Journal of Machine Learning Research 1 (2019) 1-48 Submitted 4/00; Published 10/00

Microsoft Recommenders: best practices for building industry-grade recommender system

Andreas Argyriou ANARGYRI@MICROSOFT.COM
Miguel Gonzalez-Fierro MIGONZA@MICROSOFT.COM
Scott Graham SCGRAHAM@MICROSOFT.COM
Nikhil Joglekar NIKHILJ@MICROSOFT.COM
Jun Ki Min JUMIN@MICROSOFT.COM
Jeremy Reynolds JEREMR@MICROSOFT.COM
Tao Wu WUTAO@MICROSOFT.COM
Le Zhang ZHLE@MICROSOFT.COM
Microsoft
One Microsoft Way
Redmond, WA, USA

Editor: editors here

Abstract

This paper talks about the design and implementation of `Microsoft Recommenders`, which is aimed at providing researchers and developers with best-practices for building industry-grade recommendation system at scale. Codes and documentation can be found at https://github.com./microsoft/recommenders.

Keywords: artificial intelligence, recommender system, software engineering

1. Introduction

`Microsoft Recommenders` is an open-source repository created for sharing the best practices in building industry standard recommender system for varieties of application scenarios. The repository aims at helping developers, scientists and researchers to quickly build production-ready recommendation systems as well as to prototype novel ideas using the provided utility functions. The repository is released under MIT license. In this paper the `Microsoft Recommenders` repository is introduced with its design principles, technologies under the hood, and technical advantage over the existing approaches.

2. Microsoft Recommenders

The following sub-sections brief the design thinking of `Microsoft Recommenders`.

2.1 Principles

The `Microsoft Recommenders` repository is designed with the following principles.

1. *The repository covers a wide spectrum of recommender algorithms, including classic methods such as collaborative filtering and factorization machine, as well as more re-*

cent deep learning algorithms with enhanced model accuracy, explainability, and scalability. The wide breadth of recommendation algorithms, as well as the DevOps pipeline that serves as a backbone underneath the codebase, offers convenience to the researchers and developers for trying different algorithms and selecting the optimal one for particular use case.

2. *The algorithms are implemented and optimized for easy adoption and customization.* It is a common yet challenging task to generalize implementations of various components in a recommender system for development efficiency. The repository provides modular and reusable assets with consistent formats, coding style, and standard APIs, to enable ready-for-use scenarios as well as user-customized solutions in a recommender system. In addition, the theoretical discussion and code implementation are put into Jupyter notebooks to provide interactive experience for the developers and have them quickly understand, implement or customize different recommendation algorithms.

3. *The Recommenders repository supports heterogeneous computing platforms and data storage media that meet the requirements of the different workloads in a recommender system pipeline.* To harness the engineering problems in recommender system development, the repository provides reference architectures that demonstrates how to build an enterprise level end-to-end recommendation system.

2.2 Code design

The major deliverables in the repository are `notebooks` and `reco_utils`. Both are implemented in `Python` which is the most popular programming tool to the contemporary data science and machine learning problems.

- `notebooks` are Jupyter notebooks where particular topics like recommender algorithms, data preparation methods, evaluation metrics, etc. are detailed with both text and codes. The advantage of using Jupyter notebook is that it make it convenient to users to interactively walk through the code samples, and to convert to Python scripts if needed.

- `reco_utils` are reusable assets for common tasks in building recommender system. Such tasks include data preparation, evaluation, etc. To favor such operations with scalability consideration, some of the `reco_utils` are implemented in both Python and PySpark versions. The API design of the `reco_utils` follows the software engineering principles like "evidence-based design", "single responsibility", etc., to guarantee consistency, simplicity, and modularity.

The codebase of both `notebooks` and `reco_utils` are built with unit, integration, and smoke tests to assure their appropriate functionality.

2.3 Technologies

`Microsoft Recommenders` supports various platforms for building recommender systems with the cutting-edge recommendation algorithms. Users of the repository can conveniently

choose the desired algorithms for implementing on their supported platforms like single-node CPU computer, Spark cluster, or single-node or multi-node computer(s) with GPU device incorporated. These backend platforms are supported with corresponding programming framworks like PySpark, Tensorflow, etc. (Abadi et al., 2016)

The table below summarizes the recommender algorithms collected in the repository thus far (Ke et al., 2017; Wang et al., 2018; Lian et al., 2018; Howard et al., 2018; He et al., 2017; Salakhutdinov et al., 2007; Cheng et al., 2016; Diev, 2015; Koren et al., 2009) The table also shows the technologies used in the implementation and use case scenario of these algorithms.

Table 1: List of the collected recommender algorithms and their supported technology platforms

Algorithms	**Environment**	**Type**	**Description**
Alternating Least Square	Spark cluster	Collaborative filtering	Matrix factorization algorithm
LightGBM/Gradient Boosting Tree	CPU / Spark cluster	Content-based filtering	Gradient Boosting Tree algorithm for fast training and low memory usage in content-based problems
Deep knowledge-aware network*	CPU / GPU	Content-based filtering	Deep learning algorithm with knowledge graph enhancement
Extreme Deep Factorization Machine (xDeepFM)*	CPU / GPU	Hybrid	Deep learning based algorithm for implicit and explicit feedback with user/item features
FastAI Embedding Dot Bias (FAST)	CPU / GPU	Collaborative filtering	General purpose algorithm with embeddings and biases for users and items
Neural Collaborative filtering (NCF)*	CPU / GPU	Collaborative filtering	Deep learning algorithm with enhanced performance for implicit feedback
Restricted Boltzmann Machines (RBM)*	CPU / GPU	Collaborative filtering	Neural network based algorithm for learning the underlying probability distribution for explicit or implicit feedback
Wide and Deep	CPU / GPU	Hybrid	Deep learning algorithm that can memorize feature interactions and generalize user features
Riemannian Low-rank Matrix Completion (RLRMC)*	CPU	Collaborative filtering	Matrix factorization algorithm using Riemannian conjugate gradients optimization with small memory consumption.
Simple Algorithm for Recommendation (SAR)*	CPU	Collaborative filtering	Similarity-based algorithm for implicit feedback dataset
Surprise/Singular Value Decomposition (SVD)	CPU	Collaborative filtering	Matrix factorization algorithm for predicting explicit rating feedback in datasets that are not very large
Vowpal Wabbit Family (VW)	CPU (online training)	Content-based filtering	Fast online learning algorithms, great for scenarios where user features / context are constantly changing

Recommender algorithms labelled with "*" indicate that they are implemented natively in the repository.

There are a few machine learning and architecture engineering technologies featured in the repository. For instance, hyperparameter tuning plays a vital part in model selection. Intelligent methods for tuning hyper parameters for recommender models, especially those built with deep learning algorithms, are highly desirable. The repository provides the best practices for optimizing model performance with the state-of-the-arts tuning methods like Bayesian optimization, Hyperband, etc. (Pelikan et al., 1999; Li et al., 2016) To avail building industry-grade deployable recommender system, the repository demonstrated reference architecture of building scalable recommender system [1], by using distributed database, scalable container cluster on Kubernetes, etc.

3. Comparison

Compared to the existing packages in the open source community, `Microsoft Recommenders`, ...

4. Conclusion

In this paper, we present `Microsoft Recommenders` repository, which provides a wide collection of the classic and advanced recommender algorithms, as well as useful code modules for building enterprise-grade recommender system. The best practices shared in the repository help researchers and developers to build a whole end-to-end recommender system at scale.

Acknowledgments

We would like to acknowledge all the contributors to the repository. The complete name list of the contributors can be found here

1. The reference architecture is demonstrated on Microsoft Azure cloud platform.

References

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*, pages 7–10. ACM, 2016.

V. Diev. Sar: Smart adaptative recommendation algorithm. In *Machine Learning and Data Science Conference (MLADS) Redmond 2015*, 2015.

Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web*, pages 173–182. International World Wide Web Conferences Steering Committee, 2017.

Jeremy Howard et al. fastai. https://github.com/fastai/fastai, 2018.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, pages 3146–3154, 2017.

Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, (8):30–37, 2009.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *arXiv preprint arXiv:1603.06560*, 2016.

Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, and Guangzhong Sun. xdeepfm: Combining explicit and implicit feature interactions for recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1754–1763. ACM, 2018.

Martin Pelikan, David E Goldberg, and Erick Cantú-Paz. Boa: The bayesian optimization algorithm. In *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 1*, pages 525–532. Morgan Kaufmann Publishers Inc., 1999.

Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th international conference on Machine learning*, pages 791–798. ACM, 2007.

Hongwei Wang, Fuzheng Zhang, Xing Xie, and Minyi Guo. Dkn: Deep knowledge-aware network for news recommendation. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, pages 1835–1844. International World Wide Web Conferences Steering Committee, 2018.