Algorithm selection with `librec-auto`

Masoud Mansoury[1] and Robin Burke[2]

[1] DePaul University, Chicago IL, USA
mmansou4@depaul.edu
[2] University of Colorado, Boulder, Boulder CO, USA
robin.burke@colorado.edu

Abstract. Due to the complexity of recommendation algorithms, experimentation on recommender systems has become a challenging task. Current recommendation algorithms, while powerful, involve large numbers of hyperparameters. Tuning hyperparameters for finding the best recommendation outcome often requires execution of large numbers of algorithmic experiments particularly when multiples evaluation metrics are considered. Existing recommender systems platforms fail to provide a basis for systematic experimentation of this type. In this paper, we describe `librec-auto`, a wrapper for the well-known LibRec library, which provides an environment that supports automated experimentation.

1 Introduction

A recommender system aims to predict users' preferences and suggests desirable items to them. Due to their power and effectiveness in generating personalized recommendations and consequently increasing companies' profit, such systems have become an essential tool in electronic commerce systems. There are a wide variety of real world examples like video recommendation in Youtube, item recommendation in Amazon, and music recommendation in Pandora. In all these applications, specific recommendation algorithms are employed for generating recommendations to users.

Research on recommender systems has expanded to a wide variety of topics as new problems emerged. This expansion introduced new evaluation criteria for measuring the performance of recommender systems and made recommendation algorithms, while accurate, very complicated, computationally expensive, and sensitive to parameter values. Understanding the nature of these algorithms requires extensive experiments over combination sets of parameters, multiple data sets, and different metrics.

Depending on the characteristics of the data set and evaluation criteria, specific recommendation algorithm will be needed and assignments for its parameters may vary [1]. Some algorithms may work well on dense data sets, some may work well on binary data sets, and some may result in better recommendations on contextual data sets. Thus, performing extensive experiments is required to find the best algorithm. Also, evaluation criteria are another factor in experimentation. In addition to accuracy, which is the most important metric for achieving

The 1st Interdisciplinary Workshop on Algorithm Selection and Meta-Learning in Information Retrieval (AMIR), 14 April 2019, Cologne, Germany. Editors: Joeran Beel and Lars Kotthoff. Co-located with the 41st European Conference on Information Retrieval (ECIR). http://amir-workshop.org/

personalization, non-accuracy measures such as diversity, novelty, fairness, and coverage have become recognized as important measures of recommendation effectiveness. Since some of these metrics are in conflict with each other, extensive experiments are required to achieve a trade-off between them.

Hyperparameters play an important role on the performance of many recommendation algorithms, requiring careful tuning. For example, the integrated neighborhood and singular value decomposition model[3] [9] involves seven hyperparameters (i.e., bias regularization, user bias regularization, item bias regularization, implicit feedback bias regularization, learning rate, maximum number of iterations, and number of factors). Simple calculation shows that, for instance, exploring 5 possible values for each parameter on a single data set will entail running almost 80,000 different experiments. Manual configuration of so many experiments would be tedious and error-prone, then motivating the design of a system that can significantly reduce the configuration and setup time for each experiment.

Moreover, reproducibility of previous experiments is key for conducting further analysis in the future. Doing this, it is necessary to save all elements of each experiment and retrieve them in the future. This is very useful particularly when analyzing recommendation results with different metrics is needed.

However, while a number of recommendation platforms have been developed for recommender systems researchers: LibRec [7], Surprise [8], MyMediaLite [6], LensKit [4], Mahout [11], pyRecLab [12], LKPy [3] and others, none of these tools supports automated experimentation. LibRec is well-known for its large library of implemented recommendation algorithms (more than 70 as of this writing), its attention to efficiency, and its ability to evaluate algorithms relative to a variety of tasks and metrics. LibRec works well for single-shot algorithm and data set evaluation, but it is not well-suited for reproducible, automated, large-scale experimentation.

We have implemented `librec-auto`, a Python-based wrapper that encapsulates LibRec and supports the automation of repetitive experiments. In [10], we presented a beta version of `librec-auto` along with a demo. In this paper, we present the functionalities of `librec-auto` in detail and introduce several new features implemented in the most recent version. We also comprehensively compare `librec-auto` with well-known existing recommendation tools and discuss its advantages over other tools.

2 Some existing recommendation tools

In this section, we introduce some well-known recommendation tools for recommender systems research and discuss their main features along with their advantages and disadvantages.

[3] SVD++ in Librec.

2.1 Librec

LibRec 2.0 [7] is a mature, flexible, open-source Java-based platform for recommender systems experimentation.[4]. It supports a large variety of recommendation algorithms and a large library of evaluation metrics, supported by an active community of developers and maintainers.

However, LibRec does not naturally support large-scale experimentation. Similar to other recommender systems platforms, LibRec is implemented as a chained series of operations: splitting testing and training data, training an algorithm, and executing the algorithm and evaluating results.

This monolithic design creates several problems for researchers. One is that intermediate operations are not saved and repeating similar experiments entails redundant computation. It is also the case that training/testing splits are not saved when LibRec computes them. Running with the same random seed guarantees the same split will be computed, but because there is no explicit storage of the training data, it is difficult for a researcher to perform detailed analysis and debugging where knowledge of the specific input data of each fold would be helpful.

Finally, LibRec uses a simple key-value configuration system based on the Java Properties class. The flat structure requires a multi-part key-naming scheme, which places significant cognitive load on the experimenter to remember the meaning of each key and the appropriate set of keys required for each algorithm.

2.2 MyMediaLite

MyMediaLite 3.11 [6] is an open-source C# platform[5] for recommender systems experimentation[6]. It supports both ranking and rating prediction tasks and provides a wide range of recommendation algorithms for running experiments. It also has useful functionalities like grid search for optimizing hyperparameters and model saving .

Even though MyMediaLite saves computed models, it does not save the intermediate files and outputs. Also, it does not have a configuration file for specifying the hyperparameters for its recommendation algorithms.

2.3 Other recommendation tools

There are a wide range of other platforms for experimentations on recommender systems: Mahout [11], Surprise[7] [8], Lenskit [8] [5], pyRecLab[9] [12], LKPy[10] [3] and others.

[4] Source code available at github.com/guoguibing/librec
[5] It has Perl code as well.
[6] http://www.mymedialite.net/index.html
[7] Source code available at https://github.com/NicolasHug/Surprise
[8] Source code available at https://github.com/lenskit/lenskit
[9] Source code available at https://github.com/gasevi/pyreclab
[10] Source code available at https://github.com/lenskit/lkpy

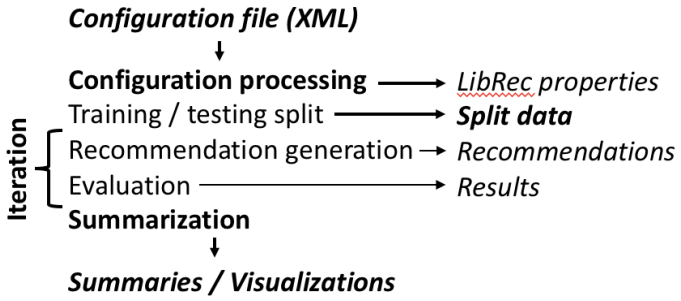

Fig. 1. Operation flow of `librec-auto`. Bold elements are added to basic LibRec functionality.

These platforms aim to provide a comprehensive tools for practitioners and researchers for experimenting and developing recommendation algorithms, but none provide tools for large-scale automated experimentation with a large library of state-of-the-art algorithms.

3 Librec-auto

`librec-auto` is a wrapper built around the core LibRec functionality to enhance the platform's ability to support reproducible large-scale experiments. The tool is currently under development with additional features planned. The core LibRec library is used unmodified, with some additional Java functionality implemented in a separate set of Java classes.

Figure 1 shows the operation of `librec-auto` and its relationship to the core LibRec system. New elements of the system are indicated in bold font. As shown, the system reads an XML configuration file, which is converted into the properties file (or files) required by LibRec. The data from each training / split is saved to that future experiments can share a common experimental configuration. Unlike in core LibRec, recommendation generation and evaluation are separate steps, so that previously-generated recommendation results can be reused where possible.

Another key element of the `librec-auto` implementation is the ability (described below) to iterate over different parameter values in one scripted experiment. The system performs all of the necessary bookkeeping and allows the user to write scripts that summarize over all of experiments, including the creation of visualizations.

3.1 Flexible configuration

LibRec uses a simple unstructured key-value properties file to define the parameters related to an experimental configuration. The file format does not capture the relationships between parameters and it can be difficult to determine, for example, what parameters are appropriate for which algorithms. `librec-auto` uses

an XML-based configuration file, with greater self-descriptive power, and the ability to perform error and consistency checking. The properties files required by LibRec are produced and managed by `librec-auto`, making the process less tedious, especially when multiple, similar, experiments are being performed.

Figure 2 shows a fragment of one such XML configuration file, showing how the element labels and embedding structure clarify the experimenter's intent. This part of the file defines an algorithm to be tested, in this case ItemKNN, with its configuration settings – the number of neighbors to be used and the similarity metric. The results are to be computed in recommendation lists of size 10 and evaluated by the nDCG measure. Note that multiple values are listed for `neighborhood-size`. The system will therefore run an experiment for each given value.

3.2 Installation

As usual for Python programs, `librec-auto` provides pip style installation. It makes the installation process easy and provides access to code everywhere for running experiments.

On a machine equipped with Python and Java and with Librec and `librec-auto` installed, the next step is to create a configuration file. The configuration specifies the data to be operated on, the type of evaluation to be performed, the algorithms to be run, and other operations.

3.3 Automation

As the example shows, one of the important capabilities of `librec-auto` is its ability to automate multiple experiments across algorithmic hyperparameters, supporting tuning and sensitivity analysis. This type of functionality will be familiar to users of scikit-learn's `GridSearchCV` model selection tool [2].

For each combination of parameters, the system creates a separate subdirectory with its own configuration and output files. A separate instance of LibRec runs on each configuration, allowing for parallel operation.

3.4 Multi-threading

Some existing tools (for example, Librec and LKPy) allow parallel operation of individual experiments by computing folds of a cross-fold validation experiment separately. This is sufficient when single experiments are performed. `librec-auto`, on the other hand, has the capability of running multiple experiments in parallel. This allows for parallelization efficiencies for a wider range of experiment types.

3.5 Intermediate results

Because recommendation algorithms can be computationally expensive at scale, `librec-auto` is designed to help minimize wasted computational effort. As much as possible, the intermediate results are stored, including training/test splits and algorithm outputs, as noted above.

```
<alg>
    <class>ItemKNNRecommender</class>
    <similarity>cos</similarity>
    <neighborhood-size>
        <value>10</value><value>20</value>
    </neighborhood-size>
</alg>
<eval>
    <metric>NDCG</metric>
    <list-size>10</list-size>
</eval>
```

Fig. 2. Fragment of `librec-auto` XML configuration file

Because intermediate outputs are recorded, it is possible in `librec-auto` to execute multiple evaluation metrics on a single set of experimental results. LibRec allows the computation of multiple metrics when the experiment is being run, but applying a different metric after the fact requires starting over, with potentially significant computational cost. Note that error-oriented and ranking-oriented experiments are not compatible: the results computed for a ranking metric are recommendation lists and can be evaluated with ranking metrics such as precision. `librec-auto` is aware of this distinction and can warn the user if there is an attempt to apply an metric incompatible with the computed results.

3.6 Summarization

The multiplicity of experiments that `librec-auto` can perform can yield a large amount of data that researchers must collate and examine. It would be tedious to have to pore over LibRec's log files to attempt to discover the differences between different parameter settings.

To make such interpretative tasks more efficient, `librec-auto` allows for a final summarization phase to follow the completion of a set of experiments. This is effectively one or more scripts that the experimenter can write to process the log file data into a useful format. Using plotting libraries such as `matplotlib`, it is possible to create and save visualizations that the researcher can quickly scan.

4 Conclusion

In this paper and our associated demo, we presented `librec-auto`, a tool for automating recommender systems experimentation using the LibRec 2.0 platform. Our tool retains the benefits of working with LibRec while making large-scale experimentation easier, more efficient, and more reproducible. Some of important features of `librec-auto` presented in this paper are: *flexible configuration*, *installation*, *automation*, *multi-threading*, *intermediate results*, and *summarization*.

References

1. Adomavicius, G., Zhang, J.: Impact of data characteristics on recommender systems performance. ACM Transactions on Management Information Systems (TMIS) **3**(3) (2012)
2. Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., VanderPlas, J., Joly, A., Holt, B., Varoquaux, G.: API design for machine learning software: experiences from the scikit-learn project. In: ECML PKDD Workshop: Languages for Data Mining and Machine Learning. pp. 108–122 (2013)
3. Ekstrand, M.D.: The lkpy package for recommender systems experiments: Next-generation tools and lessons learned from the lenskit project. arXiv preprint arXiv:1809.03125 (2018)
4. Ekstrand, M.D., Ludwig, M., Kolb, J., Riedl, J.T.: Lenskit: a modular recommender framework. In: Proceedings of the fifth ACM conference on Recommender systems. pp. 349–350 (2011)
5. Ekstrand, M.D., Ludwig, M., Kolb, J., Riedl, J.T.: Lenskit: a modular recommender framework. In: RecSys '11 Proceedings of the fifth ACM conference on Recommender systems. pp. 349–350 (2011)
6. Gantner, Z., Rendle, S., Freudenthaler, C., Schmidt-Thieme, L.: Mymedialite: a free recommender system library. In: Proceedings of the fifth ACM conference on Recommender systems. pp. 305–308 (2011)
7. Guo, G., Zhang, J., Sun, Z., Yorke-Smith, N.: Librec: A java library for recommender systems. In: UMAP Workshops (2015)
8. Hug, N.: Surprise, a python library for recommender systems. In: URL: http://surpriselib.com (2017)
9. Koren, Y.: Factorization meets the neighborhood: a multifaceted collaborative filtering model. In: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 426–434. ACM (2008)
10. Mansoury, M., Burke, R., Ordonez-Gauger, A., Sepulveda, X.: Automating recommender systems experimentation with librec-auto. In: Proceedings of the 12th ACM Conference on Recommender Systems. pp. 500–501 (2018)
11. Seminario, C.E., Wilson, D.C.: Case study evaluation of mahout as a recommender platform. In: RUE@ RecSys. pp. 45–50 (2012)
12. Sepulveda, G., Dominguez, V., Parra, D.: pyreclab: A software library for quick prototyping of recommender systems. In: ACM RecSys Poster (2017)