

LIBMF: A Library for Parallel Matrix Factorization in Shared-memory Systems

Wei-Sheng Chin

*Department of Computer Science
National Taiwan University
Taipei, Taiwan*

D01944006@CSIE.NTU.EDU.TW

Bo-Wen Yuan

*Department of Computer Science
National Taiwan University
Taipei, Taiwan*

R03944049@CSIE.NTU.EDU.TW

Meng-Yuan Yang

*Department of Computer Science
National Taiwan University
Taipei, Taiwan*

B01902037@CSIE.NTU.EDU.TW

Yong Zhuang

*Department of Computer Science
National Taiwan University
Taipei, Taiwan*

R01922139@CSIE.NTU.EDU.TW

Yu-Chin Juan

*Department of Computer Science
National Taiwan University
Taipei, Taiwan*

R01922136@CSIE.NTU.EDU.TW

Chih-Jen Lin

*Department of Computer Science
National Taiwan University
Taipei, Taiwan*

CJLIN@CSIE.NTU.EDU.TW

Editor:

Abstract

Matrix factorization (MF) plays a key role in many applications such as recommender systems and computer vision, but MF may take long running time for handling large matrices commonly seen in the big data era. Many parallel techniques have been proposed to reduce the running time, but few parallel MF packages are available. Therefore, we present an open source library, LIBMF, based on recent advances of parallel MF for shared-memory systems. LIBMF includes easy-to-use command-line tools, interfaces to C/C++ languages, and comprehensive documentation. Our experiments demonstrate that LIBMF outperforms state of the art packages. LIBMF is BSD-licensed, so users can freely use, modify, and redistribute the code.

Keywords: Matrix factorization, non-negative matrix factorization, binary matrix factorization, logistic matrix factorization, one-class matrix factorization, stochastic gradient method, adaptive learning rate, parallel computation

1. Introduction

Matrix factorization (MF) and its variants cover a wide range of applications including recommender systems, link prediction, image processing, and document clustering. Although parallel MF has been widely investigated in recent years, it is still not easy to find a MF package supporting both rich formulations and parallel computation. For example, the parallel MF algorithm in LIBPMF¹ is known to be state of the art, but LIBPMF can only solve real-valued matrix factorization with a squared loss and L2-norm regularization. Some packages (e.g., scikit-learn,² mlpack,³ and nimfa⁴) support a variety of MF problems, but unfortunately they do not implement parallel MF algorithms published in recent five years. We thus develop LIBMF for solving a family of MF problems based on the techniques proposed in Chin et al. (2015a) and Chin et al. (2015b) targeting at shared-memory systems with multi-core CPUs (e.g., modern PCs). Many computational issues were addressed to make LIBMF efficient. Our experiments show that LIBMF is faster than state of the art packages. Moreover, LIBMF allows users to use their disk as a buffer to factorize a huge matrix that may not fit into their memory. LIBMF is released under BSD license at

<http://www.csie.ntu.edu.tw/~cjlin/libmf/>.

This paper is organized as follows. Section 2 introduces LIBMF in detail. Problems solved by LIBMF are presented in Section 2.1 and we demonstrate the practical usage in Section 2.2. The documentation is summarized in Section 2.3. Comparisons of LIBMF and state of the art packages are shown in Section 3. Finally, we conclude in Section 4. We provide the implementation details in a supplementary document.⁵

2. The Software Package

LIBMF provides efficient parallel MF solvers with rich documentation for a family of MF problems. For practical usage, users can directly run LIBMF using command-line instructions, make a C/C++ function call, or use the third-party R interface. LIBMF is portable by following the C++11 standard and can be compiled on Unix-like systems as well as Microsoft Windows.

2.1 Problems Solved by LIBMF

In general, MF is a process to find two factor matrices, $P \in \mathbb{R}^{k \times m}$ and $Q \in \mathbb{R}^{k \times n}$, to describe a given m -by- n training matrix R in which some entries may be missing. MF can be found in many applications, but we only use collaborative filtering in recommender systems as examples. We further assume that the entries of R are the historical users' preferences for merchandises, and the task on hand is to predict unobserved user behavior (i.e., missing entries in R) to make a suitable recommendation. Let u and v stand for row index and column index, respectively. For rating prediction (e.g., Koren et al., 2009), the entry value $r_{u,v} \in \mathbb{R}$ indicates that the v th item was rated $r_{u,v}$ by the u th user. Once P and Q are

1. <http://www.cs.utexas.edu/~rofuyu/libpmf>

2. <http://scikit-learn.org/stable/>

3. <http://www.mlpack.org>

4. <http://nimfa.biolab.si/>

5. http://www.csie.ntu.edu.tw/~cjlin/papers/libmf/libmf_supp.pdf

learned, a missing rating at the (u', v') entry can be predicted by the inner product of the u' th column of P (i.e., $\mathbf{p}_{u'}$) and the v' th column of Q (i.e., $\mathbf{q}_{v'}$). It means that we can generate the predicted scores on all items for a user, and then the one with highest score may be recommended. Sometimes, the users preferences recorded may be binary signals (e.g., “like” or “dislike”); for example, Das et al. (2007). The entry $r_{u,v}$ is positive if we know that the u th user likes the v th item, and vice versa. In this case, we can use the sign of $\mathbf{p}_u^T \mathbf{q}_v$ as the prediction value of an unobserved signal $r_{u,v}$. As a result, the domain of the entries in R is allowed to be either real values \mathbb{R} or a binary set $\{-1, 1\}$ to cover the two scenarios.

LIBMF supports two types of matrix factorization, real-valued matrix factorization (RVMF) and binary matrix factorization (BMF). RVMF aims to find P and Q to approximate the values of entries as accurate as possible while BMF focuses on learning the correct signs of entries. Both RVMF and BMF can be formulated as a non-convex optimization problem

$$\min_{P, Q} \sum_{(u,v) \in R} \left[f(\mathbf{p}_u, \mathbf{q}_v; r_{u,v}) + \mu_p \|\mathbf{p}_u\|_1 + \mu_q \|\mathbf{q}_v\|_1 + \frac{\lambda_p}{2} \|\mathbf{p}_u\|_2^2 + \frac{\lambda_q}{2} \|\mathbf{q}_v\|_2^2 \right], \quad (1)$$

where $r_{u,v}$ is the (u, v) entry of R , $f(\cdot)$ is a non-convex loss function of \mathbf{p}_u and \mathbf{q}_v , and μ_p , μ_q , λ_p , and λ_q are regularization coefficients. For RVMF, the loss function can be a squared loss, an absolute loss, or generalized KL-divergence. If R is a binary matrix, users may select among logistic loss, hinge loss, and squared hinge loss to perform BMF. Note that, the non-negative constraints, $P \in \mathbb{R}_+^{k \times m}$ and $Q \in \mathbb{R}_+^{k \times n}$, can be activated for (1). With the general setting in (1), currently LIBMF can solve more problems than its previous versions. The newly covered problems include, for example, NMF with generalized KL-divergence (Lee and Seung, 2001), sparse NMF (Lee et al., 2007), logistic NMF (Tomé et al., 2015), and maximum-margin MF (Srebro et al., 2005).

A special case of BMF is one-class matrix factorization (OCMF) in which the training matrix is still binary but contains only positive entries. OCMF is worthy to be considered as in some online activities (e.g., Pan et al., 2008) only positive feedbacks (i.e., purchase or click) from users can be monitored. The OCMF model in LIBMF is Bayesian personalized ranking (BPR) proposed by Rendle et al. (2009).

Refer to the supplementary materials for details of RVMF, BMF, and OCMF.

2.2 Practical Usage

In a sub-directory **demo**, we prepare three data sets for helping users to understand the data format and demonstrating RVMF, BMF, and OCMF. Each data set has training and test files, of which each line is a 3-tuple of row index, column index, and the value. For RVMF, the value is a real number. For BMF, any value greater than zero would be treated as a positive label and a negative label otherwise. For OCMF, the training file should contain only positive entries. A shell script **demo.sh** contains the training and prediction commands for different types of MF problems. For example, we may use the following command to perform RVMF on **train_file** and save the result to a model file **model**.

```
$ ./mf-train train_file model
```

The model file contains two parts: the columns of $P \in \mathbb{R}^{k \times m}$ and then the columns of

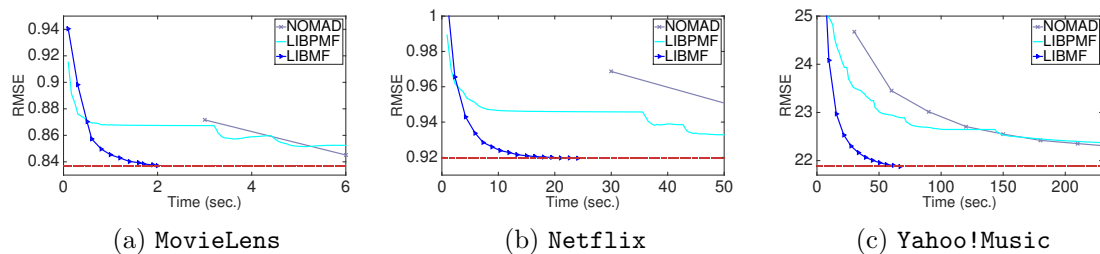


Figure 1: Comparisons with state of the art packages. The figure is generated from the results in Chin et al. (2015b). The platform for the experiments is a Linux server with two Intel Xeon 2620 CPUs and 64GB memory. Twelve threads are used for all packages.

$Q \in \mathbb{R}^{k \times n}$. To evaluate the model on a test set, users may run

```
$ ./mf-predict test_file model out
```

The file `out` contains predictions of the entries specified in `test_file`, and the default evaluation measure, root mean square error (RMSE), is printed. Users can also choose absolute error, generalized KL-divergence, logarithmic loss, accuracy, mean percentile rank, or area under the curve.

2.3 Documentation

The script `demo.sh` mentioned in Section 2.2 is a quick start guide. The `README` file covers information including an installation guide, a C/C++ API guide, and examples of command line usages. Users can refer to Chin et al. (2015b) to see how we select parameters for some representative data sets. The two files `mf-train.cpp` and `mf-predict.cpp` for the training and prediction commands, respectively, are good examples demonstrating the use of API functions. For the parallel stochastic gradient methods used in LIBMF, details are in Chin et al. (2015a), Chin et al. (2015b), and the supplementary material.

3. Comparisons

We compare LIBMF with state of the art packages, LIBPMF (Yu et al., 2012) and NOMAD (Yun et al., 2014), for parallel matrix factorization using a squared loss and L2-regularization. Results in Figure 1 indicate that LIBMF is efficient.

4. Conclusions

We develop LIBMF for fully utilizing the computational power of modern multi-core machines to solve several MF problems. The newly added features listed below make LIBMF significantly improved in compared with its previous versions.

1. More loss functions are supported for RVMF.
2. The framework is extended to cover BMF, OCMF, and L1 regularization.
3. For different type of MF problems, we implement suitable evaluation criteria such as mean percentile rank and area under the curve.
4. LIBMF can use disk to cache the training matrix, so the capability to handle huge matrices is largely enhanced.

References

- 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, 2015a. URL http://www.csie.ntu.edu.tw/~cjlin/papers/libmf/libmf_journal.pdf.
- Wei-Sheng Chin, Yong Zhuang, Yu-Chin Juan, and Chih-Jen Lin. A learning-rate schedule for stochastic gradient methods to matrix factorization. In *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, 2015b. URL http://www.csie.ntu.edu.tw/~cjlin/papers/libmf/mf_adaptive_pakdd.pdf.
- Abhinandan Das, Mayur Datar, Shyam Rajaram, and Ashutosh Garg. Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th international conference on World Wide Web*, pages 271–280, 2007.
- Yehuda Koren, Robert M. Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42:30–37, 2009.
- Daniel D. Lee and H. Sebastian Seung. Algorithms for non-negative matrix factorization. In Todd K. Leen, Thomas G. Dietterich, and Volker Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 556–562. MIT Press, 2001.
- Honglak Lee, Alexis Battle, Rajat Raina, and Andrew Y. Ng. Efficient sparse coding algorithms. In *Advances in Neural Information Processing Systems*, pages 801–808. 2007.
- Rong Pan, Yunhong Zhou, Bin Cao, Nathan N Liu, Rajan Lukose, Martin Scholz, and Qiang Yang. One-class collaborative filtering. In *IEEE International Conference on Data Mining (ICDM)*, pages 502–511, 2008.
- 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.
- Nathan Srebro, Jason D. M. Rennie, and Tommi S. Jaakola. Maximum-margin matrix factorization. In *Advances in Neural Information Processing Systems*, pages 1329–1336, 2005.
- Ana Maria Tomé, Reinhard Schachtner, Vincent Vigneron, Carlos Garcia Puntonet, and Elmar Wolfgang Lang. A logistic non-negative matrix factorization approach to binary data sets. *Multidimensional Systems and Signal Processing*, 26(1):125–143, 2015.
- 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.
- 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. In *International Conference on Very Large Data Bases (VLDB)*, 2014.