the performance of others. Parameters used for each method are in Table 4.4. In the testing phase Ω^+ is excluded because they may have been well fitted in training.

Table 4.5 shows that for nDCG, nHLU, and MAP, Full is significantly better than all other approaches. For AUC, all approaches give similar values close to one. For a recommender system, AUC is less suitable because it considers the overall number of violating pairs without emphasizing the position of top-ranked (i.e., recommended) items.

Although Ensemble improves the performance of the pure Subsampled approach on movielens1m, movielens10m, and netflix, the performance gap between Ensemble and Full is still large in Table 4.5. This observation is very different from the finding in [89], where Ensemble is able to yield competitive performance as the Full approach. Based on the observation in Table 4.5 that the larger the size of the data set, the larger the gap between Ensemble and Full, we think that the finding regarding the Ensemble performance in [89] only holds for small data sets.[9]

Note that for the data set delicious used in [89], here we see a bigger performance gap between Full and Ensemble. The reason might be that first our k is 64 rather than their 16, and second we have selected the parameter α for Full by a validation procedure.

Test time is an important concern for one-class MF because for each user i, we must calculate $\boldsymbol{w}_i^\top \boldsymbol{h}_j, \forall j$ to find the top items. The Ensemble approach particularly suffers from lengthy test time because the use of 20 models causes a 20-fold increase of the prediction cost. For example, the test time for

[9]The statistics of two data sets used in [89] are $m = 3,158; n = 1,536; |\Omega^+| = 84,117$ and $m = 3,000; n = 2,000; |\Omega^+| = 246,436$, respectively.

Table 4.5: Comparison of one-class MF approaches. The best performed method for each criterion is bold-faced.

		nDCG @1	nDCG @10	nHLU	MAP	AUC
delicious	Subsampled	41.40	27.74	26.58	14.37	0.86687
	Ensemble	41.64	28.13	27.00	14.56	0.85218
	BPR	20.70	27.49	27.71	16.32	0.88835
	Full	**56.05**	**38.53**	**36.86**	**21.51**	**0.89182**
movielens1m	Subsampled	14.69	14.94	15.79	11.17	0.93743
	Ensemble	20.60	19.00	19.73	13.82	0.94594
	BPR	7.37	15.42	16.65	11.00	0.94463
	Full	**28.91**	**23.66**	**24.11**	**16.35**	**0.94536**
movielens10m	Subsampled	9.33	12.10	13.31	10.00	0.97293
	Ensemble	12.96	15.47	16.73	12.30	0.97633
	BPR	16.48	16.66	17.90	12.20	**0.97706**
	Full	**25.64**	**23.81**	**24.94**	**17.70**	0.97372
netflix	Subsampled	10.62	11.27	12.03	8.91	0.97224
	Ensemble	15.15	15.16	15.83	11.13	0.97579
	BPR	18.40	15.97	16.19	10.36	**0.97587**
	Full	**27.04**	**22.62**	**22.72**	**13.95**	0.96962
yahoo-music	BPR	12.41	12.05	12.62	8.29	**0.99520**
	Full	**38.64**	**34.71**	**35.26**	**26.75**	0.99185

yahoo-music is about 70 times more than the time required for netflix, which is the reason why we omit the experiments for Subsampled. Developing methods to reduce the test time is an important future issue. See more discussions in Section 4.4.5.

4.3.5 Analysis of Sampling Schemes for the Subsampled Approach

The numbers reported for the Subsampled approach in Table 4.5 are the best results from all the sampling schemes. Table 4.6 shows the detailed

Table 4.6: Comparison among various sampling schemes for the Subsampled approach. See detailed descriptions for each sampling scheme in Section 4.3.1.1.

(a) delicious

Scheme	nDCG @1	nDCG @5	nDCG @10	nHLU	MAP	AUC
user	41.40	32.62	27.74	26.58	14.37	0.85985
item-f	1.55	1.84	1.89	1.98	2.04	0.69648
item-w	39.15	31.21	26.63	25.65	14.15	0.86687
item-s	29.65	24.05	21.92	21.35	12.26	0.84855
uniform	38.40	30.94	26.05	25.29	13.94	0.86578

(b) movielens1m

Scheme	nDCG @1	nDCG @5	nDCG @10	nHLU	MAP	AUC
user	13.16	11.54	12.23	12.87	9.14	0.93290
item-f	0.78	0.94	1.17	1.33	1.47	0.78903
item-w	13.56	13.67	14.74	15.67	11.12	0.93723
item-s	7.88	7.08	8.06	8.53	5.80	0.90470
uniform	14.69	13.89	14.94	15.79	11.17	0.93743

(c) movielens10m

Scheme	nDCG @1	nDCG @5	nDCG @10	nHLU	MAP	AUC
user	10.55	9.55	10.17	10.87	8.17	0.97250
item-f	0.91	1.36	1.93	2.27	1.95	0.85310
item-w	9.47	9.99	11.96	13.18	9.92	0.97304
item-s	2.84	3.29	4.21	4.81	4.35	0.95693
uniform	9.33	10.07	12.10	13.31	10.00	0.97293

(d) netflix

Scheme	nDCG @1	nDCG @5	nDCG @10	nHLU	MAP	AUC
user	10.26	9.61	9.96	10.48	7.54	0.97192
item-f	1.26	1.74	2.07	2.27	1.66	0.85662
item-w	10.43	10.02	11.01	11.77	8.75	0.97212
item-s	4.45	4.70	5.37	5.80	4.69	0.95741
uniform	10.62	10.22	11.27	12.03	8.91	0.97224

comparison among various sub-sampling schemes. We can make the following observations:

- The "uniform" scheme performs the best in general, and the "user" scheme is also competitive among all sampling schemes. This result is slightly different from the conclusion in [89], where the "user" scheme is slightly better than the "uniform" scheme.
- The "item-f" scheme is considered as a baseline in [90]. However, this scheme gives the worst performance among all schemes considered here.

4.4 Discussions

In this section before making conclusions we make some further analysis of experimental results and point out some future issues for investigation.

4.4.1 Some Explanation about the Superiority of Full over Subsampled

We give possible explanations about the superiority of Full over Subsampled. For Subsampled, by selecting only a small subset of missing entries as negative data, we fit them by using W and H but ignore others. Then the generalization ability may not be good. The importance of considering all missing data can also be seen in the best α selected by the validation procedure; see (4.26) for the definition of α. Because of the large number of negative data, we expect that α should be small. Interestingly, we find that α can be neither too small nor too large. A too small α causes W and H to underfit negative missing data, while a too large α causes W and H to wrongly fit positive missing data. In the theoretical study of PU (positive-unlabeled) learning[10] [51], the authors were able to prove an error bound under the setting of

$$\alpha = \frac{\bar{\rho}}{(2 - \bar{\rho})},$$

where $\bar{\rho}$ is the percentage of positive entries that are observed. When $\bar{\rho} \to 1$, most missing entries are negative, so $\alpha \approx 1$ causes $\boldsymbol{w}_i^\top \boldsymbol{h}_j$ to fit 0 for missing data. In contrast, when $\bar{\rho} \to 0$, some missing entries are indeed positive, so we should not let $\boldsymbol{w}_i^\top \boldsymbol{h}_j$ be very close to zero. Therefore, α should be smaller.

[10] One-class MF is a method for PU matrix completion.

4.4.2 Performance versus Various Values of α

To better understand the influence of the α parameter in (4.26), we generate a synthetic ground truth matrix A as follows:

- $W \leftarrow \texttt{rand}(m, k)$ and $H \leftarrow \texttt{rand}(n, k)$
- $\bar{A} \leftarrow WH^\top$
- $A_{ij} \leftarrow \begin{cases} 1 & \text{if } \bar{A}_{ij} \geq \bar{a}, \\ 0 & \text{otherwise,} \end{cases}$ where $\bar{a}$ is the value such that

$$|\{\bar{A}_{ij} \geq \bar{a}\}| = 0.2 \times mn.$$

As mentioned earlier, theoretical results in [51] suggest that the best α is a function of $\bar{\rho}$, which is the ratio of the observed positive entries over the entire positive entries. For a given $\bar{\rho}$, we construct a training set $\Omega^+(\bar{\rho})$ by sampling positive entries from A such that

$$\frac{|\Omega^+(\bar{\rho})|}{|\{(i,j) : A_{ij} = 1\}|} = \bar{\rho}.$$

As the ground truth is available in the synthetic data set, we report the ranking performance over the entire item set. That is, $\Omega_{\text{test}}(i) = \{1, \ldots, n\}, \forall i$ because we do not exclude Ω_i^+.

Figures 4.2–4.3 show the results on synthetic matrices A of various sizes ($m = n = 500$, $m = n = 1,000$, and $m = n = 5,000$) with $k = 10$. The x-axis denotes the value of α used in the optimization problem, and we report results for $\alpha \in \{2^{-20}, 2^{-18}, \ldots, 2^8\}$. The y-axis denotes the performance for each α with the best λ selected from $\{10^{-12}, \ldots, 10^{-1}\}$. Each curve in Figures 4.2–4.3 corresponds to a pair of training and test sets generated under a specific

$\bar{\rho}$. Four values of $\bar{\rho}$ are considered: 0.95, 0.1, 0.05, and 0.01. We make the following observations:

- The shape of curves for $\bar{\rho} = 0.1$ and $\bar{\rho} = 0.05$ is concave, which means that the best α cannot be too large or too small. The result reconfirms the discussion in Section 4.4.1. Therefore, suitable validation procedures are required to get the best performance.
- The theoretical results from [51] suggest that with an appropriate choice of λ, thresholding parameter $\bar{a}$ and

$$\alpha = \frac{\bar{\rho}}{2 - \bar{\rho}},$$

the Full approach can recover the ground truth binary matrix A from $WH^\top$ obtained by (4.4) with high probability (i.e., high point-wise accuracy). This suggests that if $\bar{\rho}$ is close to 1, the best α should also close to 1; on the other hand, if $\bar{\rho}$ is close to 0, the best α should also close to 0. Although the evaluation criteria considered here is not the point-wise accuracy considered in [51], we can still observe a similar trend on these ranking-based criteria.

4.4.3 Regularization for One-Class MF

In this section, we explain what we claimed earlier in Section 4.2.3 that the second term of (4.25), $\sum_i \sum_j \ell_{ij}^-$, can be considered as a regularization function for one-class MF. Based on the definition on ℓ_{ij}^-, it is not hard to see

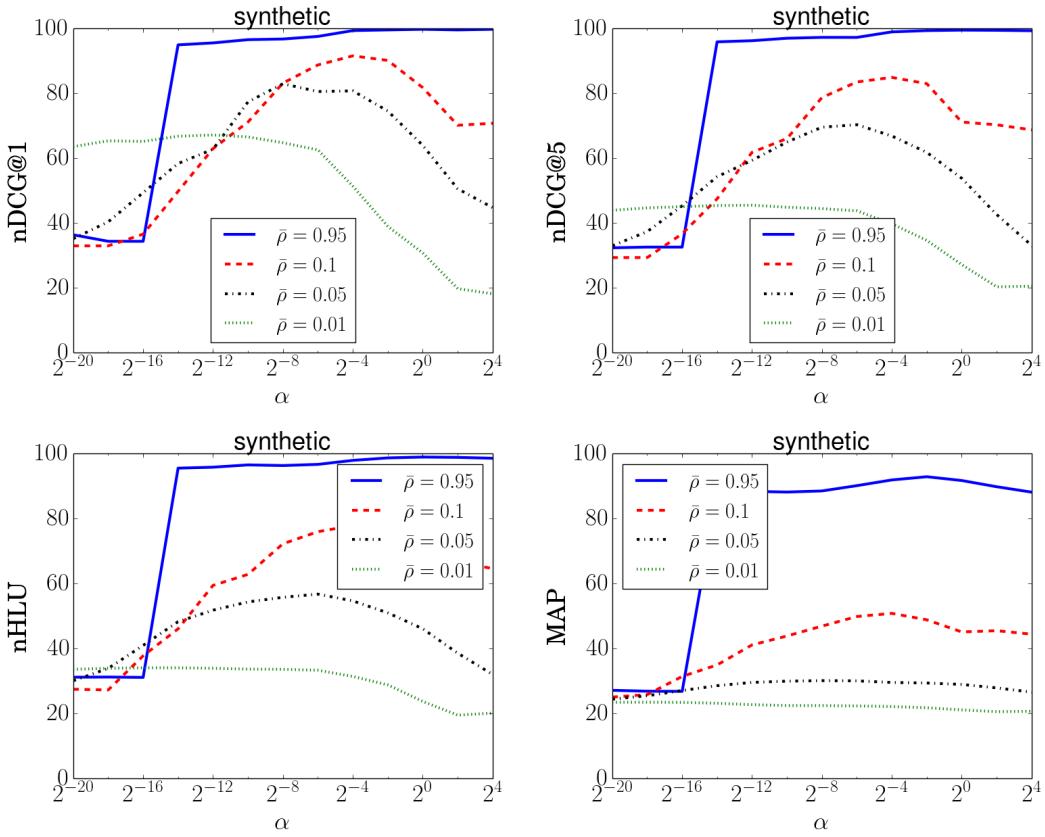


Figure 4.2: Performance versus various values of α on synthetic data sets with $m = n = 500$.

that

$$\sum_{i=1}^{m}\sum_{j=1}^{n} \ell_{ij}^{-}(\bar{a}, \boldsymbol{w}_i^\top \boldsymbol{h}_j) = \|C_w(WH^\top - \bar{a}\mathbf{1}_m \times \mathbf{1}_n^\top)C_h\|_F^2 + \lambda\|D_w W\|_F^2 + \lambda\|D_h H\|_F^2, \tag{4.28}$$

where $\mathbf{1}_m \times \mathbf{1}_n^\top$ is the all one matrix, $C_w = \sqrt{\texttt{diag}(\boldsymbol{p})}$ and $C_h = \sqrt{\texttt{diag}(\boldsymbol{q})}$ are diagonal matrices with

$$(C_w)_{ii} = \sqrt{p_i},\ \forall i, \quad \text{and} \quad (C_h)_{jj} = \sqrt{q_j},\ \forall j$$

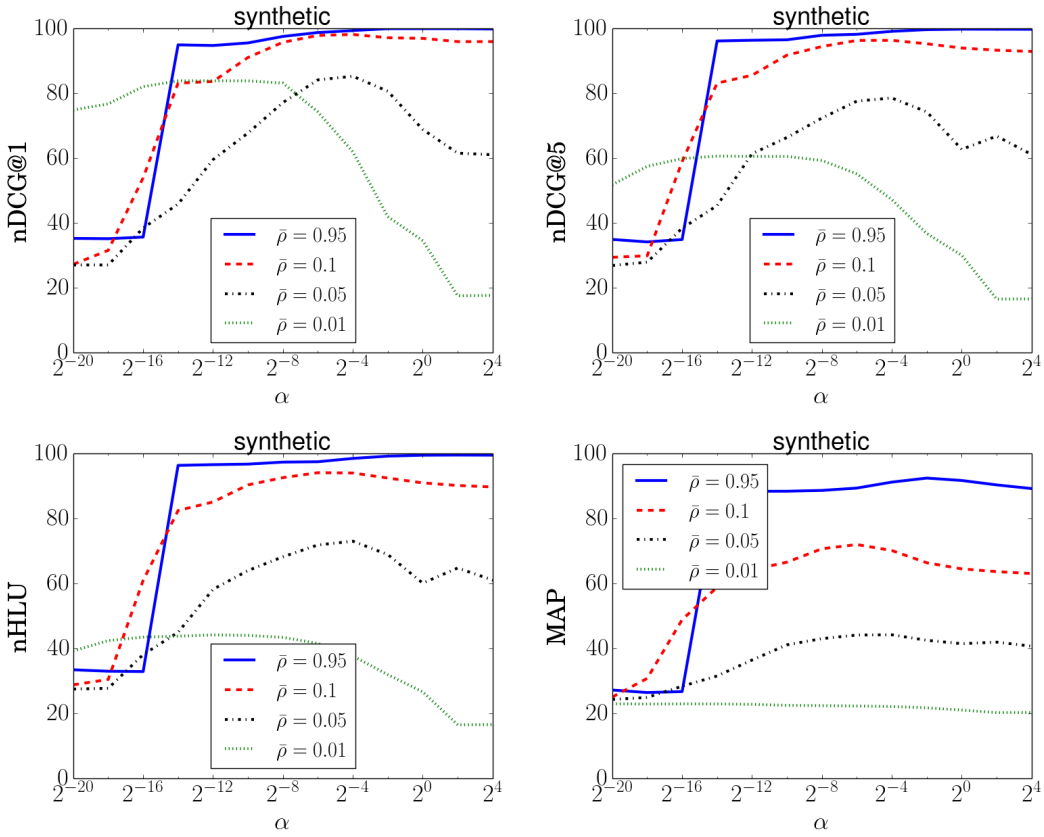


Figure 4.3: Performance versus various values of α on synthetic data sets with $m = n = 1,000$.

and D_w and D_h are diagonal matrices[11] with

$$(D_w)_{ii} = \sqrt{n|\Omega_i^+|/\big(n + |\Omega_i^+|\big)}, \ \forall i \quad \text{and} \quad (D_h)_{jj} = \sqrt{m|\bar{\Omega}_j^+|/\big(m + |\bar{\Omega}_j^+|\big)}, \ \forall j.$$

As we can see, (4.28), derived from the summation of ℓ^- terms, can be regarded as a special regularization function on the model W and H which encourages

[11] If all the regularization are distributed to ℓ_{ij}^- terms in the reformulation (4.25), $(D_w)_{ii} = \sqrt{\lambda/\lambda_i}$ and $(D_h)_{jj} = \sqrt{\lambda/\bar{\lambda}_j}$.

the structure that all the missing entries are close to $\bar{a}$. From this point of view, the Full approach can be considered as an empirical risk minimization problem in the following form:

$$\sum_{(i,j)\in\Omega^+} \ell^+(A_{ij}, \boldsymbol{w}_i^\top \boldsymbol{h}_h) + \mathcal{R}(W, H),$$

where the loss function ℓ^+ only applies to the explicitly observed entries, while the implicit responses are captured by the regularization $\mathcal{R}(W, H)$. Interestingly, this interpretation to incorporate implicit responses into regularization also fits in a recent proposed implicit matrix factorization approach [40], where the likelihood of A for a given model (W, H) is

$$\log \mathbb{P}[\![A \mid W, H]\!] = \left(\sum_{(i,j):A_{ij}>0} A_{ij} \log(\boldsymbol{w}_i^\top \boldsymbol{h}_j) - A_{ij}! \right) - \boldsymbol{e}^\top W H^\top \boldsymbol{e}, \quad (4.29)$$

where $A_{ij}!$ is the factorial of the integer rating A_{ij}, and the corresponding MLE problem is

$$\min_{W,H} \quad \sum_{(i,j):A_{ij}>0} \underbrace{-A_{ij} \log(\boldsymbol{w}_i^\top \boldsymbol{h}_j)}_{\ell_{ij}^+} + \underbrace{\boldsymbol{e}^\top W H^\top \boldsymbol{e}}_{\mathcal{R}(W,H)}. \quad (4.30)$$

We can clearly see that the first term of (4.30) corresponds to the Poisson loss for the explicitly observed positive ratings, while the second term can be regarded a regularizer accounting for the implicit responses.

4.4.4 Connection to word2vec

Recently, word2vec [80] for NLP applications identifies a latent vector representing each word from a set of observed word-context pairs. It can be

considered as a one-class MF problem [67]. Currently negative entries are selected by the Subsampled approach. In particular, the skip gram negative sampling (SGNS) objective [67] tries to maximize the probability for the observed word-context pairs while simultaneously maximizing that for unobserved (i.e., negative) pairs. Under the assumption that a randomly selected context for a given word is likely not an observed pair, SGNS randomly subsamples a few "negative" contexts from the entire set of contexts for each observed word-context pair. One can see that SGNS is essentially a Subsampled approach for word embedding learning. Thus, our investigation on Subsampled versus Full may be useful for this NLP technique. It is also interesting to ask whether the efficient techniques developed in this chapter can be extended to handle non squared-L_2 loss functions such as the one used in word2vec.

4.4.5 Computational Issue in Prediction

With the proposed efficient algorithms in Section 4.2, the time complexity of the training procedure for one-class MF is only linear in $|\Omega^+|$ for both Full and Subsampled approaches. However, item prediction remains a computational challenge. Given a recommender model (W, H) with m users and n items, item prediction for the i-th user requires the ranking or the maximum over $n - \left|\Omega_i^+\right|$ inner products (i.e, $\boldsymbol{w}_i^\top \boldsymbol{h}_j,\ j \in [n] \setminus \Omega_i^+$), which is close to $O(nk)$ as $\left|\Omega_i^+\right|$ is usually small. Thus, item prediction for all the m users requires $O(mnk)$ operations, which can be orders of magnitude more than the training phase. There have been a few recent works about how to reduce the

time complexity of searching the maximum of inner products [9, 10, 85, 104].

4.5 Summary of the Contributions

In this chapter, we have developed efficient techniques to solve the hard optimization problem of the Full approach, which treats every missing entry as negative. We then conduct thorough experiments to show that the Full approach gives much better performances than the Subsampled approach. Therefore, our work has made the Full approach very useful for large-scale one-class matrix factorization. This work is presented in [142]. We have also extended the one-class MF work to incorporate various types of side information [143].

Chapter 5

LEML: Low-rank Structured Multi-label Learning with Missing Values

Large scale multi-label classification is an important learning problem with several applications to real-world problems such as image/video annotation [22, 122] and query/keyword suggestions [3]. The goal in multi-label classification is to predict a label vector $\boldsymbol{y} \in \{0,1\}^L$ for a given data point $\boldsymbol{x} \in \mathbb{R}^d$. This problem has been studied extensively in the domain of structured output learning, where the number of labels is assumed to be small and the main focus is thus, on modeling inter-label correlations and using them to predict the label vector [45].

Due to several motivating real-life applications, recent research on multi-label classification has largely shifted its focus to the other end of the spectrum where the number of labels is assumed to be extremely large, with the key challenge being the design of scalable algorithms that offer real-time predictions and have a small memory footprint. In such situations, simple methods such as 1-vs-all or Binary Relevance (BR), that treat each label as a separate binary classification problem fail miserably. For a problem with (say) 10^4 labels and

The materials in this Chapter have been published in [139]. I proposed the problem formulation, developed the algorithms and conducted the experiments.

10^6 features, which is common in several applications, these methods have a memory footprint of around 100 Gigabytes and offer slow predictions.

A common technique that has been used to handle the label proliferation problem in several recent works is "label space reduction". The key idea in this technique is to reduce the dimensionality of the label-space by using either random projections or canonical correlation analysis (CCA) based projections [25, 53, 61, 115]. Subsequently, these methods perform prediction on the smaller dimensional label-space and then recover the original labels by projecting back onto the high dimensional label-space. In particular, [25] recently proposed an efficient algorithm with both label-space and feature-space compression via a CCA type method with some orthogonality constraints. However, this method is relatively rigid and cannot handle several important issues inherent to multi-label problems; see Section 5.1.1 for more details.

In this chapter we take a more direct approach by formulating the problem as that of learning a low-rank linear model $Z \in \mathbb{R}^{d \times L}$ s.t. $\boldsymbol{y}^{pred} = Z^\top \boldsymbol{x}$. We cast this learning problem in the standard ERM framework that allows us to use a variety of loss functions and regularizations for Z. This framework unifies several existing dimension reduction approaches. In particular, we show that if the loss function is chosen to be the squared-L_2 loss, then our proposed formulation has a closed form solution, and surprisingly, the conditional principal label space transformation (CPLST) method of [25] can be derived as a *special case*. However, the flexibility of the framework allows us to use other loss functions and regularizers that are useful for preventing overfitting and

increasing scalability.

Moreover, we can extend our formulation to handle missing labels; in contrast, most dimension reduction formulations (including CPLST) cannot accommodate missing labels. The ability to learn in the presence of missing labels is crucial as for most real-world applications, one cannot expect to accurately obtain (either through manual or automated labeling) all the labels for a given data point. For example, in image annotation, human labelers tag only prominent labels and typically miss out on several objects present in the image. Similarly, in online collections such as Wikipedia, where articles get tagged with categories, human labelers usually tag only with categories they know about. Moreover, there might be considerable noise in the labeling.

In order to solve for the low-rank linear model that results from our formulation, we use the popular alternating minimization algorithm that works well despite the non-convexity of the rank constraint. For general loss functions and trace-norm regularization, we exploit subtle structures present in the problem to design a fast conjugate gradient based method. For the special case of squared-L_2 loss and trace-norm regularization, we further exploit the structure of the loss function to provide a more efficient and scalable algorithm. As compared to direct computation, our algorithm is $O(\bar{d})$ faster, where $\bar{d}$ is the average number of nonzero features in an instance.

On the theoretical side, we perform an excess risk analysis for the trace-norm regularized ERM formulation with missing labels, assuming labels are observed uniformly at random. Our proofs do not follow from existing re-

sults due to missing labels and require a careful analysis involving results from random matrix theory. Our results show that while in general the low-rank promoting trace-norm regularization does not provide better bounds than learning a full-rank matrix (e.g. using Frobenius norm regularization), for several interesting data distributions, trace-norm regularization does indeed give significantly better bounds. More specifically, for isotropic data distributions, we show that trace-norm based methods have excess risk of $O(\frac{1}{\sqrt{nL}})$ while full-rank learning can only guarantee $O(\frac{1}{\sqrt{n}})$ excess risk, where n is the number of training points.

Finally, we provide an extensive empirical evaluation of our method on a variety of benchmark datasets. In particular, we compare our method against three recent label compression based methods: CPLST [25], Bayesian-CS [61], and WSABIE [127]. On almost all datasets, our method significantly outperforms these methods, both in the presence and absence of missing labels. Finally, we show the scalability of our method by applying it to a recently curated Wikipedia dataset [3], that has 881,805 training samples and 213,707 labels. The results show that our method not only provides reasonably accurate solutions for such large-scale problems, but that the training time is orders of magnitude lesser than several existing methods.

Related Work. Typically, Binary Relevance (BR), which treats each label as an independent binary classification task, is quite accurate for multi-label learning. However, for a large number of labels, this method becomes infeasible due to increased model size and prediction time. Recently, tech-

niques have been developed that either reduce the dimension of the labels, such as the Compressed Sensing Approach [53], PLST [115], CPLST [25], and Bayesian CS [61], or reduce the feature dimension, such as [114], or both, such as WSABIE [127]. Most of these techniques are tied to a specific loss function (e.g., CPLST and BCS cater only to the squared-L_2 loss, and WSABIE works with the weighted approximate ranking loss) and/or cannot handle missing labels.

Our framework models multi-label classification as a general ERM problem with a low-rank constraint, which not only generalizes both label and feature dimensionality reduction but also brings in the ability to support various loss functions and allows for rigorous generalization error analysis. We show that our formulation not only retrieves CPLST, which has been shown to be fairly accurate, as a special case, but substantially enhances it by use of regularization, other loss functions, allowing missing labels etc.

Organization. We begin by studying a generic low-rank ERM framework for multi-label learning in Section 5.1. Next, we propose efficient algorithms for the framework in Section 5.2.We then present empirical results in Section 5.4.

5.1 Problem Formulation

In this section we present a generic ERM-style framework for multi-label classification. For each training point, we shall receive a feature vector $\boldsymbol{x}_i \in \mathbb{R}^d$ and a corresponding label vector $\boldsymbol{y}_i \in \{0,1\}^L$ with L labels. For any

$j \in [L]$, $\boldsymbol{y}_i^j = 1$ will denote that the l^{th} label is "present" or "on" whereas $\boldsymbol{y}_i^j = 0$ will denote that the label is "absent" or "off". Note that although we focus mostly on the binary classification setting in this chapter, our methods easily extend to the multi-class setting where $\boldsymbol{y}_i^j \in \{1, 2, \ldots, C\}$.

Our predictions for the label vector shall be parametrized as $f(\boldsymbol{x}; Z) = Z^\top \boldsymbol{x}$, where $Z \in \mathbb{R}^{d \times L}$. Although we have adopted a linear parametrization here, our results can easily be extended for non-linear kernels as well. Let $\ell(\boldsymbol{y}, f(\boldsymbol{x}; Z)) \in \mathbb{R}$ be the loss function that computes the discrepancy between the "true" label vector and the prediction. We assume that the loss function is decomposable, i.e., $\ell(\boldsymbol{y}, f(\boldsymbol{x}; Z)) = \sum_{j=1}^{L} \ell(\boldsymbol{y}^j, f^j(\boldsymbol{x}; Z))$.

The motivation for our framework comes from the observation that although the number of labels in a multi-label classification problem might be large, there typically exist significant label correlations, thus reducing the effective number of parameters required to model them to much less than $d \times L$. We capture this intuition by restricting the matrix Z to learn only a small number of "latent" factors. This constrains Z to be a low rank matrix which not only controls overfitting but also gives computational benefits.

Given n training points our training set will be (X, Y) where $X = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n]^\top$ and $Y = [\boldsymbol{y}_1 \ \boldsymbol{y}_2 \ \ldots \ \boldsymbol{y}_n]^\top$. Using the loss function ℓ, we propose to learn the parameters Z by using the canonical ERM method, i.e.,

$$\begin{aligned} \hat{Z} = \arg\min_{Z} J(Z) = \sum_{i=1}^{n} \sum_{j=1}^{L} \ell(Y_{ij}, f^j(\boldsymbol{x}_i; Z)) + \lambda \cdot \mathcal{R}(Z), \\ s.t. \quad \text{rank}(Z) \leq k, \end{aligned} \tag{5.1}$$

where $\mathcal{R}(Z) : \mathbb{R}^{d \times L} \rightarrow \mathbb{R}$ is a regularizer. If there are missing labels, we compute the loss over the known labels:

$$\hat{Z} = \arg\min_{Z} J_{\Omega}(Z) = \sum_{(i,j)\in\Omega} \ell(Y_{ij}, f^j(\boldsymbol{x}_i; Z)) + \lambda \cdot \mathcal{R}(Z),$$
$$s.t. \quad \text{rank}(Z) \leq k, \tag{5.2}$$

where $\Omega \subseteq [n] \times [L]$ is the index set that represents "known" labels. Note that in this work, we assume the standard missing value setting, where each label can be either on, off (i.e., $Y_{ij} = 1$ or 0), or missing ($Y_{ij} =?$); several other works have considered another setting where only positive labels are known and are given as 1 in the label matrix, while negative or missing values are all denoted by 0 [3, 19].

5.1.1 Special Case: Squared-L_2 loss

In this section, we study (5.1) and (5.2) for the special case of squared L_2 loss function, i.e., $\ell(\boldsymbol{y}, f(\boldsymbol{x}; Z)) = \|\boldsymbol{y} - f(\boldsymbol{x}; Z)\|_2^2$. We show that in the absence of missing labels, the formulation in (5.1) can be solved optimally for the squared L_2 loss using SVD. Furthermore, by selecting an appropriate regularizer $r(Z)$ and λ, our solution for L_2 loss is exactly the same as that of CPLST [25].

We first show that the unregularized form of (5.1) with $\ell(\boldsymbol{y}, f(\boldsymbol{x}; Z)) = \|\boldsymbol{y} - Z^\top \boldsymbol{x}\|_2^2$ has a closed form solution.

Claim 5.1. *If* $\ell(\boldsymbol{y}, f(\boldsymbol{x}; Z)) = \|\boldsymbol{y} - Z^\top \boldsymbol{x}\|_2^2$ *and* $\lambda = 0$, *then*

$$V_X \Sigma_X^{-1} M_k = \arg\min_{Z:\text{rank}(Z)\leq k} \|Y - XZ\|_F^2, \tag{5.3}$$

where $X = U_X \Sigma_X V_X^\top$ *is the thin SVD decomposition of* X*, and* M_k *is the rank-k truncated SVD of* $M \equiv U_X^\top Y$*.*

See [139, Appendix] for a proof of Claim 5.1. We now show that this is exactly the solution obtained by [25] for their CPLST formulation.

Claim 5.2. *The solution to* (5.3) *is equivalent to* $Z^{CPLST} = W_{CPLST} H_{CPLST}^\top$ *which is the closed form solution for the CPLST scheme.*

See [139, Appendix] for a proof. Note that [25] derive their method by relaxing a Hamming loss problem and dropping constraints in the canonical correlation analysis in a relatively ad-hoc manner. The above results, on the other hand, show that the same model can be derived in a more principled manner. This helps us in extending the method for several other problem settings in a principled manner and also helps in providing excess risk bounds:

- As shown empirically, CPLST tends to overfit significantly whenever d is large. However, we can handle this issue by setting λ appropriately.
- The closed form solution in [25] cannot directly handle missing labels as it requires SVD on fully observed Y. In contrast, our framework can itself handle missing labels without any modifications.
- The formulation in [25] is tied to the L_2 loss function. In contrast, we can easily handle other loss functions; although, the optimization problem might become more difficult to solve.

We note that such links between low rank solutions to multi-variate regression problems and PCA/SVD are well known in literature [18, 56]. How-

ever, these results are mostly derived in the stochastic setting under various noise models whereas ours apply to the empirical setting. Moreover, these classical results put little emphasis on large scale implementation.

5.2 Algorithms

In this section, we apply the alternating minimization technique for optimizing (5.1) and (5.2). For a matrix Z with a known low rank k, it is inefficient to represent it using $d \times L$ entries, especially when d and L are large. Hence we consider a low-rank decomposition of the form $Z = WH^\top$, where $W \in \mathbb{R}^{d\times k}$ and $H \in \mathbb{R}^{L\times k}$. We further assume that $\mathcal{R}(Z)$ can be decomposed into $\mathcal{R}_w(W) + \mathcal{R}_h(H)$. In the following sections, we present results with the trace norm regularization, i.e., $\mathcal{W}(Z) = \|Z\|_{\mathrm{tr}}$, which can be decomposed as $\|Z\|_{\mathrm{tr}} = \frac{1}{2}(\|W\|_F^2 + \|H\|_F^2)$. Thus, $\min_Z J_\Omega(Z)$ with the rank constraint is equivalent to minimizing over W, H:

$$J_\Omega(W, H) = \sum_{(i,j)\in\Omega} \ell(Y_{ij}, \boldsymbol{x}_i^\top W \boldsymbol{h}_j) + \frac{\lambda}{2}\big(\|W\|_F^2 + \|H\|_F^2\big) \tag{5.4}$$

where $\boldsymbol{h}_j^\top$ is the j-th row of H. Note that when either of W or H is fixed, $J_\Omega(W, H)$ becomes a convex function. This allows us to apply alternating minimization, a standard technique for optimizing functions with such a property, to (5.4). For a general loss function, after proper initialization, a sequence

$\{(W^{(t)}, H^{(t)})\}$ is generated by

$$H^{(t)} \leftarrow \arg\min_{H} \quad J_\Omega(W^{(t-1)}, H),$$
$$W^{(t)} \leftarrow \arg\min_{W} \quad J_\Omega(W, H^{(t)}).$$

For a convex loss function, $(W^{(t)}, H^{(t)})$ is guaranteed to converge to a stationary point when the minimum for both $\min_H J_\Omega(W^{(t-1)}, H)$ and $\min_W J_\Omega(W, H^{(t)})$ are uniquely defined [see 14, Proposition 2.7.1]. In fact, when the squared loss is used and Y is fully observed, the case considered in Section 5.2.2, we can prove that $(W^{(t)}, H^{(t)})$ converges to the global minimum of (5.4) when either $\lambda = 0$ or X is orthogonal.

Once W is fixed, updating H is easy as each row $\boldsymbol{h}_j$ of H can be independently updated as follows:

$$\boldsymbol{h}_j \leftarrow \arg\min_{\boldsymbol{h}\in\mathbb{R}^k} \sum_{i:(i,j)\in\Omega} \ell(Y_{ij}, \boldsymbol{x}_i^\top W\boldsymbol{h}) + \frac{1}{2}\lambda \cdot \|\boldsymbol{h}\|_2^2, \tag{5.5}$$

which is easy to solve as k is small in general. Based on the choice of the loss function, (5.5) is essentially a linear classification or regression problem over k variables with $|\{i : (i,j) \in \Omega\}|$ instances. If H is fixed, updating W is more involved as all variables are mixed up due to the pre-multiplication with X. Let $\tilde{\boldsymbol{x}}_{ij} = \boldsymbol{h}_j \otimes \boldsymbol{x}_i$ (where $\otimes$ denotes the outer product). It can be shown that updating W is equivalent to a regularized linear classification/regression problem with $|\Omega|$ data points $\{(Y_{ij}, \tilde{\boldsymbol{x}}_{ij}) : (i,j) \in \Omega\}$. Thus if $W^* = \arg\min_W J_\Omega(W, H)$ and we denote $\boldsymbol{w}^* := \mathrm{vec}(W^*)$, then $\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}\in\mathbb{R}^{dk}} g(\boldsymbol{w})$,

$$g(\boldsymbol{w}) \equiv \sum_{(i,j)\in\Omega} \ell(Y_{ij}, \boldsymbol{w}^\top \tilde{\boldsymbol{x}}_{ij}) + \frac{1}{2}\lambda \cdot \|\boldsymbol{w}\|_2^2. \tag{5.6}$$

Algorithm 5.1 General Loss with Missing Labels

To compute $\nabla g(\boldsymbol{w})$:

1. $A \leftarrow XW$, where $\mathrm{vec}(W) = \boldsymbol{w}$.
2. $D_{ij} \leftarrow \ell'(Y_{ij}, \boldsymbol{a}_i^\top \boldsymbol{h}_j),\ \forall (i,j) \in \Omega$.
3. **Return**: $\mathrm{vec}\big(X^\top (DH)\big) + \lambda \boldsymbol{w}$

To compute: $\nabla^2 g(\boldsymbol{w})\boldsymbol{s}$

1. $A \leftarrow XW$, where $\mathrm{vec}(W) = \boldsymbol{w}$.
2. $B \leftarrow XS$, where $\mathrm{vec}(S) = \boldsymbol{s}$.
3. $U_{ij} \leftarrow \ell''(Y_{ij}, \boldsymbol{a}_i^\top \boldsymbol{h}_j) \boldsymbol{b}_i^\top \boldsymbol{h}_j,\ \forall (i,j) \in \Omega$.
4. **Return**: $\mathrm{vec}\big(X^\top (UH)\big) + \lambda \boldsymbol{s}$.

Algorithm 5.2 Squared Loss with Full Labels

To compute $\nabla g(\boldsymbol{w})$:

1. $A \leftarrow XW$, where $\mathrm{vec}(W) = \boldsymbol{w}$.
2. $B \leftarrow YH$.
3. $M \leftarrow H^\top H$.
4. **Return**: $\mathrm{vec}\big(X^\top (AM - B)\big) + \lambda \boldsymbol{w}$

To compute: $\nabla^2 g(\boldsymbol{w})\boldsymbol{s}$

1. $A \leftarrow XS$, where $\mathrm{vec}(S) = \boldsymbol{s}$.
2. $M \leftarrow H^\top H$.
3. **Return**: $\mathrm{vec}\big(X^\top (AM)\big) + \lambda \boldsymbol{s}$

Taking the squared loss as an example, the above is equivalent to a regularized least squares problem with dk variables. When d is large, say 1M, the closed form solution, which requires inverting a $dk \times dk$ matrix, can hardly be regarded as feasible. As a result, updating W efficiently turns out to be the main challenge for alternating minimization.

In large-scale settings where both dk and $|\Omega|$ are large, iterative methods such as Conjugate Gradient (CG), which perform cheap updates and offer a good approximate solution within a few iterations, are more appro-

priate to solve (5.6). Several linear classification/regression packages such as LIBLINEAR [35] can handle such problems if $\{\tilde{\boldsymbol{x}}_{ij} : (i,j) \in \Omega\}$ are available. The main operation in such iterative methods is a gradient calculation ($\nabla g(\boldsymbol{w})$) or a multiplication of the Hessian matrix and a vector ($\nabla^2 g(\boldsymbol{w})\boldsymbol{s}$). Let $\tilde{X} = [\cdots \tilde{\boldsymbol{x}}_{ij} \cdots]_{(i,j)\in\Omega}^\top$ and $\bar{d} = \sum_{i=1}^n \|\boldsymbol{x}\|_0/n$. Then these operations require at least $nnz(\tilde{X}) = O(|\Omega|\bar{d}k)$ time to compute in general.

However, as we show below, we can exploit the structure in $\tilde{X}$ to develop efficient techniques such that both the operations mentioned above can be done in $O((|\Omega| + nnz(X) + d + L) \times k)$ time. As a result, iterative methods, such as CG, can achieve $O(\bar{d})$ speedup. Our techniques make the alternating minimization efficient enough to handle large-scale problems.

5.2.1 Fast Operations for General Loss Functions

We assume that the loss function is a general twice-differentiable function $\ell(a,b)$, where a and b are scalars. Let $\ell'(a,b) = \frac{\partial}{\partial b}\ell(a,b)$, and $\ell''(a,b) = \frac{\partial^2}{\partial b^2}\ell(a,b)$. The gradient and the Hessian matrix for $g(\boldsymbol{w})$ are:

$$\nabla g(\boldsymbol{w}) = \sum_{(i,j)\in\Omega} \ell'(Y_{ij}, \boldsymbol{w}^\top \tilde{\boldsymbol{x}}_{ij})\tilde{\boldsymbol{x}}_{ij} + \lambda \boldsymbol{w}, \tag{5.7}$$

$$\nabla^2 g(\boldsymbol{w}) = \sum_{(i,j)\in\Omega} \ell''(Y_{ij}, \boldsymbol{w}^\top \tilde{\boldsymbol{x}}_{ij})\tilde{\boldsymbol{x}}_{ij}\tilde{\boldsymbol{x}}_{ij}^\top + \lambda I. \tag{5.8}$$

A direct computation of $\nabla g(\boldsymbol{w})$ and $\nabla^2 g(\boldsymbol{w})\boldsymbol{s}$ using (5.7) and (5.8) requires at least $O(|\Omega|\bar{d}k)$ time. Below we give faster procedures to perform both operations.

Gradient Calculation. Recall that $\tilde{\boldsymbol{x}}_{ij} = \boldsymbol{h}_j \otimes \boldsymbol{x}_i = \mathrm{vec}\big(\boldsymbol{x}_i \boldsymbol{h}_j^\top\big)$. Therefore, we have

$$\sum_{(i,j)\in\Omega} \ell'(Y_{ij}, \boldsymbol{w}^\top \tilde{\boldsymbol{x}}_{ij}) \boldsymbol{x}_i \boldsymbol{h}_j^\top = X^\top D H,$$

where D is sparse with $D_{ij} = \ell'(Y_{ij}, \boldsymbol{w}^\top \tilde{\boldsymbol{x}}_{ij}),\ \forall (i,j) \in \Omega$. Thus,

$$\nabla g(\boldsymbol{w}) = \mathrm{vec}\big(X^\top D H\big) + \lambda \boldsymbol{w}. \tag{5.9}$$

Assuming that $\ell'(a,b)$ can be computed in constant time, which holds for most loss functions (e.g. squared-L_2 loss, logistic loss), the gradient computation can be done in $O((nnz(X) + |\Omega| + d) \times k)$ time. Algorithm 5.1 gives the details of computing $\nabla g(\boldsymbol{w})$ using (5.9).

Hessian-vector Multiplication. After substituting $\tilde{\boldsymbol{x}}_{ij} = \boldsymbol{h}_j \otimes \boldsymbol{x}_i$, we have

$$\nabla^2 g(\boldsymbol{w}) \boldsymbol{s} = \sum_{(i,j)\in\Omega} \ell''_{ij} \cdot \big((\boldsymbol{h}_j \boldsymbol{h}_j^\top) \otimes (\boldsymbol{x}_i \boldsymbol{x}_i^\top)\big) \boldsymbol{s} + \lambda \boldsymbol{s},$$

where $\ell''_{ij} = \ell''(Y_{ij}, \boldsymbol{w}^\top \tilde{\boldsymbol{x}}_{ij})$. Let S be the $d \times k$ matrix such that $\boldsymbol{s} = \mathrm{vec}(S)$. Using the identity $(B^\top \otimes A)\,\mathrm{vec}(X) = \mathrm{vec}(AXB)$, we have $\big((\boldsymbol{h}_j \boldsymbol{h}_j^\top) \otimes (\boldsymbol{x}_i \boldsymbol{x}_i^\top)\big) \boldsymbol{s} = \mathrm{vec}\big(\boldsymbol{x}_i \boldsymbol{x}_i^\top S \boldsymbol{h}_j \boldsymbol{h}_j^\top\big)$. Thus,

$$\begin{aligned}
\sum_{ij} \ell''_{ij} \boldsymbol{x}_i \boldsymbol{x}_i^\top S \boldsymbol{h}_j \boldsymbol{h}_j^\top &= \sum_{i=1}^{n} \boldsymbol{x}_i \Big(\sum_{j:(i,j)\in\Omega} \ell''_{ij} \cdot (S^\top \boldsymbol{x}_i)^\top \boldsymbol{h}_j \boldsymbol{h}_j^\top \Big) \\
&= \sum_{i=1}^{n} \boldsymbol{x}_i \Big(\sum_{j:(i,j)\in\Omega} U_{ij} \boldsymbol{h}_j^\top \Big) = X^\top U H,
\end{aligned}$$

Table 5.1: Computation of $\ell'(a,b)$ and $\ell''(a,b)$ for different loss functions. Note that for the logistic and L_2-hinge loss in Table 5.1, Y_{ij} is assumed to be $-1,+1$ instead of $\{0,1\}$.

	$\ell(a,b)$	$\frac{\partial}{\partial b}\ell(a,b)$	$\frac{\partial^2}{\partial b^2}\ell(a,b)$
Squared loss	$\frac{1}{2}(a-b)^2$	$b-a$	1
Logistic loss	$\log\left(1+e^{-ab}\right)$	$\frac{-a}{1+e^{-ab}}$	$\frac{-a^2e^{-ab}}{\left(1+e^{-ab}\right)^2}$
L_2-hinge loss	$(\max(0,1-ab))^2$	$-2a\max(0,1-ab)$	$2\cdot\mathcal{I}[ab<1]$

where U is sparse, and $U_{ij} = \ell''_{ij}\cdot(S^\top \boldsymbol{x}_i)^\top \boldsymbol{h}_j,\ \forall(i,j)\in\Omega$. Thus, we have

$$\nabla^2 g(\boldsymbol{w})\boldsymbol{s} = \mathrm{vec}\left(X^\top U H\right) + \lambda \boldsymbol{s}. \tag{5.10}$$

In Algorithm 5.1, we describe a detailed procedure for computing the Hessian-vector multiplication in

$O((nnz(X)+|\Omega|+d)\times k)$ time using (5.10).

Loss Functions. See Table 5.1 for expressions of $\ell'(a,b)$ and $\ell''(a,b)$ for three common loss functions: squared loss, logistic loss, and squared hinge loss. Thus, to solve (5.6), we can apply CG for squared loss and TRON [74] for the other two loss functions. Note that although L_2-hinge loss is not twice-differentiable, the sub-differential of $\frac{\partial}{\partial b}\ell(a,b)$ still can be used for TRON to solve (5.6).

5.2.2 Fast Operations for Squared Loss with Full Labels

For the situation where labels are fully observed, solving (5.1) efficiently in the large-scale setting remains a challenge. The closed form solution from (5.3) is not ideal for two reasons: firstly since it involves the SVD of both

X and $U_X^\top Y$, the solution becomes infeasible when rank of X is large. Secondly, since it is an unregularized solution, it might overfit. Indeed CPLST has similar scalability and overfitting issues due to absence of regularization and requirement of pseudo inverse calculations for X. When Y is fully observed, Algorithm 5.1, which aims to handle missing labels with a general loss function, is also not scalable as $|\Omega| = nL$ imposing a $O(nLk + nnz(X)k)$ cost per operation which is prohibitive when n and L are large.

Although, for a general loss, an $O(nLk)$ cost seems to be inevitable, for the L_2 loss, we propose fast procedures such that the cost of each operation only depends on $nnz(Y)$ instead of $|\Omega|$. In most real-world multi-label problems, $nnz(Y) \ll nL = |\Omega|$. As a result, for the squared loss, our technique allows alternating minimization to be performed efficiently even when $|\Omega| = nL$.

If the squared loss is used, the matrix D in Eq. (5.9) is $D = XWH^\top - Y$ when Y is fully observed, where W is the $d \times k$ matrix such that $\text{vec}(W) = \boldsymbol{w}$. Then, we have

$$\nabla g(\boldsymbol{w}) = \text{vec}\big(X^\top XWH^\top H - X^\top YH\big) + \lambda \boldsymbol{w}. \tag{5.11}$$

Similarly, U in Eq. (5.10) is $U = XSH^\top$ which gives us

$$\nabla^2 g(\boldsymbol{w})\boldsymbol{s} = \text{vec}\big(X^\top XSH^\top H\big) + \lambda \boldsymbol{s}. \tag{5.12}$$

With a careful choice of the sequence of the matrix multiplications, we show detailed procedures in Algorithm 5.2, which use only $O(nk + k^2)$ extra space

and $O((nnz(Y) + nnz(X))k + (n + L)k^2)$ time to compute both $\nabla g(\boldsymbol{w})$ and $\nabla^2 g(\boldsymbol{w})\boldsymbol{s}$ efficiently.

5.3 Generalization Error Bounds

In this section we analyze excess risk bounds for our learning model with trace norm regularization. Our analysis demonstrates the superiority of our trace norm regularization-based technique over BR and Frobenius norm regularization. We require a more careful analysis for our setting since standard results do not apply because of the presence of missing labels.

Our multi-label learning model is characterized by a distribution $\mathcal{D}$ on the space of data points and labels $\mathcal{X} \times \{0,1\}^L$ where $\mathcal{X} \subseteq \mathbb{R}^d$ and a distribution that decides the pattern of missing labels. We receive n training points $(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_n, \boldsymbol{y}_n)$ sampled i.i.d from the distribution $\mathcal{D}$, where $\boldsymbol{y}_i \in \{0,1\}^L$ are the *ground truth* label vectors. However we shall only be able to observe the ground truth label vectors $\boldsymbol{y}_i$ at s random locations. More specifically, for each i we only observe $\boldsymbol{y}_i$ at locations $l_i^1, \ldots, l_i^s \in [L]$ where the locations are chosen uniformly from the set $[L]$ and the choices are independent of $(\boldsymbol{x}_i, \boldsymbol{y}_i)$.

Given this training data, we learn a predictor $\hat{Z}$ by performing ERM over a constrained set of predictors as follows:

$$\hat{Z} = \underset{\mathcal{R}(Z) \leq \lambda}{\arg\inf} \quad \hat{\mathcal{L}}(Z) = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{s} \ell(\boldsymbol{y}_i^{l_i^j}, f^{l_i^j}(\boldsymbol{x}_i; Z)),$$

where $\hat{\mathcal{L}}(Z)$ is the *empirical risk* of a predictor Z. Note that although the

method in Equation 5.2 uses a regularized formulation that is rank-constrained, we analyze just the regularized version without the rank constraints for simplicity. As the class of rank-constrained matrices is smaller than the class of trace-norm constrained matrices, we can in fact expect better generalization performance than that indicated here, if the ERM problem can be solved exactly.

Our goal would be to show that $\hat{Z}$ has good generalization properties i.e. $\mathcal{L}(\hat{Z}) \leq \inf_{\mathcal{R}(Z)\leq\lambda} \mathcal{L}(Z) + \epsilon$, where $\mathcal{L}(Z) := \mathop{\mathbb{E}}_{\boldsymbol{x},\boldsymbol{y},l}\left[\!\left[\ell(\boldsymbol{y}^l, f^l(\boldsymbol{x}; Z))\right]\!\right]$ is the *population risk* of a predictor.

Theorem 5.1. *Suppose we learn a predictor using the formulation*

$$\hat{Z} = \operatorname*{arg\,inf}_{\|Z\|_{\text{tr}}\leq\lambda} \hat{\mathcal{L}}(Z)$$

over a set of n training points. Then with probability at least $1-\delta$, we have

$$\mathcal{L}(\hat{Z}) \leq \inf_{\|Z\|_{\text{tr}}\leq\lambda} \mathcal{L}(Z) + O\left(s\lambda\sqrt{\frac{1}{n}}\right) + O\left(s\sqrt{\frac{\log\frac{1}{\delta}}{n}}\right),$$

where we assume (w.l.o.g.) that $\mathbb{E}\left[\!\left[\|\boldsymbol{x}\|_2^2\right]\!\right] \leq 1$.

We refer to Appendix of [139] for the proof. Interestingly, we can show that our analysis, obtained via uniform convergence bounds, is tight and cannot be improved in general. See Appendix of [139] for the tightness argument. However, it turns out that Frobenius norm regularization is also able to offer the same excess risk bounds and thus, this result does not reveal any advantage for trace norm regularization. Nevertheless, we can still get improved bounds for a general class of distributions over $(\boldsymbol{x}, \boldsymbol{y})$:

Theorem 5.2. *Let the data distribution satisfy the following conditions: 1) The top singular value of the covariance matrix* $X = \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathcal{D}}\left[\!\left[\boldsymbol{x}\boldsymbol{x}^\top\right]\!\right]$ *is* $\|X\|_2 = \sigma_1$*, 2)* $tr(X) = \Sigma$ *and 3) the distribution on* $\mathcal{X}$ *is sub-Gaussian i.e. for some* $\eta > 0$*, for all* $\boldsymbol{v} \in \mathbb{R}^d$*,* $\mathbb{E}\left[\!\left[\exp\left(x^\top \boldsymbol{v}\right)\right]\!\right] \leq \exp\left(\|\boldsymbol{v}\|_2^2\eta^2/2\right)$*, then with probability at least* $1-\delta$*, we have*

$$\mathcal{L}(\hat{Z}) \leq \inf_{\|Z\|_{\mathrm{tr}}\leq\lambda} \mathcal{L}(Z) + O\left(s\lambda\sqrt{\frac{d(\eta^2+\sigma_1)}{nL\Sigma}} + s\sqrt{\frac{\log\frac{1}{\delta}}{n}}\right).$$

In particular, if the data points are generated from a unit normal distribution, then we have

$$\mathcal{L}(\hat{Z}) \leq \inf_{\|Z\|_{\mathrm{tr}}\leq\lambda} \mathcal{L}(Z) + O\left(s\lambda\sqrt{\frac{1}{nL}}\right) + O\left(s\sqrt{\frac{\log\frac{1}{\delta}}{n}}\right).$$

The proof of Theorem 5.2 can be found in Appendix of [139].

We note that the assumptions on the data distribution are trivially satisfied with finite σ_1 and η by any distribution with support over a compact set. However, for certain distributions, this allows us to give superior bounds for trace norm regularization. We note that Frobenius norm regularization can give no better than a $\left(\frac{\lambda}{\sqrt{n}}\right)$ style excess error bound even for such distributions (see Appendix in [139] for a proof), whereas trace norm regularization allows us to get superior $\left(\frac{\lambda}{\sqrt{nL}}\right)$ style bounds. This is especially contrasting when, for instance, $\lambda = \mathcal{O}(\sqrt{L})$, in which case trace norm regularization gives $O\left(\frac{1}{\sqrt{n}}\right)$ excess error whereas the excess error for Frobenius regularization deteriorates to $O\left(\sqrt{\frac{L}{n}}\right)$. Thus, trace norm seems better suited to exploit situations where the data distribution is isotropic.

Intuitively, we expect such results due to the following reason: when labels are very sparsely observed, such as when $s = O(1)$, we observe the value of each label on $O(n/L)$ training points. In such a situation, Frobenius norm regularization with say $\lambda = \sqrt{L}$ essentially allows an independent predictor $z_l \in \mathbb{R}^d$ to be learned for each label $l \in [L]$. Since all these predictors are being trained on only $O(n/L)$ training points, the performance accordingly suffers.

On the other hand, if we were to train a single predictor for all the labels i.e. $Z = z\mathbf{1}^\top$ for some $z \in \mathbb{R}^d$, such a predictor would be able to observe $O(n)$ points and consequently have much better generalization properties. Note that this predictor also satisfies $\|z\mathbf{1}^\top\|_{\text{tr}} \leq \sqrt{L}$. This seems to indicate that trace norm regularization can capture cross label dependencies, especially in the presence of missing labels, much better than Frobenius norm regularization.

Having said that, it is important to note that trace norm and Frobenius norm regularization induce different biases in the learning framework. It would be interesting to study the bias-variance trade-offs offered by these two regularization techniques. However, in presence of label correlations we expect both formulations to suffer similar biases.

5.4 Experimental Results

We now evaluate our proposed algorithms in terms of accuracy and stability. This discussion shall demonstrate the superiority of our method over other approaches.

Table 5.2: Data statistics. d and L are the number of features and labels, respectively, and $\bar{d}$ and $\bar{L}$ are the average number of nonzero features and positive labels in an instance, respectively.

			Training set			Test set		
Dataset	d	L	n	$\bar{d}$	$\bar{L}$	n	$\bar{d}$	$\bar{L}$
bibtex	1,836	159	4,880	68.74	2.40	2,515	68.50	2.40
autofood	9,382	162	155	143.92	15.80	38	143.71	13.71
compphys	33,284	208	161	792.78	9.80	40	899.02	11.83
delicious	500	983	12,920	18.17	19.03	3,185	18.80	19.00
eurlex	5,000	3,993	17,413	236.69	5.30	1,935	240.96	5.32
nus-wide	1,134	1,000	161,789	862.70	5.78	107,859	862.94	5.79
wiki	366,932	213,707	881,805	146.78	7.06	10,000	147.78	7.08

Datasets. We considered a variety of benchmark datasets including four standard datasets (bibtex, delicious, eurlex, and nus-wide), two datasets with $d \gg L$ (autofood and compphys), and a very large scale Wikipedia based dataset, which contains about 1M wikipages and 200K labels. See Table 5.2 for more information about the datasets. We conducted all experiments on an Intel machine with 32 cores.

Competing Methods. A list containing details of the competing methods (including ours) is given below. Note that CS [53] and PLST [115] are not included as they are shown to be suboptimal to CPLST and BCS in [25, 61].

1. LEML (**L**ow rank **E**mpirical risk minimization for **M**ulti-Label **L**earning): our proposed method. We implemented CG with Algorithms 5.1 and 5.2 for squared loss, and TRON [74] with Algorithm 5.1 for logistic and squared hinge loss.

Table 5.3: Comparison of LEML with various loss functions and WSABIE on smaller datasets. SQ denotes squared loss, LR denotes logistic regression loss, and SH denotes squared hinge loss

		Top-3 Accuracy				Average AUC			
		LEML			WSABIE	LEML			WSABIE
	k/L	SQ	LR	SH		SQ	LR	SH	
bibtex	20%	**34.16**	25.65	27.37	28.77	0.8910	0.8677	0.8541	**0.9055**
	40%	**36.53**	28.20	24.81	30.05	0.9015	0.8809	0.8467	**0.9092**
	60%	**38.00**	28.68	23.26	31.11	0.9040	0.8861	0.8505	**0.9089**
autofood	20%	**81.58**	80.70	**81.58**	66.67	0.9565	**0.9598**	0.9424	0.8779
	40%	76.32	**80.70**	78.95	70.18	0.9277	**0.9590**	0.9485	0.8806
	60%	70.18	80.70	**81.58**	60.53	0.8815	**0.9582**	0.9513	0.8518
compphys	20%	**80.00**	**80.00**	**80.00**	49.17	0.9163	0.9223	**0.9274**	0.8212
	40%	**80.00**	78.33	79.17	39.17	**0.9199**	0.9157	0.9191	0.8066
	60%	**80.00**	**80.00**	**80.00**	49.17	**0.9179**	0.9143	0.9098	0.8040
delicious	20%	**61.20**	53.68	57.27	42.87	0.8854	0.8588	**0.8894**	0.8561
	40%	**61.23**	49.13	52.95	42.05	0.8827	0.8534	**0.8868**	0.8553
	60%	**61.15**	46.76	49.58	42.22	0.8814	0.8517	**0.8852**	0.8523

2. CPLST: the method proposed in [25]. We used code provided by the authors.
3. BCS: the method proposed in [61]. We used code provided by the authors.
4. BR: Binary Relevance with various loss functions.
5. WSABIE: Due to lack of publicly available code, we implemented this method and hand-tuned learning rates and the margins for each dataset as suggested by the authors of WSABIE [126].

Evaluation Criteria. We used three criteria to compare the methods: top-K accuracy (performance on a few top predictions), Hamming loss (overall classification performance), and average AUC (ranking performance).

5.4.1 Results with full labels

We divide datasets into two groups: *small datasets* (bibtex, autofood, compphys, and delicious) to which all methods are able to scale and *large datasets* (eurlex, nus-wide, and wiki) to which only LEML and WSABIE are able to scale.

Small datasets. We first compare dimension reduction based approaches to assess their performance with varying dimensionality reduction ratios. Figure 5.1 presents these results for LEML, CPLST and BCS on the squared L_2 loss with BR included for reference. Clearly LEML consistently outperforms other methods for all ratios. Next we compare LEML to WSABIE with three surrogates (squared, logistic, and L_2-hinge), which approximately optimize a weighted approximate ranking loss. Table 5.3 shows that although the best loss function for each dataset varies, LEML is always superior to or competitive with WSABIE. Based on Figure 5.1, Table 5.3, and further results in [139, Appendix],we make the following observations. 1) LEML can deliver accuracies competitive with BR even with a severe reduction in dimensionality, 2) On bibtex and compphys, LEML is even shown to outperform BR. This is a benefit brought forward by the design of LEML, wherein the relation between labels can be captured by a low rank Z. This enables LEML to better utilize label information than BR and yield better accuracies. 3) On autofood and compphys, CPLST seems to suffer from overfitting and demonstrates a significant dip in performance. In contrast, LEML, which brings regularization into the formulation performs well consistently on all datasets.

Larger data. Table 5.4 shows results for LEML and WSABIE on the three larger datasets. We implemented LEML with the squared L_2 loss using Algorithm 5.2 for comparison in the full labels case. Note that Hamming loss is not used here as it is not clear how to convert the label ranking given by WSABIE to a 0/1 encoding. For LEML, we report the time and the accuracies obtained after five alternating iterations. For WSABIE, we ran the method on each dataset with the hand-tuned parameters for about two days, and reported the time and results for the epoch with the highest average AUC. On eurlex and nus-wide, LEML is clearly superior than WSABIE on all evaluation criteria. On wiki, although both methods share a similar performance for $k = 250$, on increasing k to 500, LEML again outperforms WSABIE. Also clearly noticeable is the stark difference in the running times of the two methods. Whereas LEML takes less than 6 hours to deliver 0.9374 AUC on wiki, WSABIE requires about 1.6 days to achieve 0.9058 AUC. More specifically, WSABIE takes about 7,000s for the first epoch, 16,000s for the second and 36,000s for the third epoch which result in it spending almost two days on just 5 epochs. Although this phenomenon is expected due to the sampling scheme in WSABIE [127], it becomes more serious as L increases. We leave the issue of designing a better sampling scheme with large L for future work. Figure 5.2(a) further illustrates this gap in training times for the nus-wide dataset. All in all, the results clearly demonstrate the scalability and efficiency of LEML.

Table 5.4: Comparison of LEML and WSABIE on large datasets

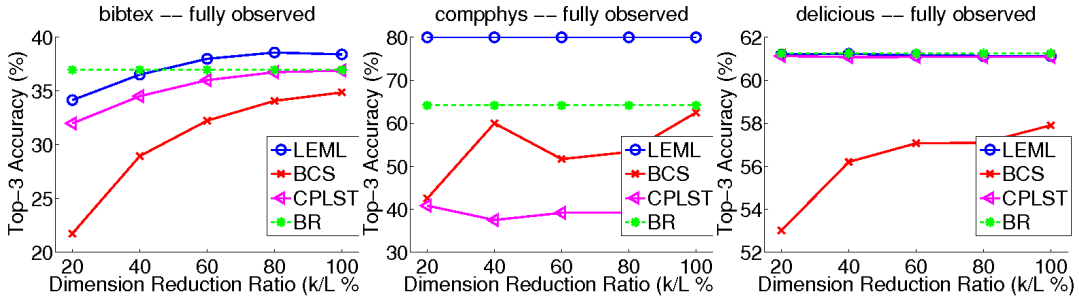
		LEML				WSABIE			
dataset	k	time (s)	top-1	top-3	AUC	time (s)	top-1	top-3	AUC
eurlex	250	**175**	**51.99**	**39.79**	**0.9425**	373	33.13	25.01	0.8648
	500	**487**	**56.90**	**44.20**	**0.9456**	777	31.58	24.00	0.8651
nus-wide	50	**574**	**20.71**	**15.96**	**0.7741**	4,705	14.58	11.37	0.7658
	100	**1,097**	**20.76**	**16.00**	**0.7718**	6,880	12.46	10.21	0.7597
wiki	250	**9,932**	**19.56**	14.43	**0.9086**	79,086	18.91	**14.65**	0.9020
	500	**18,072**	**22.83**	**17.30**	**0.9374**	139,290	19.20	15.66	0.9058

Table 5.5: Comparison between various dimensionality reduction approaches on Y with 20% observed entries, and $k = 0.4L$.

	Top-3 Accuracy			Hamming loss			Average AUC		
	LEML	BCS	BR	LEML	BCS	BR	LEML	BCS	BR
bibtex	**28.50**	23.84	25.78	**0.0136**	0.2496	0.0193	**0.8332**	0.7871	0.8087
autofood	**67.54**	35.09	62.28	**0.0671**	0.2445	0.0760	**0.8634**	0.6322	0.8178
compphys	**65.00**	35.83	31.67	**0.0518**	0.2569	0.0566	**0.7964**	0.6442	0.7459

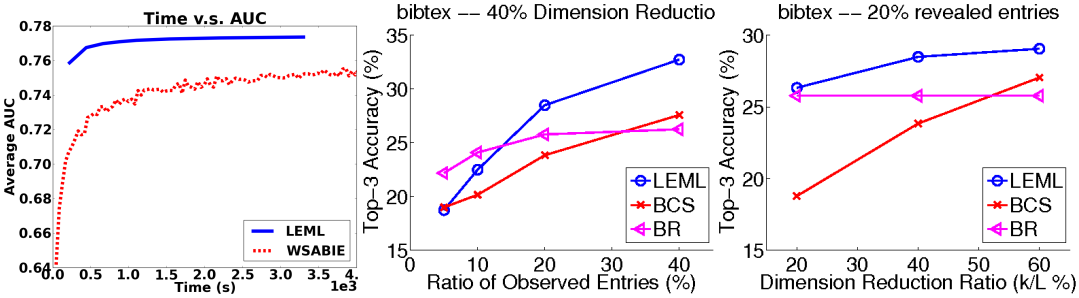
5.4.2 Results with missing labels

For experiments with missing labels, we compare LEML, BCS, and BR. We implemented BR with missing labels by learning an L_2-regularized binary classifier/regressor for each label on observed instances. Thus, the model derived from BR corresponds to the minimizer of (5.2) with Frobenius norm regularization. Table 5.5 shows the results when 20% entries were revealed (i.e. 80% missing rate) and squared loss function was used for training. We used $k = 0.4L$ for both LEML and BCS. The results clearly show that LEML outperforms BCS and LEML with respect to all three evaluation criteria. On bibtex, we further present results for various rates of observed labels in Figure 5.2(b) and results for various dimension reduction ratios in Figure 5.2(c).



(a) bibtex (b) compphys (c) delicious

Figure 5.1: Comparison between different dimension reduction methods with fully observed Y by varying the reduction ratio.



(a) (b) (c)

Figure 5.2: Results for (a): running time on nus-wide. (b): various observed ratios on bibtex. (c): various reduction ratios on bibtex.

LEML clearly shows superior performance over other approaches. More empirical results for other loss functions, various observed ratios and dimension reduction ratios can be found in [139, Appendix].

5.5 Extensions

As mentioned earlier in Figure 1.3, if we treat the label matrix as the rating matrix and $\boldsymbol{x}_i$ as the feature vector for the i-th row, LEML can be

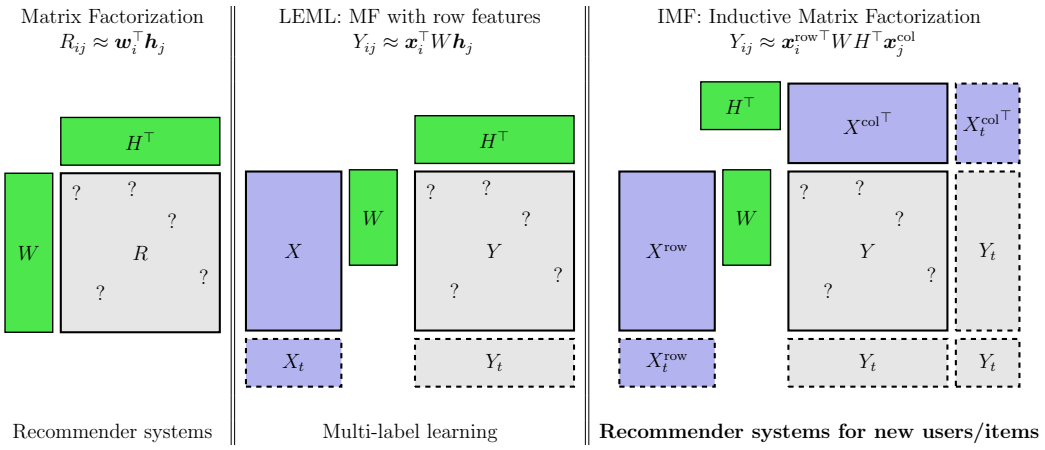


Figure 5.3: Inductive Matrix Factorization: an extension of LEML.

regarded as an extension of matrix factorization with numerical side information. We can further extend this idea to handle matrix factorization with numerical information on both row and column sides, called inductive matrix factorization (IMF). In particular, we can have $\boldsymbol{x}_i^{\text{row}}$ as the feature vector for the i-th row, and $\boldsymbol{x}_j^{\text{col}}$ as the feature vector for the j-th column. The interaction between the i-th row and the j-th column is estimated by

$$Y_{ij} \approx \boldsymbol{x}_i^{\text{row}\top} W H^\top \boldsymbol{x}_j^{\text{col}},$$

which leads to the problem formulation for IMF:

$$\min_{W,H} \sum_{(i,j)\in\Omega} \ell\big(Y_{ij}, \boldsymbol{x}_i^{\text{row}\top} W H^\top \boldsymbol{x}_j^{\text{col}}\big) + \lambda \|W\|_F^2 + \lambda \|H\|_F^2. \tag{5.13}$$

See Figure 5.3 for an illustration for IMF. IMF not only empowers recommender systems to incorporate rich side information on both users and items but also enables the recommendations for new users and new items. Note that the efficient algorithms proposed for LEML can be directly applied to solve

the IMF formulation (5.13). There have been few IMF works based on our proposed LEML algorithms [52, 81, 107].

Another extension of LEML is the combination of the IMF formulation (5.13) and the Full approach of the one-class MF formulation described in Chapter 4. In particular, the problem formulation is defined as follows:

$$\min_{W,H} \sum_{(i,j)\in[m]\times[n]} C_{ij}\ell\big(Y_{ij}, \boldsymbol{x}_i^{\text{row}\top} W H^\top \boldsymbol{x}_j^{\text{col}}\big) + \lambda\|W\|_F^2 + \lambda\|H\|_F^2, \tag{5.14}$$

where

$$C_{ij} = \begin{cases} 1 & \text{if } (i,j) \in \Omega \\ p_i q_j & \text{otherwise} \end{cases} \quad \text{and} \quad Y_{ij} = \bar{a},\ \forall (i,j) \notin \Omega,$$

where $\bar{a}$ is a fixed value for all un-observed entries and $\boldsymbol{p} \in \mathbb{R}^m$ and $\boldsymbol{q} \in \mathbb{R}^n$ are vectors chosen so that $p_i q_j < 1$ because weights for observed entries should be higher. The extension of Algorithm 5.1 and Algorithm 5.2 to solve (5.14) is not trivial. We have worked on a unified algorithm to handle (5.14) in a working manuscript [143].

5.6 Summary of the Contributions

In this chapter we studied the multi-label learning problem with missing labels in the standard ERM framework. We modeled our framework with rank constraints and regularizers to increase scalability and efficiency. To solve the obtained non-convex problem, we proposed an alternating minimization based method that critically exploits structure in the loss function to make our method scalable. We showed that our learning framework admits excess risk bounds that indicate better generalization performance for our methods

than the existing methods like BR, something which our experiments also confirmed. Our experiments additionally demonstrated that our techniques are much more efficient than other large scale multi-label classifiers and give superior performance than the existing label compression based approaches. The work of this chapter is published in ICML 2014 [139].

Chapter 6

GRALS: Collaborative Filtering with Graph Information

Low rank matrix completion approaches are among the most widely used collaborative filtering methods, where a partially observed matrix is available to the practitioner, who needs to impute the missing entries. Specifically, suppose there exists a ratings matrix $Y \in \mathbb{R}^{m \times n}$, and we only observe a subset of the entries $Y_{ij}, \forall (i,j) \in \Omega$, $|\Omega| = N \ll mn$. The goal is to estimate $Y_{i,j}, \forall (i,j) \notin \Omega$. To this end, one typically looks to solve one of the following (equivalent) programs:

$$\hat{Z} = \arg\min_{Z} \quad \frac{1}{2}\|\mathcal{P}_{\Omega}(Y - Z)\|_F^2 + \lambda_z \|Z\|_* \tag{6.1}$$

$$\hat{W}, \hat{H} = \arg\min_{W,H} \quad \frac{1}{2}\|\mathcal{P}_{\Omega}(Y - WH^{\top})\|_F^2 + \frac{\lambda_w}{2}\|W\|_F^2 + \frac{\lambda_h}{2}\|H\|_F^2 \tag{6.2}$$

where the nuclear norm $\|Z\|_*$, given by the sum of singular values, is a tight convex relaxation of the non convex rank penalty, and is equivalent to the regularizer in (6.2). $\mathcal{P}_{\Omega}(\cdot)$ is the projection operator that only retains those entries of the matrix that lie in the set Ω.

The materials in this Chapter have been published in [98]. I developed the algorithms, established theoretical guarantees, and conducted the experiments.

In many cases however, one not only has the partially observed ratings matrix, but also has access to additional information about the relationships between the variables involved. For example, one might have access to a social network of users. Similarly, one might have access to attributes of items, movies, etc. The nature of the attributes can be fairly arbitrary, but it is reasonable to assume that "similar" users/items share "similar" attributes. A natural question to ask then, is if one can take advantage of this additional information to make better predictions. In this chapter, we assume that the row and column variables lie on graphs. The graphs may naturally be part of the data (social networks, product co-purchasing graphs) or they can be constructed from available features. The idea then is to incorporate this additional structural information into the matrix completion setting.

We not only require the resulting optimization program to enforce additional constraints on Z, but we also require it to admit efficient optimization algorithms. We show in the sections that follow that this in fact is indeed the case. We also perform a theoretical analysis of our problem when the observed entries of Y are corrupted by additive white Gaussian noise. To summarize this chapter:

- We provide a *scalable* algorithm for matrix completion graph with structural information. Our method relies on efficient Hessian-vector multiplication schemes, and is orders of magnitude faster than (stochastic) gradient descent based approaches.
- We make connections with other structured matrix factorization frame-

works. Notably, we show that our method generalizes the weighted nuclear norm [112], and methods based on Gaussian generative models [148].

- We derive consistency guarantees for graph regularized matrix completion, and empirically show that our bound is smaller than that of traditional matrix completion, where graph information is ignored.
- We empirically validate our claims, and show that our method achieves comparable error rates to other methods, while being significantly more scalable.

Related Work and Key Differences

For convex methods for matrix factorization, [43] provided a framework to use regularizers with norms other than the Euclidean norm in (6.2). [1] considered a kernel based embedding of the data, and showed that the resulting problem can be expressed as a norm minimization scheme. [112] introduced a weighted nuclear norm, and showed that the method enjoys superior performance as compared to standard matrix completion under a non-uniform sampling scheme. We show that the graph based framework considered in this chapter is in fact a generalization of the weighted nuclear norm problem, with non-diagonal weight matrices.

In the context of matrix factorization with graph structural information, [20] considered a graph regularized nonnegative matrix factorization framework and proposed a gradient descent based method to solve the problem.

In the context of recommendation systems in social networks, [77] modeled the weight of a graph edge[1] explicitly in a re-weighted regularization framework. [72] considered a similar setting to ours, but a key point of difference between all the aforementioned methods and our work in this chapter is that we consider the partially observed ratings case. There are some works developing algorithms for the situation with partially observations [60, 147, 148]; however, none of them provides statistical guarantees. Weighted norm minimization has been considered before ([82, 112]) in the context of low rank matrix completion. The thrust of these methods has been to show that despite suboptimal conditions (correlated data, non-uniform sampling), the sample complexity does not change. None of these methods use graph information. We are interested in a complementary question: *Given variables conforming to graph information, can we obtain better guarantees under uniform sampling to those achieved by traditional methods?*

6.1 Graph-Structured Matrix Factorization

Assume that the "true" target matrix can be factorized as $Z^\star = W^\star (H^\star)^\top$, and there exist a graph (V^w, G^w) whose adjacency matrix encodes the relationships between the m rows of $W^\star$ and a graph (V^h, G^h) for n rows of $H^\star$. In particular, two rows (or columns) connected by an edge in the graph are "close" to each other in the Euclidean distance. In the context of graph-based

[1] The authors call this the "trust" between links in a social network

embedding, [11, 12] proposed a smoothing term of the form

$$\frac{1}{2}\sum_{i,j} G^w_{ij}(\boldsymbol{w}_i - \boldsymbol{w}_j)^2 = \mathrm{tr}(W^\top \mathbf{Lap}(G^w)W) \tag{6.3}$$

where $\mathbf{Lap}(G^w) := D^w - G^w$ is the graph Laplacian for (V^w, G^w), where D^w is the diagonal matrix with $D^w_{ii} = \sum_{j\sim i} G^w_{ij}$. Adding (6.3) into the minimization problem (6.2) encourages solutions where $\boldsymbol{w}_i \approx \boldsymbol{w}_j$ when G^w_{ij} is large. A similar argument holds for $H^\star$ and the associated graph Laplacian $\mathbf{Lap}(G^h)$.

We would thus not only want the target matrix to be low rank, but also want the variables W, H to be faithful to the underlying graph structure. To this end, we consider the following problem:

$$\min_{W,H} \ \frac{1}{2}\|\mathcal{P}_\Omega\big(Y - WH^\top\big)\|_F^2 + \frac{\lambda_L}{2}\{\mathrm{tr}(W^\top \mathbf{Lap}(G^w)W) + \mathrm{tr}(H^\top \mathbf{Lap}(G^h)H)\} + \tag{6.4}$$

$$\frac{\lambda_w}{2}\|W\|_F^2 + \frac{\lambda_h}{2}\|H\|_F^2$$

$$\equiv \min_{W,H} \ \frac{1}{2}\|\mathcal{P}_\Omega\big(Y - WH^\top\big)\|_F^2 + \frac{1}{2}\big\{\mathrm{tr}(W^\top L_w W) + \mathrm{tr}(H^\top L_h H)\big\} \tag{6.5}$$

where $L_w := \lambda_L \mathbf{Lap}(G^w) + \lambda_w I_m$, and L_h is defined similarly. Note that we subsume the regularization parameters in the definition of L_w, L_h. Note that $\|W\|_F^2 = \mathrm{tr}(W^\top I_m W)$.

The regularizer in (6.5) encourages solutions that are smooth with respect to the corresponding graphs. However, the Laplacian matrix can be replaced by other (positive, semi-definite) matrices that encourage structure by different means. Indeed, a very general class of Laplacian based regularizers

was considered in [109], where one can replace L_w by a function:

$$\langle \boldsymbol{x}, \tau(\mathbf{Lap}(G))\boldsymbol{x}\rangle \quad \text{where} \quad \tau(\mathbf{Lap}(G)) \equiv \sum_{i=1}^{|V|} \tau(\lambda_i)\boldsymbol{q}_i\boldsymbol{q}_i^\top,$$

where $\{(\lambda_i, \boldsymbol{q}_i)\}$ constitute the eigen-system of $\mathbf{Lap}(G)$ and $\tau(\lambda_i)$ is a scalar function of the eigenvalues. Our case corresponds to $\tau(\cdot)$ being the identity function. We briefly summarize other schemes that fit neatly into (6.5), apart from the graph regularizer we consider:

Covariance matrices for variables: [148] proposed a kernelized probabilistic matrix factorization (KPMF), which is a generative model to incorporate covariance information of the variables into matrix factorization. They assumed that each row of $W^\star$, $H^\star$ is generated according to a multivariate Gaussian, and solving the corresponding MAP estimation procedure yields exactly (6.5), with $L_w = C_w^{-1}$ and $L_h = C_h^{-1}$, where C_w, C_h are the associated covariance matrices.

Feature matrices for variables: Assume that there is a feature matrix $X \in \mathbb{R}^{m\times d}$ for objects associated rows. For such X, one can construct a graph (and hence a Laplacian) using various methods such as k-nearest neighbors, ϵ-nearest neighbors etc. Moreover, one can assume that there exists a kernel $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ that encodes pairwise relations, and we can use the Kernel Gram matrix as a Laplacian.

6.2 GRALS: Graph Regularized Alternating Least Squares

In this section, we propose efficient algorithms for (6.5), which is convex with respect to W or H separately. This allows us to employ alternating minimization methods [132] to solve the problem. When Y is fully observed, [72] propose an alternating minimization scheme using block steepest descent. We deal with the partially observed setting, and propose to apply conjugate gradient (CG), which is known to converge faster than steepest descent, to solve each subproblem. We propose a very efficient Hessian-vector multiplication routine that results in the algorithm being highly scalable, compared to the (stochastic) gradient descent approaches in [77, 148].

We assume that $Y \in \mathbb{R}^{m\times n}$, $W \in \mathbb{R}^{m\times k}$ and $H \in \mathbb{R}^{n\times k}$. When optimizing H with W fixed, we obtain the following sub-problem.

$$\min_{H} \; f(H) = \frac{1}{2}\|\mathcal{P}_\Omega\big(Y - WH^\top\big)\|_F^2 + \frac{1}{2}\operatorname{tr}(H^\top L_h H). \tag{6.6}$$

Optimizing W while H fixed is similar, and thus we only show the details for solving (6.6). Since L_h is nonsingular, (6.6) is strongly convex.[2] We first present our algorithm for the fully observed case, since it sets the groundwork for the partially observed setting.

[2] In fact, a nonsingular L_h can be handled using proximal updates, and our algorithm will still apply

6.2.1 Fully Observed Case

As in [20, 72] among others, there may be scenarios where Y is completely observed, and the goal is to find the row/column embeddings that conform to the corresponding graphs. In this case, the loss term in (6.6) is simply $\|Y - WH^\top\|_F^2$. Thus, setting $\nabla f(H) = 0$ is equivalent to solving the following Sylvester equation for an $n \times k$ matrix H:

$$HW^\top W + L_h H = Y^\top W. \tag{6.7}$$

(6.7) admits a closed form solution. However the standard Bartels-Stewart algorithm for the Sylvester equation requires transforming both $W^\top W$ and L_h into Schur form (diagonal in our case where $W^\top W$ and L_h are symmetric) by the QR algorithm, which is time consuming for a large L_h. Thus, we consider applying conjugate gradient (CG) to minimize $f(H)$ directly. We define the following quadratic function:

$$g(\boldsymbol{s}) := \frac{1}{2}\boldsymbol{s}^\top M\boldsymbol{s} - \mathrm{vec}\big(Y^\top W\big)^\top \boldsymbol{s}, \quad \boldsymbol{s} \in \mathbb{R}^{nk}, \quad M = I_k \otimes L_h + (W^\top W) \otimes I_n$$

It is not hard to show that $f(H) = g(\mathrm{vec}(H))$ and so we apply CG to minimize $g(\boldsymbol{s})$.

The most crucial step in CG is the Hessian-vector multiplication. Using the identity $(B^\top \otimes A)\,\mathrm{vec}(X) = \mathrm{vec}(AXB)$, it follows that

$$(I_k \otimes L_h)\boldsymbol{s} = \mathrm{vec}(L_h S), \quad \text{and} \quad \big((W^\top W) \otimes I_n\big)\boldsymbol{s} = \mathrm{vec}\big(SW^\top W\big),$$

where $\text{vec}(S) = \boldsymbol{s}$. Thus the Hessian-vector multiplication can be implemented by a series of matrix multiplications as follows.

$$M\boldsymbol{s} = \text{vec}\big(L_h S + S(W^\top W)\big),$$

where $W^\top W$ can be pre-computed and stored in $O(k^2)$ space. The details are presented in Algorithm 6.1. The time complexity for a single CG iteration is $O(nnz(L_h)k + nk^2)$, where $nnz(\cdot)$ is the number of non zeros. Since in most practical applications k is generally small, the complexity is essentially $O(nnz(L_h)k)$ as long as $nk \leq nnz(L_h)$.

Algorithm 6.1 Hv-Multiplication for $g(\boldsymbol{s})$

- **Given:** Matrices L_h, W
- **Initialization:** $\boldsymbol{G} = W^\top W$
- **Multiplication:** $\nabla^2 g(\boldsymbol{s}_0)\boldsymbol{s}$**:**
  1 **Input:** $S \in \mathbb{R}^{n\times k}$ s.t. $\boldsymbol{s} = \text{vec}(S)$
  2 $A \leftarrow S\boldsymbol{G} + L_h S$
  3 **Return**: $\text{vec}(A)$

Algorithm 6.2 Hv-Multiplication for $g_\Omega(\boldsymbol{s})$

- **Given:** Matrices L_h, W, Ω
- **Multiplication:** $\nabla^2 g(\boldsymbol{s}_0)\boldsymbol{s}$**:**
  1 **Input:** $S \in \mathbb{R}^{k\times n}$ s.t. $\boldsymbol{s} = \text{vec}(S)$
  2 Compute $K = [\boldsymbol{k}_1, \ldots, \boldsymbol{k}_n]$ s.t. $\boldsymbol{k}_j \leftarrow \sum_{i\in\Omega_j} (\boldsymbol{w}_i^\top \boldsymbol{s}_j)\boldsymbol{w}_i$
  3 $A \leftarrow K + SL_h$
  4 **Return**: $\text{vec}(A)$

6.2.2 Partially Observed Case

In this case, the loss term of (6.6) becomes $\sum_{(i,j)\in\Omega}(Y_{ij}-\boldsymbol{w}_i^\top \boldsymbol{h}_j)^2$, where $\boldsymbol{w}_i^\top$ is the i-th row of W and $\boldsymbol{h}_j$ is the j-th column of $H^\top$. Similar to the fully observed case, we can define:

$$g_\Omega(\boldsymbol{s}) := \frac{1}{2}\boldsymbol{s}^\top M_\Omega \boldsymbol{s} - \mathrm{vec}\big(W^\top Y\big)^\top \boldsymbol{s},$$

where $M_\Omega = \bar{B} + L_h \otimes I_k$, $\bar{B} \in \mathbb{R}^{nk\times nk}$ is a block diagonal matrix with n diagonal blocks $B_j \in \mathbb{R}^{k\times k}$. $B_j = \sum_{i\in\Omega_j} \boldsymbol{w}_i\boldsymbol{w}_i^\top$, where $\Omega_j = \{i : (i,j) \in \Omega\}$. Again, we can see $f(H) = g_\Omega(\mathrm{vec}\big(H^\top\big))$. Note that the transpose $H^\top$ is used here instead of H, which is used in the fully observed case.

For a given $\boldsymbol{s}$, let $S = [\boldsymbol{s}_1, \ldots \boldsymbol{s}_j, \ldots \boldsymbol{s}_n]$ be a matrix such that $\mathrm{vec}(S) = \boldsymbol{s}$ and $K = [\boldsymbol{k}_1, \ldots, \boldsymbol{k}_j, \ldots, \boldsymbol{k}_n]$ with $\boldsymbol{k}_j = B_j\boldsymbol{s}_j$. Then $\bar{B}\boldsymbol{s} = \mathrm{vec}(K)$. Note that since n can be very large in practice, it may not be feasible to compute and store all B_j in the beginning. Alternatively, $B_j\boldsymbol{s}_j$ can be computed in $O(|\Omega_j|k)$ time as follows.

$$B_j\boldsymbol{s}_j = \sum_{i\in\Omega_j}(\boldsymbol{w}_i^\top \boldsymbol{s}_j)\boldsymbol{w}_i.$$

Thus $\bar{B}\boldsymbol{s}$ can be computed in $O(|\Omega|k)$ time, and the Hessian-vector multiplication $M_\Omega\boldsymbol{s}$ can be done in $O(|\Omega|k + nnz(L_h)k)$ time. See Algorithm 6.2 for a detailed procedure. As a result, each CG iteration for minimizing $g_\Omega(\boldsymbol{s})$ is also very cheap.

Remark on Convergence. In [8], it is shown that any local minimizer of (6.5) is a global minimizer of (6.5) if k is larger than the true rank of

the underlying matrix.[3] From [132], the alternating minimization procedure is guaranteed to globally converge to a block coordinate-wise minimum[4] of (6.5). The converged point might not be a local minimizer, but it still yields good performance in practice. Most importantly, since the updates are cheap to perform, our algorithm scales well to large datasets.

6.3 Convex Connection via Generalized Weighted Nuclear Norm

We now show that the regularizer in (6.5) can be cast as a generalized version of the weighted nuclear norm. The weights in our case will correspond to the scaling factors introduced on the matrices W, H due to the eigenvalues of the shifted graph Laplacians L_w, L_h respectively.

6.3.1 A weighted atomic norm:

From [23], we know that the nuclear norm is the gauge function induced by the atomic set: $\mathscr{A}_* = \{\boldsymbol{w}_i \boldsymbol{h}_i^\top : \|\boldsymbol{w}_i\| = \|\boldsymbol{h}_i\| = 1\}$. Note that all rank one matrices in $\mathscr{A}_*$ have unit Frobenius norm. Now, assume $P = [\mathbf{p}_1, \ldots, \mathbf{p}_m] \in \mathbb{R}^{m \times m}$ is a basis of $\mathbb{R}^m$ and $S_p^{-1/2}$ is a diagonal matrix with $(S_p^{-1/2})_{ii} \geq 0$ encoding the "preference" over the space spanned by $\mathbf{p}_i$. The more the preference, the larger the value. Similarly, consider the basis Q and the preference $S_q^{-1/2}$ for $\mathbb{R}^n$. Let $A = PS_p^{-1/2}$ and $B = QS_q^{-1/2}$, and consider the following

[3] The authors actually show this for a more general class of regularizers.
[4] Nash equilibrium is used in [132].

"preferential" atomic set:

$$\mathscr{A} := \{\mathbb{a}_i = \boldsymbol{w}_i \boldsymbol{h}_i^\top : \boldsymbol{w}_i = A\boldsymbol{u}_i, \boldsymbol{h}_i = B\boldsymbol{v}_i, \|\boldsymbol{u}_i\| = \|\boldsymbol{v}_i\| = 1\}. \tag{6.8}$$

Clearly, each atom $\mathbb{a}$ in $\mathscr{A}$ has non-unit Frobenius norm. This atomic set allows for biasing of the solutions towards certain atoms. We then define a corresponding atomic norm:

$$\|Z\|_{\mathscr{A}} = \inf \sum_{\mathbb{a}_i \in \mathscr{A}} |c_i| \quad \text{s.t. } Z = \sum_{\mathbb{a}_i \in \mathscr{A}} c_i \mathbb{a}_i. \tag{6.9}$$

It is not hard to verify that $\|Z\|_{\mathscr{A}}$ is a norm and $\{Z : \|Z\|_{\mathscr{A}} \leq \tau\}$ is closed and convex.

6.3.2 Equivalence to Graph Regularization

The graph regularization (6.5) can be shown to be a special case of the atomic norm (6.9), as a consequence of the following result:

Theorem 6.1. *For any $A = PS_p^{-1/2}$, $B = QS_q^{-1/2}$, and corresponding weighted atomic set $\mathscr{A}$,*

$$\|Z\|_{\mathscr{A}} = \inf_{W,H} \quad \frac{1}{2}\{\|A^{-1}W\|_F^2 + \|B^{-1}H\|_F^2\} \quad s.t. \quad Z = WH^\top.$$

See the Appendix in [98] for a proof for a proof. Theorem 6.1 immediately leads us to the following equivalence result:

Corollary 6.1. *Let $L_w = U_w S_w U_w^\top$ and $L_h = U_h S_h U_h^\top$ be the eigen decomposition for L_w and L_h. We have*

$$\mathrm{tr}\big(W^\top L_w W\big) = \|A^{-1}W\|_F^2, \qquad \text{and} \quad \mathrm{tr}\big(H^\top L_h H\big) = \|B^{-1}H\|_F^2,$$

where $A = U_w S_w^{-1/2}$ and $B = U_h S_h^{-1/2}$. As a result, $\|M\|_{\mathscr{A}}$ with the preference pair $(U_w, S_w^{-1/2})$ for the column space and the preference pair $(U_h, S_h^{-1/2})$ for row space is a weighted atomic norm equivalent for the graph regularization using L_w and L_h.

The results above allow us to obtain the dual weighted atomic norm for a matrix Z

$$\|Z\|_{\mathscr{A}}^* = \|A^\top Z B\,\| = \|S_w^{-\frac{1}{2}} U_w^\top Z U_h S_h^{-\frac{1}{2}}\| \tag{6.10}$$

which is a weighted spectral norm. An elementary proof of this result can be found in [98, Appendix]. Note that we can then write

$$\|Z\|_{\mathscr{A}} = \|A^{-1} Z B^{-T}\|_* = \|S_w^{\frac{1}{2}} U_w^{-1} Z U_h^{-T} S_h^{\frac{1}{2}}\|_* \tag{6.11}$$

In [112], the authors consider a norm similar to (6.11), but with A, B being diagonal matrices. In the spirit of their nomenclature, we refer to the norm in (6.11) as the *generalized* weighted nuclear norm.

6.4 Statistical Consistency in the Presence of Noisy Measurements

In this section, we derive theoretical guarantees for the graph regularized low rank matrix estimators. We first introduce some additional notation.

We assume that there is a $m \times n$ matrix $Z^\star$ of rank k with $\|Z^\star\|_F = 1$, and $N = |\Omega|$ entries of $Z^\star$ are uniformly sampled[5] and revealed to us (i.e., $Y = \mathcal{P}_\Omega(Z^\star)$). We further assume an one-to-one mapping between the set of observed indices Ω and $\{1, 2, \ldots, N\}$ so that the t^{th} measurement is given by

$$y_t = Y_{i(t),j(t)} = \langle \boldsymbol{e}_{i(t)} \boldsymbol{e}_{j(t)}^\top, Z^\star \rangle + \frac{\sigma}{\sqrt{mn}} \eta_t \qquad \eta_t \sim \mathcal{N}(0, 1). \tag{6.12}$$

where $\langle \cdot, \cdot \rangle$ denotes the matrix trace inner product, and $i(t), j(t)$ is a randomly selected coordinate pair from $[m] \times [n]$. Let A, B are corresponding matrices defined in Corollary 6.1 for the given L_w, L_h. W.L.O.G, we assume that the minimum singular value of both L_w and L_h is 1. We then define the following graph based complexity measures:

$$\alpha_g(Z) := \sqrt{mn} \frac{\|A^{-1} Z B^{-T}\|_\infty}{\|A^{-1} Z B^{-T}\|_F}, \qquad \beta_g(Z) := \frac{\|A^{-1} Z B^{-T}\|_*}{\|A^{-1} Z B^{-T}\|_F} \tag{6.13}$$

where $\|\cdot\|_\infty$ is the element-wise ℓ_∞ norm. Finally, we assume that the true matrix $Z^\star$ can be expressed as a linear combination of atoms from (6.8) (we define $\alpha^\star := \alpha_g(Z^\star)$):

$$Z^\star = A U^\star (V^\star)^\top B^\top, \quad U^\star \in \mathbb{R}^{m \times k}, V^\star \in \mathbb{R}^{n \times k}, \tag{6.14}$$

Our goal in this section will be to characterize the solution to the following *convex* program, where the constraint set precludes selection of overly complex matrices in the sense of (6.13):

$$\hat{Z} = \arg\min_{Z \in \mathcal{C}} \frac{1}{N} \|\mathcal{P}_\Omega(Y - Z)\|_F^2 + \lambda \|Z\|_{\mathscr{A}}, \tag{6.15}$$

[5] Our results can be generalized to non uniform sampling schemes as well.

where

$$\mathcal{C} := \left\{ Z : \alpha_g(Z)\beta_g(Z) \leq \bar{c}_0 \sqrt{\frac{N}{\log(m+n)}} \right\},$$

and $\bar{c}_0$ is a constant depending on $\alpha^\star$.

A quick note on solving (6.15): since $\|\cdot\|_{\mathscr{A}}$ is a weighted nuclear norm, one can resort to proximal point methods [21], or greedy methods developed specifically for atomic norm constrained minimization [97, 117]. The latter are particularly attractive, since the greedy step reduces to computing the maximum singular vectors which can be efficiently computed using power methods. However, such methods will first involve computing the eigen decompositions of the graph Laplacians, and then storing the large, dense matrices A, B. We refrain from resorting to such methods in Section 6.5, and instead use the efficient framework derived in Section 6.2. We now state our main theoretical result:

Theorem 6.2. *Suppose we observe N entries of the form (6.12) from a matrix $Z^\star \in \mathbb{R}^{m\times n}$, with $\alpha^\star := \alpha_g(Z^\star)$ and which can be represented using at most k atoms from (6.8). Let $\hat{Z}$ be the minimizer of the convex problem (6.15) with $\lambda \geq C_1\sqrt{\frac{(m+n)\log(m+n)}{N}}$. Then, with high probability, we have*

$$\|\hat{Z} - Z^\star\|_F^2 \leq C\alpha^{\star 2} \max\{1, \sigma^2\} \frac{k(m+n)\log(m+n)}{N} + O\left(\frac{\alpha^{\star 2}}{N}\right)$$

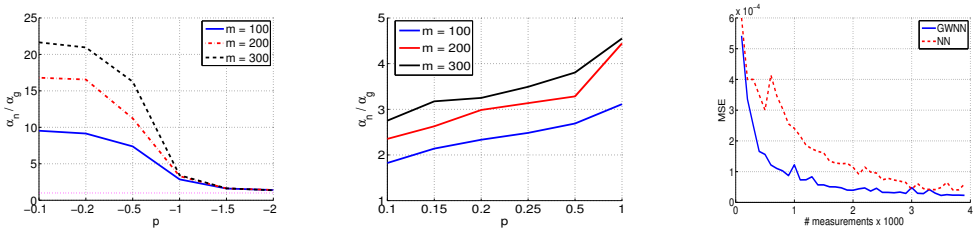
where C, C_1 are positive constants.

See [98, Appendix] for a proof.

6.4.1 Comparison to Standard Matrix Completion:

It is instructive to consider our result in the context of noisy matrix completion with uniform samples. In this case, one would replace L_w, L_h by identity matrices, effectively ignoring graph information available. Specifically, the "standard" notion of spikiness ($\alpha_n := \sqrt{mn}\frac{\|Z\|_\infty}{\|Z\|_F}$) defined in [82] will apply, and the corresponding error bound (Theorem 6.2) will have $\alpha^\star$ replaced by $\alpha_n(Z^\star)$. In general, it is hard to quantify the relationship between α_g and α_n, and a detailed comparison is an interesting topic for future work. However, we show below using simulations for various scenarios that the former is much smaller than the latter. We generate $m \times m$ matrices of rank $k = 10$, $M = U\Sigma V^\top$ with U, V being random orthonormal matrices and Σ having diagonal elements picked from a uniform$[0, 1]$ distribution. We generate graphs at random using the schemes discussed below, and set $Z = AMB^\top$, with A, B as defined in Corollary 6.1. We then compute α_n, α_g for various m.

Comparing α_g to α_n: Most real world graphs exhibit a power law degree distribution. We generated graphs with the i^{th} node having degree $(m \times i^p)$ with varying negative p values. Figure 6.1(a) shows that as $p \to 0$ from below, the gains received from using our norm is clear compared to the standard nuclear norm. We also observe that in general the weighted formulation is never worse then unweighted (The dotted magenta line is $\alpha_n/\alpha_g = 1$). The same applies for random graphs, where there is an edge between each (i, j) with varying probability p (Figure 6.1(b)).



(a) Power Law (b) Random (c) Sample Complexity

Figure 6.1: (a), (b): Ratio of spikiness measures for traditional matrix completion and our formulation. (c): Sample complexity for the nuclear norm (NN) and generalized weighted nuclear norm (GWNN)

Sample Complexity: We tested the sample complexity needed to recover a $m = n = 200$, $k = 20$ matrix, generated from a power law distributed graph with $p = -0.5$. Figure 6.1(c) again outlines that the atomic formulation requires fewer examples to get an accurate recovery. We average the results over 10 independent runs, and we used [97] to solve the atomic norm constrained problem.

6.5 Experimental Results

Comparison to Related Formulations: We compare GRALS to other methods that incorporate side information for matrix completion: the ADMM method of [60] that regularizes the entire target matrix; using known features (IMC) [57, 131]; and standard matrix completion (MC). We use the MOVIELENS $100k$ dataset,[6] that has user/movie features along with the ratings matrix. The dataset contains user features (such as age (numeric), gender

[6] http://grouplens.org/datasets/movielens/

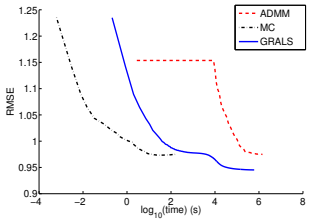


Figure 6.2: Time comparison of different methods on MOVIELENS 100k

Method	**RMSE**
IMC	1.653
Global mean	1.154
User mean	1.063
Movie mean	1.033
ADMM	0.996
MC	0.973
GRALS	**0.945**

Table 6.1: Performance comparison: RMSE on the Movielens dataset.

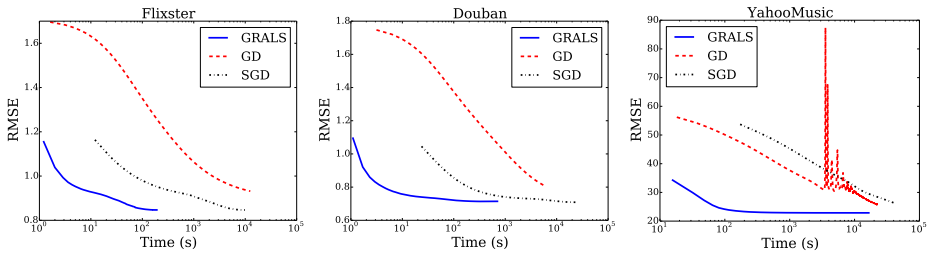
Table 6.2: Data statistics for graph regularized MF datasets.

Dataset	# users	# items	# ratings	# links	rank used
Flixster ([58])	147,612	48,794	8,196,077	2,538,746	10
Douban ([77])	129,490	58,541	16,830,839	1,711,802	10
Yahoo-Music2 ([34])	249,012	296,111	55,749,965	57,248,136	20

(binary), and occupation), which we map into a 22 dimensional feature vector per user. We then construct a 10-nearest neighbor graph using the euclidean distance metric. We do the same for the movies, except in this case we have an 18 dimensional feature vector per movie. For IMC, we use the feature vectors directly. We trained a model of rank 10, and chose optimal parameters by cross validation. Table 6.1 shows the RMSE obtained for the methods considered. Figure 6.2 shows that the ADMM method, while obtaining a reasonable RMSE does not scale well, since one has to compute an SVD at each iteration.

Scalability of GRALS: We now demonstrate that the proposed GRALS method is more efficient than other state-of-the-art methods for solving the

graph-regularized matrix factorization problem (6.5). We compare GRALS to the SGD method in [148], and GD: ALS with simple gradient descent. We consider three large-scale real-world collaborate filtering datasets with graph information: see Table 6.2 for details.[7] We randomly select 90% of ratings as the training set and use the remaining 10% as the test set. All the experiments are performed on an Intel machine with Xeon CPU E5-2680 v2 Ivy Bridge and enough RAM. Figure 6.3 shows orders of magnitude improvement in time compared to SGD. More experimental results are provided in the supplementary material.



(a) Flixster (b) Douban (c) Yahoo-Music2

Figure 6.3: Comparison of GRALS, GD, and SGD. The x-axis is the computation time in log-scale.

6.6 Summary of the Contributions

In this chapter, we consider the problem of collaborative filtering with graph information for users and/or items, and show that it can be cast as a generalized weighted nuclear norm problem. We derive statistical consistency

[7] See more details in [98, Appendix].

guarantees for our method, and develop a highly scalable alternating minimization method. Experiments on large real world datasets show that our method achieves ~ 2 orders of magnitude speedups over competing approaches. The work of this chapter is published in NIPS 2015 [98].

Chapter 7

TRMF: Temporal Regularized Matrix Factorization for High-dimensional Time Series Prediction

Time series analysis is a central problem in many applications such as demand forecasting and climatology. Often, such applications require methods that are highly scalable to handle a very large number (n) of possibly inter-dependent one-dimensional time series and/or have a large time frame (T). For example, climatology applications involve data collected from possibly thousands of sensors, every hour (or less) over several years. Similarly, a store tracking its inventory would track thousands of items every day for multiple years. Not only is the scale of such problems huge, but they might also involve missing values, due to sensor malfunctions, occlusions or simple human errors. Thus, modern time series applications present two challenges to practitioners: scalability to handle large n and T and the flexibility to handle missing values.

Most approaches in the traditional time series literature such as autore-

The materials in this Chapter have been presented in [141, 144]. I proposed the problem formulation, developed the algorithms, established theoretical results, and conducted the experiments.

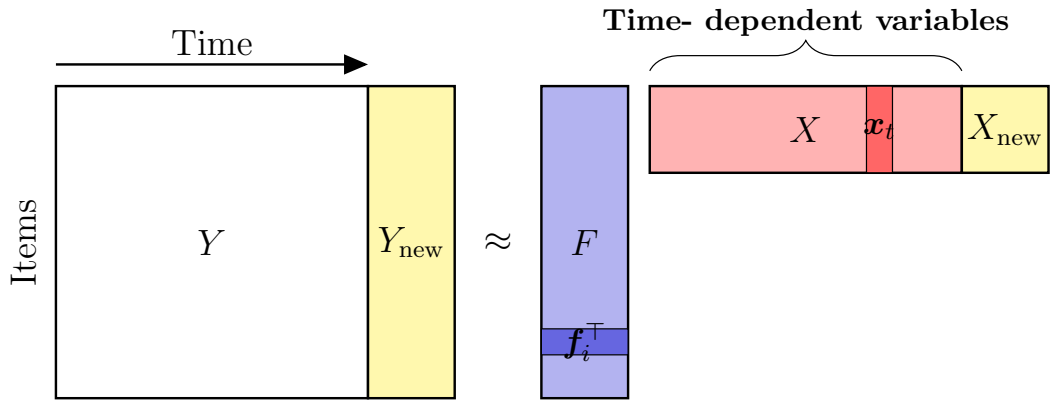


Figure 7.1: Matrix Factorization model for multiple time series. F captures features for each time series in the matrix Y, and X captures the latent and time-varying variables.

gressive (AR) models or dynamic linear models (DLM)[59, 125] focus on low-dimensional time-series data and fall short of handling the two aforementioned issues. For example, an AR model of order L requires $O(TL^2n^4 + L^3n^6)$ time to estimate $O(Ln^2)$ parameters, which is prohibitive even for moderate values of n. Similarly, Kalman filter based DLM approaches need $O(kn^2T + k^3T)$ computation cost to update parameters, where k is the latent dimensionality, which is usually chosen to be larger than n in many situations [92]. As a specific example, the maximum likelihood estimator implementation in the widely used R-DLM package [91], which relies on a general optimization solver, cannot scale beyond n in the tens. (See Appendix 7.5.4 for details). On the other hand, for models such as AR, the flexibility to handle missing values can also be very challenging even for one-dimensional time series [4], let alone the difficulty to handle high dimensional time series.

A natural way to model high-dimensional time series data is in the form of a matrix, with rows corresponding to each one-dimensional time series and columns corresponding to time points. In light of the observation that n time series are usually highly correlated with each other, there have been some attempts to apply low-rank matrix factorization (MF) or matrix completion (MC) techniques to analyze high-dimensional time series [26, 95, 103, 130, 146]. Unlike the AR and DLM models above, state-of-the-art MF methods *scale linearly* in n, and hence can handle large datasets. Let $Y \in \mathbb{R}^{n \times T}$ be the matrix for the observed n-dimensional time series with Y_{it} being the observation at the t-th time point of the i-th time series. Under the standard MF approach, Y_{it} is estimated by the inner product $\boldsymbol{f}_i^\top \boldsymbol{x}_t$, where $\boldsymbol{f}_i \in \mathbb{R}^k$ is a k-dimensional latent embedding for the i-th time series, and $\boldsymbol{x}_t \in \mathbb{R}^k$ is a k-dimensional latent temporal embedding for the t-th time point. We can stack the $\boldsymbol{x}_t$s into the columns into a matrix $X \in \mathbb{R}^{k \times T}$ and $\boldsymbol{f}_i^\top$ into the rows of $F \in \mathbb{R}^{n \times k}$ (Figure 7.1) to get $Y \approx FX$. We can then solve:

$$\min_{F,X} \sum_{(i,t) \in \Omega} \left(Y_{it} - \boldsymbol{f}_i^\top \boldsymbol{x}_t\right)^2 + \lambda_f \mathcal{R}_f(F) + \lambda_x \mathcal{R}_x(X), \tag{7.1}$$

where Ω is the set of the observed entries. $\mathcal{R}_f(F)$, $\mathcal{R}_x(X)$ are regularizers for F and X, which usually play a role to avoid overfitting and/or to encourage some specific *temporal structures* among the embeddings. It is clear that the common choice of the regularizer $\mathcal{R}_x(X) = \|X\|_F$ is no longer appropriate for time series applications, as it does not take into account the ordering among the temporal embeddings $\{\boldsymbol{x}_t\}$. Most existing MF approaches [26, 95, 103, 130, 146] adapt

graph-based approaches to handle temporal dependencies. Specifically, the dependencies are described by a weighted similarity graph and incorporated through a Laplacian regularizer [109]. However, graph-based regularization fails in cases where there are negative correlations between two time points. Furthermore, unlike scenarios where explicit graph information is available with the data (such as a social network or product co-purchasing graph for recommender systems), explicit temporal dependency structure is usually unavailable and has to be inferred or approximated. Hence, this approach requires practitioners to either perform a separate procedure to estimate the dependencies or consider very short-term dependencies with simple fixed weights. Moreover, existing MF approaches, while yielding good estimations for missing values in past points, are poor in terms of forecasting future values, which is the problem of interest in time series analysis.

In this chapter, we propose a novel temporal regularized matrix factorization framework (TRMF) for high-dimensional time series analysis. In TRMF, we consider a principled approach to **describe** the structure of temporal dependencies among latent temporal embeddings $\{\boldsymbol{x}_t\}$ and design a temporal regularizer to **incorporate** this temporal structure into the standard MF formulation. Unlike most existing MF approaches, our TRMF method supports data-driven temporal dependency learning and also brings the ability to **forecast** future values to a matrix factorization approach. In addition, inherited from the property of MF approaches, TRMF can easily handle high-dimensional time series data even in the presence of many missing values. As

a specific example, we demonstrate a novel autoregressive temporal regularizer which encourages AR structure among temporal embeddings $\{\boldsymbol{x}_t\}$. We also make connections between the proposed regularization framework and graph-based approaches [109], where even negative correlations can be accounted for. This connection not only leads to better understanding about the dependency structure incorporated by our framework but also brings the benefit of using off-the-shelf efficient solvers such as GRALS [98] directly to solve TRMF.

Organization. In Section 7.1, we review the existing approaches and their limitations on data with temporal dependencies. We present the proposed TRMF framework in Section 7.2, and show that the method is highly general and can be used for a variety of time series applications. We introduce a novel AR temporal regularizer in Section 7.3, and make connections to graph-based regularization approaches. We demonstrate the superiority of the proposed approach via extensive experimental results in Section 7.4.

7.1 Motivations: Existing Approaches and Limitations

7.1.1 Classical Time-Series Models

Models such as AR and DLM are not suitable for modern multiple high-dimensional time series data (i.e., both n and T are large) due to their inherent computational inefficiency. To avoid overfitting in AR models, there have been studies with various structured transition matrices such as low rank and sparse matrices [44, 78, 86]. The focus of most of this research has been on obtaining better statistical guarantees, and the issue of scalability of AR

models remains open. On the other hand, it is also challenging for many classic time-series models to deal with data that has many missing values [4].

In many situations where the model parameters are either given or designed by practitioners, the Kalman filter approach is used to perform forecasting, while the Kalman smoothing approach is used to impute missing entries. When model parameters are unknown, EM algorithms are applied to estimate both the model parameters and latent embeddings for DLM [38, 69, 70, 106, 113]. As most EM approaches for DLM contain the Kalman filter as a building block, they cannot scale to very high dimensional time series data. Indeed, as shown in Section 7.4, the popular R package for DLM's does not scale beyond data with tens of dimensions.

7.1.2 Existing Matrix Factorization Approaches for Data with Temporal Dependencies

In standard MF (7.1), the squared Frobenius norm $\mathcal{R}_x(X) = \|X\|_F^2 = \sum_{t=1}^{T} \|\boldsymbol{x}_t\|^2$ is usually the regularizer of choice for X. Because squared Frobenius norm assumes no dependencies among $\{\boldsymbol{x}_t\}$, standard MF formulation is *invariant to column permutation* and not applicable to data with temporal dependencies. Hence most existing temporal MF approaches turn to the framework of graph-based regularization [109] for temporally dependent $\{\boldsymbol{x}_t\}$, with a graph encoding the temporal dependencies.

Graph regularization for temporal dependencies: The framework of graph-based regularization is an approach to describe and incorporate general

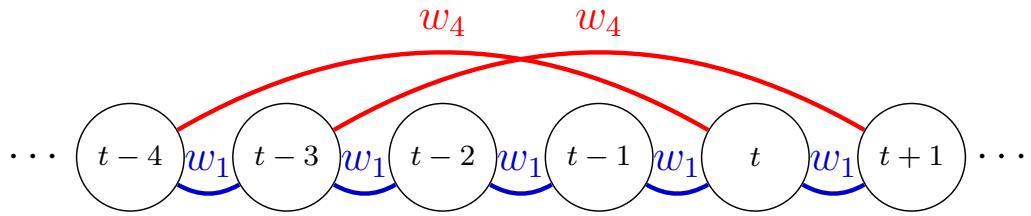


Figure 7.2: Graph-based regularization for temporal dependencies.

dependencies among variables. Let G be a graph over $\{\boldsymbol{x}_t\}$ and G_{ts} be the edge weight between the t-th node and s-th node. A popular regularizer to include as part of an objective function is the following:

$$\mathcal{R}_x(X) = \mathcal{G}(X \mid G, \eta) := \frac{1}{2}\sum_{t\sim s} G_{ts}\|\boldsymbol{x}_t - \boldsymbol{x}_s\|^2 + \frac{\eta}{2}\sum_t \|\boldsymbol{x}_t\|^2, \tag{7.2}$$

where $t \sim s$ denotes an edge between t-th node and s-th node, and the second summation term is used to guarantee strong convexity. A large G_{ts} will ensure that $\boldsymbol{x}_t$ and $\boldsymbol{x}_s$ are close to each other in Euclidean distance, when (7.2) is minimized. Note that to guarantee the convexity of $\mathcal{G}(X \mid G, \eta)$, we need $G_{ts} \geq 0$.

To apply graph-based regularizers to *temporal dependencies*, we need to specify the (repeating) dependency pattern by a lag set $\mathcal{L}$ and a weight vector $\boldsymbol{w}$ such that all the edges $t \sim s$ of distance l (i.e., $|s-t| = l$) share the same weight $G_{ts} = w_l$. See Figure 7.2 for an example with $\mathcal{L} = \{1, 4\}$. Given $\mathcal{L}$ and $\boldsymbol{w}$, the corresponding graph regularizer becomes

$$\mathcal{G}(X \mid G, \eta) = \frac{1}{2}\sum_{l\in\mathcal{L}}\sum_{t:t>l} w_l(\boldsymbol{x}_t - \boldsymbol{x}_{t-l})^2 + \frac{\eta}{2}\sum_t \|\boldsymbol{x}_t\|^2. \tag{7.3}$$

This direct use of graph-based approach, while intuitive, has two issues: **a)** there might be negatively correlated dependencies between two time points;

b) unlike many applications where such regularizers are used, the explicit temporal dependency structure is usually not available and has to be inferred. As a result, most existing approaches consider only very simple temporal dependencies such as a small size of $\mathcal{L}$ (e.g., $\mathcal{L} = \{1\}$) and/or uniform weights (e.g., $w_l = 1,\ \forall l \in \mathcal{L}$). For example, a simple chain graph is considered to design the smoothing regularizer in TCF [130]. This leads to poor forecasting abilities of existing MF methods for large-scale time series applications.

7.1.3 Challenges to Learn Temporal Dependencies

One could try to learn the weights w_l automatically, by using the same regularizer as in (7.3) but with the weights unknown. This would lead to the following optimization problem:

$$\min_{F,X,\boldsymbol{w}\geq\mathbf{0}} \sum_{(i,t)\in\Omega} \left(Y_{it} - \boldsymbol{f}_i^\top \boldsymbol{x}_t\right)^2 + \lambda_f \mathcal{R}_f(F) + \frac{\lambda_x}{2} \sum_{l\in\mathcal{L}} \sum_{t:t-l>0} w_l (\boldsymbol{x}_t - \boldsymbol{x}_{t-l})^2 + \frac{\lambda_x \eta}{2} \sum_t \|\boldsymbol{x}_t\|^2, \quad (7.4)$$

where $\mathbf{0}$ is the zero vector, and $\boldsymbol{w} \geq \mathbf{0}$ is the constraint imposed by graph regularization.

It is not hard to see that the above optimization yields the trivial all-zero solution for $\boldsymbol{w}^*$, meaning the objective function is minimized when no temporal dependencies exist! To avoid the all zero solution, one might want to impose a simplex constraint on $\boldsymbol{w}$ (i.e., $\sum_{l\in\mathcal{L}} w_l = 1$). Again, it is not hard to see that this will result in $\boldsymbol{w}^*$ being a 1-sparse vector, with w_{l^*} being 1, where $l^* = \arg\min_{l\in\mathcal{L}} \sum_{t:t>l} \|\boldsymbol{x}_t - \boldsymbol{x}_{t-l}\|^2$. Thus, looking to learn the weights

automatically by simply plugging in the regularizer in the MF formulation is not a viable option.

7.2 Temporal Regularized Matrix Factorization

In order to resolve the limitations mentioned in Sections 7.1.2 and 7.1.3, we propose the **T**emporal **R**egularized **M**atrix **F**actorization (TRMF) framework, which is a novel approach to incorporate temporal dependencies into matrix factorization models. Unlike the aforementioned graph-based approaches, we propose to use well-studied time series models to describe temporal dependencies among $\{\boldsymbol{x}_t\}$ explicitly. Such models take the form:

$$\boldsymbol{x}_t = M_\Theta(\{\boldsymbol{x}_{t-l} : l \in \mathcal{L}\}) + \boldsymbol{\epsilon}_t, \tag{7.5}$$

where $\boldsymbol{\epsilon}_t$ is a Gaussian noise vector, and M_Θ is the time-series model parameterized by $\mathcal{L}$ and Θ. $\mathcal{L}$ is a set containing the lag indices l, denoting a dependency between t-th and $(t-l)$-th time points, while Θ captures the weighting information of temporal dependencies (such as the transition matrix in AR models). To incorporate the temporal dependency into the standard MF formulation (7.1), we propose to design a new regularizer $\mathcal{T}_{\mathrm{M}}(X \mid \Theta)$ which encourages the structure induced by M_Θ.

Taking a standard approach to model time series, we set $\mathcal{T}_{\mathrm{M}}(X \mid \Theta)$ be the negative log likelihood of observing a particular realization of the $\{\boldsymbol{x}_t\}$ for a given model M_Θ:

$$\mathcal{T}_{\mathrm{M}}(X \mid \Theta) = -\log \mathbb{P}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T \mid \Theta). \tag{7.6}$$

When Θ is given, we can use $\mathcal{R}_x(X) = \mathcal{T}_{\mathrm{M}}(X \mid \Theta)$ in the MF formulation (7.1) to encourage $\{\boldsymbol{x}_t\}$ to follow the temporal dependency induced by M_Θ. When the Θ is unknown, we can treat Θ as another set of variables and include another regularizer $\mathcal{R}_\theta(\Theta)$ into (7.1):

$$\min_{F,X,\Theta} \quad \sum_{(i,t)\in\Omega} \left(Y_{it} - \boldsymbol{f}_i^\top \boldsymbol{x}_t\right)^2 + \lambda_f \mathcal{R}_f(F) + \lambda_x \mathcal{T}_{\mathrm{M}}(X \mid \Theta) + \lambda_\theta \mathcal{R}_\theta(\Theta), \tag{7.7}$$

which be solved by an alternating minimization procedure over F, X, and Θ.

Data-driven Temporal Dependency Learning in TRMF: Recall that in Section 7.1.3, we showed that directly using graph based regularizers to incorporate temporal dependencies leads to trivial solutions for the weights. TRMF circumvents this issue. When F and X are fixed, (7.7) is reduced to:

$$\min_{\Theta} \quad \lambda_x \mathcal{T}_{\mathrm{M}}(X \mid \Theta) + \lambda_\theta \mathcal{R}_\theta(\Theta), \tag{7.8}$$

which is a maximum-a-posterior (MAP) estimation problem (in the Bayesian sense) to estimate the best Θ for a given $\{\boldsymbol{x}_t\}$ under the M_Θ model. There are well-developed algorithms to solve (7.8) and obtain non-trivial Θ. Thus, unlike most existing temporal matrix factorization approaches where the strength of dependencies is fixed, Θ in TRMF can be learned automatically from data.

Time Series Analysis with TRMF: We can see that TRMF (7.7) lends itself seamlessly to handle a variety of commonly encountered tasks in analyzing data with temporal dependency:

- **Time-series Forecasting:** Once we have M_Θ for latent embeddings $\{\boldsymbol{x}_t : 1, \ldots, T\}$, we can use it to predict future latent embeddings

$$\{\boldsymbol{x}_t : t > T\}$$

  and have the ability to obtain non-trivial forecasting results for $\boldsymbol{y}_t = F\boldsymbol{x}_t$ for $t > T$.
- **Missing-value Imputation:** In some time-series applications, some entries in Y might be unobserved, for example, due to faulty sensors in electricity usage monitoring or occlusions in the case of motion recognition in video. We can use $\boldsymbol{f}_i^\top \boldsymbol{x}_t$ to impute these missing entries, much like standard matrix completion, and is useful in recommender systems [130] and sensor networks [146].
- **Time-series classification/clustering:** The obtained $\boldsymbol{f}_i$ can be used as the latent embedding for the i-th time series of Y. These latent features can be used to perform classification/clustering of the time series. Note that this can be done even when there are missing entires in the observed data, as missing entries are not a bottleneck for learning F.

Extensions to Incorporate Extra Information: In the same vein as matrix factorization approaches, TRMF (7.7) can be extended to incorporate additional information as we do in Chapters 5-6:

- **Known features for time series:** In many applications, one is given additional features along with the observed time series. Specifically,

given a set of feature vectors $\{\boldsymbol{a}_i \in \mathbb{R}^d\}$ for each row of Y, we can look to solve

$$\min_{F,X,\Theta} \quad \sum_{(i,t)\in\Omega} \left(Y_{it} - \boldsymbol{a}_i^\top F \boldsymbol{x}_t\right)^2 + \lambda_f \mathcal{R}_f(F) + \lambda_x \mathcal{T}_{\mathrm{M}}(X \mid \Theta) + \lambda_\theta \mathcal{R}_\theta(\Theta). \tag{7.9}$$

That is, the observation Y_{it} is posited to be a bilinear function of the feature vector $\boldsymbol{a}_i$ and the latent vector $\boldsymbol{x}_t$. Such an inductive framework has two advantages: we can generalize TRMF to a new time series without any observations up to time T (i.e., a new row i' of Y without any observations). As long as the feature vector $\boldsymbol{a}_{i'}$ is available, the model learned by TRMF can be used to estimate $Y_{i't} = \boldsymbol{a}_{i'}^\top F \boldsymbol{x}_t, \ \forall t$. Furthermore, prediction can be significantly sped up when $d \ll n$, since the dimension of F is reduced from $n \times k$ to $d \times k$. Such methods for standard multi-label learning and matrix completion have been previously considered in [57, 131, 139].

- **Graph information among time series:** Often, separate features for the time series are not known, but other relational information is available. When a graph that encodes pairwise interactions among multiple time series is available, one can incorporate this graph in our framework using the graph regularization approach (7.2). Such cases are common in inventory and sales tracking, where sales of one item is related to sales of other items. Given a graph G^f describing the relationship among

multiple time series, we can formulate a graph regularized problem:

$$\min_{F,X,\Theta} \quad \sum_{(i,t)\in\Omega} \big(Y_{it} - \boldsymbol{f}_i^\top \boldsymbol{x}_t\big)^2 + \lambda_f \mathcal{G}\big(F \mid G^f, \eta\big) + \lambda_x \mathcal{T}_{\mathrm{M}}(X \mid \Theta) + \lambda_\theta \mathcal{R}_\theta(\Theta), \tag{7.10}$$

where $\mathcal{G}\big(F \mid G^f, \eta\big)$ is the graph regularizer defined in (7.2) capturing pairwise interactions between time series. Graph regularized matrix completion methods have been previously considered in [98, 148].

- **Temporal-regularized tensor factorization:** Naturally, TRMF can be easily extended to analyze temporal collaborative filtering applications [113, 130], where the targeted data is a tensor with certain modes evolving over time. For example, consider $\mathcal{Y} \in \mathbb{R}^{m\times n\times T}$ be a 3-way tensor with Y_{ijt} encoding the rating of the i-th user for the j-th item at time point t. We can consider the following temporal regularization tensor factorization (TRTF) with $\mathcal{T}_{\mathrm{M}}(X \mid \Theta)$ as follows:

$$\min_{P,Q,X,\Theta} \quad \sum_{(i,j,t)\in\Omega} \big(Y_{ijt} - \langle \boldsymbol{p}_i, \boldsymbol{q}_j, \boldsymbol{x}_t\rangle\big)^2 + \lambda_p \mathcal{R}_p(P) + \mathcal{R}_q(Q) + \mathcal{T}_{\mathrm{M}}(X \mid \Theta) + \mathcal{R}_\theta(\Theta), \tag{7.11}$$

where $P = [\boldsymbol{p}_1, \cdots, \boldsymbol{p}_m]^\top \in \mathbb{R}^{m\times k}$ and $Q = [\boldsymbol{q}_1, \cdots, \boldsymbol{q}_n]^\top \in \mathbb{R}^{n\times k}$ are the latent embeddings for the m users and n items, respectively, and with some abuse of notation, we define $\langle \boldsymbol{p}_i, \boldsymbol{q}_j, \boldsymbol{x}_t\rangle = \sum_{r=1}^k p_{ir} q_{jr} x_{tr}$.

7.3 A Novel Autoregressive Temporal Regularizer

In Section 7.2, we described the TRMF framework in a very general sense, with the regularizer $\mathcal{T}_{\mathrm{M}}(X \mid \Theta)$ incorporating dependencies specified by the time series model M_Θ. In this section, we specialize this to the case of AR models, which are parameterized by a lag set $\mathcal{L}$ and weights $\mathcal{W} = \left\{W^{(l)} \in \mathbb{R}^{k \times k} : l \in \mathcal{L}\right\}$. Assume that $\boldsymbol{x}_t$ is a noisy linear combination of some previous points; that is, $\boldsymbol{x}_t = \sum_{l \in \mathcal{L}} W^{(l)} \boldsymbol{x}_{t-l} + \boldsymbol{\epsilon}_t$, where $\boldsymbol{\epsilon}_t$ is a Gaussian noise vector. For simplicity, we assume that the $\boldsymbol{\epsilon}_t \sim \mathcal{N}(0, \sigma^2 I_k)$, where I_k is the $k \times k$ identity matrix[1]. The temporal regularizer $\mathcal{T}_{\mathrm{M}}(X \mid \Theta)$ corresponding to this AR model can be written as:

$$\mathcal{T}_{\mathrm{AR}}(X \,|\mathcal{L}, \mathcal{W}, \eta) := \frac{1}{2} \sum_{t=m}^{T} \left\| \boldsymbol{x}_t - \sum_{l \in \mathcal{L}} W^{(l)} \boldsymbol{x}_{t-l} \right\|^2 + \frac{\eta}{2} \sum_{t} \|\boldsymbol{x}_t\|^2, \tag{7.12}$$

where $m := 1 + L$, $L := \max(\mathcal{L})$, and $\eta > 0$ to guarantee the strong convexity of (7.12).

TRMF allows us to learn the weights $\left\{W^{(l)}\right\}$ when they are unknown. Since each $W^{(l)} \in \mathbb{R}^{k \times k}$, there will be $|\mathcal{L}|k^2$ variables to learn, which may lead to overfitting. To prevent this and to yield more interpretable results, we consider diagonal $W^{(l)}$, reducing the number of parameters to $|\mathcal{L}|k$. To simplify notation, we use $\mathcal{W}$ to denote the $k \times L$ matrix where the l-th column constitutes the diagonal elements of $W^{(l)}$. Note that for $l \notin \mathcal{L}$, the l-th column of $\mathcal{W}$ is a zero vector. Let $\bar{\boldsymbol{x}}_r^\top = [\cdots, X_{rt}, \cdots]$ be the r-th row of X and

[1] If the (known) covariance matrix is not identity, we can suitably modify the regularizer.

$\bar{\boldsymbol{w}}_r^\top = [\cdots, \mathcal{W}_{rl}, \cdots]$ be the r-th row of $\mathcal{W}$. Then (7.12) can be written as $\mathcal{T}_{\text{AR}}(X \,|\mathcal{L}, \mathcal{W}, \eta) = \sum_{r=1}^{k} \mathcal{T}_{\text{AR}}(\bar{\boldsymbol{x}}_r \,|\mathcal{L}, \bar{\boldsymbol{w}}_r, \eta)$, where we define

$$\mathcal{T}_{\text{AR}}(\bar{\boldsymbol{x}} \,|\mathcal{L}, \bar{\boldsymbol{w}}, \eta) = \frac{1}{2}\sum_{t=m}^{T}\left(x_t - \sum_{l\in\mathcal{L}} w_l x_{t-l}\right)^2 + \frac{\eta}{2}\|\bar{\boldsymbol{x}}\|^2, \tag{7.13}$$

with x_t being the t-th element of $\bar{\boldsymbol{x}}$, and w_l being the l-th element of $\bar{\boldsymbol{w}}$.

Correlations among Multiple Time Series. Even when $\{W^l\}$ is diagonal, TRMF retains the power to capture the correlations among time series via the factors $\{\boldsymbol{f}_i\}$, since it has an effect only on the structure of latent embeddings $\{\boldsymbol{x}_t\}$. Indeed, as the i-th dimension of $\{\boldsymbol{y}_t\}$ is modeled by $\boldsymbol{f}_i^\top X$ in (7.7), the low rank F is a k dimensional latent embedding of multiple time series. This embedding captures correlations among multiple time series. Furthermore, $\{\boldsymbol{f}_i\}$ acts as time series features, which can be used to perform classification/clustering even in the presence of missing values.

Choice of Lag Index Set $\mathcal{L}$. Unlike most approaches mentioned in Section 7.1.2, the choice of $\mathcal{L}$ in TRMF is more flexible. Thus, TRMF can provide important advantages: First, because there is no need to specify the weight parameters $\mathcal{W}$, $\mathcal{L}$ can be chosen to be larger to account for long range dependencies, which also yields more accurate and robust forecasts. Second, the indices in $\mathcal{L}$ can be discontinuous so that one can easily embed *domain knowledge* about periodicity or seasonality. For example, one might consider $\mathcal{L} = \{1, 2, 3, 51, 52, 53\}$ for weekly data with a one year seasonality.

Connections to Graph Regularization. We now establish connections between $\mathcal{T}_{\text{AR}}(\bar{\boldsymbol{x}} \,|\mathcal{L}, \bar{\boldsymbol{w}}, \eta)$ and graph regularization (7.2) for matrix

factorization. Let $\bar{\mathcal{L}} := \mathcal{L} \cup \{0\}$, $w_0 = -1$ so that (7.13) is

$$\mathcal{T}_{\mathrm{AR}}(\bar{\boldsymbol{x}} \,|\mathcal{L}, \bar{\boldsymbol{w}}, \eta) = \frac{1}{2}\sum_{t=m}^{T}\left(\sum_{l\in\bar{\mathcal{L}}} w_l x_{t-l}\right)^2 + \frac{\eta}{2}\|\bar{\boldsymbol{x}}\|^2,$$

and let $\delta(d) := \left\{l \in \bar{\mathcal{L}} : l - d \in \bar{\mathcal{L}}\right\}$. We then have the following result:

Theorem 7.1. *Given a lag index set $\mathcal{L}$, weight vector $\bar{\boldsymbol{w}} \in \mathbb{R}^L$, and $\bar{\boldsymbol{x}} \in \mathbb{R}^T$, there is a weighted signed graph G^{AR} with T nodes and a diagonal matrix $D \in \mathbb{R}^{T\times T}$ such that*

$$\mathcal{T}_{AR}(\bar{\boldsymbol{x}} \,|\mathcal{L}, \bar{\boldsymbol{w}}, \eta) = \mathcal{G}\left(\bar{\boldsymbol{x}} \mid G^{AR}, \eta\right) + \frac{1}{2}\bar{\boldsymbol{x}}^\top D\bar{\boldsymbol{x}}, \tag{7.14}$$

where $\mathcal{G}\left(\bar{\boldsymbol{x}} \mid G^{AR}, \eta\right)$ is the graph regularization (7.2) with $G = G^{AR}$. Furthermore, $\forall t$ and d

$$G^{AR}_{t,t+d} = \begin{cases} \displaystyle\sum_{l\in\delta(d)}\sum_{m\le t+l\le T} -w_l w_{l-d} & \text{if } \delta(d) \neq \phi, \\ 0 & \text{otherwise,} \end{cases}$$

and

$$D_{tt} = \left(\sum_{l\in\bar{\mathcal{L}}} w_l\right)\left(\sum_{l\in\bar{\mathcal{L}}} w_l[m \le t+l \le T]\right)$$

See Section 7.5.1 for a detailed proof. From Theorem 7.1, we see that $\delta(d)$ is non-empty if and only if there are edges between time points separated by d in G^{AR}. Thus, we can construct the dependency graph for $\mathcal{T}_{\mathrm{AR}}(\bar{\boldsymbol{x}} \,|\mathcal{L}, \bar{\boldsymbol{w}}, \eta)$ by checking whether $\delta(d)$ is empty. Figure 7.3 demonstrates an example with $\mathcal{L} = \{1, 4\}$. We can see that besides edges of distance $d = 1$ and $d = 4$, there are also edges of distance $d = 3$ (dotted edges in Figure 7.3) because $4 - 3 \in \bar{\mathcal{L}}$ and $\delta(3) = \{4\}$.

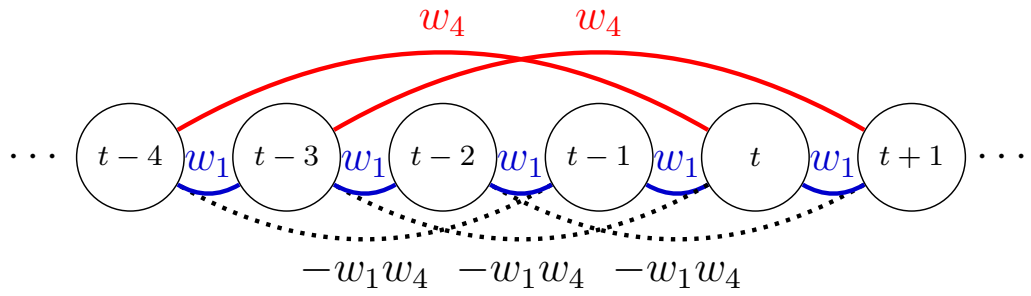


Figure 7.3: The graph structure induced by the AR temporal regularizer (7.13) with $\mathcal{L} = \{1, 4\}$.

Although Theorem 7.1 shows that AR-based regularizers are similar to the graph-based regularization framework, we note the following key differences:

- The graph G^{AR} in Theorem 7.1 contains ***both*** positive and negative edges. This implies that the AR temporal regularizer is able to support negative correlations, which the standard graph-based regularizer cannot. This can make $\mathcal{G}\big(\bar{\boldsymbol{x}} \mid G^{\text{AR}}, \eta\big)$ non-convex. The addition of the second term in (7.14), however, still leads to a convex regularizer $\mathcal{T}_{\text{AR}}(\bar{\boldsymbol{x}} \,|\mathcal{L}, \bar{\boldsymbol{w}}, \eta)$.
- Unlike (7.3) where there is freedom to specify a weight for each distance, in the graph G^{AR}, the weight values for the edges are ***more structured*** (e.g., the weight for $d = 3$ in Figure 7.3 is $-w_1w_4$). Hence, minimization w.r.t. $w's$ is not trivial, and neither are the obtained solutions.

Plugging $\mathcal{T}_{\text{M}}(X \mid \Theta) = \mathcal{T}_{\text{AR}}(X \,|\mathcal{L}, \mathcal{W}, \eta)$ into (7.7), we obtain the fol-

lowing problem:

$$\min_{F,X,\mathcal{W}} \quad \sum_{(i,t)\in\Omega} \left(Y_{it} - \boldsymbol{f}_i^\top \boldsymbol{x}_t\right)^2 + \lambda_f \mathcal{R}_f(F) + \sum_{r=1}^{k} \lambda_x \mathcal{T}_{\text{AR}}(\bar{\boldsymbol{x}}_r \,|\mathcal{L}, \bar{\boldsymbol{w}}_r, \eta) + \lambda_w \mathcal{R}_w(\mathcal{W}), \tag{7.15}$$

where $\mathcal{R}_w(\mathcal{W})$ is a regularizer for $\mathcal{W}$. We will refer to (7.15) as TRMF-AR. We can apply alternating minimization to solve (7.15). In fact, solving for each variable reduces to well known methods, for which highly efficient algorithms exist:

Updates for F. When X and $\mathcal{W}$ are fixed, the subproblem of updating F is the same as updating F while X fixed in (7.1). Thus, fast algorithms such as alternating least squares or coordinate descent can be applied directly to find F.

Updates for X. We solve

$$\arg\min_{X} \quad \sum_{(i,t)\in\Omega} \left(Y_{it} - \boldsymbol{f}_i^\top \boldsymbol{x}_t\right)^2 + \lambda_x \sum_{r=1}^{k} \mathcal{T}_{\text{AR}}(\bar{\boldsymbol{x}}_r \,|\mathcal{L}, \bar{\boldsymbol{w}}_r, \eta).$$

From Theorem 7.1, we see that $\mathcal{T}_{\text{AR}}(\bar{\boldsymbol{x}} \,|\mathcal{L}, \bar{\boldsymbol{w}}, \eta)$ shares the same form as the graph regularizer, and we can apply GRALS [98] to find X.

Updates for $\mathcal{W}$. How to update $\mathcal{W}$ while F and X fixed depends on the choice of $\mathcal{R}_w(\mathcal{W})$. There are many parameter estimation techniques developed for AR with various regularizers [86, 123]. For simplicity, we consider the squared Frobenius norm: $\mathcal{R}_w(\mathcal{W}) = \|\mathcal{W}\|_F^2$. As a result, each row of $\bar{\boldsymbol{w}}_r$

of $\mathcal{W}$ can be updated by solving the following one-dimensional autoregressive problem.

$$\begin{aligned}&\arg\min_{\bar{\boldsymbol{w}}} \lambda_x \mathcal{T}_{\mathrm{AR}}(\bar{\boldsymbol{x}}_r \,|\mathcal{L}, \bar{\boldsymbol{w}}, \eta) + \lambda_w \|\bar{\boldsymbol{w}}\|^2 \\ &\quad \equiv \arg\min_{\bar{\boldsymbol{w}}} \sum_{t=m}^{T} \left(x_t - \sum_{l \in \mathcal{L}} w_l x_{t-l} \right)^2 + \frac{\lambda_w}{\lambda_x} \|\bar{\boldsymbol{w}}\|^2,\end{aligned}$$

which is a simple $|\mathcal{L}|$ dimensional ridge regression problem with $T - m + 1$ instances, which can be solved efficiently by Cholesky factorization.

Note that since our method is highly modular, one can resort to *any* method to solve the optimization subproblems that arise for each module. Moreover, as mentioned in Section 7.2, TRMF can also be used with different regularization structures, making it highly adaptable.

7.3.1 Connections to Existing MF Approaches

TRMF-AR is a generalization of many existing MF approaches to handle data with temporal dependencies. Specifically, Temporal Collaborative Filtering [130] corresponds to $W^{(1)} = I_k$ on $\{\boldsymbol{x}_t\}$. The NMF method of [26] is an AR(L) model with $W^{(l)} = \alpha^{l-1}(1-\alpha) I_k, \ \forall l$, where α is pre-defined. The AR(1) model of [103, 146] has $W^{(1)} = I_n$ on $\{F\boldsymbol{x}_t\}$. Finally the DLM [59] is a latent AR(1) model with a general $W^{(1)}$, which can be estimated by EM algorithms.

Table 7.1: Data statistics for time-series datasets.

	synthetic	electricity	traffic	walmart-1	walmart-2
n	16	370	963	1,350	1,582
T	128	26,304	10,560	187	187
missing ratio	0%	0%	0%	55.3%	49.3%

7.3.2 Connections to Learning Gaussian Markov Random Fields

The Gaussian Markov Random Field (GMRF) is a general way to model multivariate data with dependencies. GMRF assumes that data are generated from a multivariate Gaussian distribution with a covariance matrix Σ which describes the dependencies among T dimensional variables i.e., $\bar{\boldsymbol{x}} \sim \mathcal{N}(0, \Sigma)$. If the unknown $\bar{\boldsymbol{x}}$ is assumed to be generated from this model, The negative log likelihood of the data can be written as $\bar{\boldsymbol{x}}^\top \Sigma^{-1} \bar{\boldsymbol{x}}$, ignoring the constants and where Σ^{-1} is the inverse covariance matrix of the Gaussian distribution. This prior can be incorporated into an empirical risk minimization framework as a regularizer. Furthermore, it is known that if $(\Sigma^{-1})_{st} = 0$, x_t and x_s are conditionally independent, given the other variables. In Theorem 7.1 we established connections to graph based regularizers, and that such methods can be seen as regularizing with the inverse covariance matrix for Gaussians [148]. We thus have the following result:

Corollary 7.1. *For any lag set $\mathcal{L}$, $\bar{\boldsymbol{w}}$, and $\eta > 0$, the inverse covariance matrix Σ_{AR}^{-1} of the GMRF model corresponding to the quadratic regularizer $\mathcal{R}_x(\bar{\boldsymbol{x}}) := \mathcal{T}_{AR}(\bar{\boldsymbol{x}} \,|\mathcal{L}, \bar{\boldsymbol{w}}, \eta)$ shares the same off-diagonal non-zero pattern as G^{AR} defined in Theorem 7.1. Moreover, we have $\mathcal{T}_{AR}(\bar{\boldsymbol{x}} \,|\mathcal{L}, \bar{\boldsymbol{w}}, \eta) = \bar{\boldsymbol{x}}^\top \Sigma_{AR}^{-1} \bar{\boldsymbol{x}}$.*

A detailed proof is in Section 7.5.2. As a result, our proposed AR-based regularizer is equivalent to imposing a Gaussian prior on $\bar{\boldsymbol{x}}$ with a structured inverse covariance described by the matrix G^{AR} defined in Theorem 7.1. Moreover, the step to learn $\mathcal{W}$ has a natural interpretation: the lag set $\mathcal{L}$ imposes the non-zero pattern of the graphical model on the data, and then we solve a simple least squares problem to learn the weights corresponding to the edges. As an application of Theorem 1 from [98] and Corollary 7.1, when $\mathcal{R}_f(F) = \|F\|_F^2$,we can relate $\mathcal{T}_{\mathrm{AR}}$ to a weighted nuclear norm:

$$\|ZB\|_* = \frac{1}{2} \inf_{F,X:Z=FX} \|F\|_F^2 + \sum_r \mathcal{T}_{\mathrm{AR}}(\bar{\boldsymbol{x}}_r \,|\mathcal{L}, \bar{\boldsymbol{w}}, \eta), \tag{7.16}$$

where $B = US^{1/2}$ and $\Sigma_{\mathrm{AR}}^{-1} = USU^\top$ is the eigen-decomposition of $\Sigma_{\mathrm{AR}}^{-1}$. (7.16) enables us to apply the results from [98] to obtain guarantees for the use of AR temporal regularizer when $\mathcal{W}$ is given. For simplicity, we assume $\bar{\boldsymbol{w}}_r = \bar{\boldsymbol{w}}$, $\forall r$ and consider a relaxed convex formulation for (7.15) as follows:

$$\hat{Z} = \arg\min_{Z \in \mathcal{C}} \frac{1}{N} \sum_{(i,j)\in\Omega} (Y_{ij} - Z_{ij})^2 + \lambda_z \|ZB\|_*, \tag{7.17}$$

where $N = |\Omega|$, and $\mathcal{C}$ is a set of matrices with low *spikiness*. Full details are provided in Section 7.5.3. As an application of Theorem 2 from [98], we have the following corollary.

Corollary 7.2. *Let $Z^\star = FX$ be the ground truth $n \times T$ time series matrix of rank k. Let Y be the matrix with $N = |\Omega|$ randomly observed entries corrupted with additive Gaussian noise with variance σ^2. Then if $\lambda_z \geq C_1 \sqrt{\frac{(n+T)\log(n+T)}{N}}$, with high probability for the $\hat{Z}$ obtained by (7.17),*

$$\left\| Z^\star - \hat{Z} \right\|_F \leq C_2 \alpha^2 \max(1, \sigma^2) \frac{k(n+T)\log(n+T)}{N} + O(\alpha^2/N),$$

Table 7.2: Forecasting results: ND/ NRMSE for each approach. Lower values are better. "-" indicates an unavailability due to scalability or an inability to handle missing values. TRMF-AR, SVD-AR(1), and TCT are matrix factorization based approaches, while AR(1), DLM, and R-DLM are classic time series approaches.

	Time Series Forecasting with Full Observation			Time Series Forecasting with Missing Values	
	synthetic	electricity	traffic	walmart-1	walmart-2
TRMF-AR	**0.373**/**0.487**	0.255/ **1.397**	**0.185**/ **0.421**	**0.533**/ **1.958**	**0.432**/ **1.065**
SVD-AR(1)	0.444/ 0.872	0.257/ 1.865	0.555/ 1.194	-/ -	-/ -
TCF	1.000/ 1.424	0.349/ 1.838	0.624/ 0.931	0.540/2.231	0.446/1.124
AR(1)	0.928/ 1.401	**0.219**/ 1.439	0.275/ 0.536	-/ -	-/ -
DLM	0.936/ 1.391	0.435/ 2.753	0.639/ 0.951	0.602/ 2.293	0.453/ 1.110
R-DLM	0.996/ 1.420	-/ -	-/ -	-/ -	-/ -
MEAN	1.000/ 1.424	1.410/ 4.528	0.560/ 0.826	1.239/3.103	1.097/2.088

where C_1,C_2 are positive constants, and α depends on the product $Z^\star B$.

See Section 7.5.3 for details. From the results in Table 7.3, we observe superior performance of TRMF-AR over standard MF, indicating that $\bar{\boldsymbol{w}}$ learnt from our data-driven approach (7.15) does aid in recovering the missing entries for time series. We would like to point out that establishing a theoretical guarantee for TRMF with $\mathcal{W}$ is unknown remains a challenging research direction.

7.4 Experimental Results

Datasets: We consider five datasets (See Table 7.1).

- synthetic: a small synthetic dataset with $n = 16, T = 128$. We generate $\{\boldsymbol{x}_t \in \mathbb{R}^4 : t = 1, \ldots, 128\}$ from the autoregressive process with a lag index set $\mathcal{L} = \{1, 8\}$, randomly generated $\{W^{(l)}\}$, and an additive

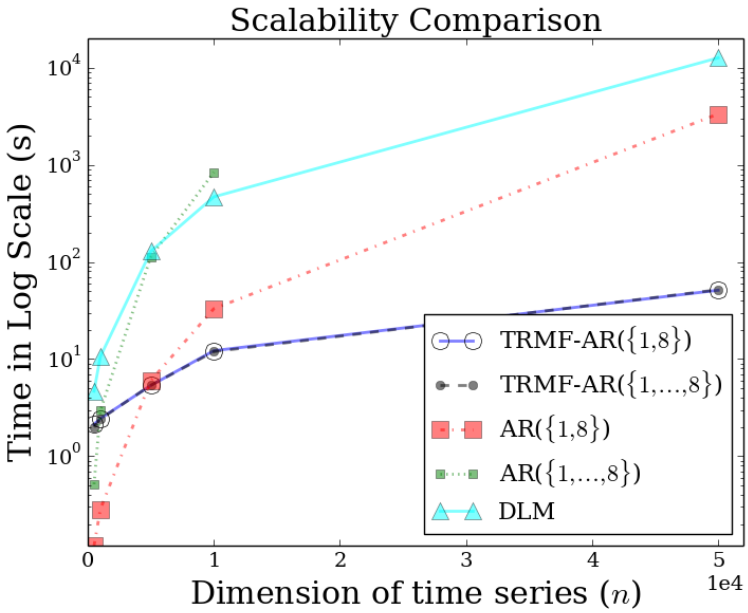


Figure 7.4: Scalability of TRMF. $T = 512$. $n \in \{500, \ldots, 50000\}$. AR($\{1, \ldots, 8\}$) cannot finish in 1 day.

white Gaussian noise of $\sigma = 0.1$. We then randomly generate a matrix $F \in \mathbb{R}^{16\times 4}$ and obtain $\boldsymbol{y}_t = F\boldsymbol{x}_t + \boldsymbol{\epsilon}_t$, where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, 0.1)$.

- electricity[2]: the electricity usage in kW recorded every 15 minutes, for $n = 370$ clients. We convert the data to reflect hourly consumption, by aggregating blocks of 4 columns, to obtain $T = 26,304$.
- traffic[3]: A collection of 15 months of daily data from the California Department of Transportation. The data describes the occupancy rate, between 0 and 1, of different car lanes of San Francisco bay area freeways. The data was sampled every 10 minutes, and we again aggregate the columns to obtain hourly traffic data to finally get $n = 963$, $T = 10,560$.
- walmart-1 & walmart-2: two propriety datasets from Walmart E-commerce contain weekly sale information of 1,350 and 1,582 items for 187 weeks, respectively. The time-series of sales for each item start and end at different time points; for modeling purposes we assume one start and end timestamp by padding each series with missing values. This along with some other missing values due to out-of-stock reasons lead to 55.3% and 49.3% of entries being missing.

Evaluation Criteria: We compute the normalized deviation (ND)

[2] https://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014.
[3] https://archive.ics.uci.edu/ml/datasets/PEMS-SF.

and normalized RMSE (NRMSE).

$$\text{ND:} \quad \left(\frac{1}{|\Omega_{test}|}\sum_{(i,t)\in\Omega_{test}}\left|\hat{Y}_{it}-Y_{it}\right|\right)\Bigg/\left(\frac{1}{|\Omega_{test}|}\sum_{(i,t)\in\Omega_{test}}|Y_{ij}|\right)$$

$$\text{NRMSE:} \quad \sqrt{\frac{1}{|\Omega_{test}|}\sum_{(i,t)\in\Omega_{test}}\left(\hat{Y}_{it}-Y_{it}\right)^2}\Bigg/\left(\frac{1}{|\Omega_{test}|}\sum_{(i,t)\in\Omega_{test}}|Y_{ij}|\right)$$

For each method and data set, we perform the grid search over various parameters (such as k, λ values) following a rolling validation approach described in [86]. We search $k \in \{2, 4, 8\}$ for synthetic and $\in \{20, 40\}$ for other datasets. For TRMF-AR, SVD-AR(1), TCF, and AR(1), we search $\lambda \in \{50, 5, 0.5, 0.05\}$

Methods/Implementations Compared:

- TRMF-AR: The proposed formulation (7.15) with $\mathcal{R}_w(\mathcal{W}) = \|\mathcal{W}\|_F^2$. For $\mathcal{L}$, we use $\{1, 2, \ldots, 8\}$ for synthetic, $\{1, \ldots, 24\} \cup \{7 \times 24, \ldots, 8 \times 24 - 1\}$ for electricity and traffic, and $\{1, \ldots, 10\} \cup \{50, \ldots, 56\}$ for walmart-1 and walmart-2 to capture seasonality.
- SVD-AR(1): The rank-k approximation of $Y = USV^\top$ is first obtained by SVD. After setting $F = US$ and $X = V^\top$, a k-dimensional AR(1) is learned on X for forecasting.
- TCF: Matrix factorization with the simple temporal regularizer proposed in [130].
- AR(1): n-dimensional AR(1) model.
- DLM: two implementations: the widely used R-DLM package [91] and the code provided in [69].

- Mean: The baseline, which predicts everything to be the mean of the observed portion of Y.

For each method and data set, we perform a grid search over various parameters (such as k, λ values) following a rolling validation approach described in [86].

Scalability: Figure 7.4 shows that traditional time-series approaches such as AR or DLM suffer from the scalability issue for large n, while TRMF-AR scales much better with n. Specifically, for $n = 50,000$, TRMF is 2 orders of magnitude faster than competing AR/DLM methods. Note that the results for R-DLM are not available because the R package cannot scale beyond n in the tens (See Appendix 7.5.4 for the source code to demonstrate that R-DLM fails when $n = 32$). Furthermore, the `dlmMLE` routine in R-DLM uses a general optimization solver, which is orders of magnitude slower than the implementation provided in [69].

7.4.1 Forecasting

We compare the forecasting performance of various approaches. Results are shown in Table 7.2.

Forecasting with Full Observations. We first compare various methods on the task of forecasting values in the test set, given fully observed training data. For synthetic, we consider one-point ahead forecasting task and use the last ten time points as the test periods. For electricity and traffic, we consider the 24-hour ahead forecasting task and use last seven days as the test

Table 7.3: Missing value imputation results: ND/ NRMSE for each approach. Note that TRMF outperforms all competing methods in almost all cases.

	$\frac{\lvert\Omega\rvert}{n\times T}$	Matrix Factorization Models			Time Series Models	
		TRMF-AR	TCF	MF	DLM	Mean
synthetic	20%	**0.467**/ **0.661**	0.713/ 1.030	0.688/ 1.064	0.933/ 1.382	1.002/ 1.474
	30%	**0.336**/ **0.455**	0.629/ 0.961	0.595/ 0.926	0.913/ 1.324	1.004/ 1.445
	40%	**0.231**/ **0.306**	0.495/ 0.771	0.374/ 0.548	0.834/ 1.259	1.002/ 1.479
	50%	**0.201**/ **0.270**	0.289/ 0.464	0.317/ 0.477	0.772/ 1.186	1.001/ 1.498
electricity	20%	**0.245**/ **2.395**	0.255/ 2.427	0.362/ 2.903	0.462/ 4.777	1.333/ 6.031
	30%	**0.235**/ **2.415**	0.245/ 2.436	0.355/ 2.766	0.410/ 6.605	1.320/ 6.050
	40%	0.231/ 2.429	0.242/ 2.457	0.348/ 2.697	**0.196**/ **2.151**	1.322/ 6.030
	50%	0.223/ 2.434	0.233/ 2.459	0.319/ 2.623	**0.158**/ **1.590**	1.320/ 6.109
traffic	20%	**0.190**/ **0.427**	0.208/ 0.448	0.310/ 0.604	0.353/ 0.603	0.578/ 0.857
	30%	**0.186**/ **0.419**	0.199/ 0.432	0.299/ 0.581	0.286/ 0.518	0.578/ 0.856
	40%	**0.185**/ **0.416**	0.198/ 0.428	0.292/ 0.568	0.251/ 0.476	0.578/ 0.857
	50%	**0.184**/ **0.415**	0.193/ 0.422	0.251/ 0.510	0.224/ 0.447	0.578/ 0.857

periods. From Table 7.2, we can see that TRMF-AR outperforms all the other methods on both metrics considered.

Forecasting with Missing Values. We next compare the methods on the task of forecasting in the presence of missing values in the data. We use the Walmart datasets here, and consider 6-week ahead forecasting and use last 54 weeks as the test periods. Note that SVD-AR(1) and AR(1) cannot handle missing values. The second part of Table 7.2 shows that we again outperform other methods.

7.4.2 Missing Value Imputation

We next consider the case of imputing missing values in the data. As in [70], we assume that blocks of data are missing, corresponding to sensor malfunctions for example, over a length of time.

To create data with missing entries, we first fixed the percentage of data that we were interested in observing, and then uniformly at random occluded blocks of a predetermined length (2 for synthetic data and 5 for the real datasets). The goal was to predict the occluded values. Table 7.3 shows that TRMF outperforms the methods we compared to on almost all cases.

7.5 Detailed Proofs and the Scalability Issue of R-DLM package

7.5.1 Proof of Theorem 7.1

Proof. In this proof, we use the notations and summation manipulation techniques introduced by Knuth [41]. To prove (7.14), it suffices to prove that

$$\sum_{m\leq t\leq T}\left(\sum_{l\in\bar{\mathcal{L}}} w_l x_{t-l}\right)^2 = \sum_{1\leq t\leq T}\sum_{1\leq d\leq L} G^{\mathrm{AR}}_{t,t+d}(x_t - x_{t+d})^2 + \bar{\boldsymbol{x}}^\top D\bar{\boldsymbol{x}}. \tag{7.18}$$

The LHS of the (7.18) can be expanded and regrouped as follows.

$$
\begin{aligned}
&\sum_{m\le t\le T}\left(\sum_{l\in\bar{\mathcal{L}}} w_l x_{t-l}\right)^2 \\
&= \sum_{m\le t\le T}\left(\sum_{l\in\bar{\mathcal{L}}} w_l^2 x_{t-l}^2 + \sum_{1\le d\le L}\sum_{l\in\delta(d)} 2w_l w_{l-d} x_{t-l} x_{t-l+d}\right) \\
&= \sum_{m\le t\le T}\left(\sum_{l\in\bar{\mathcal{L}}} w_l^2 x_{t-l}^2 + \sum_{1\le d\le L}\sum_{l\in\delta(d)} \left(-w_l w_{l-d}(x_{t-l}-x_{t-l+d})^2 + w_l w_{l-d}\big(x_{t-l}^2 + x_{t-l+d}^2\big)\right)\right) \\
&= \underbrace{\sum_{m\le t\le T}\sum_{1\le d\le L}\sum_{l\in\delta(d)} -w_l w_{l-d}(x_{t-l}-x_{t-l+d})^2}_{\mathcal{G}(\bar{\boldsymbol{x}})} + \\
&\qquad \underbrace{\sum_{m\le t\le T}\left(\sum_{l\in\bar{\mathcal{L}}} w_l^2 x_{t-l}^2 + \sum_{1\le d\le L}\sum_{l\in\delta(d)} w_l w_{l-d}\big(x_{t-l}^2 + x_{t-l+d}^2\big)\right)}_{\mathcal{D}(\bar{\boldsymbol{x}})}
\end{aligned}
$$

Let's look at the first term $\mathcal{G}(\bar{\boldsymbol{x}})$:

$$
\begin{aligned}
\mathcal{G}(\bar{\boldsymbol{x}}) &= \sum_{1\le d\le L}\sum_{l\in\delta(d)}\sum_{m\le t\le T} -w_l w_{l-d}(x_{t-l}-x_{t-l+d})^2 \\
&= \sum_{1\le d\le L}\sum_{l\in\delta(d)}\sum_{m-l\le t\le T-l} -w_l w_{l-d}(x_t - x_{t+d})^2 \\
&= \sum_{1\le d\le L}\sum_{l\in\delta(d)}\sum_{1\le t\le T} -w_l w_{l-d}(x_t - x_{t+d})^2 [m-l\le t\le T-l] \\
&= \sum_{1\le t\le T}\sum_{1\le d\le L}\left(\sum_{l\in\delta(d)} -w_l w_{l-d}[m-l\le t\le T-l]\right)(x_t - x_{t+d})^2 \\
&= \sum_{1\le t\le T}\sum_{1\le d\le L}\underbrace{\left(\sum_{\substack{l\in\delta(d)\\ m\le t+l\le T}} -w_l w_{l-d}\right)}_{G_{t,t+d}}(x_t - x_{t+d})^2,
\end{aligned}
$$

where we can see that $\mathcal{G}(\bar{\boldsymbol{x}})$ is equivalent to the first term of RHS of (7.18).

Now, we consider the second term $\mathcal{D}(\bar{\boldsymbol{x}})$:

$$\begin{aligned}\mathcal{D}(\bar{\boldsymbol{x}}) &= \sum_{m\le t\le T}\left(\sum_{l\in\bar{\mathcal{L}}} w_l^2 x_{t-l}^2 + \sum_{1\le d\le L}\sum_{l\in\delta(d)} w_l w_{l-d}\big(x_{t-l}^2 + x_{t-l+d}^2\big)\right)\\ &= \underbrace{\sum_{m\le t\le T}\sum_{l\in\bar{\mathcal{L}}} w_l^2 x_{t-l}^2}_{\mathcal{D}_1(\bar{\boldsymbol{x}})} + \underbrace{\sum_{m\le t\le T}\sum_{1\le d\le L}\sum_{l\in\delta(d)} w_l w_{l-d} x_{t-l}^2}_{\mathcal{D}_2(\bar{\boldsymbol{x}})} + \\ &\quad \underbrace{\sum_{m\le t\le T}\sum_{1\le d\le L}\sum_{l\in\delta(d)} w_l w_{l-d} x_{t-l+d}^2}_{\mathcal{D}_3(\bar{\boldsymbol{x}})}\end{aligned}$$

$$
\begin{aligned}
\mathcal{D}_1(\bar{\boldsymbol{x}}) &= \sum_{l\in\bar{\mathcal{L}}}\sum_{m\le t\le T} w_l^2 x_{t-l}^2 = \sum_{l\in\bar{\mathcal{L}}}\sum_{m-l\le t\le T-l} w_l^2 x_t^2 \\
&= \sum_{1\le t\le T}\left(\sum_{l\in\bar{\mathcal{L}}} w_l^2[m\le t+l\le T]\right)x_t^2 \\
&= \sum_{1\le t\le T}\left(\sum_{l,l'\in\bar{\mathcal{L}}} w_l w_{l'}[m\le t+l\le T][l'=l]\right)x_t^2 \\
\mathcal{D}_2(\bar{\boldsymbol{x}}) &= \sum_{m\le t\le T}\sum_{1\le d\le L}\sum_{l\in\delta(d)} w_l w_{l-d} x_{t-l}^2 \\
&= \sum_{1\le t\le T}\left(\sum_{1\le d\le L}\sum_{l\in\delta(d)} w_l w_{l-d}[m\le t+l\le T]\right)x_t^2 \\
&= \sum_{1\le t\le T}\left(\sum_{l,l'\in\bar{\mathcal{L}}} w_l w_{l'}[m\le t+l\le T][l'<l]\right)x_t^2 \\
\mathcal{D}_3(\bar{\boldsymbol{x}}) &= \sum_{m\le t\le T}\sum_{1\le d\le L}\sum_{l\in\delta(d)} w_l w_{l-d} x_{t-l+d}^2 \\
&= \sum_{1\le t\le T}\left(\sum_{1\le d\le L}\sum_{l\in\delta(d)} w_l w_{l-d}[m\le t+l-d\le T]\right)x_t^2 \\
&= \sum_{1\le t\le T}\left(\sum_{l',l\in\bar{\mathcal{L}}} w_l w_{l'}[m\le t+l\le T][l'>l]\right)x_t^2
\end{aligned}
$$

Let $D \in R^{T\times T}$ be a diagonal matrix with D_{tt} be the coefficient associated with x_t^2 in $\mathcal{D}(\bar{\boldsymbol{x}})$. Combining the results of $\mathcal{D}_1(\bar{\boldsymbol{x}}), \mathcal{D}_2(\bar{\boldsymbol{x}})$, and $\mathcal{D}_3(\bar{\boldsymbol{x}})$, D_t can be written as follows.

$$
D_{tt} = \left(\sum_{l\in\bar{\mathcal{L}}} w_l\right)\left(\sum_{l\in\bar{\mathcal{L}}} w_l[m\le t+l\le T]\right) \quad \forall t.
$$

It is clear that $\mathcal{D}(\bar{\boldsymbol{x}}) = \bar{\boldsymbol{x}}^\top D \bar{\boldsymbol{x}}$. Note that $\forall t = m, \ldots, T-L$, $D_{tt} = \left(\sum_{l \in \bar{\mathcal{L}}} w_l\right)^2$. □

7.5.2 Proof of Corollary 7.1

Proof. It is well known that graph regularization can be written in the quadratic form [109] as follows.

$$\frac{1}{2} \sum_{t \sim s} G_{ts} (x_t - x_s)^2 = \bar{\boldsymbol{x}}^\top \mathbf{Lap}(G) \bar{\boldsymbol{x}},$$

where $\mathbf{Lap}(G)$ is the $T \times T$ graph Laplacian for G defined as:

$$\mathbf{Lap}(G)_{ts} = \begin{cases} \sum_j G_{tj}, & t = s \\ -G_{ts}, & t \neq s \text{ and there is an edge } t \sim s \\ 0, & \text{otherwise.} \end{cases}$$

Based on the above fact and the results from Theorem 7.1, we obtain the quadratic form for $\mathcal{T}_{\text{AR}}(\bar{\boldsymbol{x}} \,|\mathcal{L}, \bar{\boldsymbol{w}}, \eta)$ as follows.

$$\mathcal{T}_{\text{AR}}(\bar{\boldsymbol{x}} \,|\mathcal{L}, \bar{\boldsymbol{w}}, \eta) = \frac{1}{2} \bar{\boldsymbol{x}}^\top \left(\mathbf{Lap}\big(G^{\text{AR}}\big) + \underbrace{D + \eta I}_{\text{diagonal}} \right) \bar{\boldsymbol{x}}.$$

Because $D + \eta I$ is diagonal, the non-zero pattern of the off-diagonal entries of the inverse covariance Σ_{AR}^{-1} for $\mathcal{T}_{\text{AR}}(\bar{\boldsymbol{x}} \,|\mathcal{L}, \bar{\boldsymbol{w}}, \eta)$ is determined by $\mathbf{Lap}\big(G^{\text{AR}}\big)$ which shares the same non-zero pattern as G^{AR}. □

7.5.3 Details for Corollary 7.2

We use results developed in [98] to arrive at our result. Assume we are given the matrix B. We first define the following quantities:

$$\alpha := \sqrt{nT} \frac{\|Z^\star B\|_\infty}{\|Z^\star B\|_F} \qquad \beta := \frac{\|Z^\star B\|_*}{\|Z^\star B\|_F} \tag{7.19}$$

where the $\|\cdot\|_\infty$ norm is taken element-wise. Note that, the above quantities capture the "simplicity" of the matrix Z. For example, a small value of α implies that the matrix ZB is not overly spiky, meaning that the entries of the matrix are well spread out in magnitude. Next, define the set

$$\mathcal{C} := \left\{ Z : \alpha\beta \leq C\sqrt{\frac{N}{\log(n+T)}} \right\}, \tag{7.20}$$

with C being a constant that depends on α.

Finally, we assume the following observation model: for each $i, j \in \Omega$, suppose we observe

$$Y_{ij} = Z^\star_{ij} + \frac{\sigma}{\sqrt{nT}}\eta_{ij} \quad \eta_{ij} \sim \mathcal{N}(0,1)$$

Then, we can see that the setup is identical to that considered in [98] with the difference being that there is no graph present that relates the rows of $Z^\star$. Hence, setting $A = I$ in Theorem 1 in the aforementioned paper yields our result.

7.5.4 Details: Scalability Issue of R-DLM package

In this section, we show the source code demonstrating that R-DLM fails to handle high-dimensional time series even with $n = 32$. Interested readers can run the following R code to see that the `dlmMLE()` function in R-DLM is able to run on a 16-dimensional time series. However, when we increase the dimension to 32, `dlmMLE()` crashes the entire R program.

```
library(dlm)
```

```
builderFactory <- function(n,k) {
    n = n;
    k = k;
    init = c(rep(0,k), rep(0.1,3),0.1*rnorm(n*k), 0.1*rnorm(k*k))
    build = function(x) {
        m0 = x[1:k]
        C0 = (abs(x[k+1]))*diag(k)
        V  = (abs(x[k+2]))*diag(n)
        W  = (abs(x[k+3]))*diag(k)
        FF = matrix(nrow=n,ncol=k, data=x[(k+3+1):(k+3+n*k)])
        GG = matrix(nrow=k,ncol=k, data=x[(k+3+n*k+1):(k+3+n*k+k*k)])
        return (dlm( m0=m0, C0=C0, FF=FF, GG=GG, V=V, W=W))
    }
    return (list(n=n,k=k,init=init,build=build))
}

Rdlm_train <- function(Y, k, maxit) {
    if(missing(maxit)) { maxit=10 }

    if(ncol(Y)==3) {
        Ymat = matrix(nrow=max(Y(,1)),ncol=max(Y(,2)))
        Ymat[cbind(Y(,1),Y(,2))] = Y(,3)
    } else {
```

```
        Ymat = Y;
    }
    n = nrow(Ymat)
    TT = ncol(Ymat)
    dlm_builder = builderFactory(n, k)
    mle = dlmMLE(Ymat,dlm_builder$init,build=dlm_builder$build,
                 control=list(maxit=10))
    dlm = dlm_builder$build(mle$par)
    dlm_filt = dlmFilter(Ymat,dlm)
    return (dlm_filt)
}

tmp = t(as.matrix(Nile));
tmp=rbind(tmp,tmp); tmp=rbind(tmp,tmp);
tmp=rbind(tmp,tmp); tmp=rbind(tmp,tmp);

print(nrow(tmp))
Rdlm_train(tmp,4);
print('works')

tmp=rbind(tmp,tmp);
print(nrow(tmp))
Rdlm_train(tmp,4);
```

7.6 Summary of the Contributions

In this chapter, we have proposed a novel temporal regularized matrix factorization framework (TRMF) for large-scale multiple time series problems with missing values. TRMF not only models temporal dependency among the data points, but also supports data-driven dependency learning. Our method generalizes several well known methods, and also yields superior performance when compared to other state-of-the-art methods on real-world datasets. A preliminary work of this chapter is presented in the time series workshop at NIPS 2015 [141].

Chapter 8

Greedy-MIPS: A Greedy Approach for Budgeted Maximum Inner Product Search

In this chapter, we study the computational issue in the prediction phase for many MF-based latent embedding models in recommender systems. Because of the large number of items, the prediction phase for an embedding-based model usually becomes a problem of maximum inner product search (MIPS) with a very large number of candidate embeddings. Specifically, given a large collection of n candidate vectors

$$\mathcal{H} = \left\{ \boldsymbol{h}_j \in \mathbb{R}^k : 1, \dots, n \right\}$$

and a query vector $\boldsymbol{w} \in \mathbb{R}^k$, MIPS aims to identify a subset of candidates that have top largest inner product values with $\boldsymbol{w}$. We also denote by $H = [\boldsymbol{h}_1, \dots, \boldsymbol{h}_j, \dots, \boldsymbol{h}_n]^\top$ as the candidate matrix. A naive linear search procedure to solve MIPS for a given query $\boldsymbol{w}$ requires $O(nk)$ operations to compute n inner products and $O(n \log n)$ operations to obtain the sorted ordering of the n candidates.[1]

[1] When only the largest B elements are required, the sorting procedure can be reduced to $O(n + B \log B)$ on average using a selection algorithm [33].

Recently, MIPS has drawn a lot of attention in the machine learning community. Matrix factorization (MF) based recommender system [34, 65] is one of the most important applications. In a MF based recommender system, each user i is associated with a vector $\boldsymbol{w}_i$ of dimension k, while each item j is associated with a vector $\boldsymbol{h}_j$ of dimension k. The interaction (such as preference) between a user and an item is modeled by the value of the inner product between $\boldsymbol{w}_i$ and $\boldsymbol{h}_j$. It is clear that identifying top-ranked items in such a system for a user is exactly a MIPS problem. Because both the number of users (the number of queries) and the number of items (size of vector pool in MIPS) can easily grow to millions, a naive linear search is extremely expensive; for example, to compute the preference for all m users over n items with latent embeddings of dimension k in a recommender system requires at least $O(mnk)$ operations. When both m and n are large, the prediction procedure is extremely time consuming; it is even *slower* than the training procedure used to obtain the $m+n$ embeddings, which costs only $O(|\Omega|k)$ operations per iteration. Taking the yahoo-music dataset as an example, we have $m = 1M$, $n = 0.6M$, $|\Omega| = 250M$, and

$$mn = 600B \gg 250M = |\Omega|.$$

As a result, the development of efficient algorithms for MIPS is needed in large-scale recommender systems. In addition, MIPS can be found in many other machine learning applications, such as the prediction for a multi-class or multi-label classifier [127, 139], an object detector, a structure SVM predicator, and many others.

There have been some works on how to accelerate MIPS for large n recently such as [9, 10, 64, 85, 96, 104]. However, most of them do not have the flexibility to control the trade-off between search efficiency and search quality in the prediction phase. In this chapter, we consider the budgeted MIPS problem, which is a generalized version of the standard MIPS with a computation budget: *how to generate a top-ranked candidates under a given* **budget** *on the number of inner products one can perform.* By carefully studying the problem structure of MIPS, we develop a novel Greedy-MIPS algorithm, which handles budgeted MIPS by design. While simple and intuitive, Greedy-MIPS yields surprisingly superior performance compared to existing state-of-the-art approaches.

Contributions. Our contributions can be summarized as follows:

- We carefully study the MIPS problem and develop Greedy-MIPS, which is a novel algorithm without any nearest neighbor search reduction that is essential in many state-of-the-art approaches [9, 85, 104].
- Greedy-MIPS is orders of magnitudes faster than many state-of-the-art MIPS approaches in order to obtain a desired search performance. As a specific example, on the yahoo-music data sets with $n = 624,961$ and $k = 200$, Greedy-MIPS runs 200x faster than the naive approach and yields search results with the top-5 precision more than 75%, while the search performance of other state-of-the-art approaches under the similar speedup drops to less than 3% precision.
- Greedy-MIPS supports MIPS with a budget, which brings the ability to

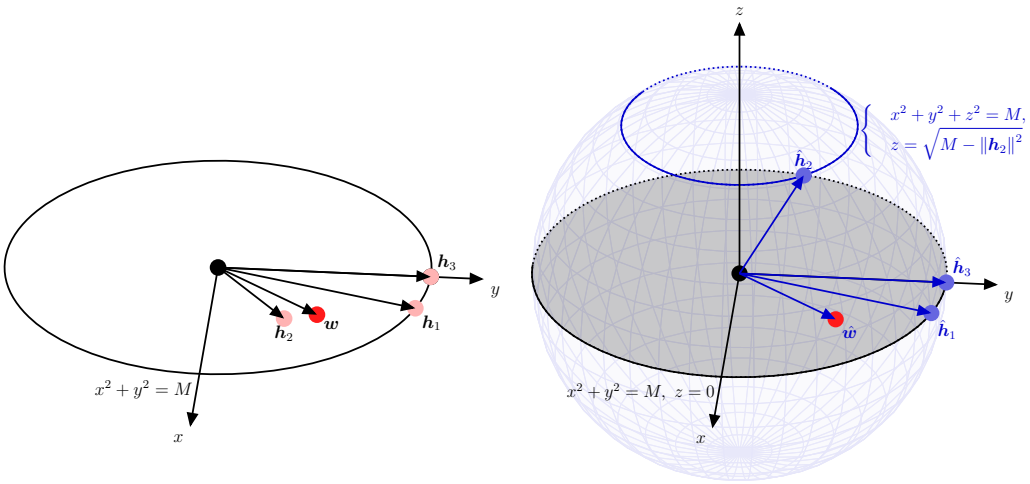
control of the trade-off between the computation efficiency and the search quality in the prediction phase. To the best of our knowledge, among existing MIPS approaches, only the sampling approaches proposed in [10, 32] support the similar flexibility under a limited situation where all the candidates and query vectors are non-negative.

Organization. We first review existing fast MIPS approaches in Section 8.1 and introduce the budgeted MIPS problem in Section 8.2. In Section 8.3, we propose a novel greedy budgeted MIPS approach called Greedy-MIPS. We then show the empirical comparison in Section 8.4 and conclude this chapter in Section 8.5.

8.1 Existing Approaches for Fast MIPS

Because of its wide applicability, there have been some attempts to design efficient algorithms for MIPS. Most of existing approaches consider to reduce the MIPS problem to the nearest neighbor search problem (NNS), where the goal is to identify the nearest candidates of the given query, and apply an existing efficient NNS algorithm to solve the reduced problem [7, 9, 85, 104, 105]. [9] is the first MIPS work which adopts such a MIPS-to-NNS reduction. Variants MIPS-to-NNS reduction reduction are also proposed in [104, 105]. Experimental results in [9] show the superiority of the NNS reduction over the traditional branch-and-bound search approaches for MIPS [64, 96].

Fast MIPS approaches with sampling schemes have become popular recently [10, 32]. Various sampling schemes have been proposed to handle



(a) Original MIPS in $\mathbb{R}^2$. (b) Reduced NNS in $\mathbb{R}^3$.

Figure 8.1: MIPS-to-NN reduction. In 8.1(a), all the candidate vectors $\{\boldsymbol{h}_j\}$ and the query vector $\boldsymbol{w}$ are in $\mathbb{R}^2$. $\boldsymbol{h}_2$ is the nearest neighbor of $\boldsymbol{w}$, while $\boldsymbol{h}_1$ is the vector yielding the maximum value of the inner product with $\boldsymbol{w}$. In 8.1(b), the reduction proposed in [9] is applied to $\boldsymbol{w}$ and $\{\boldsymbol{h}_j\}$: $\hat{\boldsymbol{w}} = [\boldsymbol{w}; 0]^\top$ and $\hat{\boldsymbol{h}}_j = [\boldsymbol{h}_j; \sqrt{M - \|\boldsymbol{h}_j\|^2}]^\top$, $\forall j$, where $M = \max_j \|\boldsymbol{h}_j\|^2$. All the transformed vectors are in $\mathbb{R}^3$. In particular, all the transformed candidate vectors are in the sphere with radius $\sqrt{M}$. As a result, the nearest neighbor of $\hat{\boldsymbol{w}}$ in this transformed 3-dimensional NNS problem, $\hat{\boldsymbol{h}}_1$, corresponds to the vector $\boldsymbol{h}_1$ which yields the maximum inner product value with $\boldsymbol{w}$ in the original 2-dimensional MIPS problem.

MIPS problem with different *constraints*. We will briefly review two popular sampling schemes in Section 8.1.2.

8.1.1 Approaches with Nearest Neighbor Search Reduction

We briefly introduce the concept of the reduction proposed in [9]. First, we consider the relationship between the Euclidean distance and the inner

product:

$$\|\boldsymbol{w} - \boldsymbol{h}_{j_1}\|^2 = \|\boldsymbol{w}\|^2 + \|\boldsymbol{h}_{j_1}\|^2 - 2\boldsymbol{w}^\top \boldsymbol{h}_{j_1}$$
$$\|\boldsymbol{w} - \boldsymbol{h}_{j_2}\|^2 = \|\boldsymbol{w}\|^2 + \|\boldsymbol{h}_{j_2}\|^2 - 2\boldsymbol{w}^\top \boldsymbol{h}_{j_2}.$$

It is clear to see that that when all the candidate vectors $\boldsymbol{h}_j$ share the same length; that is,

$$\|\boldsymbol{h}_1\| = \|\boldsymbol{h}_2\| = \cdots = \|\boldsymbol{h}_n\|,$$

the MIPS problem is exactly the same as the NNS problem: because $\|\boldsymbol{h}_{j_1}\| = \|\boldsymbol{h}_{j_2}\|$ we have

$$\|\boldsymbol{w} - \boldsymbol{h}_{j_1}\| > \|\boldsymbol{w} - \boldsymbol{h}_{j_2}\| \iff \boldsymbol{w}^\top \boldsymbol{h}_{j_1} < \boldsymbol{w}^\top \boldsymbol{h}_{j_2}. \tag{8.1}$$

Note that when $\|\boldsymbol{h}_{j_1}\| \neq \|\boldsymbol{h}_{j_2}\|$, (8.1) no longer holds. See Figure 8.1(a) for an example where not all the candidate vectors have the same length. We can see that $\boldsymbol{h}_1$, the candidate vector yielding the maximum inner product value with $\boldsymbol{w}$ is not the nearest neighbor candidate, i.e., $\boldsymbol{h}_2$ in this example.

To handle the situation where candidates have multiple lengths, [9] proposes the following transform to reduce the original MIPS problem with $\mathcal{H}$ and $\boldsymbol{w}$ in a k dimensional space to a new NNS problem with $\hat{\mathcal{H}}$ and $\hat{\boldsymbol{w}}$ in a $k+1$ dimensional space:

$$\hat{\boldsymbol{w}} = [\boldsymbol{w}; 0]^\top,$$
$$\hat{\boldsymbol{h}}_j = \left[\boldsymbol{h}_j; \sqrt{M - \|\boldsymbol{h}_j\|^2}\right]^\top, \ \forall j = 1, \ldots, n, \tag{8.2}$$

where M is the maximum squared length over the entire candidate set $\mathcal{H}$:

$$M = \max_{j=1,\ldots,n} \|\boldsymbol{h}_j\|^2.$$

First, we can see that with the above transform, $\left\|\hat{\boldsymbol{h}}_j\right\|^2 = M$ for all j:

$$\left\|\hat{\boldsymbol{h}}_j\right\|^2 = \|\boldsymbol{h}_j\|^2 + M - \|\boldsymbol{h}_j\|^2 = M, \ \forall j.$$

Then, for any $j_1 \neq j_2$, we have

$$\begin{aligned}
&\left\|\hat{\boldsymbol{w}} - \hat{\boldsymbol{h}}_{j_1}\right\| < \left\|\hat{\boldsymbol{w}} - \hat{\boldsymbol{h}}_{j_2}\right\| \\
\iff & M + \|\boldsymbol{w}\|^2 - 2\boldsymbol{w}^\top \boldsymbol{h}_{j_1} < M + \|\boldsymbol{w}\|^2 - 2\boldsymbol{w}^\top \boldsymbol{h}_{j_2} \\
\iff & \boldsymbol{w}^\top \boldsymbol{h}_{j_1} > \boldsymbol{w}^\top \boldsymbol{h}_{j_2}.
\end{aligned}$$

With the above relationship, the original k dimensional MIPS problem is equivalent to the transformed $k+1$ dimensional NNS problem. In Figure 8.1(b), we show the transformed NNS problem for the original MIPS problem presented in Figure 8.1(a).

In [105], another MIPS-to-NNS reduction has been proposed. The high level idea is to apply a transformation to $\mathcal{H}$ such that all the candidate vectors *roughly* have the same length by appending additional $\bar{k}$ dimensions. In the procedure by [105], all the $\boldsymbol{h}_j$ vectors are assumed (or scaled) to have $\|\boldsymbol{h}_j\| \leq U, \ \forall j$, where $U < 1$ is a positive constant. Then the following transform is applied to reduce the original MIPS problem of k dimension to a new NNS problem with $\hat{\mathcal{H}}$ and $\hat{\boldsymbol{w}}$ of $k + \bar{k}$ dimension:

$$\begin{aligned}
\hat{\boldsymbol{w}} &= [\boldsymbol{w}; \boldsymbol{0}_{\bar{k}}]^\top \\
\hat{\boldsymbol{h}}_j &= \left[\boldsymbol{h}_j; 1/2 - \|\boldsymbol{h}_j\|^{2^1}; 1/2 - \|\boldsymbol{h}_j\|^{2^2}; \ldots; 1/2 - \|\boldsymbol{h}_j\|^{2^{\bar{k}}}\right]^\top,
\end{aligned} \tag{8.3}$$

where $\mathbf{0}_{\bar{k}}$ is a zero vector of dimension $\bar{k}$. Because $U < 1$, [105] shows that with the transform (8.3), we have $\left\|\hat{\boldsymbol{h}}_j\right\|^2 = \bar{k}/4 + \|\boldsymbol{h}_j\|^{2^{\bar{k}+1}}$, with the second term vanishing as $\bar{k} \to \infty$. Thus, all the candidates $\hat{\boldsymbol{h}}_j$ approximately have the same length. We can see the idea behind (8.3) is similar to (8.2): transforming $\mathcal{H}$ to $\hat{\mathcal{H}}$ such that all the candidates have the same length. Note that (8.2) achieves this goal exactly while (8.3) achieves this goal approximately. Both transforms show a similar empirical performance in [85].

There are many choices to solve the transformed NNS problem after the MIPS-to-NN reduction has been applied. In [85, 104, 105], various locality sensitive hashing schemes have been considered. In [9], a PCA-tree based approach is proposed, and shows better performance than LSH-based approaches, which is consistent to the empirical observations in [7] and our experimental results shown in Section 8.4. In [7], a simple K-means clustering algorithm is proposed to handled the transformed NNS problem.

8.1.2 Sampling-based Approaches

The idea of the sampling-based MIPS approach is first proposed in [32] as an approach to perform approximate matrix-matrix multiplications. Its applicability on MIPS problems is studied very recently [10]. The idea behind a sampling-based approach called Sample-MSIPS, is about designing an efficient sampling procedure such that the j-th candidate is selected with the probability $p(j)$:

$$p(j) \sim \boldsymbol{h}_j^\top \boldsymbol{w}.$$

In particular, Sample-MSIPS is an efficient scheme to sample $(j, t) \in [n] \times [k]$ with the probability $p(j, t)$:

$$p(j, t) \sim h_{jt} w_t.$$

Each time a pair (j, t) is sampled, we increase the count for the j-th item by one. By the end of the sampling process, the spectrum of the counts forms an estimation of n inner product values. Due to the nature of the sampling approach, it can only handle the situation where all the candidate vectors and query vectors are *nonnegative*.

Diamond-MSIPS, a diamond sampling scheme proposed in [10], is an extension of Sample-MSIPS to handle the maximum **squared** inner product search problem (MSIPS) where the goal is to identify candidate vectors with largest values of $\left(\boldsymbol{h}_j^\top \boldsymbol{w}\right)^2$. If both $\boldsymbol{w}$ and $\mathcal{H}$ are nonnegative or $\boldsymbol{h}_j^\top \boldsymbol{w} \geq 0,\ \forall j$, MSIPS can be used to generate the solutions for MIPS. However, the solutions to MSIPS can be very different from the solutions to MIPS in general. For example, if all the inner product values are negative, the ordering for MSIPS is the exactly reverse ordering induced by MIPS. Here we can see that the applicability of both Sample-MSIPS and Diamond-MSIPS to MIPS is very limited.

8.2 Budgeted Maximum Inner Product Search

The core idea behind the fast approximate MIPS approaches is to trade the search quality for the shorter query latency: the shorter the search latency,

the lower the search quality. In most existing fast MIPS approaches, the trade-off depends on the approach-specific parameters such as the depth of the PCA tree in [9] or the number of hash functions in [85, 104, 105]. Such approach-specific parameters are usually required to construct approach-specific data structures before any query is given, which means that the trade-off is somewhat *fixed* for all the queries. Particularly, the computation cost for all the query requests is fixed. However, in many real-world scenarios, each query request might have a different computational budget, which raises the question: *Can we design a fast MIPS approach which supports the dynamic adjustment of the trade-off in the query phase?*

In this section, we formally define the budgeted MIPS problem which is an extension of the standard MIPS problem with a computational budget as a parameter given in the query phase. We first summarize the essential components for fast MIPS approaches in Section 8.2.1 and give the problem definition of budgeted MIPS in Section 8.2.2.

8.2.1 Essential Components for Fast MIPS Approaches

Before diving into the details of budgeted MIPS, we first review the essential components in an algorithm for fast MIPS:

- Before any query request:
  - *Query-Independent Data Structure Construction:* A pre-processing procedure is performed on the entire candidate sets to construct an approach-specific data structure $\mathcal{D}$ to store information about $\mathcal{H}$,

such as the LSH hash tables [85, 104, 105], space partition trees (e.g., KD-tree or PCA-tree [9]), or cluster centroids [7].

- For each query request:
  - *Query-dependent* **P***re-processing:* In some approaches, a query dependent pre-processing is needed. For example, a vector augmentation is required in all approaches with the MIPS-to-NNS reduction [7, 9, 85, 104]. In addition, [9] also requires another normalization. T_P is used to denote the time complexity of this stage.
  - *Candidate* **S***creening:* In this stage, based on the pre-constructed data structure $\mathcal{D}$, an efficient procedure is performed to filter candidates such that only a subset of candidates $\mathcal{C}(\boldsymbol{w}) \subset \mathcal{H}$ is selected. In a naive linear approach, no screening procedure is performed, so $\mathcal{C}(\boldsymbol{w})$ simply contains all the n candidates. For a tree-based structure, $\mathcal{C}(\boldsymbol{w})$ contains all the candidates stored in the leaf node of the query vector. In a sampling-based MIPS approach, an efficient sampling scheme is designed to generate highly possible candidates to form $\mathcal{C}(\boldsymbol{w})$. T_S denotes the computational cost of the screening stage.
  - *Candidate* **R***anking:* An exact ranking is performed on the selected candidates in $\mathcal{C}(\boldsymbol{w})$ obtained from the screening stage. This involves the computation of $|\mathcal{C}(\boldsymbol{w})|$ inner products and the sorting procedure among these $|\mathcal{C}(\boldsymbol{w})|$ values. The overall time complexity T_R is

$$T_R = O(|\mathcal{C}(\boldsymbol{w})|k + |\mathcal{C}(\boldsymbol{w})| \log|\mathcal{C}(\boldsymbol{w})|).$$

The per-query computational cost T_Q is

$$T_Q = T_P + T_S + T_R.$$

It is clear that the candidate screening stage is the *key* component for a fast MIPS approach. In terms of the search quality, the performance highly depends on whether the screening procedure can identify highly possible candidates. In terms of the query latency, the efficiency highly depends on the size of $\mathcal{C}(\boldsymbol{w})$ and how fast to generate $\mathcal{C}(\boldsymbol{w})$. The major difference between various fast MIPS approaches is the choice of the data structure $\mathcal{D}$ and the corresponding screening procedure.

8.2.2 Budgeted MIPS: Problem Definition

Budgeted maximum inner product search is an extension of the standard approximate MIPS problem with a computation budget: how to generate top-ranked candidates under a given **budget** on the number of inner product operations one can perform. Budgeted MIPS has a wide applicability. For example, a real-time recommender system must provide a list of recommended items for its users in a very short response time.

Note that the cost for the candidate ranking (T_R) is inevitable in the per-query cost: $T_Q = T_P + T_S + T_R$. A viable approach to support budgeted MIPS must include a screening procedure which satisfies the following requirements:

- the flexibility to control the size of $\mathcal{C}(\boldsymbol{w})$ in the candidate screening stage

such that $|\mathcal{C}(\boldsymbol{w})| \leq B$, where B is a given budget, and

- an efficient screening procedure to obtain $\mathcal{C}(\boldsymbol{w})$ in $O(Bk)$ time such that the overall per-query cost is

$$T_Q = O(Bk + B \log B).$$

As mentioned earlier, most recently proposed efficient algorithms such as PCA-MIPS [9] and LSH-MIPS [85, 104, 105] adopt the approach to reduce the MIPS problem to an instance of NNS problem, and apply various search space partition data structures or techniques (e.g., LSH, KD-tree, or PCA-tree) designed for NNS to index the candidates $\mathcal{H}$ in the *query-independent pre-processing* stage. As the construction of $\mathcal{D}$ is query independent, both the **search performance** and the **computation cost** are *fixed* when the construction is done. For example, the performance of a PCA-MIPS depends on the depth of the PCA-tree. Given a query vector $\boldsymbol{w}$, there is no control to the size of $\mathcal{C}(\boldsymbol{w})$ in the candidate generating phase. LSH-based approaches also have the similar issue. As a result, it is not clear how to generalize PCA-MIPS and LSH-MIPS in a principled way to handle the situation with a computational budget: how to reduce the size of $\mathcal{C}(\boldsymbol{w})$ under a limited budget and how to improve the performance when a larger budget is given.

Unlike other NNS-based algorithms, the design of Sample-MSIPS naturally enables it to support budgeted MIPS for a nonnegative candidate matrix H and a nonnegative query $\boldsymbol{w}$. Recall that the core idea behind Sample-MSIPS

is to draw a sample candidate j among n candidates such that

$$p(j) \propto \boldsymbol{h}_j^\top \boldsymbol{w}.$$

The more the number of samples, the lower the variance of the estimated frequency spectrum. Clearly, Sample-MSIPS has the flexibility to control the size of $\mathcal{C}(\boldsymbol{w})$. As a result, Sample-MSIPS can be a viable approach for the budgeted MIPS problem. However, Sample-MSIPS works only on the situation where the entire $\mathcal{H}$ and $\boldsymbol{w}$ are non-negative. Diamond-MSIPS has the similar issue.

8.3 Greedy-MIPS: A Novel Approach for Budgeted MIPS

In this section, we carefully study the problem structure of MIPS and develop a simple but novel algorithm called Greedy-MIPS, which handles budgeted MIPS by design. Unlike the most recent approaches [7, 9, 85, 104, 105], Greedy-MIPS is an approach without any reduction to a NNS problem. Moreover, Greedy-MIPS is a viable approach for the budgeted MIPS problem without the non-negativity limitation inherited in the sampling approaches.

As mentioned earlier that the key component for a fast MIPS approach is the algorithm used in the candidate screening phase. In budgeted MIPS, for any given budget B and query $\boldsymbol{w}$, an *ideal procedure* for the candidate screening phase costs $O(Bk)$ time to generate $\mathcal{C}(\boldsymbol{w})$ which contains the B items with the largest B inner product values over the n candidates in $\mathcal{H}$. The requirement on the time complexity $O(Bk)$ implies that the procedure is independent from

$n = |\mathcal{H}|$, the number of candidates in $\mathcal{H}$. One might wonder whether such an ideal procedure exists or not. In fact, designing such an ideal procedure under such a requirement on the time complexity is even more challenging than the original budgeted MIPS problem.

8.3.1 A Motivating Example for Greedy-MIPS

Although the existence of an *ideal procedure* for a general budgeted MIPS problem seems to be impossible, we demonstrate that an *ideal approach* exists for budgeted MIPS when $k = 1$. It is not hard to observe that Property 8.1 holds for any given $\mathcal{H} = \{h_1, \ldots, h_n \mid h_j \in \mathbb{R}\}$:

Property 8.1. *For any nonzero query $w \in \mathbb{R}$ and any budget $B > 0$, there are only two possible results for that top B inner products between w and $\mathcal{H}$:*

$$w > 0 \Rightarrow \text{ Largest } B \text{ elements in } \mathcal{H},$$
$$w < 0 \Rightarrow \text{ Smallest } B \text{ elements in } \mathcal{H}.$$

This property leads to the following simple approach, which is an ideal procedure for the budgeted MIPS problem when $k = 1$:

- *Query-independent data structure:* a **sorted** list of indices of $\mathcal{H}$: $\mathtt{s}[\mathtt{r}]$, $\mathtt{r} = 1, \ldots, n$ such that $\mathtt{s}[\mathtt{r}]$ stores the index to the r-th largest candidate. That is

$$h_{\mathtt{s}[1]} \geq h_{\mathtt{s}[2]} \geq \cdots \geq h_{\mathtt{s}[\mathtt{n}]},$$

- *Candidate screening phase:* for any given $w \neq 0$ and $B > 0$,

$$\textbf{return} \begin{cases} \text{first } B \text{ elements: } \{\texttt{s}[1], \ldots, \texttt{s}[\texttt{B}]\} & \text{if } w > 0, \\ \text{last } B \text{ elements: } \{\texttt{s}[\texttt{n}], \ldots, \texttt{s}[\texttt{n} - \texttt{B} + 1]\} & \text{if } w < 0 \end{cases}$$

  as the indices of the exact largest-B candidates.

Note that for this simple scenario ($k = 1$), neither the *query dependent preprocessing* nor the *candidate ranking* is needed. Thus, the overall time complexity per query is $T_Q = O(B)$. We can see that Property 8.1 is the key to the correctness of the above procedure. Nevertheless, it is not clear how to generalize Property 8.1 for MIPS problems with $k \geq 2$. Fortunately, we can utilize the fact that Property 8.1 holds for $k = 1$ to design an efficient greedy procedure for the candidate screening when $k \geq 2$.

8.3.2 A Greedy Procedure to Candidate Screening

To better describe the idea of the proposed algorithm Greedy-MIPS, we consider the following definition (8.4):

Definition 8.1. *The* **rank** *of an item x among a set of items $\mathcal{X} = \{x_1, \ldots, x_{|\mathcal{X}|}\}$ is defined as*

$$\textbf{rank}(x \mid \mathcal{X}) := \sum_{j=1}^{|\mathcal{X}|} \mathbb{I}[x_j \geq x], \tag{8.4}$$

where $\mathbb{I}[\cdot]$ is the indicator function. A ranking induced by $\mathcal{X}$ is a function $\pi(\cdot) : \mathcal{X} \to \{1, \ldots, |\mathcal{X}|\}$ such that $\pi(x_j) = \textbf{rank}(x_j \mid \mathcal{X}) \quad \forall x_j \in \mathcal{X}$.

One way to store a ranking $\pi(\cdot)$ induced by $\mathcal{X}$ is by a sorted index array

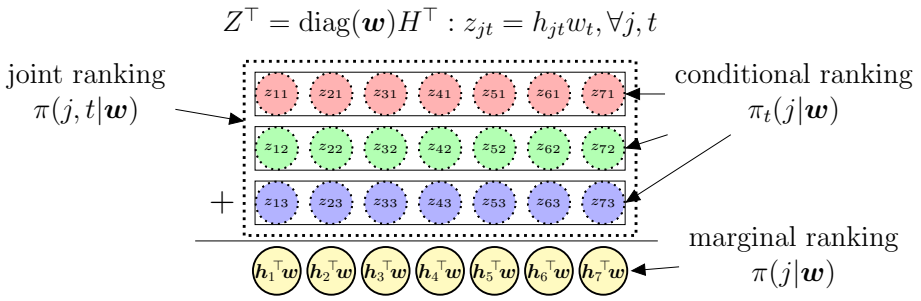


Figure 8.2: nk multiplications in a naive linear MIPS approach.

$\mathtt{s}[\mathtt{r}]$ of size $|\mathcal{X}|$ such that

$$\pi(x_{\mathtt{s}[1]}) \leq \pi(x_{\mathtt{s}[2]}) \leq \cdots \leq \pi(x_{\mathtt{s}[|\mathcal{X}|]}).$$

We can see that $\mathtt{s}[\mathtt{r}]$ stores the index to the item x with $\pi(x) = r$.

In order to design an efficient candidate screening procedure, we carefully study the operations required for MIPS. In the naive linear MIPS approach, nk multiplication operations are required to obtain n inner product values $\left\{\boldsymbol{h}_1^\top \boldsymbol{w}, \ldots, \boldsymbol{h}_n^\top \boldsymbol{w}\right\}$. nk operations form an *implicit matrix* $Z \in \mathbb{R}^{n \times k}$:

$$Z = H\,\mathrm{diag}(\boldsymbol{w}),$$

where $\mathrm{diag}(\boldsymbol{w}) \in \mathbb{R}^{k \times k}$ is a matrix with $\boldsymbol{w}$ as it diagonal. The (j, t) entry of Z denotes the multiplication operation $z_{jt} = h_{jt}w_t$ and $\boldsymbol{z}_j = \mathrm{diag}(\boldsymbol{w})\boldsymbol{h}_j$ denotes the j-th row of Z. In Figure 8.2, we use $Z^\top$ to demonstrate the implicit matrix. The implicit matrix Z is query dependant, that is, Z is different for a different $\boldsymbol{w}$. Note that n inner product values can be obtained by taking the column-wise summation of $Z^\top$. In particular, we have

$$\boldsymbol{h}_j^\top \boldsymbol{w} = \sum_{t=1}^{k} z_{jt}, \;\; j = 1, \ldots, n.$$

Thus, the ranking induced by the n inner product values can be characterized by the *marginal ranking* $\pi(j|\boldsymbol{w})$ defined on the implicit matrix Z as follows:

$$\begin{aligned}\pi(j|\boldsymbol{w}) &:= \mathbf{rank}\left(\sum_{t=1}^{k} z_{jt} \;\middle|\; \left\{\sum_{t=1}^{k} z_{1t}, \cdots, \sum_{t=1}^{k} z_{nt}\right\}\right) \\ &= \mathbf{rank}\big(\boldsymbol{h}_j^\top \boldsymbol{w} \mid \{\boldsymbol{h}_1^\top \boldsymbol{w}, \ldots, \boldsymbol{h}_n^\top \boldsymbol{w}\}\big).\end{aligned} \tag{8.5}$$

As mentioned earlier, it is hard to design an ideal candidate screening procedure which generates $\mathcal{C}(\boldsymbol{w})$ based on the marginal ranking. Because the main goal for the candidate screening phase is to quickly identify candidates which are highly possible to be top-ranked items, it suffices to have an efficient procedure generating $\mathcal{C}(\boldsymbol{w})$ by an approximation ranking. Here we propose a greedy heuristic ranking:

$$\bar{\pi}(j|\boldsymbol{w}) := \mathbf{rank}\Big(\max_{t=1}^{k} z_{jt} \;\Big|\; \Big\{\max_{t=1}^{k} z_{1t}, \cdots, \max_{t=1}^{k} z_{nt}\Big\}\Big), \tag{8.6}$$

which is obtained by replacing the summation terms in (8.5) by max operators. The intuition behind this heuristic is that the largest element of $\boldsymbol{z}_j$ multiplied by k is an upper bound of $\boldsymbol{h}_j^\top \boldsymbol{w}$:

$$\boldsymbol{h}_j^\top \boldsymbol{w} = \sum_{t=1}^{k} z_{jt} \leq k\Big(\max_{t=1}^{k} z_{jt}\Big).$$

Thus, $\bar{\pi}(j|\boldsymbol{w})$, which is induced by such an upper bound of $\boldsymbol{h}_j^\top \boldsymbol{w}$, could be a reasonable approximation ranking for the marginal ranking $\pi(j|\boldsymbol{w})$.

Next we design an efficient procedure which generates $\mathcal{C}(\boldsymbol{w})$ according to the ranking $\bar{\pi}(j|\boldsymbol{w})$ defined in (8.6). First, based on the relative orderings of $\{z_{jt}\}$, we consider the *joint ranking* and the *conditional ranking* defined as follows:

- Joint ranking: $\pi(j,t|\boldsymbol{w})$ is the exact ranking over the nk entries of Z.

$$\pi(j,t|\boldsymbol{w}) := \mathbf{rank}(z_{jt} \mid \{z_{11},\ldots,z_{nk}\}).$$

- Conditional ranking: $\pi_t(j|\boldsymbol{w})$ is the exact ranking over the n entires of the t-th row of $Z^\top$.

$$\pi_t(j|\boldsymbol{w}) := \mathbf{rank}(z_{jt} \mid \{z_{1t},\ldots,z_{nt}\}).$$

See Figure 8.2 for an illustration for both rankings. Similar to the marginal ranking, both joint and conditional rankings are query dependent.

Observe that, in (8.6), for each j, only a single maximum entry of Z, $\max_{t=1}^{k} z_{jt}$, is considered to obtain the ranking $\bar{\pi}(j|\boldsymbol{w})$. To generate $\mathcal{C}(\boldsymbol{w})$ based on $\bar{\pi}(j|\boldsymbol{w})$, we can iterate (j,t) entries of Z in a greedy sequence such that (j_1,t_1) is visited before (j_2,t_2) if $z_{j_1t_1} > z_{j_2t_2}$, which is exactly the sequence corresponding to the joint ranking $\pi(j,t|\boldsymbol{w})$. Each time an entry (j,t) is visited, we can include the index j into $\mathcal{C}(\boldsymbol{w})$ if $j \notin \mathcal{C}(\boldsymbol{w})$. In Theorem 8.1, we show that the sequence to include a newly observed j into $\mathcal{C}(\boldsymbol{w})$ is exactly the sequence induced by the ranking $\bar{\pi}(j|\boldsymbol{w})$ defined in (8.6).

Theorem 8.1. *For all j_1 and j_2 such that $\bar{\pi}(j_1|\boldsymbol{w}) < \bar{\pi}(j_2|\boldsymbol{w})$, j_1 will be included into $\mathcal{C}(\boldsymbol{w})$ before j_2 if we iterate (j,t) pairs following the sequence induced by the joint ranking $\pi(j,t|\boldsymbol{w})$.*

Proof. Let $t_1 = \arg\max_{t=1}^{k} z_{j_1t}$ and $t_2 = \arg\max_{t=1}^{k} z_{j_2t}$. By the definition of t_1, we have $\pi(j_1,t_1|\boldsymbol{w}) < \pi(j_1,t|\boldsymbol{w})$, $\forall t \neq t_1$. Thus, (j_1,t_1) will be first entry

among $\{(j_1, 1), \ldots, (j_1, k)\}$ to be visited in the sequence corresponding to the joint ranking $\pi(j, t|\boldsymbol{w})$. Similarly, (j_2, t_2) will be the first visited entry among $\{(j_2, 1,), \ldots, (j_2, k)\}$. We also have

$$\bar{\pi}(j_1|\boldsymbol{w}) < \bar{\pi}(j_2|\boldsymbol{w}) \Rightarrow z_{j_1 t_1} > z_{j_2 t_2} \Rightarrow \pi(j_1, t_1|\boldsymbol{w}) < \pi(j_2, t_2|\boldsymbol{w}).$$

Thus, j_1 will be included into $\mathcal{C}(\boldsymbol{w})$ before j_2. □

At first glance, generating (j, t) in the sequence according to the joint ranking $\pi(j, t|\boldsymbol{w})$ might require the access to all the nk entries of Z and cost $O(nk)$ time. In fact, based on Property 8.2 of conditional rankings, we can design an efficient variant of the k-way merge algorithm [62, Chapter 5.4.1] to generate (j, t) pairs in the desired sequence iteratively.

Property 8.2. *Given a fixed candidate matrix H, for any possible $\boldsymbol{w}$ with $w_t \neq 0$, the conditional ranking $\pi_t(j|\boldsymbol{w})$ is either $\pi_{t+}(j)$ or $\pi_{t-}(j)$:*

- $\pi_{t+}(j) = \mathbf{rank}(h_{jt} \mid \{h_{1t}, \ldots, h_{nt}\})$,
- $\pi_{t-}(j) = \mathbf{rank}(-h_{jt} \mid \{-h_{1t}, \ldots, -h_{nt}\})$.

In particular, we have

$$\pi_t(j|\boldsymbol{w}) = \begin{cases} \pi_{t+}(j) & \text{if } w_t > 0, \\ \pi_{t-}(j) & \text{if } w_t < 0. \end{cases}$$

Similar to Property 8.1, Property 8.2 enables us to characterize a *query dependent* conditional ranking $\pi_t(j|\boldsymbol{w})$ by two *query independent* rankings $\pi_{t+}(j)$ and $\pi_{t-}(j)$. As a result, similar to the motivating example in

Algorithm 8.1 `ConditionalIterator`: an iterator iterates $j \in \{1, \ldots, n\}$ based on the conditional ranking $\pi_t(j|\boldsymbol{w})$. This pseudo code assumes that the k sorted index arrays $\mathtt{s_t[r]},\ r = 1, \ldots, n,\ t = 1, \ldots, k$ are available.

class `ConditionalIterator`:

def `constructor(dim_idx, query_val)`:
  `t, w, ptr` $\leftarrow$ `dim_idx, query_val, 1`

def `current()`: **return** $\begin{cases} \mathtt{s_t[ptr]} & \text{if } \mathtt{w} > 0, \\ \mathtt{s_t[n - ptr + 1]} & \text{otherwise.} \end{cases}$

def `hasNext()`: **return** $(\mathtt{ptr} < n)$

def `getNext()`: `ptr` $\leftarrow$ `ptr` $+ 1$ and **return** `current()`

Section 8.3.1,for each t, we can construct and store a **sorted** index array $\mathtt{s_t[r]},\ r = 1, \ldots, n$ such that

$$\pi_{t+}(\mathtt{s_t[1]}) \leq \pi_{t+}(\mathtt{s_t[2]}) \leq \cdots \leq \pi_{t+}(\mathtt{s_t[n]}), \tag{8.7}$$

or

$$\pi_{t-}(\mathtt{s_t[1]}) \geq \pi_{t-}(\mathtt{s_t[2]}) \geq \cdots \geq \pi_{t-}(\mathtt{s_t[n]}), \tag{8.8}$$

equivalently. Thus, in the phase of *query-independent data structure construction* of Greedy-MIPS, we compute and store *query-independent* rankings $\pi_{t+}(\cdot),\ t = 1, \ldots, k$ by k sorted index arrays of length n: $\mathtt{s_t[r]},\ r = 1, \ldots, n,\ t = 1, \ldots, n$ such that (8.7) holds. The entire construction costs $O(kn \log n)$ time and $O(kn)$ space.

Next we describe the details of the proposed Greedy-MIPS algorithm when a query $\boldsymbol{w}$ and the budget B are given. As mentioned earlier, Greedy-MIPS utilizes the idea of the k-way merge algorithm to visit (j, t) entries of Z

according to the joint ranking $\pi(j, t|\boldsymbol{w})$. Designed to merge k **sorted** sublists into a single sorted list, the k-way merge algorithm uses 1) k pointers, one for each sorted sublist, and 2) a binary tree structure (either a heap or a selection tree) containing the elements pointed by these k pointers to obtain the next element to be appended into the sorted list [62, Chapter 5.4.1.].

8.3.2.1 Query-dependent Pre-processing

In Greedy-MIPS, we divide nk entries of (j, t) into k groups. The t-th group contains n entries: $\{(j, t) : j = 1, \ldots, n\}$. Here we need an *iterator* playing a similar role as the pointer which can iterate index $j \in \{1, \ldots, n\}$ in the *sorted* sequence induced by the conditional ranking $\pi_t(\cdot|\boldsymbol{w})$. Utilizing Property 8.2, the t-th pre-computed sorted arrays $\mathtt{s_t[r]}$, $r = 1, \ldots, n$ can be used to construct such an iterator, called `ConditionalIterator`, which iterates an index j one by one in the desired sorted sequence. `ConditionalIterator` needs to support `current()` to access the currently pointed index j and `getNext()` to advance the iterator. In Algorithm 8.1, we describe a pseudo code for `ConditionalIterator`, which utilizes the facts (8.7) and (8.8) such that both the construction and the index access cost $O(1)$ space and $O(1)$ time. For each t, we use $\mathtt{iters}[t]$ to denote the `ConditionalIterator` for the t-th conditional ranking $\pi_t(j|\boldsymbol{w})$.

Regarding the binary tree structure used in Greedy-MIPS, we consider a max-heap `Q` of (z, t) pairs. $z \in \mathbb{R}$ is the *compared key* used to maintain the heap property of `Q`, and $t \in \{1, \ldots, k\}$ is an integer to denote the index to a

Algorithm 8.2 Query-dependent pre-processing procedure in Greedy-MIPS.

- **Input:** query $\boldsymbol{w} \in \mathbb{R}^k$
- For $t = 1, \ldots, k$
  - $\texttt{iters}[t] \leftarrow \texttt{ConditionalIterator}(t,\ w_t)$
  - $j \leftarrow \texttt{iters}[t].\texttt{current}()$
  - $z \leftarrow h_{jt} w_t$
  - $\texttt{Q}.\texttt{push}((z, t))$
- **Output:**
  - $\texttt{iters}[\texttt{t}],\ t = 1, \ldots, k$: iterators for conditional ranking $\pi_t(\cdot|\boldsymbol{w})$.
  - $\texttt{Q}$: a max-heap containing $\left\{(z, t) \mid z = \max_{j=1}^{n} z_{jt},\ t = 1, \ldots, k\right\}$.

entry group. Each $(z, t) \in \texttt{Q}$ denotes the (j, t) entry of Z where

$$j = \texttt{iters}[t].\texttt{current}() \quad \text{and} \quad z = z_{jt} = h_{jt} w_t.$$

Note that there are most k elements in the max-heap at any time. Thus, we can implement $\texttt{Q}$ by a binary heap such that it supports

- $\texttt{Q}.\texttt{top}()$: returns the maximum pair (z, t) of $\texttt{Q}$ in $O(1)$ time,
- $\texttt{Q}.\texttt{pop}()$: deletes the maximum pair of $\texttt{Q}$ in $O(\log k)$ time, and
- $\texttt{Q}.\texttt{push}((z, t))$: inserts a new pair in $O(\log k)$ time.

Note that the entire Greedy-MIPS can also be implemented using a selection tree among the k entries pointed by the k iterators. For the simplicity of presentation, we use a max-heap to describe the idea of Greedy-MIPS first and describe the details of Greedy-MIPS with a selection tree in the end of Section 8.3.2.2.

In the *query-dependent pre-processing* phase of Greedy-MIPS, we need to construct $\texttt{iters}[t],\ t = 1, \ldots, k$, one for each conditional ranking $\pi_t(j|\boldsymbol{w})$, and

a max-heap `Q` which is initialized to contain $\{(z,t) \mid z = \max_{j=1}^{n} z_{jt},\ t = 1,\ldots,k\}$. A detailed procedure is described in Algorithm 8.2, which costs $O(k \log k)$ time and $O(k)$ space.

8.3.2.2 Candidate Screening

Recall the requirements for a viable candidate screening procedure to support budgeted MIPS: 1) the flexibility to control the size $|\mathcal{C}(\boldsymbol{w})| \leq B$; and 2) an efficient procedure runs in $O(Bk)$. The core idea of Greedy-MIPS is iteratively traversing (j,t) entries of Z in a *greedy* sequence and collect newly observed indices j into $\mathcal{C}(\boldsymbol{w})$ until $|\mathcal{C}(\boldsymbol{w})| = B$. In particular, if $r = \pi(j,t|\boldsymbol{w})$, then (j,t) entry is visited at the r-th iterate. Utilizing the max-heap `Q` and the k iterators: `iters`$[t]$, we can design an iterator, called `JointIterator`, which iterates (j,t) pairs one by one in the desired greedy sequence induced by joint ranking $\pi(j,t|\boldsymbol{w})$. Following the k-way merge algorithm, in Algorithm 8.3, we describe a detailed pseudo code for such an iterator. `JointIterator` costs $O(k \log k)$ time to run Algorithm 8.2 to construct and initialize `Q` and `iters`$[t]$, and costs $O(\log k)$ time to advance to the next entry. In Algorithm 8.4, we describe our first *candidate screening* procedure with a budget B for Greedy-MIPS, which is a simple **while**-loop to iterate (j,t) entries using the `JointIterator` with $\boldsymbol{w}$ until $|\mathcal{C}(\boldsymbol{w})| = B$.

To analyze the time complexity of Algorithm 8.4, we need to know the number of the iterations of the **while**-loop before the stop condition is satisfied. The following Theorem 8.2 gives an upper bound on this number of

Algorithm 8.3 `JointIterator`: an iterator generates (j, t) pairs one by one based on the joint ranking $\pi(j, t|\boldsymbol{w})$. The constructor costs $O(k \log k)$ time to build a max-heap `Q`. The time complexity to generate a pair is $O(\log k)$.

class `JointIterator`:

def `constructor`($\boldsymbol{w}$): $\cdots O(k \log k)$

Run Algorithm 8.2 with $\boldsymbol{w}$ to initialize `Q` and `iters`$[t]$, $t = 1, \ldots, k$

`ptr` $\leftarrow$ `1`.

def `current()`: $\cdots O(1)$

$(z, t) \leftarrow$ `Q.top()`

$j \leftarrow$ `iters`$[t]$`.current()`

return (j, t)

def `hasNext()`: **return** (`ptr` $< nk$) $\cdots O(1)$

def `getNext()`: $\cdots O(\log k)$

$(z, t) \leftarrow$ `Q.pop()` $\cdots O(\log k)$

if `iters`$[t]$`.hasNext()`:

$j \leftarrow$ `iters`$[t]$`.getNext()`

$z \leftarrow h_{jt} w_t$

`Q.push`$((z, t))$ $\cdots O(\log k)$

`ptr` $\leftarrow$ `ptr` $+ 1$

return `current()`

Algorithm 8.4 Candidate screening procedure in Greedy-MIPS.

- **Input:** $\boldsymbol{w}$ and an empty $\mathcal{C}(\boldsymbol{w})$
- `jointIter` $\leftarrow$ `JointIterator`$(\boldsymbol{w})$ $\cdots O(k \log k)$
- $(j, t) \leftarrow$ `jointIter.current()`
- **while** $|\mathcal{C}(\boldsymbol{w})| < B$:
  - **if** $j \notin \mathcal{C}(\boldsymbol{w})$: append j to $\mathcal{C}(\boldsymbol{w})$
  - $(j, t) \leftarrow$ `jointIter.getNext()` $\cdots O(\log k)$
- **Output:** $\mathcal{C}(\boldsymbol{w}) = \{j \mid \bar{\pi}(j|\boldsymbol{w}) \leq B\}$

iterations.

Theorem 8.2. *There are at least B distinct indices j in the first Bk entries (j,t) in terms of the joint ranking $\pi(j,t|\boldsymbol{w})$ for any $\boldsymbol{w}$; that is,*

$$|\{j \mid \forall(j,t) \text{ such that } \pi(j,t|\boldsymbol{w}) \leq Bk\}| \geq B. \tag{8.9}$$

Proof. By grouping these first Bk entries by the index t and applying the pigeonhole principle, we know that there exists a group G such that it contains at least B entries. Because each entry in the same group has a distinct j index, we know that the group G contains at least B distinct indices j. □

Theorem 8.1 guarantees the correctness of Algorithm 8.4 to generate $\mathcal{C}(\boldsymbol{w})$ based on $\bar{\pi}(j|\boldsymbol{w})$ defined in (8.6). By Theorem 8.2, the overall time complexity of Algorithm 8.4 is $O(Bk \log k)$ as each iteration of the **while**-loop costs $O(\log k)$ time.

The $O(Bk \log k)$ time complexity of Algorithm 8.4 does not satisfy the efficiency requirement of a viable budgeted MIPS approach. Here we propose an improved candidate screening procedure which reduces the overall time complexity to $O(Bk)$. Observe that the $\log k$ term comes from the $\texttt{Q.push}((\texttt{z}_{\texttt{jt}}, \texttt{t}))$ and $\texttt{Q.pop}()$ operations of the max-heap for each visited (j,t) entry. As the goal of the screening procedure is to identify j indices only, we can skip the $\texttt{Q.push}((\texttt{z}_{\texttt{jt}}, \texttt{t}))$ for an entry (j,t) with the j having been included in $\mathcal{C}(\boldsymbol{w})$. As a result, $\texttt{Q.pop}()$ is executed at most $B+k-1$ times when

Algorithm 8.5 An improved candidate screening procedure in Greedy-MIPS. The overall time complexity is $O(Bk)$.

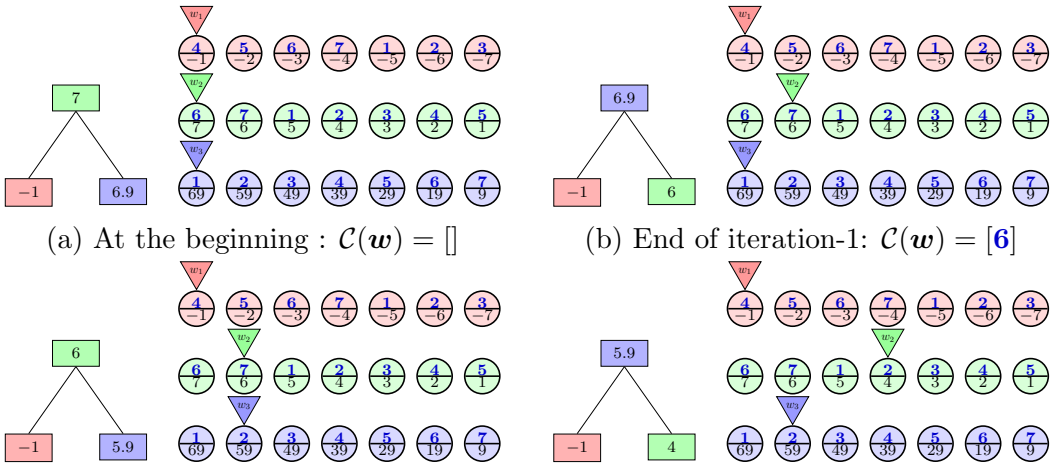
- **Input:**
  - $\mathcal{H}$, $\boldsymbol{w}$, and the computational budget B
  - `Q` and `iters`$[t]$: output of Algorithm 8.2
  - $\mathcal{C}(\boldsymbol{w})$: an empty list
  - `visited`$[j] = 0,\ j = 1, \ldots, n$: a zero-initialized array of length n
- **while** $|\mathcal{C}(\boldsymbol{w})| < B$:
  - $(z, t) \leftarrow$ `Q.pop()` $\cdots O(\log k)$
  - $j \leftarrow$ `iters`$[t]$`.current()`
  - **if** `visited`$[j] = 0$:
    - append j into $\mathcal{C}(\boldsymbol{w})$
    - `visited`$[j] \leftarrow 1$
  - **while** `iters`$[t]$`.hasNext()`:
    - $j \leftarrow$ `iters`$[t]$`.getNext()`
    - **if** `visited`$[j] = 0$:
      - $z \leftarrow h_{jt} w_t$
      - `Q.push`$((z, t))$ $\cdots O(\log k)$
      - **break**
- `visited`$[j] \leftarrow 0, \forall j \in \mathcal{C}(\boldsymbol{w})$ $\cdots O(B)$
- **Output:** $\mathcal{C}(\boldsymbol{w}) = \{j \mid \bar{\pi}(j|\boldsymbol{w}) \leq B\}$

$|\mathcal{C}(\boldsymbol{w})| = B$. The extra $k - 1$ times occurs in the situation that

$$\texttt{iters}[1]\texttt{.current}() = \cdots \texttt{iters}[1]\texttt{.current}() = \cdots = \texttt{iters}[k]\texttt{.current}()$$

at the beginning of the entire screening procedure.

In Algorithm 8.5, we give a detailed description for this improved candidate screening procedure for Greedy-MIPS. See Figure 8.3 for a detailed illustration of this algorithm on a toy example. Note that in Algorithm 8.5, we use an auxiliary zero-initialized array of length n: `visited`$[j],\ j = 1, \ldots, n$ to de-



(a) At the beginning : $\mathcal{C}(\boldsymbol{w}) = []$ (b) End of iteration-1: $\mathcal{C}(\boldsymbol{w}) = [\mathbf{6}]$

(c) End of iteration-2: $\mathcal{C}(\boldsymbol{w}) = [6, \mathbf{1}]$ (d) End of iteration-3: $\mathcal{C}(\boldsymbol{w}) = [6, 1, \mathbf{7}]$

Figure 8.3: Illustration of Algorithm 8.5 with $\boldsymbol{w} = [1, 1, 0.1]^\top$ and $B = 3$. The left plot for each sub-figure shows the heap structure in the max-heap `Q`: the value in each rectangle denotes z, and each index t is shown in a different color (red for 1, green for 2, and blue for 3). The sorted index arrays are shown in the upper part of circles on the right plot for each sub-figure; for example, $\mathtt{s}_1[4] = 7$, $\mathtt{s}_2[1] = 6$, and $\mathtt{s}_3[5] = 5$. The value in lower part of circles is the corresponding h_{jt}; for example, $h_{71} = -4$, $h_{62} = 7$, and $h_{53} = 29$. Three downward triangles denote the current position of $\mathtt{iters}[t]$, $t = 1, 2, 3$. Figure 8.3(a) shows the status for each data data structure at the beginning of Algorithm 8.5. Figures 8.3(b)-8.3(c) show the status in the end of the first and the second iterations of the outer **while**-loop in Algorithm 8.5. In Figure 8.3(c), we show that at the third iteration, after $(z, t) = (6, 2) \leftarrow \mathtt{Q.pop}()$ is executed and $7 = \mathtt{iters}[2].\mathtt{current}()$ is appended into $\mathcal{C}(\boldsymbol{w})$, we need to advance $\mathtt{iters}[2]$ twice because the index $j = 1$ has been included in $\mathcal{C}(\boldsymbol{w})$. Note that for this example $\boldsymbol{h}_1$ is the candidate with the largest inner product value with $\boldsymbol{w}$.

note whether an index j has been included in $\mathcal{C}(\boldsymbol{w})$ or not. As $\mathcal{C}(\boldsymbol{w})$ contains at most B indices, only B elements of this auxiliary array will be modified during the screening procedure. Furthermore, the auxiliary array can be reset to zero using $O(B)$ time in the end of Algorithm 8.5, so this auxiliary array

can be utilized again for a different query vector $\boldsymbol{w}$.

Notice that Algorithm 8.5 still iterates Bk entries of Z but at most $B + k - 1$ entries will be pushed into or pop from the max-heap. Thus, the overall time complexity of Algorithm 8.5 is $O(Bk + B \log k) = O(Bk)$, which satisfies the efficiency requirement for a viable approach for budgeted MIPS.

Greedy-MIPS with a Selection Tree. As there are at most k pairs in the max-heap `Q`, one from each `iters`$[t]$, the max-heap can be replaced by a selection tree to achieve a slightly faster implementation as suggested in [62, Chapter 5.4.1]. In Algorithm 8.6, we give a pseudo code for the selection tree with a $O(k)$ time constructor, a $O(1)$ time maximum element look-up, and a $O(\log k)$ time updater. To apply the section tree for our Greedy-MIPS, we only need to the following modifications:

- In Algorithm 8.2, remove `Q.push`$((z, t))$ from the **for**-loop and construct `Q` by `Q` $\leftarrow$ `SelectionTree`$(\boldsymbol{w}, k,$ `iters`$)$.
- In Algorithm 8.3 and Algorithm 8.5, replace `Q.pop`$()$ by `Q.top`$()$ and replace `Q.push`$((z, t))$ by `Q.updateValue`(t, z).

8.3.2.3 Connection to Sampling-based MIPS Approaches

Sample-MSIPS, as mentioned earlier, is essentially a sampling algorithm with replacement scheme to draw entries of Z such that (j, t) is sampled with the probability proportional to z_{jt}. Thus, Sample-MSIPS can be thought as a traversal of (j, t) entries using in a stratified random sequence determined by a distribution of the *values* of $\{z_{jt}\}$, while the core idea of Greedy-MIPS is to

Algorithm 8.6 A pseudo code of a selection tree used for Greedy-MIPS.

```
class SelectionTree:
    def constructor(w, k, iters) :                                  ··· O(k)
        K̄ ← min{2^i | 2^i ≥ k}
        for i = 1, ..., 2K̄:
            buf[i] ← (−∞, 0)
        for t = 1, ..., k:
            j ← iters[t].current()
            buf[K̄ + t] ← (h_jt w_t, t)
        for i = K̄, ..., 1:
            if buf[2i].first > buf[2i + 1].first:
                buf[i] ← buf[2i]
            else:
                buf[i] ← buf[2i + 1]
    def top(): return buf[1]                                        ··· O(1)
    def updateValue(t, z):                                          ··· O(log k)
        i ← K̄ + t
        buf[i] ← (z, t)
        while i > 1:
            i ← ⌊i/2⌋
            if buf[2i].first > buf[2i + 1].first:
                buf[i] ← buf[2i]
            else:
                buf[i] ← buf[2i + 1]
```

iterate (j, t) entries of Z in a greedy sequence induced by the *ordering* of $\{z_{jt}\}$. Next, we discuss the differences between Greedy-MIPS and Sample-MSIPS in a few perspectives:

Applicability: Sample-MSIPS can be applied to the situation where both $\mathcal{H}$ and $\boldsymbol{w}$ are nonnegative because of the nature of sampling scheme. In contrast,

Greedy-MIPS can work on any MIPS problems as only the ordering of $\{z_{jt}\}$ matters in Greedy-MIPS. Instead of $\boldsymbol{h}_j^\top \boldsymbol{w}$, Diamond-MSIPS is designed for the MSIPS problem which is to identify candidates with largest $\left(\boldsymbol{h}_j^\top \boldsymbol{w}\right)^2$ or $\left|\boldsymbol{h}_j^\top \boldsymbol{w}\right|$ values [10]. In fact, for nonnegative MIPS problems, the diamond sampling is equivalent to Sample-MSIPS. Moreover, for MSIPS problems with negative entries, when the number of samples is set to be the budget B,[2] the Diamond-MSIPS is equivalent to apply Sample-MSIPS to sample (j, t) entries with the probability $p(j, t) \propto |z_{jt}|$. Thus, the applicability of the existing sampling-based approaches is still very limited for general MIPS problems.

Flexibility to Control $|\mathcal{C}(\boldsymbol{w})|$: By Theorem 8.2, we know that Greedy-MIPS can guarantee both the time complexity of the candidate screening procedure and the size of output $|\mathcal{C}(\boldsymbol{w})|$ for any $\mathcal{H}$, $\boldsymbol{w}$, and B. For a sampling-based approach, one can easily control either the time complexity of the sampling procedure or the size of $\mathcal{C}(\boldsymbol{w})$, but not both. Because all the existing sampling-based approaches are a sampling scheme with replacement, the same entry (j, t) could be sampled repeatedly. Thus, the time complexity to guarantee that $\mathcal{C}(\boldsymbol{w}) = B$ depends on the distribution of values of $\boldsymbol{w}$ and $\mathcal{H}$. Hence, Greedy-MIPS is more flexible than sampling-based approaches in terms of the controllability of $\mathcal{C}(\boldsymbol{w})$.

[2] This setting is used in the experiments in [10]

8.4 Experimental Results

In this section, we perform extensive empirical comparisons to compare Greedy-MIPS with other state-of-the-art fast MIPS approaches on both real-world and synthetic datasets.

- We use netflix and yahoo-music as our real-world recommender system datasets. There are 17,770 and 624,961 items in netflix and yahoo-music, respectively. In particular, we obtain the low rank model (W, H) by the standard low-rank matrix factorization:

$$\min_{W,H} \sum_{(i,j)\in\Omega} \left(A_{ij} - \boldsymbol{w}_i^\top \boldsymbol{h}_j\right)^2 + \lambda\left(\sum_{i=1}^{m} \frac{|\Omega_i|}{n}\|\boldsymbol{w}_i\|^2 + \sum_{j=1}^{n} \frac{|\bar{\Omega}_j|}{m}\|\boldsymbol{h}_j\|^2\right),$$

  where A_{ij} is the rating of the j-th item given by the i-th user, Ω is the set of observed ratings, $\Omega_i = \{j \mid (i,j) \in \Omega\}$, and $\bar{\Omega}_j = \{i \mid (i,j) \in \Omega\}$, and λ is a regularization parameter. We use the CCD++ [138] algorithm implemented in LIBPMF[3] to solve the above optimization problem and obtain the user embeddings $\{\boldsymbol{w}_i\}$ and item embeddings $\{\boldsymbol{h}_j\}$. We use the same λ used in [28]. We also obtain (W, H) with a different k: 50, 100, and $k = 200$.

- We also generate synthetic datasets with various $n = 2^{\{17,18,19,20\}}$ and $k = 2^{\{2,5,7,10\}}$. For each synthetic dataset, both candidate vector $\boldsymbol{h}_j$ and query $\boldsymbol{w}$ vector are drawn from the normal distribution.

[3] http://www.cs.utexas.edu/~rofuyu/libpmf

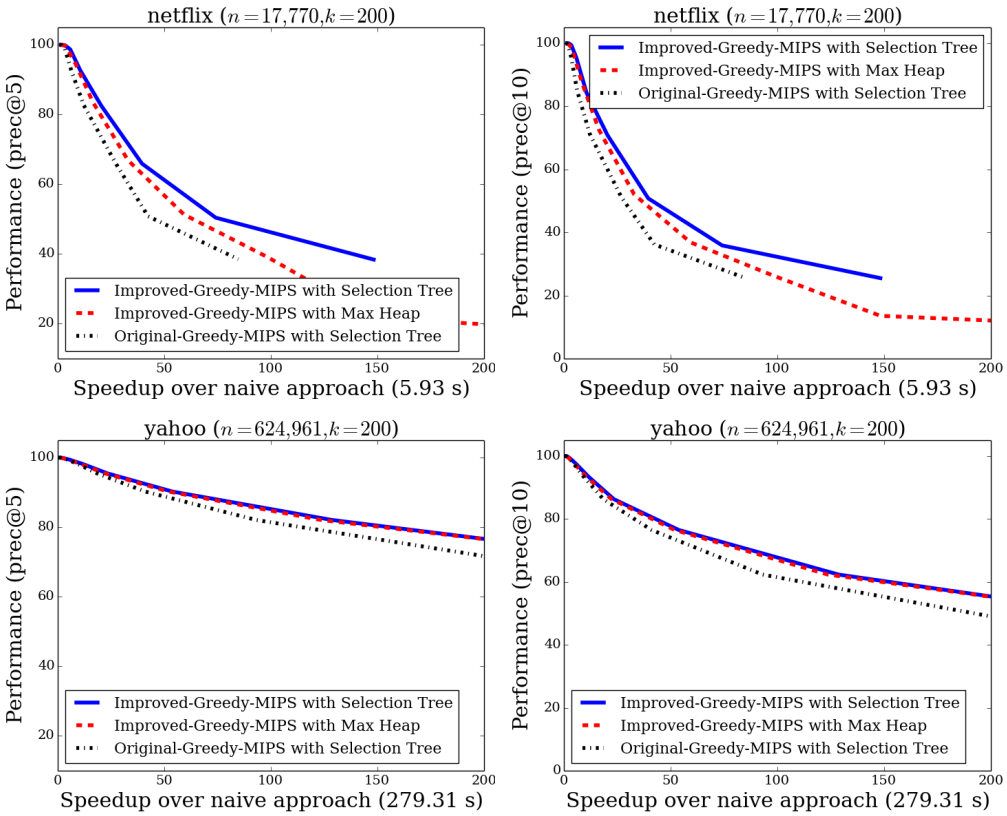


Figure 8.4: Comparison of variants of Greedy-MIPS.

8.4.1 Experimental Settings and Evaluation Criteria

All the experiments are performed on a Linux machine with 20 cores and 256 GB memory. We ensure that only single core/thread is used for our experiments. To have a fair comparison, all the compared approaches are implemented in C++:

- Greedy-MIPS: our proposed approach in Section 8.3. We compare the following variants in Section 8.4.2:

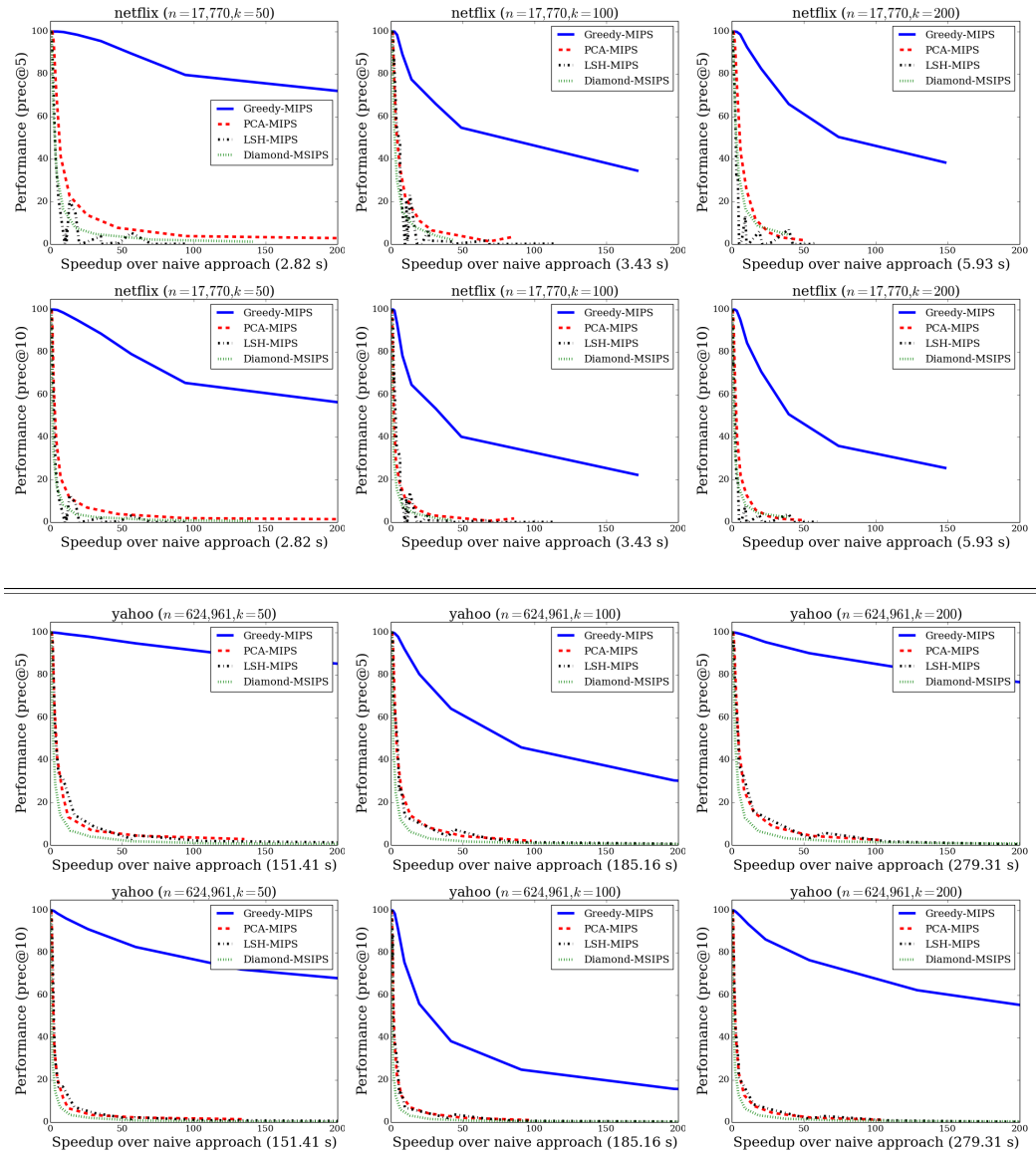


Figure 8.6: MIPS Comparison on netflix and yahoo-music.

– The improved Greedy-MIPS in Algorithm 8.5 with the selection tree in Algorithm 8.6.

– The improved Greedy-MIPS in Algorithm 8.5 with a max-heap.

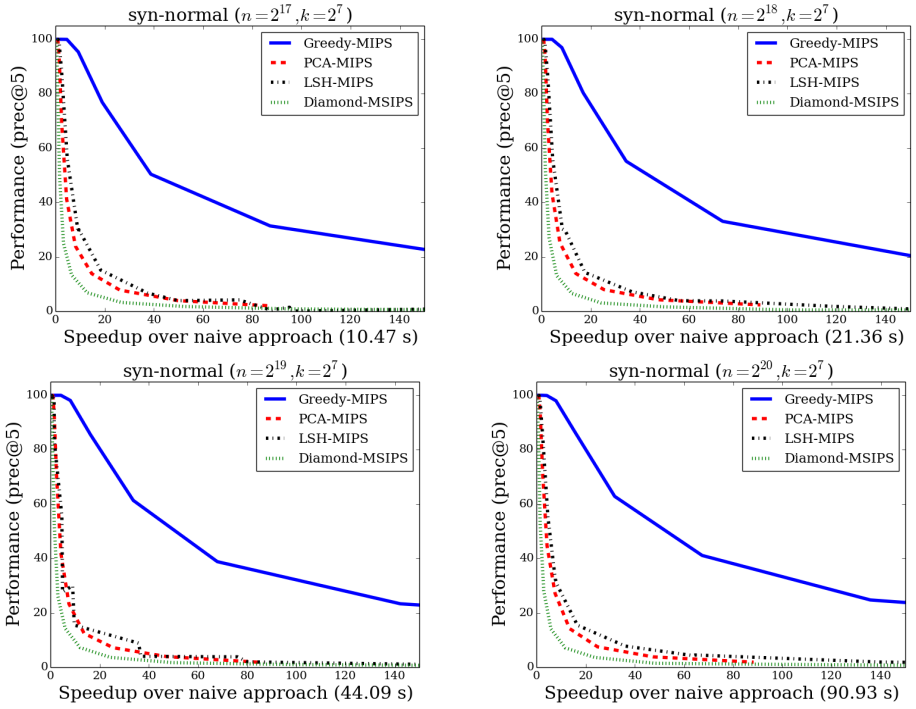


Figure 8.7: MIPS Comparison on synthetic datasets with $n \in 2^{\{17,18,19,20\}}$ and $k = 128$. The datasets used to generate results are created with each entry drawn from a normal distribution.

  - The original Greedy-MIPS in Algorithm 8.4 with the selection tree in Algorithm 8.6.

- NNS-based MIPS approaches:
  - PCA-MIPS: the approach proposed in [9], which is shown to be the state-of-the-art among tree-based approaches [9]. We implement a complete PCA-Tree with the neighborhood boosting techniques described in [9]. We vary the depth of PCA tree to control the trade-off between the search quality and the search efficiency.

  - LSH-MIPS: the approach proposed in [85, 104]. We use the nearest neighbor transform function proposed in [9, 85] and use the random projection scheme as the LSH function as suggested in [85]. We also implement the standard amplification procedure with an OR-construction of b hyper LSH hash functions. Each hyper LSH function is a result of an AND-construction of a random projections. We vary the values (a, b) to control the trade-off between the search quality and the search efficiency.

- Diamond-MSIPS: the sampling scheme proposed in [10] for the maximum squared inner product search. As it shows better performance than LSH-MIPS in [10] in terms of MIPS problems, we also include Diamond-MSIPS into our comparison. F+Tree [140] is implemented as the multinomial sampling procedure.
- Naive-MIPS: the baseline approach which applies a linear search to identify the exact top-K candidates.

Evaluation Criteria. For each dataset, the actual top-20 items for each query are regarded as the ground truth. We report the average performance on a randomly selected 2,000 query vectors. To evaluate the search quality, we use the precision on the top-K prediction (prec@K), is obtained by selecting top-K items from $\mathcal{C}(\boldsymbol{w})$ returned by the candidate screening procedure of a compared MIPS approach. $K = 5$ and $K = 10$ are used in our experiments. To evaluate the search efficiency, we report the relative speedups

over the Naive-MIPS approach as follows:

$$\text{speedup} = \frac{\text{prediction time required by Naive-MIPS}}{\text{prediction time by a compared approach}}.$$

Remarks on Budgeted MIPS versus Non-Budgeted MIPS. As mentioned in Section 8.2, PCA-MIPS and LSH-MIPS cannot handle MIPS with a budget. Both the search computation cost and the search quality are fixed when the corresponding data structure is constructed. As a result, to understand the trade-off between search efficiency and search quality for these two approaches, we can only try various values for its parameters (such as the depth for PCA tree and the amplification parameters (a, b) for LSH). For each combination of parameters, we need to re-run the entire query-independent pre-processing procedure to construct a new data structure.

8.4.2 Experimental Results

Results on Variants of Greedy-MIPS. In Figure 8.4, we shows the comparison between the three variants of Greedy-MIPS on netflix and yahoo-music. We can see that the difference between the use of a selection-tree and a max-heap is small, while the different between the use of Algorithm 8.4 and the use of Algorithm 8.5 is more significant. For the comparison to other MIPS approaches, we use Greedy-MIPS to denote the results obtained from the version with the combination of Algorithm 8.5 and Algorithm 8.6.

Results on Real-World Data sets. Comparison results for netflix and yahoo-music are shown in Figure 8.6. The first, second, and third columns

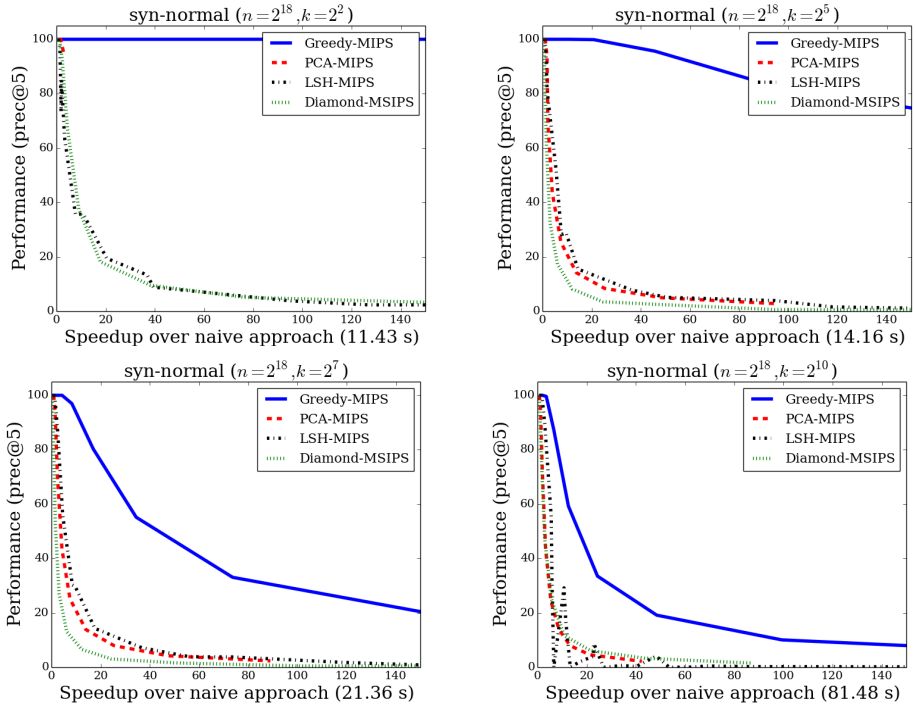


Figure 8.8: MIPS Comparison on synthetic datasets with $n = 2^{18}$ and $k \in 2^{\{2,5,7,10\}}$. The datasets used to generate results on are created with each entry drawn from a normal distribution.

present the results with $k = 50$, $k = 100$, and $k = 200$, respectively. It is clearly observed that given a fixed speedup, Greedy-MIPS yields predictions with much higher search quality. In particular, on the yahoo-music data set with $k = 200$, Greedy-MIPS runs 200x faster than Naive-MIPS and yields search results with $p@5 = 70\%$, while none of PCA-MIPS, LSH-MIPS, and Diamond-MSIPS can achieve a $p@5 > 10\%$ while maintaining the similar 200x speedups.

Results on Synthetic Data Sets. We also perform comparisons on synthetic datasets. The comparison with various $n \in 2^{\{17,18,19,20\}}$ is shown

in Figure 8.7, while the comparison with various $k \in 2^{\{2,5,7,10\}}$ is shown in Figure 8.8. We observe that the performance gap between Greedy-MIPS over other approaches remains when n increases, while the gap becomes smaller when k increases. However, Greedy-MIPS still outperforms other approaches significantly.

8.5 Summary of the Contributions

In this chapter, we study the computational issue in the prediction phase for many MF-based models: a maximum inner product search problem (MIPS) with a very large number of candidate embeddings. By carefully studying the problem structure of MIPS, we develop a novel Greedy-MIPS algorithm, which can handle budgeted MIPS by design. While simple and intuitive, Greedy-MIPS yields surprisingly superior performance compared to state-of-the-art approaches. As a specific example, on a candidate set containing half a million vectors of dimension 200, Greedy-MIPS runs 200x faster than the naive approach while yielding search results with the top-5 precision greater than 75%.

Bibliography

[1] Jacob Abernethy, Francis Bach, Theodoros Evgeniou, and Jean-Philippe Vert. Low-rank matrix factorization with attributes, 2006. arXiv preprint cs/0611124.

[2] Alekh Agarwal and John C. Duchi. Distributed delayed stochastic optimization. In J. Shawe-Taylor, R.S. Zemel, P. Bartlett, F.C.N. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 873–881, 2011.

[3] Rahul Agrawal, Archit Gupta, Yashoteja Prabhu, and Manik Varma. Multi-label learning with millions of labels: Recommending advertiser bid phrases for web pages. In *Proceedings of the International World Wide Web Conference*, 2013.

[4] Oren Anava, Elad Hazan, and Assaf Zeevi. Online time series prediction with missing data. In *Proceedings of the International Conference on Machine Learning*, pages 2191–2199, 2015.

[5] Arthur Asuncion, Padhraic Smyth, and Max Welling. Asynchronous distributed learning of topic models. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 81–88, Cambridge, MA, 2008. MIT Press.

[6] Arthur Asuncion, Max Welling, Padhraic Smyth, and Yee Whye Teh. On smoothing and inference for topic models. In *UAI*, pages 27–34, 2009.

[7] Alex Auvolat, Sarath Chandar, Pascal Vincent, Hugo Larochelle, and Yoshua Bengio. Clustering is efficient for approximate maximum inner product search, 2016. arXiv preprint arXiv:1507.05910.

[8] Francis Bach, Julien Mairal, and Jean Ponce. Convex sparse matrix factorizations, 2008. arXiv preprint arXiv:0812.1869.

[9] Yoram Bachrach, Yehuda Finkelstein, Ran Gilad-Bachrach, Liran Katzir, Noam Koenigstein, Nir Nice, and Ulrich Paquet. Speeding up the xbox recommender system using a euclidean transformation for inner-product spaces. In *Proceedings of the 8th ACM Conference on Recommender Systems*, pages 257–264, 2014.

[10] Grey Ballard, Seshadhri Comandur, Tamara Kolda, and Ali Pinar. Diamond sampling for approximate maximum all-pairs dot-product (MAD) search. In *Proceedings of the IEEE International Conference on Data Mining*, 2015.

[11] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, volume 14, pages 585–591, 2001.

[12] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396, 2003.

[13] Robert M. Bell, Yehuda Koren, and Chris Volinsky. Modeling relationships at multiple scales to improve accuracy of large recommender systems. In *Proceedings of the Thirteenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2007.

[14] Dimitri P. Bertsekas. *Nonlinear Programming*. Athena Scientific, Belmont, MA 02178-9998, second edition, 1999.

[15] David Blei, Andrew Ng, and Michael Jordan. Latent Dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[16] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In Yves Lechevallier and Gilbert Saporta, editors, *Proceedings of the 19th International Conference on Computational Statistics*, pages 177–187. Springer, August 2010.

[17] John S. Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 43–52, 1998.

[18] Leo Breiman and Jerome H. Friedman. Predicting Multivariate Re-

sponses in Multiple Linear Regression. *Journal of the Royal Stat. Soc.: Series B*, 59(1):3–54, 1997.

[19] Serhat Selcuk Bucak, Rong Jin, and Anil K. Jain. Multi-label learning with incomplete class assignments. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2011.

[20] Deng Cai, Xiaofei He, Jiawei Han, and Thomas S Huang. Graph regularized nonnegative matrix factorization for data representation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 33(8):1548–1560, 2011.

[21] Jian-Feng Cai, Emmanuel J Candès, and Zuowei Shen. A singular value thresholding algorithm for matrix completion. *SIAM Journal on Optimization*, 20(4):1956–1982, 2010.

[22] Gustavo Carneiro, Antoni B. Chan, Pedro J. Moreno, and Nuno Vasconcelos. Supervised learning of semantic classes for image annotation and retrieval. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(3):394–410, 2007.

[23] Venkat Chandrasekaran, Benjamin Recht, Pablo A Parrilo, and Alan S Willsky. The convex geometry of linear inverse problems. *Foundations of Computational Mathematics*, 12(6):805–849, 2012.

[24] Po-Lung Chen, Chen-Tse Tsai, Yao-Nan Chen, Ku-Chun Chou, Chun-Liang Li, Cheng-Hao Tsai, Kuan-Wei Wu, Yu-Cheng Chou, Chung-Yi

Li, Wei-Shih Lin, Shu-Hao Yu, Rong-Bing Chiu, Chieh-Yen Lin, Chien-Chih Wang, Po-Wei Wang, Wei-Lun Su, Chen-Hung Wu, Tsung-Ting Kuo, Todd G. McKenzie, Ya-Hsuan Chang, Chun-Sung Ferng, Chia-Mau Ni, Hsuan-Tien Lin, Chih-Jen Lin, and Shou-De Lin. A linear ensemble of individual and blended models for music. In *JMLR Workshop and Conference Proceedings: Proceedings of KDD Cup 2011 Competition*, volume 18, pages 21–60, 2012.

[25] Yao-Nan Chen and Hsuan-Tien Lin. Feature-aware label space dimension reduction for multi-label classification. In P. Bartlett, F.C.N. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1538–1546, 2012.

[26] Zhe Chen and Andrzej Cichocki. Nonnegative matrix factorization with temporal smoothness and/or spatial decorrelation constraints. *Laboratory for Advanced Brain Signal Processing, RIKEN, Tech. Rep*, 68, 2005.

[27] Wei-Sheng Chin, Yong Zhuang, Yu-Chin Juan, and Chih-Jen Lin. A fast parallel stochastic gradient method for matrix factorization in shared memory systems. *ACM Transactions on Intelligent Systems and Technology*, 6:2:1–2:24, 2015.

[28] Wei-Sheng Chin, Yong Zhuang, Yu-Chin Juan, and Chih-Jen Lin. A learning-rate schedule for stochastic gradient methods to matrix factor-

ization. In *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, 2015.

[29] Wei-Sheng Chin, Bo-Wen Yuan, Meng-Yuan Yang, Yong Zhuang, Yu-Chin Juan, and Chih-Jen Lin. LIBMF: A library for parallel matrix factorization in shared-memory systems. *Journal of Machine Learning Research*, 2016. To appear.

[30] Fan Chung, Linyuan Lu, and Van Vu. Spectral of random graphs with given expected degrees. *Internet Mathematics*, 1(3):257–275, 2003.

[31] Andrzej Cichocki and Anh-Huy Phan. Fast local algorithms for large scale nonnegative matrix and tensor factorizations. *IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences*, E92-A(3):708–721, 2009.

[32] Edith Cohen and David D. Lewis. Approximating matrix multiplication for pattern recognition tasks. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 682–691, 1997.

[33] Thomas H. Cormen, Charles E. Leiserson, and Ronald L Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1990.

[34] Gideon Dror, Noam Koenigstein, Yehuda Koren, and Markus Weimer. The Yahoo! music dataset and KDD-Cup'11. In *JMLR Workshop and Conference Proceedings: Proceedings of KDD Cup 2011 Competition*, volume 18, pages 3–18, 2012.

[35] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

[36] Peter M. Fenwick. A new data structure for cumulative frequency tables. *Software: Practice and Experience*, 24(3):327–336, 1994.

[37] Rainer Gemulla, Peter J. Haas, Erik Nijkamp, and Yannis Sismanis. Large-scale matrix factorization with distributed stochastic gradient descent. In *Proceedings of the Sixteenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2011.

[38] Zoubin Ghahramani and Geoffrey E. Hinton. Parameter estimation for linear dynamical systems. Technical report, Technical Report CRG-TR-96-2, University of Totronto, Dept. of Computer Science, 1996.

[39] Joseph E. Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2012.

[40] Prem Gopalan, Jake M. Hofman, and David M. Blei. Scalable recommendation with hierarchical poisson factorization. In *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 326–335, 2015.

[41] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley, 2nd edition, 1994.

[42] Thomas L Griffiths and Mark Steyvers. Finding scientific topics. *Proceedings of the National Academy of Sciences*, 101:5228–5235, 2004.

[43] Benjamin Haeffele, Eric Young, and Rene Vidal. Structured low-rank matrix factorization: Optimality, algorithm, and applications to image processing. In *Proceedings of the 31st International Conference on Machine Learning*, pages 2007–2015, 2014.

[44] Fang Han and Han Liu. Transition matrix estimation in high dimensional time series. In *Proceedings of the International Conference on Machine Learning*, pages 172–180, 2013.

[45] Bharath Hariharan, Lihi Zelnik-Manor, S. V. N. Vishwanathan, and Manik Varma. Large scale max-margin multi-label classification with priors. In *Proceedings of the International Conference on Machine Learning*, June 2010.

[46] Gregor Heinrich. Parameter estimation for text analysis. Technical report, Technical Report, 2008.

[47] Ngoc-Diep Ho and Paul Van Doorenand Vincent D. Blondel. Descent methods for nonnegative matrix factorization. In Paul Van Dooren, Shankar P. Bhattacharyya, Raymond H. Chan, Vadim Olshevsky, and

Aurobinda Routray, editors, *Numerical Linear Algebra in Signals, Systems and Control*, volume 80 of *Lecture Notes in Electrical Engineering*, pages 251–293. Springer Netherlands, 2011.

[48] Cho-Jui Hsieh and Inderjit S. Dhillon. Fast coordinate descent methods with variable selection for non-negative matrix factorization. In *Proceedings of the Sixteenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2011.

[49] Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S. Sathiya Keerthi, and Sellamanickam Sundararajan. A dual coordinate descent method for large-scale linear SVM. In *Proceedings of the Twenty Fifth International Conference on Machine Learning (ICML)*, 2008.

[50] Cho-Jui Hsieh, Mátyás A. Sustik, Inderjit. S. Dhillon, and Pradeepr K. Ravikumar. Sparse inverse covariance matrix estimation using quadratic approximation. In J. Shawe-Taylor, R.S. Zemel, P. Bartlett, F.C.N. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, 2011.

[51] Cho-Jui Hsieh, Nagarajan Natarajan, and Inderjit Dhillon. PU learning for matrix completion. In *Proceedings of the International Conference on Machine Learning*, pages 2445–2453, 2015.

[52] Cho-Jui Hsieh, Hsiang-Fu Yu, and Inderjit S. Dhillon. PASSCoDe: Parallel ASynchronous Stochastic dual Co-ordinate Descent. In *Proceedings*

of the International Conference on Machine Learning, pages 2370 – 2379, jul 2015.

[53] Daniel Hsu, Sham Kakade, John Langford, and Tong Zhang. Multi-label prediction via compressed sensing. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 772–780, 2009.

[54] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *Proceedings of the IEEE International Conference on Data Mining*, pages 263–272, 2008.

[55] ander Ihler and David Newman. Understanding errors in approximate distributed latent dirichlet allocation. *Knowledge and Data Engineering, IEEE Transactions on*, 24(5):952–960, 2012.

[56] Alan Julian Izenman. Reduced-Rank Regression for the Multivariate Linear Model. *Journal of Multivariate Analysis*, 5:248–264, 1975.

[57] Prateek Jain and Inderjit S. Dhillon. Provable inductive matrix completion. *arXiv preprint arXiv:1306.0626*, 2013.

[58] Mohsen Jamali and Martin Ester. A matrix factorization technique with trust propagation for recommendation in social networks. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, pages 135–142, 2010.

[59] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Journal of Fluids Engineering*, 82(1):35–45, 1960.

[60] Vassilis Kalofolias, Xavier Bresson, Michael Bronstein, and Pierre Vandergheynst. Matrix completion on graphs, 2014. arXiv preprint arXiv:1408.1717.

[61] Ashish Kapoor, Raajay Viswanathan, and Prateek Jain. Multilabel classification using bayesian compressed sensing. In P. Bartlett, F.C.N. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2654–2662, 2012.

[62] Donald E. Knuth. *The Art of Cmoputer Programming, Volumne 3: Sorting and Searching*. Addison-Wesley, 2nd edition, 1998.

[63] Noam Koenigstein and Ulrich Paquet. Xbox movies recommendations: Variational bayes matrix factorization with embedded feature selection. In *Proceedings of the 7th ACM Conference on Recommender Systems*, pages 129–136, 2013.

[64] Noam Koenigstein, Parikshit Ram, and Yuval Shavitt. Efficient retrieval of recommendations in a matrix factorization framework. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, CIKM '12, pages 535–544, 2012.

[65] Yehuda Koren, Robert M. Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *IEEE Computer*, 42:30–37, 2009.

[66] John Langford, Alexander J. Smola, and Martin Zinkevich. Slow learners are fast. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 2331–2339, 2009.

[67] Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In *Advances in Neural Information Processing Systems*, pages 2177–2185, 2014.

[68] Aaron Q. Li, Amr Ahmed, Sujith Ravi, and Alexander J. Smola. Reducing the sampling complexity of topic models. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2014.

[69] Lei Li and B Aditya Prakash. Time series clustering: Complex is simpler! In *Proceedings of the International Conference on Machine Learning*, pages 185–192, 2011.

[70] Lei Li, James McCann, Nancy S Pollard, and Christos Faloutsos. DynaMMo: Mining and summarization of coevolving sequences with missing values. In *ACM SIGKDD International Conference on Knowledge discovery and data mining*, pages 507–516. ACM, 2009.

[71] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 583–598, 2014.

[72] Wu-Jun Li and Dit-Yan Yeung. Relation regularized matrix factorization. In *21st International Joint Conference on Artificial Intelligence*, 2009.

[73] Darryl Lim, Julian McAuley, and Gert Lanckriet. Top-n recoomendation with missing implicit feedback. In *Proceedings of the Sixth ACM Conference on Recommender Systems*, pages 309–312, 2015.

[74] Chih-Jen Lin, Ruby C. Weng, and S. Sathiya Keerthi. Trust region Newton method for large-scale logistic regression. *Journal of Machine Learning Research*, 9:627–650, 2008.

[75] Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph M. Hellerstein. Graphlab: A new framework for parallel machine learning. *CoRR*, abs/1006.4990, 2010.

[76] Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph M. Hellerstein. Distributed GraphLab: A framework for machine learning in the cloud. *PVLDB*, 5(8):716–727, 2012.

[77] Hao Ma, Dengyong Zhou, Chao Liu, Michael R Lyu, and Irwin King. Recommender systems with social regularization. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 287–296. ACM, 2011.

[78] Igor Melnyk and Arindam Banerjee. Estimating structured vector autoregressive model. In *Proceedings of the Thirty Third International Conference on Machine Learning (ICML)*, 2016.

[79] Aditya Krishna Menon and Charles Elkan. Link prediction via matrix factorization. In *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD)*, pages 437–452, 2011.

[80] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[81] Nagarajan Natarajan and Inderjit S. Dhillon. Inductive matrix completion for predicting gene-disease associations. *Bioinformatics*, 30:160–168, 2014.

[82] Sahand N. Negahban and Martin J. Wainwright. Restricted strong convexity and weighted matrix completion: Optimal bounds with noise. *Journal of Machine Learning Research*, 13(1):1665–1697, 2012.

[83] Sahand N. Negahban, Pradeep K. Ravikumar, Martin J. Wainwright, and Bin Yu. A unified framework for high-dimensional analysis of m-estimators with decomposable regularizers. *Statistical Science*, 27(4): 538–557, 2012.

[84] David Newman, Arthur Asuncion, Padhraic Smyth, and Max Welling. Distributed algorithms for topic models. *Journal of Machine Learning Research*, 10:1801–1828, 2009.

[85] Behnam Neyshabur and Nathan Srebro. On symmetric and asymmetric lshs for inner product search. In *Proceedings of the International Conference on Machine Learning*, pages 1926–1934, 2015.

[86] William B. Nicholson, David S. Matteson, and Jacob Bien. Structured regularization for large vector autoregressions. Technical report, Technical Report, University of Cornell, 2014.

[87] Feng Niu, Benjamin Recht, Christopher Re, and Stephen J. Wright. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In J. Shawe-Taylor, R.S. Zemel, P. Bartlett, F.C.N. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 693–701, 2011.

[88] Rong Pan and Martin Scholz. Mind the gaps: Weighting the unknown in large-scale one-class collaborative filtering. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 667–676, 2009.

[89] Rong Pan, Yunhong Zhou, Bin Cao, Nathan N Liu, Rajan Lukose, Martin Scholz, and Qiang Yang. One-class collaborative filtering. In *Proceedings of the IEEE International Conference on Data Mining*, pages 502–511, 2008.

[90] Ulrich Paquet and Noam Koenigstein. One-class collaborative filtering with random graphs. In *Proceedings of the International World Wide Web Conference*, pages 999–1008, 2013.

[91] Giovanni Petris. An R package for dynamic linear models. *Journal of Statistical Software*, 36(12):1–16, 2010.

[92] Giovanni Petris, Sonia Petrone, and Patrizia Campagnoli. *Dynamic Linear Models with R*. Use R! Springer, 2009.

[93] István Pilászy, Dávid Zibriczky, and Domonkos Tikk. Fast ALS-based matrix factorization for explicit and implicit feedback datasets. In *Proceedings of the fourth ACM conference on Recommender systems*, ACM RecSys '10, pages 71–78, 2010.

[94] Martin Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[95] Swati Rallapalli, Lili Qiu, Yin Zhang, and Yi-Chao Chen. Exploiting temporal stability and low-rank structure for localization in mobile networks. In *International Conference on Mobile Computing and Networking*, MobiCom '10, pages 161–172. ACM, 2010.

[96] Parikshit Ram and Alexander G. Gray. Maximum inner-product search using cone trees. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 931–939, 2012.

[97] Nikhil Rao, Parikshit Shah, and Stephen Wright. Conditional gradient with enhancement and truncation for atomic-norm regularization. In *NIPS Workshop on Greedy Algorithms*, 2013.

[98] Nikhil Rao, Hsiang-Fu Yu, Pradeep K. Ravikumar, and Inderjit S. Dhillon. Collaborative filtering with graph information: Consistency and scalable methods. In *Advances in Neural Information Processing Systems 27*, 2015.

[99] Benjamin Recht. A simpler approach to matrix completion. *Journal of Machine Learning Research*, 12:3413–3430, 2011.

[100] Benjamin Recht and Christopher Ré. Parallel stochastic gradient algorithms for large-scale matrix completion. *Mathematical Programming Computation*, 5(2):201–226, June 2013.

[101] Steffen Rendle and Christoph Freudenthaler. Improving pairwise learning for item recommendation from implicit feedback. In *Proceedings of the 7th ACM international conference on Web search and data mining (WSDM)*, pages 273–282, 2014.

[102] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 452–461, 2009.

[103] Matthew Roughan, Yin Zhang, Walter Willinger, and Lili Qiu. Spatio-temporal compressive sensing and internet traffic matrices (extended version). *IEEE/ACM Transactions on Networking*, 20(3):662–676, June 2012.

[104] Anshumali Shrivastava and Ping Li. Asymmetric lsh (ALSH) for sublinear time maximum inner product search (MIPS). In Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2321–2329, 2014.

[105] Anshumali Shrivastava and Ping Li. Improved asymmetric locality sensitive hashing lsh (ALSH) for maximum inner product search (MIPS). In *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 812–821, 2015.

[106] Robert H. Shumway and David S. Stoffer. An approach to time series smoothing and forecasting using the EM algorithm. *Journal of time series analysis*, 3(4):253–264, 1982.

[107] Si Si, Kai-Yang Chiang, Cho-Jui Hsieh, Nikhil Rao, and Inderjit S. Dhillon. Goal-directed inductive matrix completion. In *Proceedings of*

the 2ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2016.

[108] Vikas Sindhwani, Serhat S Bucak, Jianying Hu, and Aleksandra Mojsilovic. One-class matrix completion with low-density factorizations. In *Proceedings of the IEEE International Conference on Data Mining (ICDM)*, pages 1055–1060, 2010.

[109] Alexander J. Smola and Risi Kondor. Kernels and regularization on graphs. In *Learning theory and kernel machines*, pages 144–158. Springer, 2003.

[110] Alexander J. Smola and Shravan Narayanamurthy. An architecture for parallel topic models. *Proceedings of the VLDB Endowment*, 3(1):703–710, 2010.

[111] Peter Snyder. tmpfs: A virtual memory file system. In *Proceedings of the Autumn 1990 European UNIX Users' Group Conference*, 1990.

[112] Nathan Srebro and Ruslan Salakhutdinov. Collaborative filtering in a non-uniform world: Learning with the weighted trace norm. In J. Lafferty, C. Williams, J. Shawe-taylor, R.s. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 2056–2064, 2010.

[113] John Z. Sun, Kush R. Varshney, and Karthik Subbian. Dynamic matrix factorization: A state space approach. In *Proceedings of Inter-*

national Conference on Acoustics, Speech and Signal Processing, pages 1897–1900. IEEE, 2012.

[114] Liang Sun, Shuiwang Ji, and Jieping Ye. Canonical correlation analysis for multi-label classification: A least squares formulation, extensions and analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):194–200, 2011.

[115] Farbound Tai and Hsuan-Tien Lin. Multi-label classification with principal label space transformation. *Neural Computation*, 24(9):2508–2542, 2012.

[116] Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. Scalable collaborative filtering approaches for large recommender systems. *Journal of Machine Learning Research*, 10:623–656, 2009.

[117] Ambuj Tewari, Pradeep K. Ravikumar, and Inderjit S. Dhillon. Greedy algorithms for structurally constrained high dimensional problems. In J. Shawe-taylor, R.s. Zemel, P. Bartlett, F.c.n. Pereira, and K.q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 882–890, 2011.

[118] Rajeev Thakur and William Gropp. Improving the performance of collective operations in MPICH. In *Proceedings of European PVM/MPI Users' Group Meeting*, 2003.

[119] Roman Vershynin. A note on sums of independent random matrices after ahlswede-winter. *Lecture notes*, 2009.

[120] Michael D. Vose. A linear algorithm for generating random numbers with a given distribution. *IEEE Transactions on Software Engineering*, 17(9):972–975, 1991.

[121] Alastair J. Walker. An efficient method for generating discrete random variables with general distributions. *ACM Transactions on Mathematical Software*, 3(3):253–256, September 1977.

[122] Changhu Wang, Shuicheng Yan, Lei Zhang, and Hong-Jiang Zhang. Multi-Label Sparse Coding for Automatic Image Annotation. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2009.

[123] Hansheng Wang, Guodong Li, and Chih-Ling Tsai. Regression coefficient and autoregressive order shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 69 (1):63–78, 2007.

[124] Yi Wang, Hongjie Bai, Matt Stanton, Wen-Yen Chen, and Edward Y. Chang. PLDA: Parallel latent Dirichlet allocation for large-scale applications. In *International Conference on Algorithmic Aspects in Information and Management*, 2009.

[125] Mike West and Jeff Harrison. *Bayesian Forecasting and Dynamic Models*. Springer Series in Statistics. Springer, 2013.

[126] Jason Weston. Personal Communication, 2013.

[127] Jason Weston, Samy Bengio, and Nicolas Usunier. Large scale image annotation: learning to rank with joint word-image embeddings. *Mach. Learn.*, 81(1):21–35, October 2010.

[128] R. Clint Whaley, Antoine Petitet, and Jack J. Dongarra. Automated empirical optimizations of software and the atlas project. *Parallel Computing*, 27:3–35, 2001.

[129] Chak-Kuen Wong and Malcolm C. Easton. An efficient method for weighted sampling without replacement. *SIAM Journal on Computing*, 9(1):111–113, 1980.

[130] Liang Xiong, Xi Chen, Tzu-Kuo Huang, Jeff G Schneider, and Jaime G. Carbonell. Temporal collaborative filtering with Bayesian probabilistic tensor factorization. In *SIAM International Conference on Data Mining*, pages 223–234, 2010.

[131] Miao Xu, Rong Jin, and Zhi-Hua Zhou. Speedup matrix completion with side information: Application to multi-label learning. In C.j.c. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2301–2309, 2013.

[132] Yangyang Xu and Wotao Yin. A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion. *SIAM Journal on Imaging Sciences*, 6(3):1758–1789, 2013.

[133] Feng Yan, Ningyi Xu, and Yuan Qi. Parallel inference for latent Dirichlet allocation on graphics processing units. In Y. Bengio, D. Schuurmans, J.D. Lafferty, C.K.I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 2134–2142, 2009.

[134] Limin Yao, David Mimno, and Andrew McCallum. Efficient methods for topic model inference on streaming document collections. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 937–946. ACM, 2009.

[135] Hsiang-Fu Yu, Fang-Lan Huang, and Chih-Jen Lin. Dual coordinate descent methods for logistic regression and maximum entropy models. *Machine Learning*, 85(1-2):41–75, 2011.

[136] Hsiang-Fu Yu, Cho-Jui Hsieh, Si Si, and Inderjit S. Dhillon. Scalable coordinate descent approaches to parallel matrix factorization for recommender systems. In *Proceedings of the IEEE International Conference on Data Mining*, pages 765–774, 2012.

[137] Hsiang-Fu Yu, Chia-Hua Ho, Yu-Chin Juan, and Chih-Jen Lin. Libshorttext: A library for short-text classification and analysis. Technical report, National Taiwan University, 2013.

[138] Hsiang-Fu Yu, Cho-Jui Hsieh, Si Si, and Inderjit S. Dhillon. Parallel matrix factorization for recommender systems. *Knowledge and Information Systems*, 41(3):793–819, 2014.

[139] Hsiang-Fu Yu, Prateek Jain, Purushottam Kar, and Inderjit S. Dhillon. Large-scale multi-label learning with missing labels. In *Proceedings of the International Conference on Machine Learning*, pages 593–601, 2014.

[140] Hsiang-Fu Yu, Cho-Jui Hsieh, Hyokun Yun, S.V.N. Vishwanathan, and Inderjit S. Dhillon. A scalable asynchronous distributed algorithm for topic modeling. In *Proceedings of the International World Wide Web Conference*, pages 1340–1350, may 2015.

[141] Hsiang-Fu Yu, Nikhil Rao, and Inderjit S. Dhillon. Temporal regularized matrix factorization. In *Time Series Workshop in NIPS*, 2015.

[142] Hsiang-Fu Yu, Mikhail Bilenko, and Chih-Jen Lin. Selection of negative samples for one-class matrix factorization. Technical report, National Taiwan University, 2016. URL http://www.csie.ntu.edu.tw/~cjlin/papers/one-class/biased-mf-jmlr.pdf.

[143] Hsiang-Fu Yu, Hsing-Yi Huang, Inderjit S. Dhillon, and Chih-Jen Lin. A unified algorithm for one-class structured matrix factorization with side information, 2016. Working Manuscript.

[144] Hsiang-Fu Yu, Nikhil Rao, and Inderjit S. Dhillon. High-dimensional

time series prediction with missing values, 2016. arXiv preprint arXiv:1509.08333.

[145] Hyokun Yun, Hsiang-Fu Yu, Cho-Jui Hsieh, S. V. N. Vishwanathan, and Inderjit S. Dhillon. NOMAD: Non-locking, stOchastic Multi-machine algorithm for Asynchronous and Decentralized matrix completion. *Proceedings of the VLDB Endowment*, 7(11):975–986, 2014.

[146] Yin Zhang, Matthew Roughan, Walter Willinger, and Lili Qiu. Spatio-temporal compressive sensing and internet traffic matrices. *SIGCOMM Comput. Commun. Rev.*, 39(4):267–278, August 2009. ISSN 0146-4833.

[147] Zhou Zhao, Lijun Zhang, Xiaofei He, and Wilfred Ng. Expert finding for question answering via graph regularized matrix completion. *Transactions on Knowledge and Data Engineering*, 27(4):993–1004, 2015.

[148] Tinghui Zhou, Hanhuai Shan, Arindam Banerjee, and Guillermo Sapiro. Kernelized probabilistic matrix factorization: Exploiting graphs and side information. In *SDM*, volume 12, pages 403–414. SIAM, 2012.

[149] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. Large-scale parallel collaborative filtering for the Netflix prize. In *Proceedings of International Conference on Algorithmic Aspects in Information and Management*, 2008.

[150] Martin Zinkevich, Markus Weimer, Alexander J. Smola, and Lihong Li. Parallelized stochastic gradient descent. In J. Lafferty, C. K. I. Williams,

J. Shawe-Taylor, R.S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 2595–2603, 2010.

Vita

Hsiang-Fu Yu is a Ph.D. student at the University of Texas at Austin. His research focus is developing new algorithms and optimization techniques for large-scale machine learning problems. Hsiang-Fu obtained his B.S. degree in 2008 and M.S. degree in 2010 from National Taiwan University (advisor: Prof. Chih-Jen Lin). Currently, he is a member of the Center for Big Data Analytics led by Prof. Inderjit Dhillon. He is the recipient of the Intel Ph.D. fellowship in 2014-2015, the best research paper award in KDD 2010, and the best paper award in ICDM 2012.

Email address: rofuyu@cs.utexas.edu

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.