libreс-auto: A Tool for Recommender Systems Experimentation

Nasim Sonboli
nasim.sonboli@colorado.edu
University of Colorado, Boulder
Boulder, Colorado, USA

Masoud Mansoury
m.mansoury@tue.nl
Eindhoven University of Technology
Netherlands

Ziyue Guo
Ziyue.Guo@colorado.edu
University of Colorado, Boulder
Boulder, Colorado, USA

Shreyas Kadekodi
Shreyas.Kadekodi@colorado.edu
University of Colorado, Boulder
Boulder, Colorado, USA

Weiwen Liu
wwliu@cse.cuhk.edu.hk
The Chinese University of Hong Kong
Shatin, Hong Kong, China

Zijun Liu
zl3031@columbia.edu
Columbia University
New York, New York, USA

Andrew Schwartz
andrew.t.schwartz@colorado.edu
University of Colorado, Boulder
Boulder, Colorado, USA

Robin Burke
Robin.Burke@colorado.edu
University of Colorado, Boulder
Boulder, Colorado, USA

ABSTRACT

Recommender systems are complex. They integrate the individual needs of users with the characteristics of particular domains of application which may span items from large and potentially heterogeneous collections. Extensive experimentation is required to understand the multidimensional properties of recommendation algorithms and the fit between algorithm and application. `librec-auto` is a tool that automates many aspects of off-line batch recommender system experimentation. It has a large library of state-of-the-art and historical recommendation algorithms and a wide variety of evaluation metrics. It further supports the study of diversity and fairness in recommendation through the integration of re-ranking algorithms and fairness-aware metrics. It supports declarative configuration for reproducible experiment management and supports multiple forms of hyperparameter optimization.

CCS CONCEPTS

• **Information systems → Recommender systems**; **Relevance assessment**; • **Human-centered computing → Collaborative filtering**; • **Social and professional topics → User characteristics**; • **Software and its engineering →** *Software libraries and repositories.*

KEYWORDS

Recommender systems, Reproducibility, Software tools, Algorithmic fairness

ACM Reference Format:
Nasim Sonboli, Masoud Mansoury, Ziyue Guo, Shreyas Kadekodi, Weiwen Liu, Zijun Liu, Andrew Schwartz, and Robin Burke. 2021. librec-auto: A Tool for Recommender Systems Experimentation. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management (CIKM '21), November 1–5, 2021, Virtual Event, QLD, Australia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3459637.3482006

1 INTRODUCTION

`librec-auto` is a Python-based tool for recommender systems experimentation[1]. It is designed to support researchers by automating experimental methodology, thereby making experimental results more reproducible, and allowing simplified access to the large collection of algorithm implementations found in the LibRec platform. The system supports off-line batch evaluation in which ratings data (either explicit or implicit) are divided into training and test sets and the algorithm is evaluated on test set performance. Although other evaluation methods are gaining currency, particularly those that support reinforcement learning [25–27], this type of evaluation is still extremely common in research practice and in educational settings.

1.1 Reproducibility in Recommender Systems Research

Reproducibility remains a challenge in recommender systems research [7, 38, 45], particularly in the areas that are not well-established such as fairness-aware recommendation. Even minor differences in parameters and experimental settings can yield incompatible results, which make it difficult to provide definitive answers about the relative properties of different algorithms. Additionally, many of the existing libraries are not comprehensive. They mostly support experimentation for a very specific class of algorithms, making it difficult to readily re-use their capabilities for general research purposes.

We believe that progress towards reproducibility is best supported by providing a platform on which comparative experiments can be conducted using declarative experimental configuration (so that experimental settings can be easily shared), with pre-implemented methodological workflows, and with a large library of algorithms for easy of benchmarking against prior work. `librec-auto` is such a platform and it has been steadily enhanced over multiple years to meet the needs of researchers.

[1]https://librec-auto.readthedocs.io/en/latest/quickstart.html

1.2 Contributions

This paper represents the first full discussion of `librec-auto`. The system was introduced through a demo at the ACM Recommender Systems conference in 2018 [32] and also presented to the audience of the ACM Conference on Fairness, Accountability and Transparency as a tutorial in 2020 [42]. In this paper, we provide a detailed account of the capabilities of `librec-auto` in its current 0.2 release, the typical usage of the system, and description of plans for future releases.

2 LIBREC AND LIBREC-AUTO

`librec-auto` was originally designed as an open-source command-line Python package providing a wrapper for the LibRec recommender systems algorithm library[2]. LibRec is a Java-based recommendation generation platform, available under the GPL 3.0 open-source license. It was introduced in 2015 [22], and is maintained by a group led by Professor Guibing Guo at Northeastern University, Shenyang, China. The group has implemented a large library of recommendation algorithms including deep learning algorithms (more than 70 as of this writing) drawn from the recommender systems research literature. In addition to numerous algorithms, the LibRec platform supports a variety of evaluation metrics and evaluation methodologies.

Despite the significant capabilities of LibRec, we found that for practical experimentation and reproducibility research, the system by itself is not sufficient. For example, intermediate computational outputs, such as recommendation results, cannot be reused as input for new evaluation metrics. If an experimenter runs an experiment and then wishes to follow-up and explore additional properties of the results, LibRec would require re-execution of the entire set of recommendation computations. In addition, result re-ranking is not supported because of the close tie between algorithm execution and result evaluation built into LibRec. This makes it difficult to use LibRec as it is to explore the areas of fairness-aware and diversity-enhanced recommendation.

`librec-auto` was conceived and implemented as a wrapper for core aspects of LibRec's functionality, in particular, to control and organize inputs (experiment configuration) and outputs (predictions and metric calculations), while preserving an experimenter's ability to access the large collection of implemented algorithms and metrics found in LibRec. Based on our internal research needs, we have extended this platform in a number of ways, particularly to support research in fairness-aware recommendation. We have also enhanced LibRec with a suite of metrics for measuring the fairness of recommendation outcomes. In addition, the tool now supports recommendation re-ranking, a common approach to enhancing fairness, diversity, and other non-accuracy properties of recommendation outcomes.

Except for a small chunk of Java wrapper code, `librec-auto` is implemented in Python and (because it inherits heavily from LibRec) is available under the same GPL 3.0 license. It can be installed using the `pip` package manager or by downloading the source package from GitHub[3]. Because of its ease of configuration and use, `librec-auto` also supports students and teachers in learning about recommendation algorithms and their evaluation. The system has been used for courses at the Technical University of Eindhoven and the University of Colorado, Boulder.

[2]https://guoguibing.github.io/librec/index.html
[3]https://github.com/that-recsys-lab/librec-auto

3 KEY FEATURES

A sketch of the functionality of `librec-auto` is provided in Figure 1. As the figure indicates, LibRec is encapsulated by `librec-auto`, and its various component elements are used to execute particular portions of the experimental workflow. The user experience of the system is organized around the concept of a *study*, which involves experimentation with a single data set and a single recommendation algorithm. Multiple hyperparameters may be investigated as part of a study, either through grid search or black-box optimization as described in Section 3.3. Each combination of hyperparameters is evaluated through an *experiment*, which may entail multiple training / test iterations if cross-validation is used. Although the figure indicates a straight-line of execution, parallelism is built into `librec-auto` at the level of experiment execution. Because experiments can have lengthy execution times, the post-processing phase allows for integration with messaging platforms, including Slack, so that experimenters are notified when their tasks are complete. These messages can include visualizations of experimental output, to provide a quick overview of results.

We can follow the workflow of a study through the processes depicted in Figure 1.

XML configuration The experimenter designs a study and crafts a configuration file. The `librec-auto` release includes a simple set-up wizard to automate some aspects of this process.

Configuration processing The system processes the configuration file and produces internal data sources necessary for the study. These include folders for the experiments to be run, experiment-specific configuration files, and a translation of the LibRec-specific parts of the configuration into the key-value properties file that LibRec consumes.

CV split The ratings data file is processed, and cross-validation train / test splits are produced. Ratings data is assumed to be in the form of a CSV file of sparse $\langle user, item, rating \rangle$ triples. Unlike vanilla LibRec, split files are saved, so that they can be used for further analysis, calculation of training data properties, for example, or for use by non-LibRec recommendation libraries, a feature under active development.

Algorithm The configured recommendation generation algorithm from LibRec is run, producing predictions over the training data. Depending on the nature of the evaluation metric, it will either generate a prediction for each training data triple or it will generate a ranked list of recommendations for each user in the training data.

Re-ranking (optional) If the study uses a re-ranking algorithm, the output of the (list-generating) recommender is input to this algorithm, and a new list of re-ranked results is produced for each user. The system includes a set of re-ranker implementations (see below), written as Python scripts.

Evaluation The metrics established in the configuration file are executed over the recommendation results. As is the case with LibRec, multiple metrics can be applied to the results

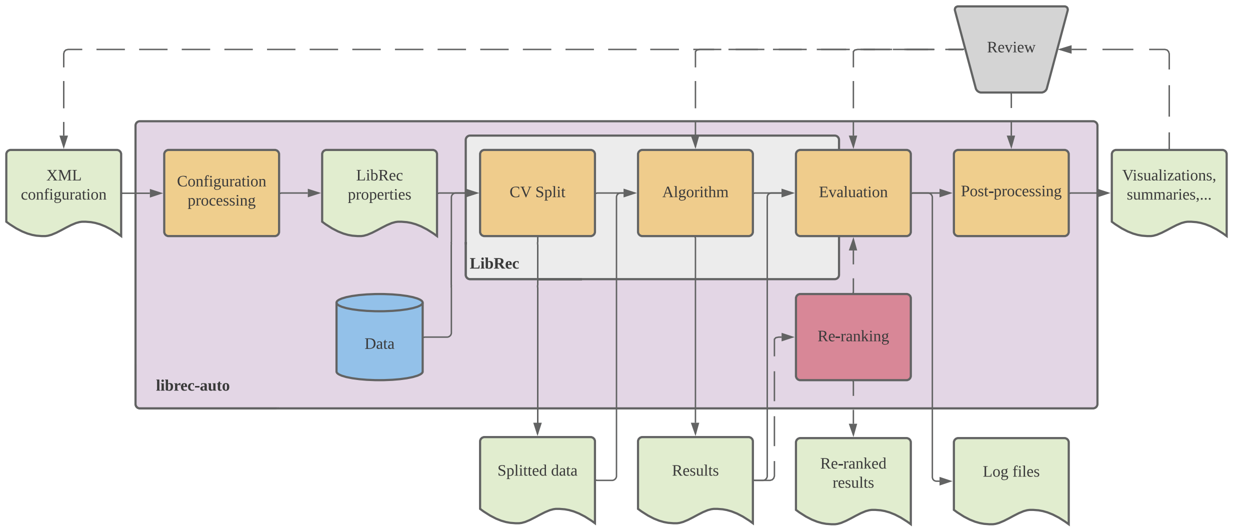


Figure 1: Schematic of experimentation workflow with `librec-auto`. The LibRec library (Java, shown in grey) is encapsulated by `librec-auto` (Python, shown in purple), which manages configuration, experimental outputs and post-processing. Added from [32] is the new re-ranking module shown in red.

of a study. These metrics can be the ones implemented in LibRec or, as the figure indicates, can be implemented in Python and executed by the `librec-auto` process.

Post-processing (optional) Once a study is complete, post-processing operations are available to put the results of a study into human-readable form through visualizations, or export to forms that can be read by other tools. These may lead to revisions in the experimental protocol or to the launching of additional studies.

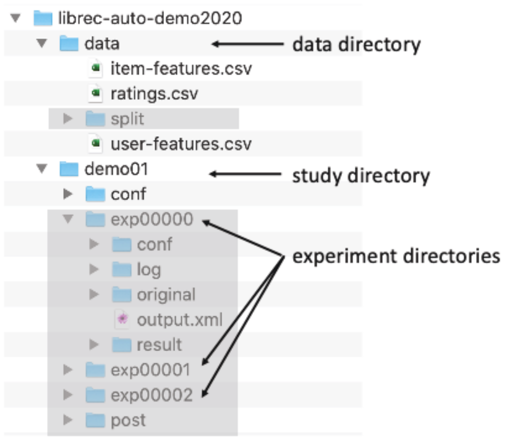
Note that `librec-auto` supports incremental updating of study execution. Later portions of the experimental pipeline can be executed without re-computing earlier steps, a capability not available in vanilla LibRec. For example, a re-ranker implementation can be tweaked and tested or an additional evaluation metric can be applied without having to generate new results.

3.1 Study structure

A study is organized in a multi-directory file structure. Some parts of the structure are managed by `librec-auto` itself, others are set up by the experimenter. Figure 2 shows an example of the file layout for a study called “demo01”. In this case, the data is being used in more than one study, so it is stored in a directory outside of the study structure. As the shaded region in figure indicates, most of the study files are managed by `librec-auto` itself.

Key files in the study structure include:

Data files Ratings files and files with user and item meta-data.
Configuration files XML-based study configuration (described in detail below).
Properties files Input to the LibRec library.
Log files Output from the LibRec execution.
Output files Summary files for individual experiments and the study as whole.



```
▼ librec-auto-demo2020
  ▼ data
      item-features.csv
      ratings.csv
    ▶ split
      user-features.csv
  ▼ demo01
    ▶ conf
    ▼ exp00000
      ▶ conf
      ▶ log
      ▶ original
        output.xml
      ▶ result
    ▶ exp00001
    ▶ exp00002
    ▶ post
```

Figure 2: Directory structure for a `librec-auto` study. The shaded directories are experiment outputs managed by the system.

Post-processing outputs These can take a variety of forms including image, CSV and others.

For reproducibility, it is sufficient for experimenters to distribute their data and configuration files (`conf` directory). All other aspects of a study can be reproduced by running the `librec-auto` application.

3.2 Fairness-aware recommendation

With the extensive research in literature focusing on the assessment and mitigation of unfair outcome of algorithms, several toolkits for fair machine learning have recently emerged to make such methods widely accessible. Examples of such toolkits include: AI Fairness

360 – an open source toolkit for mitigating bias in AI application lifecycle designed by IBM [8], Fairlearn – a toolkit for assessing and improving fairness in AI designed by Microsoft [12], Aequitas – an open-source bias audit toolkit that allows auditing machine learning models for discrimination and bias [39], LinkedIn Fairness Toolkit (LiFT) [48], and FairSight: Visual Analytics for Fairness in Decision Making [2], just to name a few. However these toolkits have been proposed to address issues of unfairness in classification, and the metrics and approaches they encode are not always applicable to recommender systems research. What distinguishes `librec-auto`'s fairness extensions from other toolkits is its focus on evaluating and mitigating unfairness in recommender system algorithms in particular.

Although `librec-auto` has been under development since 2018, the latest release incorporates several key advances that specifically support common tasks in the study of recommendation fairness. These specific fairness-aware capability of `librec-auto` are (1) evaluation metrics that report on fairness aspects of recommendation output, (2) support for working with user (demographic) and item (content) features in algorithms and metrics, and (3) an optional re-ranking step in the experiment pipeline, to support what is one of the most common category of fairness enhancing techniques. With these features, `librec-auto` can support a wide range of research activities in fairness-aware recommendation, and we will be adding additional capabilities in future releases.

3.2.1 Fairness metrics. There are two types of assumptions behind fairness-aware systems Friedler et al. [20]: (1) *What you see is what you get (WYSIWYG)*, and (2) *We're all equal*. Having a WYSIWYG assumption in a system means that the observations of features do not encode bias, therefore the system uses that unbiased information to make decisions about individuals in a way that similar individuals are treated similarly. WYSISYG seeks to achieve fairness for individuals rather than a group of users. However, the second assumption (WAE) acknowledges that the biases in the society are reflected in data which lead to disparate treatment of different groups with different sensitive features. Therefore WAE supports group fairness. In `librec-auto`, we have concentrated on fairness-aware metrics that measure group fairness. However, some individual fairness metrics are also included. We intend to extend the coverage of both types of metrics in future releases.

In addition to the group vs individual fairness distinction, recommender system fairness is also distinguished by the fact that fairness concerns may be formulated relative to multiple different stakeholders [1, 13]. In particular, we may be concerned about fairness towards consumers of recommendations (end-users) and providers of items being recommended. Consumer-side group fairness asks whether the system is fair to different groups of users: for example, male, female and non-binary job seekers getting recommendations of job listings. Provider-side group fairness asks whether the system is fair to different groups of item providers: for example, male, female and non-binary musical artists whose tracks are being recommended. We have included metrics for both types of stakeholders in `librec-auto`.

The WEA assumption is concerned with equitable treatment of users in protected groups. The WAE assumption requires that recommender systems provide individuals of a protected group (regardless of stakeholder position) to experience a similar quality of service as those in the unprotected group, according to some objective or metric deemed important to that group [18, 19, 44, 51]. For consumers, the common approach to measure the quality of recommendations is using accuracy-based metrics. These values are computed and compared across groups. As an example, [51] measures and compares different types of errors among user groups. [14] also compared the statistical parity of the precision of recommendations for different demographic groups. A WSIWYG assumption asserts that all users should get similar quality of service and does not make use of a protected / unprotected group assumption. [44] uses a diversity-based metric (KL-divergence) to measure the discrepancy between the distribution of item categories in user profile and that of her recommendations. A system that scores well on this calibration measure is considered fair in that it is providing recommendations well matched to the tastes of individual recommendation consumers, even when those tastes vary widely from the average user.

Table 1: Fairness metrics in `librec-auto`.

Metric	Stakeholder Focus	Fairness Type
Calibration	consumer	individual
DPF	consumer	group
Error-based	consumer	group
Gini Index	provider	individual
PPR	consumer & provider	group
Statistical Parity	provider	group

Similarly for providers, we provide metrics appropriate under either of the WAE or WYSIWYG assumptions. Under the first assumption, we concentrate on the results associated with protected vs unprotected provider groups. A simple way to compare these results is to look at *exposure*, the likelihood that particular providers will have their items displayed in the recommendation lists of consumers [30, 43]. However, displaying a recommendation to an uninterested user might not be of much value, so a utility-based (or *hit-based*) alternative considers both how often items are displayed and the quality of match between the user and the item and compares these utilities between protected and unprotected providers [11, 41]. An example of individual fairness metrics for providers is the Gini index metric, where a high (unfair) value indicates that recommendation results are highly concentrated among a few providers.

As this discussion suggests, fairness metrics for recommender systems form a fairly complex space with many different proposals [9, 11, 16, 29, 44, 47, 50, 51]. We implement the following metrics in `librec-auto` and where possible, include both consumer-side and provider-side versions of the metric, as listed in Table 1. All metrics listed here are implemented in Java and integrated with the LibRec code base:

- **Calibration** [44], a distribution-based metric that uses KL-Divergence to measure the difference in item category distribution between the preferences of users and their respective recommendation lists.
- **Discounted Proportional Fairness** (DPF), a hit-based fairness metric similar to the metric offered in [16] where it

measures the ranking utility (nDCG) of the protected group with respect to the other groups.

- **Error-based** metrics proposed in Yao et al. [51] including value-unfairness, absolute unfairness, underestimation unfairness, overestimation unfairness. Non-parity unfairness as defined by Kamishima et al. [28] is also in this group.
- **Gini Index** calculated over the exposure of all the providers in all recommendation lists.
- **P-Percent-Rule** (PPR) discussed in [10], is a two-sided extension of statistical parity [6].
- **Statistical parity**, based on the ideas discussed in [36, 53], measuring the difference in outcomes between protected and unprotected groups relative to various recommendation outcomes. Both ranking and prediction accuracy measures are supported.

In addition to this set of fairness metrics, we provide an implementation of the *Intra-List Distance (ILD)* measure [55], a pairwise distance between all the item features in each user's recommendation list. This is a user-focused measure of the diversity that a recommender system provides.

3.2.2 Item and user metadata. LibRec does not directly support algorithms that use item and user metadata: content-based or demographic recommendation. The ability to make use of such data is an enhancement that was made in order to implement fairness-aware algorithms and metrics: it is clear, for example, that you need to know if a user is in a protected group to know how to apply metrics like statistical parity or value unfairness.

Our enhancements allow the algorithms and metrics to access data stored in user and item feature files. These files have a simple sparse triple format: item id (or user id), feature name, feature value. Because it is a sparse format, rows with zero values can be omitted. If the value for a feature is binary, the feature value can also be omitted and all $\langle item, feature\rangle$ pairs that appear in the file will have a feature value of 1.

For the specific case of group fairness algorithms and metrics, the feature file must contain a feature with a binary value that represents the protected / unprotected group distinctions. Items with a value 1 for this feature will be considered protected for the purposes of metrics or algorithms. We plan to generalize this capability in future releases.

3.2.3 Re-ranking. A re-ranking algorithm takes the ranked output of a base system (usually an information retrieval or recommender system) and performs a permutation of the ordering (and usually a truncation) before sending the list to an end user. A typical application of these techniques is to enhance output diversity or fairness when the list of top candidates returned by the base system lacks these properties.

Re-rankers strive to achieve a reasonable balance between accuracy and other output properties, like fairness or diversity, by offering a tunable tradeoff between boosting protected or diverse items and including items ranked highly by the base system.

`librec-auto` includes implementations of the following re-ranking algorithms:

- **FAR**, defined in [30], combines a personalization-induced and fairness-induced scores with hyper-parameter λ;
- **PFAR**, from [30], adds a personalized weight to FAR, calculated based on item-features in user profile, representing the tolerance of the user for diverse results
- **OFAiR** is based on PFAR and allows fine-grained control of protected group promotion when there are multiple protected groups [43].
- **FA*IR** [52] builds a queues of protected and unprotected items and draws from each queue to build the final re-ranked list.
- **MMR** diversifies result lists by greedily adding items with maximal marginal relevance [15].
- **XQuAD** defined in [40] has similar goal to MMR algorithm, but it enhances diversity with respect to specific aspects.
- **Calibrated Recommendations**, an algorithm closely tied to the Calibration metric above, which re-ranks recommendations to ensure a close match to the user's distribution of interests in item features [44].

The re-ranking methods are part of `librec-auto` and are implemented in Python. A re-ranking script loads the original set of (large) recommendation lists computed for each user and then computes new re-ranked sub-lists for output. As noted above, re-rankers can participate in the optimization process. However, it would be inefficient to compute LibRec results multiple times if the only parameters changing were within the re-ranker. `librec-auto` detects this situation and only computes results once for a given set of algorithm parameters.

3.3 Optimization

As noted above, `librec-auto` supports grid search, an exhaustive method of checking each combination of values specified in order to find the optimal value. The number of experiments is determined by a cross-product of the variables. The syntax to run grid-search involves specifying specific values for a given metric. For example,

```
<item-reg><value>0.01</value>
      <value>0.05</value></item-reg>
```

While grid search is guaranteed to find the optimal result within the specified parameter-value combinations, it can be expensive and time-consuming to run, particularly with a large number of variables, and depends heavily on experimenter expertise in choosing specific parameter values to sample. To overcome these difficulties, we have introduced Bayesian black-box optimization (using `optuna`), which aims to reduce the time to find an optimal combination of values [3]. The `optuna` package includes a number of optimization methods including the Tree Of Parzen Estimators (TPE) method, which makes probabilistically sampled estimates of optimal parameter values based on prior experimental results. In our experiments, this method outperformed other options that we considered.

In order to use this system, an `optimize` element must be added to the configuration file (see below), and instead of the `value` syntax used to indicate possible values for grid search, the experimenter includes `lower` and `upper` elements indicating the legal range for each variable. If the metric being used is a LibRec metric then the direction of optimization is automatically determined. For user-defined metrics, the experimenter must specify the direction of optimization.

4 STUDY CONFIGURATION

`librec-auto` uses an XML configuration file to specify all aspects of the experimental pipeline. As noted above, a configuration file defines a *study*, which computes evaluation results for a single algorithm and a single data set, possibly over multiple choices of hyperparameters, each combination of which constitutes an *experiment*. The configuration file is divided into sections, some of which are optional. We devote some attention here to this file because a study of its components gives a good overview of the capabilities and flexibility of `librec-auto`. A sample configuration file is shown in Figure 3.

XML was chosen because configuration files need to be managed both by human experimenters and by the system itself. The unstructured key-value properties format used by LibRec was found to be difficult for experimenters to manage as it does not impose a natural structure on the different aspects of the configuration and makes it difficult to re-use and adapt configurations. JSON was also considered, but XML is more self-describing, which is important when the configuration must provide documentation for what experiments were performed. XML enables configuration files to be easily modularized, so that, for example, multiple studies can share the same methodology elements, preventing inadvertent misconfiguration. XML also provides facilities for validating configuration files against an XML schema, which in future versions will allow us to provide error checking of configuration files before processing them.

4.1 Global elements (all optional)

The global elements establish conditions under which all aspects of the configuration file interpretation and study execution will take place.

random-seed : An integer that will be used as the seed for any randomized actions that the platform takes. This ensures repeatability for experiments.

thread-count : If this is greater than zero, librec-auto will spawn multiple threads for various tasks, including parallel execution of experiments.

library : There can be multiple `library` elements, from which algorithms, metrics and other elements can be imported. There is a default system library for algorithms (referenced in the example). An element from the library can be imported using the `ref` attribute.

In the example file in Figure 3, `<alg ref="alg:biasedmf"/>`‘ refers to the `biasedmf` (Biased Matrix Factorization) algorithm as implemented in LibRec with the default hyperparameters given in the library. The library file is also a useful reference for experimenters to see what hyperparameters a given algorithm accepts. These can be overridden by local declarations in the configuration file.

4.2 Data Section

The data section indicates where the data for the study can be found. The data can be in any convenient place. However, `librec-auto`

```
<librec-auto>
    <random-seed>202001</random-seed>
    <thread-count>1</thread-count>
    <library src="system">default-algorithms.xml</library>

<data>
        <data-dir>../data</data-dir>
        <format>UIR</format>
        <data-file format="text">ratings.csv</data-file>
</data>

<splitter>
        <model count="3">kcv</model>
        <dim>userfixed</dim>
        <ratio>0.8</ratio>
        <save>true</save>
</splitter>

<alg ref="alg:biasedmf">
        <similarity type="item">pcc</similarity>
        <iterator-max>25</iterator-max>
        <item-reg><value>0.01</value>
                  <value>0.05</value></item-reg>
        <num-factors>20</num-factors>
</alg>

<metric>
        <ranking>true</ranking>
        <list-size>10</list-size>
        <class>precision</class>
        <script lang="python3" src="system">
            <script-name>ndcg_metric.py</script-name>
            <param name="list_size">10</param>
        </script>
</metric>

<post>
        <script lang="python3" src="system">
            <script-name>results_to_csv.py</script-name>
            <param name="option">all</param>
        </script>
</post>
</librec-auto>
```

Figure 3: Sample configuration file

will need to be able to write to this directory since it will by default add new data split directories here. [4]

4.3 Feature Section (optional)

LibRec is heavily focused on collaborative algorithms, so the basic form for input data is the user/rating matrix. However, as noted above, for content-oriented algorithms and for fairness-aware algorithms and metrics, it may be necessary to have item metadata

[4]Note that, for LibRec compatibility, there are two different places where the label “format” is used. The `format` element indicates the columns in the ratings file. The `format` attribute of the `data-file` element is the file format of the ratings file: LibRec supports text and AIFF file formats.

and/or user demographic data. The `feature` element allows experimenters to include either item or user features as input to a study's algorithm or metrics.

4.4 Splitter Section

A variety of different evaluation strategies are employed by recommender systems researchers to evaluate algorithms. K-fold cross-validation is a common one, and LibRec supports multiple variants on this approach.

The example in Figure 3 directs `librec-auto` to perform five-fold cross-validation using 80% of each user's data for training and 20% for testing in each fold. The splits will be saved to the data directory, and can be re-used in subsequent experimentation.

The configuration file also supports fixed training and test files supplied by the experimenter and temporal splitting (when dates are available for ratings).

4.5 Algorithm Section

LibRec supports more than 70 recommendation algorithms, each linked to a paper from the scientific literature that it implements. A default algorithms library contains a number of the most common algorithms and complete lists of their hyperparameters with default values.

Typically, a study will consist of multiple experiments over different algorithm hyperparameters. As noted in Section 3.3, `librec-auto` supports both grid search and Bayesian black-box optimization In Figure 3, the weight for the item regularization term (`item-reg`) is being varied across two values, 0.01 and 0.05. This information tells the system to conduct two experiments using the given weight values. Any number of hyperparameters can be searched over. The system will conduct an experiment for every combination of values (Cartesian product), so the number of experiments can be quite large.

4.6 Optimize Section (optional)

The black-box optimization capability requires the use of a separate `optimize` element to specify what metric is used for optimization and how many iterations are to be performed. For example,

```
<optimize><metric>precision</metric>
          <iterations>25</iterations></optimize>
```

This option cannot be combined with grid search. If it is used, instead of providing a list of values associated with a parameter (the `value` element), we provide an upper and lower bound to the search range.

```
<item-reg><lower>0.01</lower>
          <upper>0.05</upper></item-reg>
```

4.7 Metrics Section

A study can employ multiple metrics. LibRec supports over 20 different metrics for evaluating recommender system performance. There are two basic types: *error metrics*, which compare the predictions produced by an algorithm with known ratings from the test set, (for example, RMSE), and *ranking metrics*, which evaluate a ranked list of results produced for each user in the training data (for example, Precision@10). Although multiple metrics can be used, only one type of metric can be used at a time since these methodologies are quite different in how they make use of the training data. Ranking metrics (like Precision@10 shown in the example) require the `ranking` element to be true and a list-size to be specified.

Fairness-aware metrics require a `protected-feature` element. As described in Section 3.2, the current release requires this to be a binary feature drawn from the item or user feature file. Items (or users) associated feature value of 1 will be considered protected for the purposes of a fairness metric. This value is also used by fairness-aware algorithms in LibRec.

`librec-auto` supports custom metrics defined as Python scripts, similar to re-ranking scripts described below. Implementers can make use of generic error or ranking metric abstract classes for building their metrics, minimizing the amount of implementation effort. This eliminates the need to program such metrics in Java and re-compile the LibRec executable.

If black-box optimization is used, the optimizer needs to know which direction of a metric is considered "good": note that large RMSE is bad, but large nDCG is good. This property is pre-defined for built-in metrics, but for custom metrics, a flag must be included in the metric `script` declaration to indicate whether to optimize for larger or small values of the metric.

4.8 Rerank Section (optional)

For a study that includes re-ranking, the re-ranking script is specified in a `rerank` element. Note that all re-ranking is done by script resources and these can be easily customized by experimenters. Currently, only Python scripts are supported.

The configuration for a re-ranker indicates the parameters to be passed. In the following example, we see that the `far-rerank.py` script is being used, which implements the re-ranker described in [30]. The `max_len` parameter tells the script how many items will be returned for each user from the re-ranked set. Other parameters are algorithm-specific controls on the re-ranking process. The FAR algorithm defined in [30] uses the `lambda` and `binary` parameters shown here.

```
<rerank>
   <script lang="python3" src="system">
      <script-name>far-rerank.py</script-name>
         <param name="max_len">10</param>
         <param name="lambda">
            <value>0.3</value>
            <value>0.0</value>
         </param>
         <param name="binary">False</param>
   </script>
</rerank>
```

Note that a re-ranking step can participate in the optimization aspects of `librec-auto`. In the example above, we explore two different values of `lambda` in combination with whatever other hyperparameters are being searched. (Black-box optimization is not yet available for re-ranker parameters, but is a planned feature.)

4.9 Post-Processing Section (optional)

`librec-auto` supports the post-processing of study results. There are existing scripts for producing simple visualizations (see Figure 4), for producing CSV files for further analysis, and for posting experimental results to Slack and Dropbox (see Figure 5). The latter are useful for experiments with longer run-times. There is a script provided for encrypting application API keys so that they do not need to be shared (insecurely) in the configuration file.

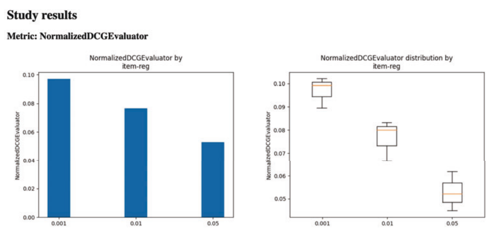
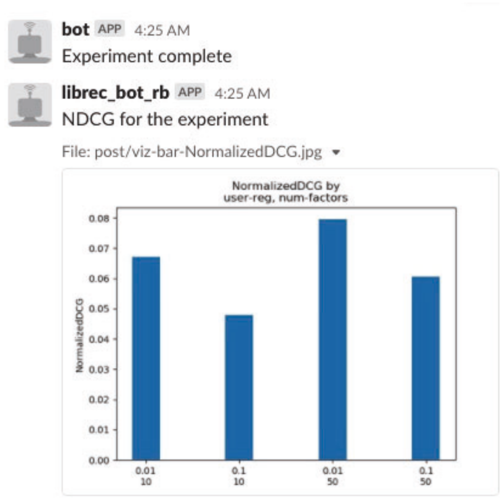


Figure 4: Sample auto-generated study output visualizations. These simple plots support experimenters in providing a quick analysis of study outcomes.


bot APP 4:25 AM
Experiment complete

librec_bot_rb APP 4:25 AM
NDCG for the experiment

File: post/viz-bar-NormalizedDCG.jpg

Figure 5: Sample auto-generated Slack posts. Such notifications provide experimenters and research teams with asynchronous notifications of study progress.

5 RELATED WORK

With the rapid growth of published recommendation algorithms, the concern about reproducibility of models in the research community has given rise to the need of having standard implementations for models and evaluation metrics. Many of the recommendation algorithms lack public implementations or those that have one, are inconsistent in details. `librec-auto` is a response to this need, but it should be noted that there many other recommender systems libraries have been created and offered to the research community, beginning with earlier systems such as MyMediaLite [21] and continuing through newer releases such as Microsoft Recommenders [4] and RecBole [54]. However, none of these libraries offer the precise combination of features that are found in `librec-auto` and some are no longer being maintained.

One of the key aspects of `librec-auto` is its incorporation of fairness-aware metrics and algorithms. As noted above, the platform has strong support for re-ranking, perhaps the most common fairness-aware recommendation technique, and it includes examples of re-ranking algorithms from the research literature, as described in Section 3.2.3. Also, `librec-auto` supports metrics for both consumer- and provider-side fairness as noted in Section 3.2.1. No other existing library has these capacities.

Another important difference between `librec-auto` and existing libraries is its large collection of algorithms (70) drawn from LibRec itself. Many existing libraries, for example, RecommenderLab [23], LKPY [17], and Surprise [24] concentrate on a relatively small handful of algorithms. Microsoft Recommenders [4] and RecBole [54] have large algorithm libraries including dozens of implementations, but these have almost no overlap with those included in LibRec and so can be thought of as useful complements. We plan to integrate the algorithms from the Microsoft library in particular as part of our next major release, as described below.

Finally, `librec-auto` stands apart for its use of explicit algorithm configuration and support for unattended operation. While Jupyter notebooks are very convenient for interactive exploration of algorithm properties, we have found that large-scale experimentation (especially for researchers on limited compute budgets) is best supported through the ability to configure and run batch processes and this has been considered a primary use case since the project's inception.

6 DISCUSSION AND FUTURE DIRECTIONS

The `librec-auto` project is driven by the needs of recommender systems experimentation and education and continues to evolve. Its key features (explicit configuration, built-in parameter optimization, computationally-efficient result handling) were all requirements that surfaced within our research work. Newer capabilities in the area of fairness-aware recommendation and re-ranking have a similar genesis and proven practical value.

We note that `librec-auto` does not directly support all popular methodologies for recommender systems evaluation. In particular, it supports only batch evaluation, and not iterative evaluation as required by bandit and reinforcement learning paradigms. There are other platforms, for example RecoGym [37] and RecSim NG [33], that support this type of methodology and we do not anticipate the need to evolve our system in this direction.

6.1 Algorithm integration

One of the key requirements for future releases is that the platform makes it easy to experiment with algorithms other than those implemented in LibRec. By the time of publication, we expect to have released interfaces to two popular recommendation libraries: LibFM [35] and Microsoft's Recommenders library [5].

LibFM[35] is a factorization machine (FM) software tool written in C++, that features stochastic gradient descent (SGD) and alternating least-squares (ALS) optimization, as well as Bayesian inference using Markov Chain Monto Carlo (MCMC) for regression and classification tasks. It also contains methods for optimizing an FM model with respect to ranking.

As noted above, Microsoft Recommenders [4] is an open-source platform for recommender systems development and evaluation. This Python-based tool includes 31 recommendation algorithms including some recent deep learning algorithms, for example, Neural Recommendation with Multi-Head Self-Attention (NRMS) [49]. While the platform includes many capabilities for data handling and evaluation, these are handled by `librec-auto` in our integration. We draw only from the algorithm implementations.

An experimenter accesses non-LibRec platforms through custom wrapper scripts to which parameters are passed. For example, the algorithm configuration element below invokes the xDeepFM algorithm using the `msrec-wrapper.py` and passing arguments needed by this algorithm.

```
<alg>
   <script lang="python3" src="system">
      <script-name>msrec-wrapper.py</script-name>
         <param name="model">xdeepfm</param>
         <param name="epochs">200</param>
         <param name="batch_size">256</param>
         <param name="learning_rate">1e-3</param>
         <param name="cross_reg">0.0001</param>
         <param name="embed_reg">0.0001</param>
         <param name="num_factors">100</param>
         <param name="feature_count">1000</param>
         <param name="field_size">10</param>
   </script>
</alg>
```

6.2 Fairness-aware aspects

`librec-auto` is unique in its library of recommendation re-ranking algorithms, and this aspect of the project is being actively developed. The current library of re-rankers only includes *listwise* re-rankers that optimize one recommendation list at a time. There is a thread of fairness- and diversity-aware research that studies *batch* re-rankers that optimize over the entire collection of recommendation lists at once: see, for example, [31, 34, 46]. We plan to extend our library of re-rankers to include this class of algorithm, and, as we have with the listwise class, provide base implementations with which researchers can construct their own.

Similarly, while `librec-auto` has a large variety of fairness metrics and supports both consumer-side and provider-side analysis, this capability will be enhanced in future versions. There remain some fairness definitions in the literature that we have not yet implemented, and some of our existing implementations are not as flexible as might be desired. In particular, we note that currently all fairness-aware aspects of any given study must be focused on a single side of the analysis (consumer or provider) and all group fairness aspects are controlled by the same definition of what is protected. We are extending this capability to allow (for example) re-ranking under one fairness definition and evaluating based on another. We are also planning to relax the constraint that protected groups must be defined by binary features, so that experimenters can control how protected groups are defined without having to produce new feature files in which the distinction is encoded.

6.3 Other enhancements

While the system supports experimenters in planning, executing and evaluating individual studies, it does not support more general research workflows that might involve the comparison of multiple algorithms. For this higher-level support, we are working to integrate ClearML[5], an open-source machine learning management platform. A prototype of this integration is already in use.

`librec-auto` also does not currently support Bayesian optimization in combination with re-ranking. We anticipate adding support for this in the near future.

Finally, we are seeking to make `librec-auto` more accessible to new users through the development of a setup wizard that automates the creation of study directories and configurations for typical use cases. The initial version of this wizard is part of the current codebase and we have plans for its continued enhancement.

ACKNOWLEDGMENTS

The authors would like to acknowledge the many contributors to the `librec-auto` project including Aldo Ordoñez-Gauger, Xavier Sepulveda, Bum Kim, Emily Rooney and Will Mardick-Kanter. Authors Burke, Sonboli, Schwartz and Kadekodi were supported in part by the National Science Foundation under Grant Number: 1911025. Schwartz, Liu and Kadekodi were also supported by the University of Colorado through the DLA and UROP programs.

REFERENCES

[1] Himan Abdollahpouri, Gediminas Adomavicius, Robin Burke, Ido Guy, Dietmar Jannach, Toshihiro Kamishima, Jan Krasnodebski, and Luiz Pizzato. 2020. Multistakeholder recommendation: Survey and research directions. *User Modeling and User-Adapted Interaction* 30, 1 (2020), 127–158.

[2] Yongsu Ahn and Yu-Ru Lin. 2019. Fairsight: Visual analytics for fairness in decision making. *IEEE transactions on visualization and computer graphics* 26, 1 (2019), 1086–1095.

[3] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2623–2631.

[4] Andreas Argyriou, Miguel González-Fierro, and Le Zhang. 2020. Microsoft Recommenders: Best Practices for Production-Ready Recommendation Systems. In *Companion Proceedings of the Web Conference 2020* (Taipei, Taiwan) *(WWW ’20)*. Association for Computing Machinery, New York, NY, USA, 50–51. https://doi.org/10.1145/3366424.3382692

[5] Andreas Argyriou, Miguel González-Fierro, and Le Zhang. 2020. Microsoft Recommenders: Best Practices for Production-Ready Recommendation Systems. In *Companion Proceedings of the Web Conference 2020* (Taipei, Taiwan) *(WWW ’20)*. Association for Computing Machinery, New York, NY, USA, 50–51. https://doi.org/10.1145/3366424.3382692

[6] Solon Barocas and Andrew D Selbst. 2016. Big data’s disparate impact. *Calif. L. Rev.* 104 (2016), 671.

[7] Joeran Beel, Corinna Breitinger, Stefan Langer, Andreas Lommatzsch, and Bela Gipp. 2016. Towards reproducibility in recommender-systems research. *User modeling and user-adapted interaction* 26, 1 (2016), 69–101.

[8] Rachel KE Bellamy, Kuntal Dey, Michael Hind, Samuel C Hoffman, Stephanie Houde, Kalapriya Kannan, Pranay Lohia, Jacquelyn Martino, Sameep Mehta, Aleksandra Mojsilović, et al. 2019. AI Fairness 360: An extensible toolkit for detecting and mitigating algorithmic bias. *IBM Journal of Research and Development* 63, 4/5 (2019), 4–1.

[9] Alex Beutel, Jilin Chen, Tulsee Doshi, Hai Qian, Li Wei, Yi Wu, Lukasz Heldt, Zhe Zhao, Lichan Hong, Ed H Chi, et al. 2019. Fairness in recommendation ranking through pairwise comparisons. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2212–2220.

[10] Dan Biddle. 2006. *Adverse impact and test validation: A practitioner’s guide to valid and defensible employment testing.* Gower Publishing, Ltd.

[5]https://clear.ml/

[11] Asia J Biega, Krishna P Gummadi, and Gerhard Weikum. 2018. Equity of attention: Amortizing individual fairness in rankings. In *The 41st international acm sigir conference on research & development in information retrieval*. 405–414.

[12] Sarah Bird, Miro Dudík, Richard Edgar, Brandon Horn, Roman Lutz, Vanessa Milan, Mehrnoosh Sameki, Hanna Wallach, and Kathleen Walker. 2020. *Fairlearn: A toolkit for assessing and improving fairness in AI*. Technical Report.

[13] Robin Burke. 2017. Multisided Fairness for Recommendation. In *Workshop on Fairness, Accountability and Transparency in Machine Learning (FATML)*. Halifax, Nova Scotia, 5 pages.

[14] Robin Burke, Nasim Sonboli, and Aldo Ordonez-Gauger. 2018. Balanced neighborhoods for multi-sided fairness in recommendation. In *Conference on Fairness, Accountability and Transparency*. PMLR, 202–214.

[15] Jaime Carbonell and Jade Goldstein. 1998. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*. 335–336.

[16] Carlos Castillo. 2019. Fairness and transparency in ranking. 52, 2 (2019), 64–71.

[17] Michael D. Ekstrand. 2020. *LensKit for Python: Next-Generation Software for Recommender Systems Experiments*. Association for Computing Machinery, New York, NY, USA, 2999–3006. https://doi.org/10.1145/3340531.3412778

[18] Michael D Ekstrand and Daniel Kluver. 2021. Exploring author gender in book rating and recommendation. *User Modeling and User-Adapted Interaction* (2021), 1–44.

[19] Michael D Ekstrand, Mucun Tian, Ion Madrazo Azpiazu, Jennifer D Ekstrand, Oghenemaro Anuyah, David McNeill, and Maria Soledad Pera. 2018. All the cool kids, how do they fit in?: Popularity and demographic biases in recommender evaluation and effectiveness. In *Conference on Fairness, Accountability and Transparency*. PMLR, 172–186.

[20] Sorelle A. Friedler, Carlos Scheidegger, and Suresh Venkatasubramanian. 2021. The (Im)Possibility of Fairness: Different Value Systems Require Different Mechanisms for Fair Decision Making. *Commun. ACM* 64, 4 (March 2021), 136–143. https://doi.org/10.1145/3433949

[21] Zeno Gantner, Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2011. MyMediaLite: A Free Recommender System Library. In *Proceedings of the Fifth ACM Conference on Recommender Systems* (Chicago, Illinois, USA) *(RecSys '11)*. Association for Computing Machinery, New York, NY, USA, 305–308. https://doi.org/10.1145/2043932.2043989

[22] Guibing Guo, Jie Zhang, Zhu Sun, and Neil Yorke-Smith. 2015. LibRec: A Java Library for Recommender Systems.. In *UMAP Workshops*, Vol. 4.

[23] Michael Hahsler. 2015. *recommenderlab: A framework for developing and testing recommendation algorithms*. Technical Report.

[24] Nicolas Hug. 2020. Surprise: A Python library for recommender systems. *Journal of Open Source Software* 5, 52 (2020), 2174. https://doi.org/10.21105/joss.02174

[25] Thorsten Joachims, Maria Dimakopoulou, Adith Swaminathan, Yves Raimond, Olivier Koch, and Flavian Vasile. 2019. REVEAL 2019: closing the loop with the real world: reinforcement and robust estimators for recommendation. In *Proceedings of the 13th ACM Conference on Recommender Systems*. 568–569.

[26] Thorsten Joachims, Yves Raimond, Olivier Koch, Maria Dimakopoulou, Flavian Vasile, and Adith Swaminathan. 2020. REVEAL 2020: Bandit and Reinforcement Learning from User Interactions. In *Fourteenth ACM Conference on Recommender Systems*. 628–629.

[27] Thorsten Joachims, Adith Swaminathan, Yves Raimond, Olivier Koch, and Flavian Vasile. 2018. REVEAL 2018: offline evaluation for recommender systems. In *Proceedings of the 12th ACM Conference on Recommender Systems*. 514–515.

[28] Toshihiro Kamishima, Shotaro Akaho, and Jun Sakuma. 2011. Fairness-aware learning through regularization approach. In *2011 IEEE 11th International Conference on Data Mining Workshops*. IEEE, 643–650.

[29] Caitlin Kuhlman, MaryAnn VanValkenburg, and Elke Rundensteiner. 2019. FARE: Diagnostics for Fair Ranking using Pairwise Error Metrics. In *The World Wide Web Conference*. 2936–2942.

[30] Weiwen Liu, Jun Guo, Nasim Sonboli, Robin Burke, and Shengyu Zhang. 2019. Personalized Fairness-Aware Re-Ranking for Microlending. In *Proceedings of the 13th ACM Conference on Recommender Systems* (Copenhagen, Denmark) *(RecSys '19)*. Association for Computing Machinery, New York, NY, USA, 467–471. https://doi.org/10.1145/3298689.3347016

[31] Masoud Mansoury, Himan Abdollahpouri, Mykola Pechenizkiy, Bamshad Mobasher, and Robin Burke. 2020. Fairmatch: A graph-based approach for improving aggregate diversity in recommender systems. In *Proceedings of the 28th ACM Conference on User Modeling, Adaptation and Personalization*. 154–162.

[32] Masoud Mansoury, Robin Burke, Aldo Ordonez-Gauger, and Xavier Sepulveda. 2018. Automating recommender systems experimentation with librec-auto. In *Proceedings of the 12th ACM Conference on Recommender Systems*. 500–501.

[33] Martin Mladenov, Chih-Wei Hsu, Vihan Jain, Eugene Ie, Christopher Colby, Nicolas Mayoraz, Hubert Pham, Dustin Tran, Ivan Vendrov, and Craig Boutilier. 2021. RecSim NG: Toward Principled Uncertainty Modeling for Recommender Ecosystems. arXiv:2103.08057 [cs.LG]

[34] Gourab K Patro, Arpita Biswas, Niloy Ganguly, Krishna P Gummadi, and Abhijnan Chakraborty. 2020. Fairrec: Two-sided fairness for personalized recommendations in two-sided platforms. In *Proceedings of The Web Conference 2020*. 1194–1204.

[35] Steffen Rendle. 2012. Factorization Machines with libFM. *CM Transactions on Intelligent Systems and Technology (TIST)* 3, 3, Article 57 (May 2012), 22 pages.

[36] Ya'acov Ritov, Yuekai Sun, and Ruofei Zhao. 2017. On conditional parity as a notion of non-discrimination in machine learning. arXiv:1706.08519 [stat.ML]

[37] David Rohde, Stephen Bonner, Travis Dunlop, Flavian Vasile, and Alexandros Karatzoglou. 2018. RecoGym: A Reinforcement Learning Environment for the problem of Product Recommendation in Online Advertising. arXiv:1808.00720 [cs.IR]

[38] Alan Said and Alejandro Bellogín. 2014. Comparative recommender system evaluation: benchmarking recommendation frameworks. In *Proceedings of the 8th ACM Conference on Recommender systems*. 129–136.

[39] Pedro Saleiro, Benedict Kuester, Loren Hinkson, Jesse London, Abby Stevens, Ari Anisfeld, Kit T Rodolfa, and Rayid Ghani. 2018. Aequitas: A bias and fairness audit toolkit. , 19 pages.

[40] Rodrygo LT Santos, Jie Peng, Craig Macdonald, and Iadh Ounis. 2010. Explicit search result diversification through sub-queries. In *European conference on information retrieval*. Springer, 87–99.

[41] Ashudeep Singh and Thorsten Joachims. 2018. Fairness of exposure in rankings. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2219–2228.

[42] Nasim Sonboli, Robin Burke, Zijun Liu, and Masoud Mansoury. 2020. Fairness-Aware Recommendation with Librec-Auto. In *Fourteenth ACM Conference on Recommender Systems* (Virtual Event, Brazil) *(RecSys '20)*. Association for Computing Machinery, New York, NY, USA, 594–596. https://doi.org/10.1145/3383313.3411525

[43] Nasim Sonboli, Farzad Eskandanian, Robin Burke, Weiwen Liu, and Bamshad Mobasher. 2020. Opportunistic Multi-aspect Fairness through Personalized Reranking. In *Proceedings of the 28th ACM Conference on User Modeling, Adaptation and Personalization*. 239–247.

[44] Harald Steck. 2018. Calibrated recommendations. In *Proceedings of the 12th ACM conference on recommender systems*. 154–162.

[45] Zhu Sun, Di Yu, Hui Fang, Jie Yang, Xinghua Qu, Jie Zhang, and Cong Geng. 2020. Are we evaluating rigorously? benchmarking recommendation for reproducible evaluation and fair comparison. In *Fourteenth ACM Conference on Recommender Systems*. 23–32.

[46] Özge Sürer, Robin Burke, and Edward C Malthouse. 2018. Multistakeholder recommendation with provider constraints. In *Proceedings of the 12th ACM Conference on Recommender Systems*. 54–62.

[47] Virginia Tsintzou, Evaggelia Pitoura, and Panayiotis Tsaparas. 2018. Bias Disparity in Recommendation Systems. arXiv:1811.01461 [cs.IR]

[48] Sriram Vasudevan and Krishnaram Kenthapadi. 2020. LiFT: A Scalable Framework for Measuring Fairness in ML Applications. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management* (Virtual Event, Ireland) *(CIKM '20)*. Association for Computing Machinery, New York, NY, USA, 2773–2780. https://doi.org/10.1145/3340531.3412705

[49] Chuhan Wu, Fangzhao Wu, Suyu Ge, Tao Qi, Yongfeng Huang, and Xing Xie. 2019. Neural news recommendation with multi-head self-attention. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. 6389–6394.

[50] Ke Yang and Julia Stoyanovich. 2017. Measuring fairness in ranked outputs. In *Proceedings of the 29th International Conference on Scientific and Statistical Database Management*. 1–6.

[51] Sirui Yao and Bert Huang. 2017. Beyond parity: Fairness objectives for collaborative filtering. In *Advances in Neural Information Processing Systems*. 2921–2930.

[52] Meike Zehlike, Francesco Bonchi, Carlos Castillo, Sara Hajian, Mohamed Megahed, and Ricardo Baeza-Yates. 2017. Fa* ir: A fair top-k ranking algorithm. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 1569–1578.

[53] Rich Zemel, Yu Wu, Kevin Swersky, Toni Pitassi, and Cynthia Dwork. 2013. Learning fair representations. In *International Conference on Machine Learning*. 325–333.

[54] Wayne Xin Zhao, Shanlei Mu, Yupeng Hou, Zihan Lin, Kaiyuan Li, Yushuo Chen, Yujie Lu, Hui Wang, Changxin Tian, Xingyu Pan, Yingqian Min, Zhichao Feng, Xinyan Fan, Xu Chen, Pengfei Wang, Wendi Ji, Yaliang Li, Xiaoling Wang, and Ji-Rong Wen. 2020. RecBole: Towards a Unified, Comprehensive and Efficient Framework for Recommendation Algorithms. , 23 pages.

[55] Cai-Nicolas Ziegler, Sean M McNee, Joseph A Konstan, and Georg Lausen. 2005. Improving recommendation lists through topic diversification. In *Proceedings of the 14th international conference on World Wide Web*. 22–32.