Microsoft Recommenders: Best Practices for Production-Ready Recommendation Systems

Andreas Argyriou
anargyri@microsoft.com
Microsoft, UK

Miguel González-Fierro
migonza@microsoft.com
Microsoft, UK

Le Zhang
zhle@microsoft.com
Microsoft, Singapore

ABSTRACT

Recommendation algorithms have been widely applied in various contemporary business areas, however the process of implementing them in production systems is complex and has to address significant challenges. We present *Microsoft Recommenders*, an open-source Github repository for helping researchers, developers and non-experts in general to prototype, experiment with and bring to production both classic and state-of-the-art recommendation algorithms. A focus of this repository is on *best practices* in development of recommendation systems. We have also incorporated learnings from our experience with recommendation systems in production, in order to enhance ease of use; speed of implementation and deployment; scalability and performance.

CCS CONCEPTS

• **Information systems → Recommender systems**.

KEYWORDS

Recommender systems; Algorithms; Libraries

ACM Reference Format:
Andreas Argyriou, Miguel González-Fierro, and Le Zhang. 2020. Microsoft Recommenders: Best Practices for Production-Ready Recommendation Systems. In *Proceedings of The Web Conference 2020 (WWW '20), April 20–24, 2020, Taipei, Taiwan.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3366424.3382692

1 INTRODUCTION

Recommendation systems have become ubiquitous in modern business, examples of which include recommendation of brands, news, consumer products, media content, travel packages and many others. In practice, however, application of recommendation algorithms has encountered significant challenges. First, there are *limited references and guidance* for building recommendation systems at scale to support enterprise grade scenarios. In addition, even though there exist several off-the-shelf packages, they tend to focus on specific aspects of recommendations, and may *not be compatible* with each other. On the other side, researchers, when applying their methods to real world scenarios, may lack experience of the domain of interest or awareness of the best practices with respect to data science and software engineering. Thus, it can be *time-consuming* to build a new recommendation system from an algorithm even when it is available as software, or to incorporate the algorithm into an end-to-end pipeline.

In response to these challenges, we developed the recommendation best-practice hub, `Microsoft Recommenders`, which is open-sourced on GitHub as a repository.[1] At the time of writing, the repository is the most popular one for recommendation systems on GitHub, with ~ 6800 stars.

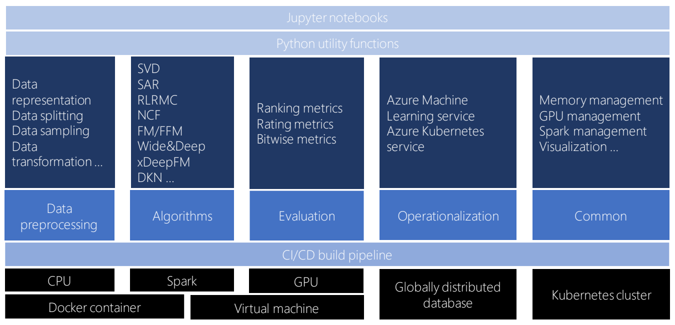

Figure 1: Microsoft Recommenders structure diagram.

2 OVERVIEW OF RECOMMENDERS

The repository is open-sourced under the MIT License. All the algorithms and utilities have been written in the Python programming language and the demonstrations are in the form of Jupyter notebooks. The platforms supported are Linux and Windows. Some of the algorithms can take advantage of GPU resources if available and some others require a Spark environment. In Figure 1 a diagram of the repository structure is shown.

2.1 Recommenders Utilities

Utilities are functions designed as the minimum layer needed for supporting common recommendation pipelines. Some utility functions are designed to support atomic operations, e.g., the utilities invoking opensource libraries while some others are meant for full implementation of fully functional components, such as recommender algorithms. Utility functions can be generally categorized as follows:

- **Common utilities:** supporting utilities like timers, loggers, GPU functions, Spark helpers, memory management, etc.
- **Data preparation:** Utility functions for data download, loading, transformation, splitting, etc.
- **Algorithms:** algorithmic implementations and/or their auxiliary functions.
- **Evaluation:** rating and ranking metrics implemented in Python+CPU and PySpark environments.

ACM ISBN 978-1-4503-7023-3/20/04.
https://doi.org/10.1145/3366424.3382692

[1]https://github.com/Microsoft/Recommenders

- **Model Selection:** utility functions developed using some hyperparameter tuning frameworks.
- **Operationalization:** model operationalization functions on platforms like Kubernetes.

2.2 Recommenders Notebooks

The second main component of Microsoft Recommenders are the Jupyter notebooks. They are classified in the following categories:

- **Quick start notebooks:** The quick start notebooks are examples of recommendation algorithms that can be run in less than 15 minutes.
- **Deep dive notebooks:** These notebooks contain a more detailed description of a system, summarizing the main mathematical development and code.
- **General notebooks:** They cover general use cases like data preparation, evaluation, hyperparameter tuning or operationalization.

3 BEST PRACTICES IN RECOMMENDATION SYSTEMS

The best practices followed in Microsoft Recommenders lie in five folds. First, the repository is designed to provide *guidance* on algorithm selection under predefined business or technical circumstances. The notebooks in the repository cover scenarios such as *news recommendations*, *sequence-based recommendations*, *item-to-item recommendations* or *recommendations based on knowledge graph*, etc.

Second, code design of the repository adopts an *evidence-based software design* methodology [3]. The evidence is gathered from extensive experience working with customers in real life recommendation scenarios. When there is no input from a customer, the second source of evidence that is used is popular machine learning libraries like Tensorflow [1], PyTorch [4] or LightGBM [2].

Third, the repository can serve as a tool for *scientific reproducibility* when a novel algorithm is designed and tested. In particular, we provide utilities for tasks which are common and repetitive when building recommendations pipelines, like data preparation with standard methods of data splitting; model training and prediction using an interface similar to Scikit-Learn [5], with `fit()` and `predict()` methods; ranking and rating evaluation methods; hyperparameter tunning and operationalization.

Fourth, we created an extensive test pipeline, with *unit tests* executed every time there is a pull request to the repository, and *smoke and integration tests*[2] that are run every night to test the overall robustness of the code. At the time of writing, there are over 400 tests, between Windows and Linux.

And finally, the repository offers a *space for collaboration* for the machine learning community to make joint contributions in the area of recommendations.

4 LESSONS LEARNED FROM BUILDING REAL-WORLD SYSTEMS

Productionizing an industry-grade recommender system for real-world problems is often challenging in various ways. For example, *model retraining* on a daily basis is often crucial in applications like e-commerce where the volume and dynamics of training data can change rapidly. In some cases, due to the size of the data or effects like concept drift, *online training* may be the best option. *Hyperparameters* of a recommender model, especially in content-based recommendation scenarios, need to be fine-tuned properly in order to ensure that the model delivers good performance. *Real-time serving* of recommendations should meet engineering specifications determined from business requirements, e.g. latencies smaller than 100 ms. An end-to-end recommendation pipeline may consist of *heterogeneous computing platforms* (distributed computing cluster, GPU etc.) where computing resources should be used effectively and economically. The *health status* of the recommendation system should be well monitored with an online mechanism so that break points of the entire pipeline can be detected. *Offline evaluation* is often used for selection and tuning of algorithms, while it is always recommended to perform *online evaluation* by using A/B testing and/or multi-armed bandits; a frequent issue however is that offline and online evaluation may not correlate well. Finally, a recommender system may be restricted by *business rules* (such as country-specific legislation, seen items allowed to be recommended only a certain amount of times, seasonal restrictions etc.) to make the recommendation results comply. All these lessons are embedded and shared publicly in the repository.

5 CONCLUSION

Microsoft Recommenders centralizes best practices and makes the processes of doing research or data science on recommendation systems more productive. In several projects involving recommendation systems in enterprise settings, we have observed significant productivity gains by using the repository, which amounted to speeding up the implementation time of a new recommendation system by 10 times or more than before. Furthermore, Microsoft Recommenders addresses one of the most frequent challenges, that is, integrating algorithms into complete end-to-end pipelines for recommendations either in an industrial or an academic setting.

ACKNOWLEDGMENTS

We would like to thank all the contributors to the repository listed at https://github.com/microsoft/recommenders/graphs/contributors.

REFERENCES

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

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

[3] Steven W Kembel, Evan Jones, Jeff Kline, Dale Northcutt, Jason Stenson, Ann M Womack, Brendan JM Bohannan, GZ Brown, and Jessica L Green. 2012. Architectural design influences the diversity and structure of the built environment microbiome. *The ISME journal* 6, 8 (2012), 1469.

[4] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. (2017).

[5] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *Journal of machine learning research* 12, Oct (2011), 2825–2830.

[2]https://github.com/Microsoft/Recommenders/tree/master/tests