Journal of Machine Learning Research 1 (2000) 1-48 Submitted 4/00; Published 10/00

Microsoft/Recommenders: Best practices for building industry-grade recommender system

Le Zhang YOUR@EMAIL.COM
One Microsoft Way
Redmond, WA 98052, USA

Jun-Ki Min JUN.MIN@MICROSOFT.COM
1 Memorial Dr
Cambridge, MA 02142, USA

Authors YOUR EMAIL
your address line 1
your address line 2
your address line 3

Editor: editors here

Abstract

This paper talks about the design philosophy and implementation methodology of Microsoft/Recommenders, which is aimed at providing researchers and developers with best-practices for building industry-grade recommendation system at scale.

Keywords: artificial intelligence, recommender system, software engineering

1. Introduction

Recent decades have witnessed a great proliferation of recommendation systems. The technology has brought significant profits to many business verticals. From earlier algorithms such as similarity based collaborative filtering to the latest deep neural network based methods, recommendation technologies have evolved dramatically, which, to some extent, makes it challenging to practitioners to select and customize the optimal algorithms for a specific business scenario. In addition, operations such as data preprocessing, model evaluation, system operationalization etc. play a significant role in the lifecycle of developing a recommendation system; however, they are often neglected by practitioners. Based on extensive experience in productization of recommendation systems in a variety of real-world application domains, an open source GitHub repository, `Microsoft/Recommenders` was 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.

In this paper we introduce Microsoft/Recommenders by describing the general principles, models and topics covered by the repository which includes implementation of modern state-of-the-art recommendation algorithms as well as more practical topics such as hyper-parameter tuning and operationalization.

2. Microsoft/Recommenders

2.1 General principles of the repository

1. *The repository covers a wide spectrum of recommender algorithms. It includes classic methods such as collaborative filtering and factorization machine, as well as more recent deep learning algorithms with enhanced model accuracy, interpretability, 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 as utility functions (e.g. temporal or stratified split between training and testing data) 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. When necessary, the notebooks can be conveniently converted to production-ready codes with minimal efforts on refactoring.

3. *The Recommenders repository avails heterogeneous computing platforms and data storage media on cloud that meet the requirements of the different workloads in a recommender system pipeline.* A commonly seen problem that arises in recommender system design is that the backend data processing pipeline and model building/serving pipeline should be architected under certain set of engineering constraints to satisfy the requirements of the frontend applications. To harness the complexity in system design, the Recommenders repository provides reference architectures that demonstrates how to build an enterprise level end-to-end recommendation system. For example, the reference architecture for a real-time recommender system shows data ingestion, data storage, model building, model deployment and model consumption by using various technologies like Spark, distributed database and Kubernetes on the cloud.

4. *The repository provides an open, transparent, and collaborative platform for academic and open-source community for contributing and testing their own algorithms.* A major design principle of the repository is openness, to facilitate collaborations among researchers and developers in the field. More than 20 people contributed to the repository [4]. This is made available by the reusable and extensible utility functions. The recommenders repository became a well-recognized resource to acquire knowledge about recommendation engine beyond the basics. As of early May 2019, the repository has more than 3000 stars on Github and is forked close to 400 times. It was also covered and recognized by Hacker News, KDNuggets and O'Reilly Data Newsletter

2.2 Data preparation and model evaluation

2.2.1 Data preparation

2.2.2 Model evaluation

2.2.3 Offline evaluation and metrics

2.2.4 Online evaluation

- A/B testing
- Multi-armed bandit

2.3 Modeling and optimization

2.3.1 Model deep dives

We covers algorithm-level details about each recommender model so that users can select a model that fits to their problem.

2.3.2 Hyper-parameter tuning

2.4 Operationalization

2.4.1 Large-scale model building and serving

2.4.2 Distributed computing platform and database

2.4.3 Containerization for real-time serving

Remainder omitted in this sample. See http://www.jmlr.org/papers/ for full paper.

3. Background

This section lists recommender algorithms covered by Recommenders repository as of May 2019.

3.1 Classic algorithms

3.1.1 Collaborative filtering algorithms

- Memory-based algorithms
  - Similarity-based models, Simple Algorithm for Recommendation (SAR), etc.
  - Implementation challenges
- Matrix factorization
  - singular value decomposition
  - alternate leasting square
  - nerual collaborative filtering

3.1.2 Content-based filtering algorithms

- logistic regression
- gradient boosting techniques
- factorization machine

3.2 Emerging approaches

3.2.1 Deep learning applications

3.2.2 Enhancement with heterogeneous information

Acknowledgments

We would like to acknowledge support for this project from the National Science Foundation (NSF grant IIS-9988642) and the Multidisciplinary Research Program of the Department of Defense (MURI N00014-00-1-0637).

Appendix A.

In this appendix we prove the following theorem from Section 6.2:
Theorem *Let u, v, w be discrete variables such that v, w do not co-occur with u (i.e., $u \neq 0 \Rightarrow v = w = 0$ in a given dataset $\mathcal{D}$). Let N_{v0}, N_{w0} be the number of data points for which $v = 0, w = 0$ respectively, and let I_{uv}, I_{uw} be the respective empirical mutual information values based on the sample $\mathcal{D}$. Then*

$$N_{v0} > N_{w0} \;\Rightarrow\; I_{uv} \leq I_{uw}$$

with equality only if u is identically 0. ■
Proof. We use the notation:

$$P_v(i) = \frac{N_v^i}{N}, \quad i \neq 0; \quad P_{v0} \equiv P_v(0) = 1 - \sum_{i \neq 0} P_v(i).$$

These values represent the (empirical) probabilities of v taking value $i \neq 0$ and 0 respectively. Entropies will be denoted by H. We aim to show that $\frac{\partial I_{uv}}{\partial P_{v0}} < 0$....

Remainder omitted in this sample. See http://www.jmlr.org/papers/ for full paper.