LibRec: A Java Library for Recommender Systems

Guibing Guo*, Jie Zhang†, Zhu Sun†, and Neil Yorke-Smith‡

*School of Information Systems, Singapore Management University, Singapore
†School of Computer Engineering, Nanyang Technological University, Singapore
‡Suliman S. Olayan School of Business, American University of Beirut, Lebanon
*gbguo@smu.edu.sg, †{zhangj,sunzhu}@ntu.edu.sg, ‡nysmith@aub.edu.lb

Abstract. The large array of recommendation algorithms proposed over the years brings a challenge in reproducing and comparing their performance. This paper introduces an open-source Java library that implements a suite of state-of-the-art algorithms as well as a series of evaluation metrics. We empirically find that *LibRec* performs faster than other such libraries, while achieving competitive evaluative performance.

1 Introduction

Recommender systems have been developed for decades, and a large number of algorithms have been proposed by the community. As more and more algorithms are being designed, the concern of reproducibility of algorithm performance grows [1]. Although multiple open-source frameworks exist for the purpose of algorithm reproduction and comparison, many of them only implement a set of classic algorithms which are now outdated [2]. Hence, we posit that more efforts are warranted in comparing the current state-of-the-art algorithms with new algorithms that are rapidly emerging.

This paper proposes an open-source Java library for recommender systems, called *LibRec*[1]. The LibRec library implements a suite of state-of-the-art recommendation algorithms as well as the traditional methods. In addition, a series of evaluation metrics are implemented including diversity-based metrics which are rarely enabled in other libraries. LibRec provides a platform for fair comparisons among different algorithms in multiple aspects, given the fact that the evaluative performance depends on data characteristic. It also provides a high flexibility for expansion with new algorithms.

2 The LibRec Library

LibRec is GPL-licensed Java software[2] (version 1.7 or higher required), which can be easily deployed and executed in any platforms including MS Windows,

[1] `http://www.librec.net`. Version 1.3 of LibRec is described in this paper
[2] Source code hosted in GitHub: `https://github.com/guoguibing/librec`

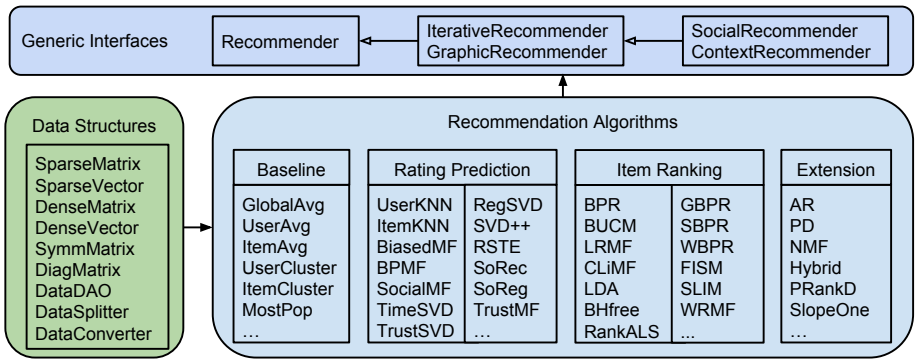

Fig. 1. The Class Structure of the LibRec Library

Linux, and Mac OS. It facilitates the study of the two classic problems of recommender systems, namely *rating prediction* and *item recommendation*. The LibRec framework consists of three major components, namely generic interfaces, data structures and recommendation algorithms, as illustrated in Figure 1.

2.1 Generic Interfaces

Generic interfaces define a set of abstract recommenders that can be extended and implemented by specific recommendation algorithms. In LibRec version 1.3, five generic recommenders are implemented. *Recommender* defines a general recommender which is extended by baselines and some other algorithms such as SlopeOne. *IterativeRecommender* defines a recommender usually based on iterative learning techniques. Matrix factorization-based approaches (e.g., SVD++, BiasedMF) are often derived from it. *GraphicRecommender* suits for the algorithms based on probabilistic graphic models (e.g., LDA, BUCM). *SocialRecommender* defines a recommender that incorporates social information, such as SocialMF and TrustSVD. Lastly, *ContextRecommender* defines a recommender that integrates additional contextual information into recommendations, for example, temporal information by TimeSVD. Although social connections are indeed a form of contextual information, we determine to expand from *IterativeRecommender* rather than from *ContexRecommender*. This is because many social recommenders have been proposed and the format of social connections is relatively simple and consistent across different data sets. Additional generic interfaces can be defined by further development. By defining those interfaces, new algorithms can be easily implemented by focusing on their own logics.

2.2 Data Structures

At least four data structures are widely and heavily used to implement recommender systems, namely sparse matrices and vectors, and dense matrices and vectors. Other structures include symmetric and diagonal matrices. The data structures have a great influence on the execution time of recommenders. An

important characteristic of LibRec is that it runs much faster than other counterparts[3]. Our implementations are mainly inspired by a Java matrix library MTJ[4], one of the most powerful matrix libraries in Java. MTJ implements several sparse matrix structures in a compressed row (column) storage for effective row (column) operations. Since it is often necessary to operate both in rows and columns, we choose to implement a sparse matrix class by keeping both compressed row and column storages, whereby additional utility functions can be easily added. Similarly, our sparse vector is an enhanced version of that in MTJ by incorporating a number of functionalities. For dense matrix and vector, we discard the implementations of MTJ which only stores data in a single array and hence not suitable for large-scale data storage. In contrast, we implement our own versions using two-dimensional Java arrays. Java caching techniques are also adopted to further boost algorithm executions. In addition, LibRec includes (1) *DataDAO*, a data access object (DAO) for input/output data operations; (2) *DataSplitter*, a utility class to split a data set into the training and test subsets; (3) *DataConverter*, a converter to transform a format of source data sets into another; and (4) other data structures and classes.

2.3 Recommendation Algorithms

We identify and implement three kinds of recommendation algorithms: (1) baselines that make little use of personalized information; (2) core algorithms that are state-of-the-art approaches based on user-item interactions and contextual information; and (3) other algorithms. A list of enabled recommendation algorithms with references are elaborated at: `http://www.librec.net/tutorial.html`.

Recommendation algorithms can be evaluated in different settings—*Given N (ratio) ratings*, *K-fold cross validation*, *cold start*—to name a few. Three kinds of evaluation metrics are implemented: (1) predictive error-based measures including (normalized) mean absolute error (MAE), root mean square error (RMSE) and mean prediction error (MPE); (2) ranking-based measures including mean average precision (MAP), normalized discounted cumulative gain (NDCG), mean reciprocal rank (MRR), area under the ROC curve (AUC), precision and recall, etc.; and (3) other novel measures: currently a similarity-based diversity measure [3] is implemented. More measures will be added in a future version.

3 Comparison with Other Frameworks

Many open-source libraries are available including Mahout[5], Duine[6], Cofi[7], LensKit[8] MyMediaLite[9] and PREA[10]. Lee et al. [2] provide a detailed comparison among

[3] The comparison is elaborated at `http://www.librec.net/example.html`
[4] Matrix Toolkits Java (MTJ): `https://github.com/fommil/matrix-toolkits-java`
[5] Mahout: `https://mahout.apache.org`
[6] Duine: `http://www.duineframework.org`
[7] Cofi: `http://www.nongnu.org/cofi/`
[8] LensKit: `http://lenskit.org/`
[9] MyMediaLite: `http://www.mymedialite.net`
[10] PREA: `http://prea.gatech.edu`

these different frameworks, and report that Mahout, Duine and Cofi focus only on memory-based algorithms and hence are outdated. LensKit provides only a few classic recommendation algorithms. We give a comparison with more advanced packages, i.e., PREA and MyMediaLite. MyMediaLite is a well-known recommendation library written in C#. Some toolkits, e.g., Rival[11] and WrapRec[12] are recently designed as a wrapper of MyMediaLite for better use (e.g., data split and evaluation). PREA is a more recently released framework implemented in Java. However, the two libraries become less active for further development. The last update of MyMediaLite was September, 2013 and that of PREA was June, 2014. Consequently, some newly proposed algorithms are not supported by these libraries, e.g., TrustMF, FISM and TrustSVD as we do. We also note that graphic recommenders are rarely provided by other libraries. Besides, LibRec provides more baseline and extension algorithms than PREA and MyMediaLite, such as UserCluster and NMF. To evaluate recommendation performance, PREA only provides predictive error-based metrics while MyMediaLite does not provide novel measures beyond accuracy. In contrast, our library provides novel measures as well as the traditional accuracy-based measures. Lastly, we empirically demonstrate that LibRec runs much faster than PREA and MyMediaLite while achieving competitive recommendation performance. The amount of performance gain differs from algorithm to algorithm. A detailed comparison in training and evaluation time can be found at: `http://www.librec.net/example.html`. Another important characteristic of our library is that LibRec configures recommenders using a configuration file, while PREA and MyMydiaLite use command lines. The advantages of a configuration file are (1) easier to configure all possible parameters; (2) portable and easier to reproduce algorithm performance; and (3) easier to debug programs by using alternative parameter settings in one time.

4 Conclusion

LibRec contributes to the community of recommender systems by providing (1) a much faster implementation of a set of recent state-of-the-art recommendation algorithms; (2) a fair and easy comparison among recommendation algorithms in terms of multi-aspect evaluation metrics; and (3) a platform for others to contribute more source codes of other algorithms as an open-source library.

References

1. Ekstrand, M.D., Ludwig, M., Konstan, J.A., Riedl, J.T.: Rethinking the recommender research ecosystem: reproducibility, openness, and lenskit. In: Proceedings of the fifth ACM conference on Recommender systems (RecSys). 133–140 (2011)
2. Lee, J., Sun, M., Lebanon, G.: Prea: Personalized recommendation algorithms toolkit. Journal of Machine Learning Research **13** 2699–2703 (2012)
3. Smyth, B., McClave, P.: Similarity vs. diversity. In: Proceedings of the International Conference on Case-Based Reasoning Research and Development. 347–361 (2001)

[11] `http://rival.recommenders.net`
[12] `https://github.com/babakx/WrapRec`