Comparative Recommender System Evaluation: Benchmarking Recommendation Frameworks

Alan Said[*]
TU-Delft
The Netherlands
alansaid@acm.org

Alejandro Bellogín[*]
Universidad Autónoma de Madrid
Spain
alejandro.bellogin@uam.es

ABSTRACT

Recommender systems research is often based on comparisons of predictive accuracy: the better the evaluation scores, the better the recommender. However, it is difficult to compare results from different recommender systems due to the many options in design and implementation of an evaluation strategy. Additionally, algorithmic implementations can diverge from the standard formulation due to manual tuning and modifications that work better in some situations.

In this work we compare common recommendation algorithms as implemented in three popular recommendation frameworks. To provide a fair comparison, we have complete control of the evaluation dimensions being benchmarked: dataset, data splitting, evaluation strategies, and metrics. We also include results using the internal evaluation mechanisms of these frameworks. Our analysis points to large differences in recommendation accuracy across frameworks and strategies, i.e. the same baselines may perform orders of magnitude better or worse across frameworks. Our results show the necessity of clear guidelines when reporting evaluation of recommender systems to ensure reproducibility and comparison of results.

Categories and Subject Descriptors: H.5.1 [Multimedia Information Systems]: Evaluation/methodology

General Terms: Experimentation, Documentation, Performance

1. INTRODUCTION

Recommender systems are a popular means of assisting users in online services, whether for music (Spotify, Last.fm), movies and videos (Netflix, YouTube) or other items (Amazon, eBay, etc.). In recent years, the recommender system-related research field has grown exponentially, and today most top-tier research venues feature tracks on recommendation (or closely related). There has been a parallel development in the industry and many of the positions in data science list knowledge of recommendation techniques as a top priority. This gain in popularity has led to an overwhelming amount of research literature over the last few years. With this in mind, it becomes increasingly important to be able to measure recommendation models *against each other* in order to objectively estimate their qualities. In today's recommender systems-related literature, a majority of papers state what datasets, algorithms, baselines and other potential settings are used in order to ensure replication, some manuscripts additionally present running times, hardware and software infrastructure, etc. In light of the vast variety of existing methods to implement and evaluate a recommender system, this is a very positive aspect. It should however be critically analyzed in order to ensure improvement in the field. It is known from Information Retrieval, that even when standard datasets are used and a well-established set of baselines are known, no overall improvement over the years is guaranteed [2].

Looking back on the accumulated amount of work in the research community, there have emerged three popular recommendation frameworks: *Apache Mahout* (Mahout) [23], *LensKit* [10] and *MyMediaLite* [12]. The latter two have been developed by researchers in academia, whereas Mahout is an Apache Software Foundation project. Even though the frameworks provide basically the same recommendation algorithms, they have differences in their implementations, data management, and evaluation methods. The frameworks provide basic evaluation packages able to calculate some of the most common evaluation metrics (e.g. precision, recall, root-mean-square error, etc.). However, due to the difference in implementation between the same algorithm across frameworks, even when using the same dataset, it becomes uncertain whether a comparison of results from different frameworks is possible.

To analyze if such a comparison between recommendation frameworks is indeed possible, we performed a standalone evaluation of recommendation results, allowing fine-grained control of a wide variety of parameters within the evaluation methodology. In this way, we aim to ensure an objective comparison of the recommendation accuracy under some common dimensions, such as the dataset used, the algorithms evaluated, the evaluation metrics analyzed and so on. Our work provides a cross-system and cross-dataset benchmark of some of the most common recommendation and rating prediction algorithms. It also highlights the differences in recommendation accuracy between implementations of the same algorithms on different frameworks, distinct levels of accuracy in different datasets, and variations of evaluation results on the same dataset and in the same framework when employing various evaluation strategies.

2. BACKGROUND

It lies in the nature of recommender systems that progress is measured by evaluating a system and benchmarking it towards other state of the art systems or algorithms. This is generally true for many machine learning-related approaches.

Recommender systems have experienced a steep evolution in the last decade, evaluation methods have seen a similar, although not as diverse development. Seminal works on recommender systems evaluation include Herlocker et al.'s survey [15] and Shani and Gunawardana's book chapter [29].

[*]Authors contributed equally to this work.

RecSys'14, October 6–10, 2014, Foster City, Silicon Valley, CA, USA.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2668-1/14/10 ...$15.00.
http://dx.doi.org/10.1145/2645710.2645746 .

Currently, recommender systems-related research literature commonly include evaluation sections. However, it still often remains unclear how the proposed recommendations perform compared to others. Given the fact that many recommendation approaches are researched, developed and deployed for specific purposes, it is imperative that the evaluation should deliver a clear and convincing overview of the systems' recommendation qualities, i.e. show in detail how the system was evaluated in order to make a comparison to other systems feasible. As an experiment, we studied all full-length (8 page, non-industry) research papers from the two, at the time of writing, most recent ACM RecSys conferences (i.e. after the release of MyMediaLite [12] and LensKit [10]) and analyzed how *reproducible* the results were, 56 papers in total. Reproducibility refers to the description of the experimental setup, the dataset used, the recommendation framework used, etc., in order to allow replication and validation of the results by a third party. Our analysis focused on four aspects of the contributions:

- **Dataset** - if the dataset used is publicly available. Excluding proprietary datasets or ones collected specifically for the paper from public sources (unless made available for download).
- **Recommendation framework** - if the recommendations were generated by openly available frameworks or if the source code was made available. Excluding software not available publicly or where nothing was mentioned.
- **Data details** - if a feasible reproduction of training/test splits is possible (irrespective of dataset) or whether not enough details were given for a successful replication.
- **Algorithmic details** - if algorithms were described in sufficient detail, e.g., neighborhood sizes for nearest neighbor (kNN) or number of factors for matrix factorization approaches.

We see these aspects related to the two issues identified by Konstan and Adomavicius [17] as being critical for making research *cumulative* and allowing *rapid progress*, i.e. ensuring reproducibility and replication by (*i*) thoroughly documenting the research and using open data, and (*ii*) following standards and using best-practice methods for coding, algorithms, data, etc.

In our study we found that only about half of the papers (30) used open datasets, a similar amount (35) presented information on training/test splits, making similar splits possible, even though a direct replication often remained unfeasible. It stands to reason that some organizations cannot share their data openly, making a detailed description of how the data is used even more important. In total, from 56 papers, only one used a combination of an open dataset, an open framework and provided all necessary details for replication of experiments and results. Even though a majority presented an evaluation and comparison towards a baseline, in most cases the information provided was not sufficient for accurate replication, i.e. only providing the algorithm type without additional details (e.g. kNN without neighborhood size, similarity metrics, etc), stating that the data were split into 80%-20% training/test sets but not providing the splitting strategy (per-user, per-item, random, etc.), or what actually constituted an accurate recommendation. Algorithmic details were fully disclosed in only 18 papers, which could perhaps be motivated by the fact that certain algorithmic aspects should be learned on the data (or withheld as part of an organization's intellectual property); there is however no clear reason for withholding data splitting details. We believe this can be due to lack of space and/or the perceived lower importance of specific details concerning the deployment and evaluation.

3. EVALUATION & RECOMMENDATION

This section presents the evaluation methodology and recommendation frameworks used in this paper.

3.1 Evaluation

To provide a fair comparison, we believe complete control over the evaluation protocol needs to be retained. Because of this, we have identified four stages that will be benchmarked using different alternatives available in the literature: *data splitting*, *item recommendation*, *candidate item generation*, and *performance measurement*.

3.1.1 Data splitting

One aspect of the evaluation setting is the dataset partition strategy. Different training and test partitions may have considerable impact on the recommendation performance. Although an exhaustive analysis of data partitioning is out of the scope of our work, we briefly discuss some common methods.

Time-based approaches [**?**, 14] consider the timeline of events, i.e. the timestamps of when user-item interactions were performed. A basic approach is to select a point in time to separate training data (all interaction records prior to that point) and test data (after the split time point). The split point can be set so as to, e.g. have a desired training/test ratio which can be global, with a single common split point for all users, or user-specific, to ensure the same ratio per user. None of the recommenders evaluated in the scope of this work take time into consideration, hence this is left as future work, where a wider set of recommenders will be tested.

If we ignore time, there are at least three strategies to withhold items: *a*) sample a fixed (different) number for each user; *b*) sample a fixed (same for all) number for each user, also known as *given n* or *all but n* protocols; *c*) sample a percentage of all interactions using *cross validation*, where non-overlapping subsets are generated, i.e. every (user, item, rating) tuple is evaluated once. Protocol *c*, appears to be the most popular [13, 28], although *all but n* (*b*) is also common [**?**]. We will use cross validation (cv) and test two variations: global splitting into (non-overlapping) folds (global cv), and each user's ratings list divided into the given number of folds and assigned to its corresponding split (per-user cv). We also try other strategies using a fixed number of sampled items, selected as a percentage of the total ratings (global ratio) or as a percentage of a user's ratings, partitioning on a user basis (per-user ratio).

Independent of the partition, the goals of an evaluation may be different in each situation, thus, a different setting (and partition) should be employed [14, 15]. If this is not the case, the results obtained in one setting could be invalid, i.e. evaluating something not sought for. As a result, the partition protocol in use needs to be clearly specified, and assessed if it is appropriate in the context. In this work we benchmark different instantiations of some of these protocols to provide a comparison. In practice the protocol should be selected such that it aligns with the evaluated approach and research goals – not necessarily the protocol obtaining best results.

3.1.2 Recommendation

Once the training and test splits are available, we produce recommendations using three common state-of-the-art methods from the recommender systems literature: user-based collaborative filtering (CF), item-based CF and matrix factorization. One reason behind this choice is the availability of the methods in the compared frameworks, which is not the case for less standard algorithms.

Item-based recommenders look at each item rated by the target user, and find items *similar* to that item [28, 30]. Item similarity is defined in terms of rating correlations between users, although cosine-based or probability-based similarities have also been proposed [8]. Adjusted cosine has been shown to perform better than other similarity methods [28]. The rating prediction computed by item-based strategies is generally estimated as follows [28]:

$$\tilde{r}(u,i) = C \sum_{j \in S_i} \text{sim}(i,j) r(u,j) \quad (1)$$

where S_i is the set of items similar to i [8]. We should note that S_i is generally replaced by I_u – all items rated by user u – since unrated items are assumed to be rated with a zero.

The user-based CF algorithm can be found in the literature under two different formulations; in the first one, rating deviations from the user's and neighbor's rating means are considered [24] (Eq. 2), whereas in the second one raw scores given by each neighbor are used [30] instead (Eq. (3)):

$$\tilde{r}(u,i) = \bar{r}(u) + C \sum_{v \in N_k(u)} \text{sim}(u,v) \left(r(v,i) - \bar{r}(v) \right) \quad (2)$$

$$\tilde{r}(u,i) = C \sum_{v \in N_k(u)} \text{sim}(u,v) r(v,i) \quad (3)$$

where $N_k(u)$ is a neighborhood of (typically) the k most similar users to u [9]; $\bar{r}(u)$ is the user's average rating. Note that both variations are referred to using the same name, making it difficult to *a priori* know which version is used.

The user similarity $\text{sim}(u,v)$ is often computed using the Pearson correlation coefficient or the cosine similarity between the user preference vectors [1]. We include here a shrunk version of these similarities, as proposed in [18]:

$$\text{sim}_s(a,b) = \frac{n_{a,b}}{n_{a,b} + \lambda_s} \text{sim}(a,b) \quad (4)$$

where $n_{a,b}$ is the number of users who rated both items (or items rated by both users), and λ_s is the shrinking parameter.

Matrix factorization recommenders use dimensionality reduction in order to uncover latent factors between users and items, e.g. by Singular Value Decomposition (SVD), probabilistic Latent Semantic Analysis or Latent Dirichlet Allocation. Generally, these systems minimize the regularized squared error on the set of known ratings to learn the factor vectors p_u and q_i [20]:

$$\min_{q^*,p^*} \sum_{(u,i)} \left(r(u,i) - q_i^T p_u \right)^2 + \lambda \left(\|q_i\|^2 + \|p_u\|^2 \right) \quad (5)$$

Since a closed formulation for these models also depends on how the parameters are learned for Eq. (5) we selected the same learning approach in each framework (when possible).

Table 1 shows the classes in each framework corresponding to user-, item- and matrix factorization-based recommenders together with their parameters. Note that FunkSVD [11] is not available in MyMediaLite, instead SVDPlusPlus [18] is used as it performs similarly to the other methods, although it requires a longer training time – due to the algorithm, not the implementation.

3.1.3 Candidate items generation

Once recommendations have been generated, the next step is to measure recommendation performance based on predictive accuracy (the error with respect to the predicted rating) and/or ranking-based precision. Both types have their limitations: the former needs the recommender to output a score meaningful as a rating, the latter is sensitive to the number of unknown items in the generated ranking, and how they are selected [4]. Regarding this aspect, the following candidate generation strategies, where L_u is for the set of target items the recommender ranks (candidate items), have been proposed:

UserTest This strategy takes the same target item sets as standard error-based evaluation: for each user u, the list L_u consists of items rated by u in the test set. The smallest set of target items for each user is selected, including no unrated items. A relevance threshold is used to indicate which of the items in the user's test are considered relevant. Threshold variations can be static for all users [16], or per-user [3].

TrainItems Every rated item in the system is selected – except those rated by the target user. This strategy is useful when simulating a real system where no test is available, i.e. no need to look into the test set to generate the rankings.

RelPlusN For each user, a set of highly relevant items is selected from the test set. Then, a set of non-relevant items is created by randomly selecting N additional items. In [6], N is set to $1,000$ stating that the non-relevant items are selected from items in the test set not rated by u. Finally, for each highly relevant item i, the recommender produces a ranking of the union between this item and the non-relevant items. Metrics for each user are calculated by averaging the values obtained for the rankings associated to each user over every highly relevant item. The final performance values are averaged over the values obtained for each user.

3.1.4 Performance measurement

The final stage in the proposed evaluation protocol is to measure the performance of the produced scores. The last step (candidate items generation) is not required for error-based metrics, e.g. mean absolute error (MAE) and root-mean-square error (RMSE) as these only consider predictions on items in the test set, whereas the recommendation step generates predictions for all items in the dataset. Nevertheless, it is crucial to report the coverage of the methods – in terms of user coverage (fraction of users that receive at least one recommendation) and catalog coverage (fraction of items that can be recommended) [29]. This is due to the fact that, in principle, no *backup* recommender is used as fallback, i.e. if we expect to receive 10 recommendations for a user but only get 8, we measure the performance based on those 8 items. This is a well-known problem [15] and the suggested solution is to report both accuracy and coverage.

Error-based metrics are useful under the assumption that a system that provides more accurate predictions is preferred by the user [29]. Although this has been studied and refuted by several authors [7,21], it still is one of the *de facto* evaluation techniques used.

Ranking-based metrics like precision, recall, nDCG, and mean reciprocal rank (MRR) aim to capture the quality of a particular ranking, taking into account that the user has expressed a preference towards some items, typically those in the test set rated above a certain threshold. The candidate item generation strategy will change the value computed with these metrics since a different ranking is evaluated, although the order of the recommenders does not have to change. Bellogín et al. [4] observed that one difference in the results is the range of the evaluation metrics. This becomes clear when using the RelPlusN strategy with different values of N, the larger the N, the smaller the value of the metric, as more items are considered in the ranking. In our experiments, after the stages described above, we used *trec_eval*[1] to produce ranking-based metrics.

3.2 Recommendation Frameworks

Our analysis is based on three open source recommendation frameworks popular in the recommender systems community; Mahout, MyMediaLite, and LensKit. Even though the frameworks have been developed with a similar goal in mind (an easy way to deploy recommender systems in research and/or production environments) and share some basic concepts, there are certain differences. These vary from how data is interacted with to differences in implementation across algorithms, similarity metrics, etc.

[1]Available at `http://trec.nist.gov/trec_eval/`

Framework	Class	Similarity	
Item-based			
LensKit	ItemItemScorer	CosineVectorSimilarity, PearsonCorrelation	
Mahout	GenericItemBasedRecommender	PearsonCorrelationSimilarity	
MyMediaLite	ItemKNN	Cosine, Pearson	
User-based			**Parameters**
LensKit	UserUserItemScorer	CosineVectorSimilarity, PearsonCorrelation	SimpleNeighborhoodFinder, NeighborhoodSize
Mahout	GenericUserBasedRecommender	PearsonCorrelationSimilarity	NearestNUserNeighborhood, neighborhood size
MyMediaLite	UserKNN	Cosine, Pearson	neighborhood size
Matrix factorization			
LensKit	FunkSVDItemScorer	IterationCountStoppingCondition, factors, iterations	
Mahout	SVDRecommender	FunkSVDFactorizer, factors, iterations	
MyMediaLite	SVDPlusPlus	factors, iterations	

Table 1: Details for the item-based, user-based and matrix factorization-based recommenders benchmarked in each framework. Namespace identifiers have been omitted due to space constraints.

This section highlights some of the features of the frameworks and gives an overview of differences and similarities between them, specifically what the frameworks provide in terms of evaluation and data management (test and training splitting), but also how certain algorithms are implemented.

3.2.1 LensKit

LensKit is a Java framework and provides a set of basic CF algorithms. The current version at the time of writing was 2.0.5.

To demonstrate some of the implementational aspects of LensKit, we look at a common similarity method, the Pearson Correlation. In LensKit, it is based on Koren and Bell's description [19] (stated in the source code), having the possibility to set the shrinkage parameter λ_s mentioned in Eq. (4). Additionally, LensKit contains an evaluator class which can perform cross validation and report evaluation results using a set of metrics. Available metrics include RMSE, nDCG, etc. (although it lacks precision and recall metrics).

3.2.2 Mahout

Mahout, also written in Java, had reached version 0.8 at the time of writing. It provides a large number of recommendation algorithms, both non distributed as well as distributed (MapReduce); in this work we focus solely on the former for the sake of comparison.

Mahout's Pearson Correlation calculates the similarity of two vectors, with the possibility to infer missing preferences in either. λ_s is not used, instead a weighting parameter suits a similar purpose.

There are several evaluator classes, we focus on the Information Retrieval evaluator (GenericRecommenderIRStatsEvaluator) which calculates e.g. precision, recall, nDCG, etc. Generally, the evaluators lack cross validation settings, instead, a percentage can be passed to the evaluator and only the specified percentage of (randomly selected) users will be evaluated. Additionally, the evaluator trains the recommendation models separately for each user. Furthermore, while preparing individual training and test splits, the evaluator requires a threshold to be given, i.e. the lowest rating to be considered when selecting items into the test set. This can either be a static value or calculated for each user based on the user's rating average and standard deviation. Finally, users are excluded from the evaluation if they have less than $2 * N$ preferences (where N is the level of recall).

3.2.3 MyMediaLite

MyMediaLite had, at the time of writing, reached version 3.10. It is implemented in C# and offers multi-platform support via Mono.

One key difference between MyMediaLite and the other two frameworks is that MyMediaLite addresses rating and item prediction as separate problems. However, it provides a rating-based recommendation setting similar to the other frameworks.

Dataset	Users	Items	Ratings	Density	Time
Movielens 100k	943	1,682	100,000	6.30%	'97-'98
Movielens 1M	6,040	3,706	1,000,000	4.47%	'00-'03
Yelp2013	45,981	11,537	229,907	0.04%	'05-'13

Table 2: Statistics of the three datasets used in this paper.

MyMediaLite's Pearson correlation class cites [19] as the basis for the implementation. Similar to LensKit, the shrinkage parameter λ_s can be used through a constructor. It is not used by default.

4. EXPERIMENTS & EVALUATION

In order to provide a comparison, we performed an analysis of results obtained using a protocol using the four steps from Section 3.1, and using the internal evaluation methods of each framework. All experiments were run on computers with Fedora Linux (v.18), 8Gb RAM and Intel Core2 Quad Q6600 2.40GHz CPUs. For Mahout and LensKit, Java (1.7.0_25-b15) was allowed to use 3Gb of RAM. MyMediaLite was run on Mono 3.2.7.

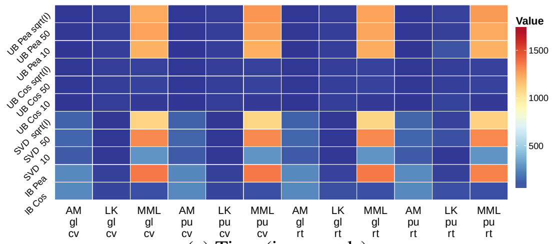
4.1 Datasets

We use three datasets, two from the movie domain (Movielens 100k & 1M[2]) and one with user reviews of business venues (Yelp[3]). All datasets contain users' ratings on items (1-5 stars). The selection of datasets was based on the fact that not all evaluated frameworks support implicit or log-based data. The first dataset, Movielens 100k (ML100k), contains 100,000 ratings from 943 users on 1,682 movies. The second dataset, Movielens 1M (ML1M), with 1 million ratings from 6,040 users on 3,706 items. The Yelp 2013 dataset contains 229,907 ratings by 45,981 users on 11,537 businesses. The datasets additionally contain item meta data, e.g. movie titles, business descriptions, these were not used in this work. A summary of data statistics is shown in Table 2.

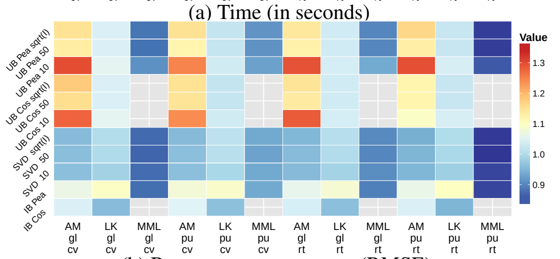
4.2 Controlled evaluation protocol

We used the Movielens 100k dataset to benchmark the four stages of the proposed controlled evaluation. More specifically, we tested the data splitting configuration experimenting with cross validation (5 folds) vs. random partition (80%-20% ratio), for both user and global splits (see Section 3.1.1) [26]. The recommendations were produced using the algorithms presented in Section 3.1.2, i.e., matrix factorization (SVD), user-based CF (UB) and item-based CF (IB) nearest neighbor. Then, error-based metrics were computed using the corresponding test splits and our own code (available in [26]), whereas the ranking-based metrics were calculated after running the different strategies to generate candidate items using the *trec_eval* tool; in this case, we considered an item relevant if it had been rated with a 5, similar to Cremonesi et al. [6]. For the RelPlusN strategy, N was set to 100, a lower value than the typical one (i.e., 1,000)

[2]`http://grouplens.org/datasets/movielens/`
[3]`http://www.kaggle.com/c/yelp-recsys-2013`



(a) Time (in seconds)



(b) Root-mean-square error (RMSE)

Figure 1: Time and RMSE for the controlled evaluation protocol. IB and UB refer to item- and user-based respectively; Pea and Cos to Pearson and Cosine; gl and pu to global and per user; AM, LK, MML to the frameworks; and cv, rt to cross validation and ratio. (NB: To be viewed in color.)

but it allows us to perform more exhaustive experiments, while keeping the comparison fair (since this parameter applies evenly to all evaluated recommenders). After that, we further extended our experiments with the other datasets, for a subset of the configurations used in the experiments with Movielens 100k.

4.3 Framework-dependent protocols

In addition to the controlled evaluation, we used each framework's internal evaluation methods. This was performed for Movielens 100k[4], default evaluation settings were used, e.g. the frameworks created the training/test splits based on default configurations. LensKit and MyMediaLite performed 5-fold cross validation, in Mahout, the recommender generates training/test splits individually for each user. Since there is no common evaluation metric calculated in these frameworks, we present nDCG values for LensKit and Mahout, and RMSE values for LensKit and MyMediaLite. The algorithms used were according to Table 1. In Mahout's case, the evaluator needs the level of recall (N) prior to recommendation and evaluation (see Section 3.2.2), thus we set $N = 50$ for Mahout. In the case of MyMediaLite, where it distinguishes between rating prediction and item recommendation, we used the rating prediction setting to generate predictions and recommendations.

5. RESULTS

Note that due to space constraints we do not present the results from all runs. Instead focus is on presenting a full coverage of the main benchmarking dataset, Movielens 100k, with additional results from the other datasets for completeness.

5.1 Controlled evaluation protocol

Figs. 1 and 2 show a subset of the most important metrics computed using the controlled evaluation protocol. The figures show a wide combination of strategies for data splitting, recommendation, and candidate items generation, along with prediction accuracy (RMSE Fig. 1b), ranking quality (nDCG@10 Fig. 2c), coverage (Figs. 2a and 2b), and running times of the different recommenders (Fig. 1a). Although the same gradient is used in all the figures, we have to note that for coverage and nDCG higher values (red) are better, whereas for time and RMSE lower values (blue) are better.

[4]Attempts to use the Yelp2013 dataset were also made. The running time and memory usage of the frameworks made it unfeasible.

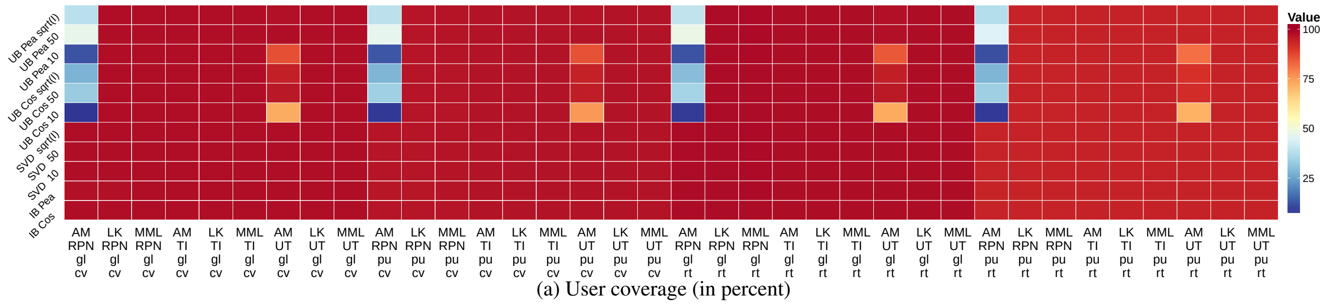
In terms of running time, we see in Fig. 1a that the most expensive algorithm is MyMediaLite's item-based method (using Pearson correlation), regardless of splitting strategy. MyMediaLite's factorization algorithm needs more time than SVD in other frameworks, this is due to the base factorizer method being more accurate but also more computationally expensive (see Section 3.1.2). In general, we observe that LensKit has the shortest running time no matter the recommendation method or data splitting strategy. Note that for datasets with other characteristics this can differ.

Looking at the RMSE values, we notice (Fig. 1b) a gap in the MyMediaLite recommenders. This is due to MyMediaLite distinguishing between rating prediction and item recommendation tasks (Section 3.2.3). In combination with the similarity method used (cosine), this means the scores are not valid ratings. Moreover, MyMediaLite's recommenders outperform the rest. There is also difference between using a cross validation strategy or a ratio partition, along with global or per user conditions. Specifically, better values are obtained for the combination of global cross validation, and best RMSE values are found for the per user ratio partition. These results can be attributed to the amount of data available in each of these combinations: whereas global splits may leave users out of the evaluation (the training or test split), cross validation ensures that they will always appear in the test set. Performance differences across algorithms are negligible, although SVD tends to outperform others within each framework.

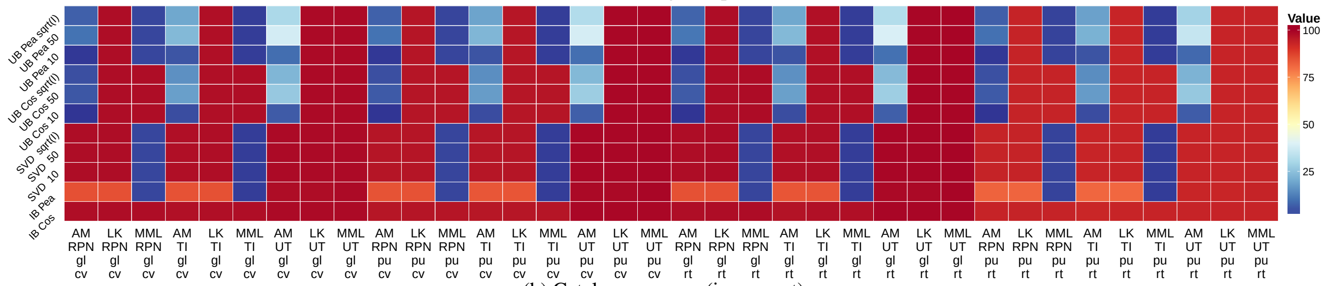
Finally, the ranking-based evaluation results should be carefully analyzed. We notice that, except for the UserTest candidate items strategy, MyMediaLite outperforms Mahout and LensKit in terms of nDCG@10. This high precision comes at the expense of lower coverage, specifically of the catalog (item) coverage. As a consequence, MyMediaLite seems to be able to recommend at least one item per user, but far from the complete set of items, in particular compared to the other frameworks. This is specifically noticeable in rating-based recommenders (as we recalled above, all except the UB and IB with cosine similarity), whereas the ranking-based approach obtains coverage similar to other combinations. In terms of nDCG@10, the best results are obtained with the UserTest strategy, with noticeable differences between recommender types, i.e. IB performs poorly, SVD performs well, UB in between, in accordance with e.g. [19]. The splitting strategy has little effect on the results in this setting.

Tables 3b and 3c show results for the Movielens 1M and Yelp2013 datasets respectively, where only a global cross validation splitting strategy is presented as we have shown that the rest of strategies do not have a strong impact on the final performance. We also include in Table 3a the corresponding results from Figs. 1 and 2 for comparison. In the tables we observe that most of the conclusions shown in Figs. 1 and 2 hold in Movielens 1M, likely due to both datasets having a similar nature (movies from Movielens, similar sparsity), even though the ratio of user vs. items and the number of ratings is different (see Table 2). For instance, the running time of IB is longest for Mahout; and Mahout and MyMediaLite have coverage problems when random items are considered in the evaluation (i.e., RPN item candidate strategy), in particular in the UB recommender. LensKit's performance is superior to Mahout's in most of the situations where a fair comparison is possible (i.e., when both frameworks have a similar coverage), while at the same time MyMediaLite outperforms LensKit in terms of nDCG@10 for the RPN strategy, although this situation is reversed for the UT strategy.

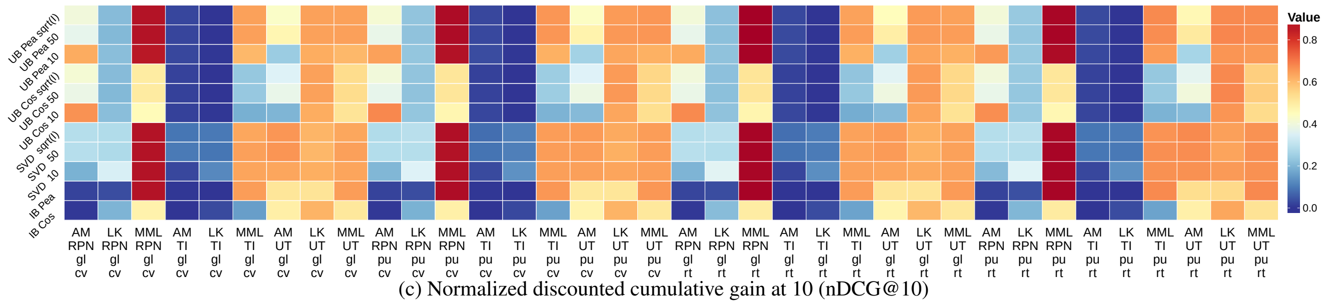
The majority of the results for the dataset based on Yelp reviews are consistent with those found for Movielens 1M. One exception is the IB recommender, where the LensKit implementation is slower and has a worse performance (in terms of RMSE and nDCG) than Mahout. This may be attributed to the very low density of this dataset, which increases the difficulty of the recommendation prob-



(a) User coverage (in percent)



(b) Catalog coverage (in percent)



(c) Normalized discounted cumulative gain at 10 (nDCG@10)

Figure 2: User and catalog coverage, nDCG and MAP for the controlled evaluation. RPN, TI and UT refer to RelPlusN, TrainItems and UserTest strategies (other acronyms from Fig. 1). (NB: To be viewed in color.)

lem; nonetheless it should be noted that LensKit is able to achieve a much higher coverage for some recommenders – especially for the user-based – than Mahout. The performance of MyMediaLite is also affected in this dataset, where we were not able to obtain recommendations for three of the methods: two of them required too much memory, and the other required too much time to finish. For the recommenders that finished, the results are better for this framework, especially compared to LensKit, the other one with full coverage with the RPN strategy.

5.2 Framework-dependent protocols

The results of the framework-dependent evaluation are shown in Table 4. Table 4a shows evaluation results in terms of nDCG, generated by Mahout and LensKit, whereas Table 4b shows the RMSE values from LensKit and MyMediaLite. We start by studying the results presented in Table 4a, where it seems that LensKit outperforms Mahout at several orders of magnitude. The highest nDCG obtained by Mahout (0.2868) is less than one third of the lowest value obtained by LensKit (0.9422). This should be taken in the context of each framework's evaluator. Recall that Mahout's evaluator will only consider users with a certain minimum number of preferences (two times the level of recall). Our level of recall was set to 50, meaning only users with at least 100 preferences are evaluated (corresponding only to circa 33% of the users in this dataset).

Looking at the RMSE results obtained by LensKit and MyMediaLite in Table 4b, the difference between the frameworks is not as large as in the previous case. All RMSE results are between 7.5% (UBCos50) and 11.7% (UBCos10) of each other. In this case, both frameworks created five instances of training/tests splits with an 80%-20% ratio. The splits are randomly seeded, meaning that even though the frameworks only create five training/test datasets, they are not the same between the two frameworks.

A further observation to be made is how the framework's internal evaluation compares to our controlled evaluation. We see that the frameworks perform better in the controlled environment than in the internal ditto. This could be the effect of different ways of calculating the RMSE, i.e. how the average is computed (overall or per user), or what happens when the recommender cannot provide a rating.

In the case of nDCG (Table 4a), we see that Mahout's values fluctuate more in both versions of the controlled evaluation (RPN and UT) than in Mahout's own evaluation. The internal evaluation results consistently remain lower than the corresponding values in the controlled setting – although the RPN values are closer to Mahout's own results. The UT (UserTest) values obtained in the controlled evaluation are several orders of magnitude higher than in the internal setting, even though the setting resembles Mahout's own evaluation closer than the RPN setting. A link to the different results could potentially be the user and item coverage. Recall that Mahout's internal evaluation only evaluates users with more than $2 * N$ preferences, whereas the controlled setting evaluates the full set of users. LensKit's internal evaluation consistently shows better results than the controlled setting. We believe this could be an effect related to how the final averaged nDCG metric is calculated (similar to the mismatch in RMSE values mentioned above).

Given the widely differing results, it seems pertinent that in order to perform an inter-framework benchmarking, the evaluation process needs to be clearly defined and both recommendations and evaluations performed in a controlled and transparent environment.

(a) Movielens 100k

Alg.	F.W.	Time	RMSE	nDCG@10		User cov.(%)		Cat. cov.(%)	
		(sec.)		RPN	UT	RPN	UT	RPN	UT
IBCos	AM	238	1.041	0.003	0.501	98.16	100	99.71	99.67
	LK	44	0.953	0.199	0.618	98.16	100	99.88	99.67
	MML	75	NA	0.488	0.521	98.16	100	100	99.67
IBPea	AM	237	1.073	0.022	0.527	97.88	100	86.66	99.31
	LK	31	1.093	0.033	0.527	97.86	100	86.68	99.31
	MML	1,346	0.857	0.882	0.654	98.16	100	2.87	99.83
SVD50	AM	132	0.950	0.286	0.657	98.12	100	99.88	99.67
	LK	7	1.004	0.280	0.621	98.16	100	100	99.67
	MML	1,324	0.848	0.882	0.648	98.18	100	2.87	99.83
UBCos50	AM	5	1.178	0.378	0.387	35.66	98.25	6.53	27.80
	LK	25	1.026	0.223	0.657	98.16	100	99.88	99.67
	MML	38	NA	0.519	0.551	98.16	100	100	99.67
UBPea50	AM	6	1.126	0.375	0.486	48.50	100	10.92	39.08
	LK	25	1.026	0.223	0.657	98.16	100	99.88	99.67
	MML	1,261	0.847	0.883	0.652	98.18	100	2.87	99.83

(b) Movielens 1M

Alg.	F.W.	Time	RMSE	nDCG@10		User cov.(%)		Cat. cov.(%)	
		(sec.)		RPN	UT	RPN	UT	RPN	UT
IBCos	AM	23,027	1.028	0.002	0.534	100	100	99.88	99.99
	LK	1,571	0.906	0.225	0.627	100	100	100	100
	MML	1,350	NA	0.525	0.515	100	100	100	100
IBPea	AM	23,148	0.972	0.019	0.578	99.98	100	94.55	99.97
	LK	1,832	1.052	0.091	0.593	99.98	100	94.59	99.97
	MML	> 5 days	–	–	–	–	–	–	–
SVD50	AM	9,643	0.858	0.337	0.692	100	100	100	100
	LK	89	0.879	0.200	0.660	100	100	100	100
	MML	25,987	0.804	0.909	0.677	100	100	2.55	100
UBCos50	AM	118	1.123	0.544	0.421	31.36	99.43	3.04	20.37
	LK	2,445	0.957	0.293	0.676	100	100	100	100
	MML	1,046	NA	0.542	0.553	100	100	100	100
UBPea50	AM	149	1.077	0.513	0.504	38.88	99.98	5.05	25.96
	LK	2,408	0.957	0.293	0.676	100	100	100	100
	MML	181,542	0.807	0.906	0.660	100	100	2.55	100

(c) Yelp2013

Alg.	F.W.	Time	RMSE	nDCG@10		User cov.(%)		Cat. cov.(%)	
		(sec.)		RPN	UT	RPN	UT	RPN	UT
IBCos	AM	737	1.146	0.109	0.761	85.32	80.60	70.05	90.01
	LK	5,955	1.231	0.065	0.824	100	100	100	100
	MML	30,718	0.229	0.668	100	100	100	100	
IBPea	AM	712	1.577	0.350	0.659	61.23	63.07	17.33	68.19
	LK	558	1.236	0.281	0.690	66.84	71.38	21.62	71.97
	MML	> 5 days	–	–	–	–	–	–	–
SVD50	AM	404	1.125	0.181	0.865	100	100	100	97.57
	LK	483	1.250	0.114	0.732	100	100	100	100
	MML	995	1.131	0.995	0.930	100	100	2.53	100
UBCos50	AM	967	1.204	0.569	0.662	43.81	76.57	5.85	32.38
	LK	4,298	1.149	0.090	0.863	100	100	99.99	97.57
	MML	ME	–	–	–	–	–	–	–
UBPea50	AM	774	1.307	0.643	0.531	29.95	46.31	3.06	29.67
	LK	4,311	1.149	0.090	0.863	100	100	99.99	97.57
	MML	ME	–	–	–	–	–	–	–

Table 3: Performance results in Movielens 100k, 1M and Yelp2013 with the controlled evaluation protocol with cross validation as splitting strategy for the frameworks (F.W) column.

6. DISCUSSION

In the light of the previous section, it stands clear that even though the evaluated frameworks implement the recommendation algorithms in a similar fashion, the results are not comparable, i.e. the performance of an algorithm implemented in one cannot be compared to the performance of the same algorithm in another. Not only do there exist differences in algorithmic implementations, but also in the evaluation methods themselves.

There are no *de facto* rules or standards on how to evaluate a recommendation algorithm. This also applies to how recommendation algorithms of a certain type should be realized (e.g., default parameter values, use of backup recommendation algorithms, and other *ad-hoc* implementations). However, this should perhaps not be seen as something negative *per se*. Yet, when it comes to performance comparison of recommendation algorithms, a standardized (or controlled) evaluation is crucial [17]. Without which, the relative performance of two or more algorithms evaluated under different conditions becomes essentially meaningless. In order to objectively

(a) nDCG for AM and LK

Alg.	**F.W.**	**nDCG**
IBCos	AM	0.000414780
	LK	0.942192050
IBPea	AM	0.005169231
	LK	0.924546132
SVD50	AM	0.105427298
	LK	0.943464094
UBCos50	AM	0.169295451
	LK	0.948413562
UBPea50	AM	0.169295451
	LK	0.948413562

(b) RMSE values for LK and MML.

Alg.	**F.W.**	**RMSE**
IBCos	LK	1.01390931
	MML	0.92476162
IBPea	LK	1.05018614
	MML	0.92933246
SVD50	LK	1.01209290
	MML	0.93074012
UBCos50	LK	1.02545490
	MML	0.95358984
UBPea50	LK	1.02545490
	MML	0.93419026

Table 4: Results using the internal evaluation methods of each framework.

and definitively characterize the performance of an algorithm, a controlled evaluation, with a defined evaluation protocol is a prerequisite.

To date, the most well-known recommender system-related event, the Netflix Prize[5], created a benchmarking environment where all participating algorithms were evaluated in the same controlled setting. It is debatable whether the research advances accomplished during the Prize's three-year run would have been matched if there had been no standardized and controlled evaluation and benchmarking environment. Having said this, we should also stress the importance of research conducted outside the scope of such actions. However, when it comes to recommender systems, given that much of the progress is measured in terms of higher precision or recall, lower RMSE, higher nDCG, etc., it seems intuitive that some form of controlled evaluation could lead to a broader understanding of recommender system algorithms' qualities in general.

As a note, we should mention that the Netflix Prize focused on improving Netflix' internal rating prediction algorithm by 10% in terms of RMSE. Even though this comparison is by no means fair (in the scope of the Netflix Prize at least), we have shown that RMSE values between the same algorithms implemented in different frameworks can often differ by more than 10%.

We have highlighted that there exist large discrepancies in recommender algorithm quality, not only for different algorithms, but specifically for the same algorithms implemented in different popular recommender system frameworks. Our analysis further shows that not only do the disparities exist in common metrics, but additionally in metrics which are perhaps more seldom used for recommender systems-related research purposes, e.g. user and catalog coverage as well as the time needed to train the algorithms. Given the recent investigation on how well common metrics actually correspond to users' taste, e.g. RMSE [22,27], precision [21], etc. we believe that it is crucial to not only report the accuracy of recommender algorithms, but the evaluation methodology as well, in order to ensure reported progress is not only due to the specific implementation details of a particular framework. Especially, given the large amount of recommender systems-related research being conducted, and the variety of open source frameworks as well as individually implemented code, it is necessary to not only describe in detail the algorithms used, but also the complete evaluation process, including strategies for data splitting, specifics on the evaluation metrics and methods, etc.

For example, by working with the Yelp dataset we observed that not all the frameworks are able to deal equally with too sparse data; in particular this dataset is not too large (around 200k ratings, one fifth of the Movielens 1M dataset) but it contains a larger number of items and users than the other compared datasets. This condition made it impossible for us to run some of the algorithms implemented in the frameworks.

[5]`http://www.netflixprize.com`

7. CONCLUSION & FUTURE WORK

We have shown the disparity of evaluation results between three common recommendation frameworks. Even though the frameworks implement similar (if not the same) algorithms, there exist large differences in the reported recommendation quality, both in the frameworks internal evaluations as well as in the external evaluations performed in the scope of this work. Using three popular and publicly available datasets, we evaluated three types of recommendation algorithms (user-based and item-based nearest neighbor CF and SVD-based matrix factorization). We evaluated the results using each framework's internal evaluation modules as well as through a *controlled evaluation* – the latter case ensuring that no implementation-specific factors affected the results of the evaluation. Our analysis shows that, even in a controlled environment, the same algorithms evaluated on identical data show performance discrepancies. These discrepancies range from minor to quite significant, e.g. *catalog coverage* ranging between 3% and 100% for the same algorithm and setting in two different frameworks (Table 3a UBPea50, MML vs. LK), or a 33% difference in RMSE (Table 3b UBPea50, AM vs. MML). Similarly, our analysis of running times show large differences between the frameworks, e.g. a factor of 1,200 between AM and MML for user-based nearest-neighbor using Pearson (Table 3b). The differences in evaluation results using the internal evaluation methods provided by the frameworks show even larger discrepancies, e.g. IBCos AM vs. LK (Table 4), where nDCG differs a factor of 2,300 (0.00041 vs. 0.9422).

In light of the presented results, it stands clear that the evaluation and presentation of recommendation results should be given in ample detail in order to ensure a fair comparison to the baseline algorithms used. The presented work shows that inter-framework comparisons of recommendation quality can potentially point to incorrect results and conclusions, unless performed with great caution and in a controlled, framework independent environment.

We are currently performing further studies on evaluation discrepancies in recommender systems and attempting to limit the effects of this by developing an open source recommender systems evaluation toolkit[6] [26] (which was also used in this work). With this toolkit, our attempt is to formalize a set of standardized evaluation methodologies, independent of specific recommendation or evaluation strategy. In the future, we aim to extend our work to other evaluation dimensions (e.g., diversity, novelty), along with considering implicit feedback datasets and other contextual dimensions – like time and social networks – in the evaluation protocols.

8. ACKNOWLEDGMENTS

The authors thank Zeno Gantner and Michael Ekstrand for support during this work. This work was partly carried out during the tenure of an ERCIM "Alain Bensoussan" Fellowship Programme. The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreements n°246016 and n°610594, and the Spanish Ministry of Science and Innovation (TIN2013-47090-C3-2).

9. REFERENCES

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. Knowl. Data Eng.*, 17(6):734–749, 2005.

[2] T. G. Armstrong, A. Moffat, W. Webber, and J. Zobel. Improvements that don't add up: ad-hoc retrieval results since 1998. In *CIKM*, pages 601–610, 2009.

[3] C. Basu, H. Hirsh, and W. W. Cohen. Recommendation as classification: Using social and content-based information in recommendation. In J. Mostow and C. Rich, editors, *AAAI/IAAI*, pages 714–720. AAAI Press / MIT Press, 1998.

[6]http://rival.recommenders.net

[4] A. Bellogín, P. Castells, and I. Cantador. Precision-oriented evaluation of recommender systems: an algorithmic comparison. In *RecSys*, pages 333–336, 2011.

[5] J. S. Breese, D. Heckerman, and C. M. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. volume abs/1301.7363, 2013.

[6] P. Cremonesi, Y. Koren, and R. Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *RecSys*, pages 39–46, 2010.

[7] P. Cremonesi, A. Sansottera, and S. Gualandi. On the cooling-aware workload placement problem. In *AI for Data Center Management and Cloud Computing*, 2011.

[8] M. Deshpande and G. Karypis. Item-based top-n recommendation algorithms. *ACM Trans. Inf. Syst.*, 22(1):143–177, Jan. 2004.

[9] C. Desrosiers and G. Karypis. A comprehensive survey of neighborhood-based recommendation methods. In Ricci et al. [25], pages 107–144.

[10] M. D. Ekstrand, M. Ludwig, J. A. Konstan, and J. Riedl. Rethinking the recommender research ecosystem: reproducibility, openness, and lenskit. In *RecSys*, pages 133–140, 2011.

[11] S. Funk. Netflix update: Try this at home. http://sifter.org/~simon/journal/20061211.html (retrieved Jan. 2014), Dec 2006.

[12] Z. Gantner, S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme. Mymedialite: A free recommender system library. In *RecSys*, pages 305–308, New York, NY, USA, 2011. ACM.

[13] K. Goldberg, T. Roeder, D. Gupta, and C. Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Inf. Retr.*, 4(2):133–151, July 2001.

[14] A. Gunawardana and G. Shani. A survey of accuracy evaluation metrics of recommendation tasks. *J. Mach. Learn. Res.*, 10:2935–2962, Dec. 2009.

[15] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1):5–53, Jan. 2004.

[16] T. Jambor and J. Wang. Optimizing multiple objectives in collaborative filtering. In *RecSys*, pages 55–62, New York, NY, USA, 2010. ACM.

[17] J. A. Konstan and G. Adomavicius. Toward identification and adoption of best practices in algorithmic recommender systems research. In *RepSys*, pages 23–28, New York, NY, USA, 2013. ACM.

[18] Y. Koren. Factorization meets the neighborhood: A multifaceted collaborative filtering model. In *KDD*, pages 426–434, New York, NY, USA, 2008. ACM.

[19] Y. Koren and R. Bell. Advances in collaborative filtering. In Ricci et al. [25], pages 145–186.

[20] Y. Koren, R. M. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *IEEE Computer*, 42(8):30–37, 2009.

[21] S. M. McNee, J. Riedl, and J. A. Konstan. Being accurate is not enough: how accuracy metrics have hurt recommender systems. In *CHI Extended Abstracts*, pages 1097–1101, 2006.

[22] T. T. Nguyen, D. Kluver, T.-Y. Wang, P.-M. Hui, M. D. Ekstrand, M. C. Willemsen, and J. Riedl. Rating support interfaces to improve user experience and recommender accuracy. In *RecSys*, RecSys '13, pages 149–156, New York, NY, USA, 2013. ACM.

[23] S. Owen, R. Anil, T. Dunning, and E. Friedman. *Mahout in Action*. Manning Publications Co., Greenwich, CT, USA, 2011.

[24] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *CSCW*, pages 175–186, 1994.

[25] F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, editors. *Recommender Systems Handbook*. Springer, 2011.

[26] A. Said and A. Bellogín. Rival – a toolkit to foster reproducibility in recommender system evaluation. In *RecSys*, 2014.

[27] A. Said, B. J. Jain, S. Narr, and T. Plumbaum. Users and noise: The magic barrier of recommender systems. In *UMAP*, pages 237–248. Springer, 2012.

[28] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *WWW*, pages 285–295, 2001.

[29] G. Shani and A. Gunawardana. Evaluating recommendation systems. In Ricci et al. [25], pages 257–297.

[30] U. Shardanand and P. Maes. Social information filtering: Algorithms for automating "word of mouth". In *CHI*, pages 210–217, 1995.