A Troubling Analysis of Reproducibility and Progress in Recommender Systems Research

Maurizio Ferrari Dacrema
Politecnico di Milano

Paolo Cremonesi
Politecnico di Milano

Dietmar Jannach
Alpen-Adria-Universität Klagenfurt, Austria

A Troubling Analysis of Reproducibility and Progress in Recommender Systems Research

MAURIZIO FERRARI DACREMA, SIMONE BOGLIO, and PAOLO CREMONESI, Politecnico di Milano, Italy
DIETMAR JANNACH, University of Klagenfurt, Austria

The design of algorithms that generate personalized ranked item lists is a central topic of research in the field of recommender systems. In the past few years, in particular, approaches based on deep learning (neural) techniques have become dominant in the literature. For all of them, substantial progress over the state-of-the-art is claimed. However, indications exist of certain problems in today's research practice, e.g., with respect to the choice and optimization of the baselines used for comparison, raising questions about the published claims. In order to obtain a better understanding of the actual progress, we have tried to reproduce recent results in the area of neural recommendation approaches based on collaborative filtering. The worrying outcome of the analysis of these recent works—all were published at prestigious scientific conferences between 2015 and 2018—is that 11 out of the 12 reproducible neural approaches can be outperformed by conceptually simple methods, e.g., based on the nearest-neighbor heuristics. None of the computationally complex neural methods was actually consistently better than already existing learning-based techniques, e.g., using matrix factorization or linear models. In our analysis, we discuss common issues in today's research practice, which, despite the many papers that are published on the topic, have apparently led the field to a certain level of stagnation.[1]

CCS Concepts: • **Information systems** → **Recommender systems**; *Collaborative filtering*; • **General and reference** → Evaluation.

Additional Key Words and Phrases: Recommender Systems, Deep Learning, Evaluation; Reproducibility

ACM Reference Format:
Maurizio Ferrari Dacrema, Simone Boglio, Paolo Cremonesi, and Dietmar Jannach. -. A Troubling Analysis of Reproducibility and Progress in Recommender Systems Research. *ACM Transactions on Information Systems* Under review, 0, Article 0 (-), 41 pages. https://doi.org/-

1 INTRODUCTION

Personalized recommendations are a common feature of many modern online services, e.g., on e-commerce, media streaming, and social media sites. In many cases, these recommendations are generated using *collaborative filtering* (CF) techniques [6]. Such techniques leverage the preference or activity profiles of a large user community to predict which are the most relevant items for the individual customer. Early technical approaches to build collaborative filtering recommender systems (RS) were based on nearest neighbor techniques and date back to the 1990s [34, 50]. Soon,

[1] This paper significantly extends or own previous work presented in [18].

Authors' addresses: Maurizio Ferrari Dacrema, maurizio.ferrari@polimi.it; Simone Boglio, simone.boglio@mail.polimi.it; Paolo Cremonesi, paolo.cremonesi@polimi.it, Politecnico di Milano, Italy, Milano; Dietmar Jannach, University of Klagenfurt, Klagenfurt, Austria, dietmar.jannach@aau.at.

1046-8188/-/0-ART0 $15.00
https://doi.org/-

all sorts of machine learning approaches were proposed for the *top-n* recommendation task, which is to rank the available items according to their assumed relevance for the user. In the most recent years, recommendation methods based on deep learning (DL) or "neural" technology have become particularly popular. While these methods are often computationally complex, their solid success in other application areas—such as language or image processing—led to deep learning techniques nowadays dominating the recommender systems research landscape.

However, there are a number of indications that using increasingly deeper learning methods is not as beneficial as one would expect. For example, in two recent papers, [33] and [67], the authors report that for a certain information retrieval task recent neural methods are actually not better than long-existing non-neural ones. In the domain of time series prediction, Makridakis et al. [39] compared various statistical methods with traditional and recent neural methods and observed that statistical methods were mostly favorable over even the most recent machine learning techniques. In the area of recommender systems, the empirical analyses in [37] and [38] showed that sometimes almost trivial methods can outperform the latest neural methods for the task of session-based recommendations. These findings are, however, not tied to recent deep learning approaches alone. The observation that the reported improvements in the literature for certain retrieval tasks "don't add up" was put forward back in 2009 [3]. Most often, the reason for such "phantom progress" lies in the choice and poor optimization of the baselines used in the experiments. For the domain of rating prediction for recommender systems, Rendle et al. in [49] found that many algorithms that were published between 2015 and 2019 actually did not outperform longer-existing methods, when these were properly tuned.

In their paper, Rendle et al. attribute the problem not only to poorly-optimized baselines but also the sometimes missing standardized benchmarks. In some sense, this finding is surprising since the research community over the last decades has indeed developed certain standards regarding the evaluation of *top-n* recommendation algorithms. Typically, the input to an algorithm is a matrix of user-item interactions and the task is to compute relevance scores for the missing entries or just to rank the items. A number of public datasets exists that can be used for reproducible evaluation, as well as various well-established evaluation metrics. Finally, the technical details of newly proposed algorithms are often laid out in great detail in scientific papers and sometimes authors even share the code they used in their experiments.

This methodological approach for benchmarking algorithms seems very solid at first sight and suitable to determine if an algorithm is favorable over another in a specific combination of (i) performance measure, (ii) evaluation procedure and (iii) dataset, at least when we assume that both algorithms are properly optimized. However, the claims made in many research papers are much more general. Many papers claim a significant improvement over the "state-of-the-art", but do not explicitly state such claim to be only supported under very specific experimental conditions. In today's recommender systems research scholarship the researcher has ample freedom in selecting the specific experimental conditions, i.e., which metrics, which protocol, which datasets, and which baselines to use. Furthermore, since research in applied machine learning is often driven by the "hunt" for the best model [61] and not by theoretical or practical considerations, usually authors do not have to justify their choices.

Given these observations regarding potential methodological issues in our field, we were wondering to what extent recent deep learning methods actually help to achieve progress in creating *top-n* recommendation lists based on user-item rating matrices. To that purpose, we conducted an extensive set of experiments in which we tried to reproduce the results reported in papers that were recently published in top-tier scientific conferences. We scanned the proceedings of several conference series and identified 26 relevant papers. We could reproduce 12 (46%) of them with reasonable effort. After fine-tuning a number of established baselines algorithms, it turned out that in 8 of the reproducible cases, simple and long-known approaches (e.g., based on nearest neighbors) outperformed the latest neural methods. Furthermore, when including established linear models based on machine learning (e.g., based on matrix factorization), we found that only 1 out of 12 recent neural methods was clearly better than the "non-neural" baselines, but only on one dataset. Overall, these results indicate certain signs of stagnation in the context of applying machine learning methods for *top-n* recommendation tasks, despite many papers published every year claiming substantial progress.

The rest of the paper is organized as follows. Next, in Section 2, we describe how we selected relevant works considered in our study. Section 3 provides details about our research methodology and Section 4 lists our results in detail. The potential reasons for the observed phenomena are finally discussed in Section 5.

2 IDENTIFYING RELEVANT AND REPRODUCIBLE WORKS

In order to make sure our research is neither focusing on a few hand-picked examples nor singling out any individual researcher, we followed a systematic approach to identify recent papers that are relevant for our analysis. To obtain a suitable sample for our research, we considered papers which fulfilled the following constraints.

(1) The paper proposed a new neural collaborative filtering recommendation method for *top-n* recommendation tasks. Hybrid techniques with a collaborative filtering component were considered as well. We limited ourselves to works that focus on the traditional item ranking task. Papers that dealt with other tasks, e.g., session-based recommendation or group recommendation, were not considered to be in the scope of our study. Furthermore, to be considered, a paper had to report at least one ranking or classification accuracy metric. Papers that focused solely on rating or relevance prediction tasks, e.g., using the RMSE, were not taken into account.
(2) The paper was published between 2015 and 2018 in one of the following conferences: SIGIR, KDD, TheWebConf (WWW), IJCAI, WSDM, ACM RecSys. All of the mentioned conferences are typical outlets for recommender systems research in computer science. Furthermore, all of them, except ACM RecSys, are rated with A* in the Australian CORE ranking system. ACM RecSys, in contrast, is entirely focused on recommender systems, which is why we included it in the study as well. We identified relevant papers by scanning the proceedings of these conference series in a manual process.
(3) The work was reproducible, with reasonable effort, based on source code published by the authors.[2] For some of the relevant papers we found that some source code was published but was missing major parts, e.g. it consisted only in a skeleton of the algorithm, or did not work. For those papers that were deemed relevant but for which the source code was not publicly available or was not runnable, we contacted all authors of the papers by e-mail. When there was no positive response after 30 days, we considered the source code of the paper to be not available.
(4) At least one of the datasets that were used in the original paper for evaluation was publicly available. In some cases, the authors also provided the data splits, train and test, that were used in their experiments. If the data splits were not available, we considered papers as

[2] In theory, research papers should contain all relevant information that are needed to implement the proposed algorithm. In reality, however, certain details are sometimes omitted in length-restricted conference papers. For that reason, and to ensure that the results that we report here are reliable, we followed a conservative approach and limited ourselves to the papers where the original authors themselves provided an implementation of their method.

reproducible only if they contained sufficient information about data preprocessing and splitting.[3]

Following this approach, we identified **26** relevant papers. Of these, **12** were considered reproducible according to our classification scheme. Table 1 summarizes which works were considered relevant and which ones were reproducible.

Table 1. Statistics of relevant and reproducible works on deep learning algorithms for *top-n* recommendation per conference series from 2015 to 2018.

Conference	Rep. ratio	Reproducible	Non-Reproducible
KDD	3/4 (75%)	[26], [31], [62]	[57]
IJCAI	5/7 (71%)	[22], [69], [68], [11], [66]	[44], [65]
WWW	2/4 (50%)	[23], [32]	[56], [17]
SIGIR	1/3 (30%)	[16]	[41], [10]
RecSys	1/7 (14%)	[70]	[55], [7], [52], [58], [28], [60]
WSDM	0/1 (0%)		[64]
Total	12/26 (46%)		

Our first contribution in our present work is therefore an analysis of the reproducibility—at least when using our specific practical definition—of research works published on neural collaborative filtering. We generally found that the share of papers that can be reproduced based on the provided source code by the authors is still relatively low. When looking at the statistics over the years, we can observe a certain trend towards authors sharing their source code more often. One possible reason is that reproducibility in general is considered a positive point in the reviewing process, e.g., at KDD.

3 EVALUATION METHODOLOGY

The core of our study was to re-run the experiments reported in the original papers following the original experimental methodology, including additional baseline methods which were systematically fine-tuned just like the newly proposed methods.

To ensure the reproducibility of this study, we share all the data, the source code used for pre-processing, hyperparameter optimization, algorithms, and the evaluation as well as all hyperparameter values and results online.[4]

3.1 Measurement Approach

Our analysis of the relevant papers shows that researchers use all sorts of datasets, evaluation protocols, and metrics in their experiments, see also Section 5.2. To make our analyses and comparisons as fair as possible, we decided to run our evaluations in exactly the same way as the authors of the originally proposed method did, i.e., using their datasets, their protocol, and their performance metrics. To obtain a broader understanding of the model performance, we also included additional baselines.[5]

[3] In case we encountered problems with the provided code, the data, or the reproduction of the results, we also contacted the authors for assistance.

[4] The code will be be made publicly available on GitHub upon acceptance. We share the code during the review phase here. https://polimi365-my.sharepoint.com/:u:/g/personal/10322330_polimi_it/ES7kUo4X5XZAuvAFQRaE2J0BkHKCQ0jBEfufgsaNawTkSg?e=6gCJoE

[5] An alternative would have been to integrate all methods in one unified framework for evaluation, as done in [37], and evaluate them on a set of representative datasets. This approach would allow a direct comparison of neural approaches as in [38], which was however not the goal of our work.

In order to ensure all algorithms are evaluated under the same conditions, we re-factored the original code so that we could include it in our evaluation framework along with the additional baselines. The core algorithm implementations remained unaltered. We evaluated the algorithms using the datasets reported in the original papers, provided that they were either publicly available or shared by the authors. We also used the original train/test split whenever the authors shared it, otherwise we created the data split ourselves following the information provided in the original paper.

Extensive hyperparameter optimization was performed for all examined baselines. For the investigated neural methods, in all but one case we relied on the optimal hyperparameters reported in the original papers. This is appropriate as we used the same datasets and evaluation procedures in the experiments. The only exception is the SpectralCF algorithm (Section 4.12), for which we performed a new hyperparameter optimization ourselves due to an issue with the provided dataset splits, as will be discussed later. Since the number of epochs and the stopping criteria are usually not described in the original papers, for all machine learning models we select the number of epochs via *early-stopping* (Section 3.3).

The optimal hyperparameters were selected via a Bayesian search [2, 19, 24], using the Scikit-Optimize[6] implementation. We considered 50 cases for each algorithm during this search. The first 15 of them were used as initial random points. Once the optimal hyperparameters were determined, including the number of epochs, the final model was fitted on the union of train and validation data using those optimal hyperparameters. The considered hyperparameter ranges and distributions are listed in Appendix A.

3.2 Baselines

Over the last 25 years, a multitude of algorithms of different types were proposed. In order to obtain a picture that is as broad as possible, we selected algorithms of different families for inclusion in our measurements. An overview of all used baselines is given in Table 2 and the relative hyperparameter ranges are reported in Appendix A.

3.2.1 Popularity-Based Ranking. Recommending the most popular items to everyone is a common strategy in practice. The method **TopPopular** implements this non-personalized recommendation approach. The popularity of an item is determined by its number of implicit or explicit ratings in the given dataset.

3.2.2 Nearest-Neighbor Methods. Nearest-neighbor techniques were used in the early GroupLens system [50] and first successful reports of collaborative filtering systems also used nearest-neighbor techniques [34]. We consider both *user-based* and *item-based* variants, **UserKNN** and **ItemKNN**.

Many variants of the basic nearest-neighbor prediction scheme were proposed over the years, see [9] for an early performance comparison. In this work, we therefore consider different variations of the nearest-neighbor techniques as well. For both UserKNN and ItemKNN, the following hyperparameters can be set and were optimized in our experiments, their ranges are reported in Appendix A.

- *Neighborhood Size*: This main parameter determines how many neighbors are considered for prediction.
- *Similarity Measure*: We made experiments with the Jaccard coefficient [48] as well as Cosine [53], Asymmetric Cosine [1], Dice-Sørensen [15] and Tversky [59] similarities. Some of these similarity measures also have their own parameters, as reported in Appendix A, which we optimized as well.

[6] https://scikit-optimize.github.io/

Table 2. Overview of Baseline Methods

Family	*Method*	*Description*
Non-personalized	TopPopular	Recommends the most popular items to everyone [14]
Nearest-Neighbor	UserKNN	User-based k-nearest neighbors [50]
	ItemKNN	Item-based k-nearest neighbors [53]
Graph-based	$P^3\alpha$	A graph-based method based on random walks [12]
	$RP^3\beta$	An extension of $P^3\alpha$ [47]
Content-Based and Hybrid	ItemKNN-CBF	ItemKNN with content-based similarity [36]
	ItemKNN-CFCBF	A simple item-based hybrid CBF/CF approach [43]
	UserKNN-CBF	UserKNN with content-based similarity
	UserKNN-CFCBF	A simple user-based hybrid CBF/CF approach
Non-Neural Machine Learning	iALS	Matrix factorization for implicit feedback data [27]
	pureSVD	A basic matrix factorization method [14]
	SLIM	A scalable linear model [30, 45]
	$EASE^R$	A recent linear model, similar to auto-encoders [54]

- *Shrinkage*: As proposed in [5], we used a parameter (the *shrink term*) to lower the similarity between items that have only few interactions in common. The shrinkage is applied to all similarities.
- *Feature Weighting*: Using feature weighting for ratings was proposed in [63]. In our experiments, we both tested configurations with no weighting and weighting with either the TF-IDF or the BM25 scheme.
- *Normalization*: This setting determines if we should consider the denominator in the similarity measure as normalization. Only some of the similarity measures have this parameter.

3.2.3 *Graph-based Methods.*

Traditional nearest-neighbor models consider "direct" neighborhoods by computing similarities between pairs of objects. Graph-based models can help to overcome this possible limitation relying on a broader interpretation of neighborhoods. In our study, we consider two such graph-based methods called $P^3\alpha$ [12] and $RP^3\beta$ [47]. Both methods often lead to good recommendation quality at low computational cost. Interestingly, these two methods appear to be almost unknown in the community and seldom used as baselines, despite the fact that they are very simple, effective and have been published in top-tier venues.

- **$P^3\alpha$**: This method implements a two-steps random walk from users to items and vice-versa, where the probabilities to jump between users and items are computed from the normalized ratings raised to the power of α. The method is equivalent to a KNN item-based CF algorithm, with the similarity matrix being computed as the dot-product of the probability vectors [12]. In addition to what described in the original algorithm, we normalize each row of the similarity matrix with its $l1$ norm. The hyperparameters of the algorithm include the size of the neighborhood and the value for α.

- ***RP*^{3}*β***: This is an improved version of P$^3\alpha$ proposed in [47]. In RP$^3\beta$, each similarity between two items is computed with P$^3\alpha$ and divided by the popularity of the items raised to the power of β. Again, we normalize each row of the similarity matrix with its $l1$ norm. If β is 0, RP$^3\beta$ is equivalent to P$^3\alpha$. The hyperparameters of the algorithm are the size of the neighborhood and the values for α and β.

3.2.4 Content-based and hybrid Methods. Some of the neural methods investigated in this paper include side information about items or users. We have therefore included two simple baselines that make usage of content information.

- **ItemKNN-CBF**, **UserKNN-CBF**: A neighborhood-based content-based-filtering (CBF) approach, where we compute the item (or user) similarities based on the items' (or user's) content features (attributes) [36]. We tested the same set of similarity measures described for the collaborative KNN methods (Jaccard coefficient, Cosine, Asymmetric Cosine, Dice-Sørensen and Tversky similarity). The hyperparameters are the same as for the ItemKNN and UserKNN methods.
- **ItemKNN-CFCBF**, **UserKNN-CFCBF**: A hybrid algorithm based on item-item (or user-user) similarities and described in [43]. The similarity between items is computed by first concatenating, for each item, the vector of implicit ratings (collaborative features) and the vector of item attributes (content features) and by later computing the similarity between the concatenated vectors. In case of user-user similarities the algorithm operates in a similar way, concatenating the vector of implicit ratings of each user with the user's content feature vector. The hyperparameters and similarity measures are the same as for ItemKNN, plus a parameter w that controls the relative importance of the content features with respect to the collaborative features. When w is 0, this algorithm is equivalent to the pure collaborative versions, either ItemKNN or UserKNN.

3.2.5 Non-Neural Machine Learning Approaches. Countless machine learning models were proposed for *top-n* recommendation tasks in the literature. In our experiments, we included a number of comparably basic models from the literature as representatives of which methods were often considered the state-of-the-art in pre-neural times.

- *Matrix Factorization (MF) Techniques*: The application of matrix decomposition methods for collaborative filtering problems was investigated already in the early years of recommender systems [8], and became a de-facto standard after the Netflix prize competition (2006-2009). We made experiments with many variants, but will limit our discussion to two main techniques which proved to consistently lead to competitive results among the different MF techniques.
  - **iALS**: In their seminal work [27], Hu et al. proposed an *Alternating Least Squares* approach for implicit feedback datasets, which turns implicit feedback signals into confidence values. The authors also proposed a particular optimization method that has the advantage of scaling well on larger datasets. A number of hyperparameters can be tuned for the method, including the number of latent factors, the confidence scaling and the regularization factor.
  - **PureSVD**: This method corresponds to a basic matrix factorization approach as proposed in [14]. To implement PureSVD, we used a standard SVD decomposition method provided in the `scikit-learn` package for Python. [7] The only hyperparameter of this method is the number of latent factors.
- *Sparse Linear Models (SLIM)*: SLIM was proposed as a well-performing regression-based method for *top-n* recommendation tasks in [45]. In our work, we use the more scalable variant proposed in [30] (**SLIM ElasticNet**) which learns the item similarity matrix one item

[7] https://scikit-learn.org/stable/modules/generated/sklearn.utils.extmath.randomized_svd.html

at a time (e.g. one column w at a time) by solving a regression problem in such a way that the interactions for the target item y are learned by using all other interactions as training data. To implement *SLIM ElasticNet* we used a standard ElasticNet solver provided in the `scikit-learn` package for Python.[8] The hyperparameters of this method include the ratio of *l1* and *l2* regularizations as well as a regularization magnitude coefficient.

- ***EASE***^{R}: In a recent article [54] the author showed that an "embarrassingly shallow" linear model, which shares similarities with an auto-encoder, can produce highly-accurate recommendations that often outperform existing and much more complex techniques. A peculiarity of this model is the existence of a closed-form solution for the training objective which results in very fast training. The only hyperparameter is the choice of the regularization factor. This algorithm has been published in 2019 and, as such, the papers covered by our study could not include EASE^R as a baseline. However, we include EASE^R to investigate whether shallow auto-encoders are able to provide, on average, more accurate recommendations with respect to complex deep-learning architectures.

3.3 Early Stopping Approach

Many machine learning models are trained for a *number of epochs* in which the model's performance varies, first increasing and then stabilizing, while usually exhibiting some degree of variance. The number of epochs therefore represents another important parameter to be determined. However, it is worth noting that in the articles we have analyzed neither the number of epochs nor the stopping criteria are usually mentioned. The procedure in which this parameter was chosen in the original articles is therefore not clear. Looking at the code shared by the authors we could observe that, in some cases, the number of epochs was inappropriately selected via an evaluation done on the test data, therefore causing information leakage from the test data. In other cases, the reported metric values were inappropriately taken from different epochs.

Early stopping is a widely used technique to select the optimal number of train epochs and is available in many libraries like Keras.[9] The idea of early stopping is to periodically evaluate the model on the validation data, while the model is being trained, and stop the training when for a certain number of validation steps the model quality has not improved over the best solution found so far. Early stopping has the advantage of selecting the number of epochs with a transparent criterion, avoiding arbitrary manual optimization, and often results in shorter training times.

To implement early stopping, we use two independent copies of the current model. One is the model that is still being trained, the other is the model frozen at the epoch with the best recommendation quality found so far. If the trained model, after further epochs, exhibits better recommendation quality than the best one found so far, the best model is updated. Since the evaluation step is time consuming, we run five train epochs before each validation step. Moreover, we choose to stop the training if for 5 consecutive validation steps the recommendation quality of the current model is worse than the best one found so far.

4 RESULTS

In this section, we summarize the main observations of our experiments. For each analyzed method, we describe *(i)* the basic idea of the method; *(ii)* the baseline algorithms and datasets that were used in the original paper; *(iii)* the outcomes reported in the original work; *(iv)* our results after including and optimizing additional baselines.

[8] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ElasticNet.html

[9] For early stopping in Keras, see https://keras.io/callbacks/#earlystopping

The experimental evaluation reported in this paper required significant computational effort. Considering all baselines and reproducible deep learning algorithms, due to the number of different datasets and preprocessing procedures, we report the recommendation quality of more than 900 models. When taking into account the hyperparameter tuning procedure, 41,000 models were fitted, corresponding to a total computation time of 253 days.[10]

The analyzed papers were, as mentioned, published between 2015 and 2018. We organize the discussion of the papers mostly by year of publication. Table 3 summarizes the main findings. An overview of the temporal development and the dependencies between the approaches can be found in Figure 1.

Fig. 1. Overview of Neural Methods, arrows indicate when a newer method used another one as baseline in the experiments.

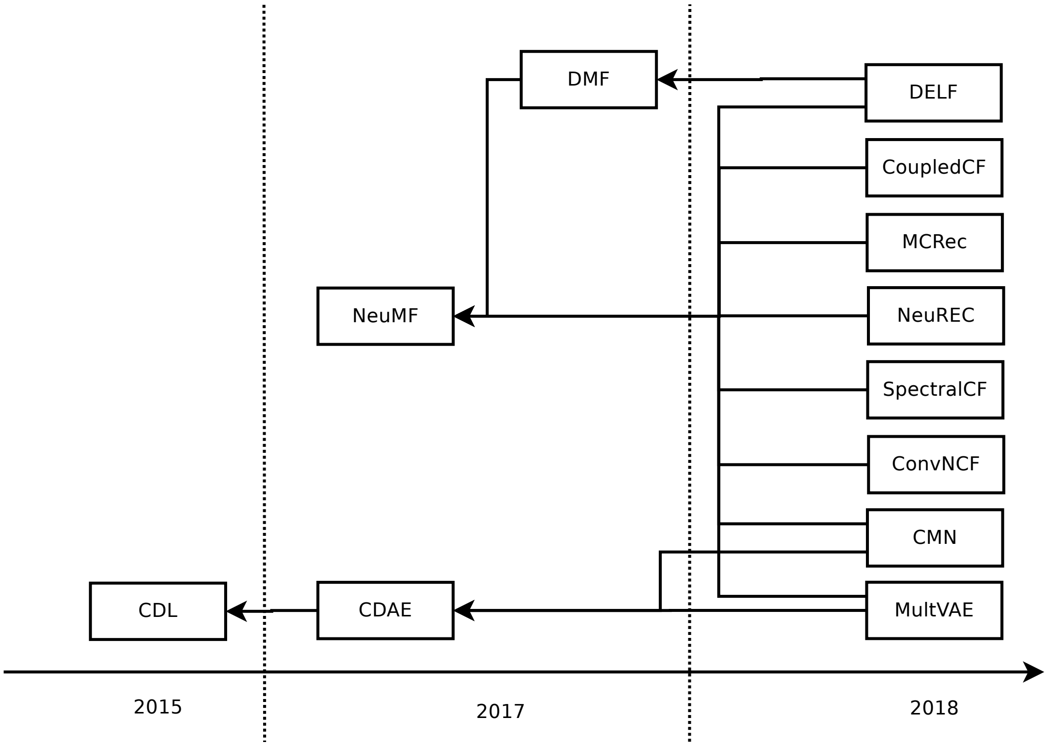


Table 3. Overview of the obtained results

Paper	Conference	Summary of Comparison with Baselines
Collaborative Deep Learning for Recommender Systems (CDL) [62]	KDD '15	Simple hybrid methods outperform CDL in three out of four dataset configurations; CDL is only better on one small and very sparse dataset with *one* interaction per user in the training set.

[10] The computation time refers to the total instance time for one AWS instance p3.2xlarge, with 8 vCPU, 30GB RAM, and one Tesla V100-SXM2-16GB GPU. The detailed measurements are available in the online material (Section 3)

Collaborative Variational Autoencoder for Recommender Systems [31]	KDD ’17	Improvements over the CDL method are observed, but CVAE is still outperformed by simple hybrids in all but one configuration with only one interaction per user in the training set.
Neural Collaborative Filtering [23]	WWW ’17	Simple methods outperform the the proposed algorithm on one dataset and non-neural machine learning outperforms it on the other, for all metrics.
Deep Matrix Factorization Models for Recommender Systems [66]	IJCAI ’17	The proposed method is outperformed on three out of four datasets by long-known baselines. In one and very sparse dataset, the proposed method was better than our baseline methods, in particular with respect to the Hit Rate.
Variational Autoencoders for Collaborative Filtering [32]	WWW ’18	The proposed method indeed outperforms all our baselines consistently on one of two datasets. For the other dataset, the SLIM method was better when the optimization target was the same as the evaluation measure (NDCG).
NeuRec: On Nonlinear Transformation for Personalized Ranking [69]	IJCAI ’18	The method is consistently outperformed by many of our baselines on three commonly used datasets. Only on one small dataset and for one individual measure at a short list length the proposed method is slightly better than our baselines.
CoupledCF: Learning Explicit and Implicit User-item Couplings in Recommendation for Deep Collaborative Filtering [68]	IJCAI ’18	For one MovieLens dataset, the non-deep machine learning models were consistently better than the proposed method. For another dataset, even nearest neighbor methods were preferable.
DELF: A Dual-Embedding based Deep Latent Factor Model for Recommendation [11]	IJCAI ’18	The method was consistently outperformed by an established implicit-feedback baseline (iALS). For one of the datasets, several existing baselines were always better than the proposed method.
Outer Product-based Neural Collaborative Filtering [22]	IJCAI ’18	Traditional nearest-neighbor methods are consistently better than the proposed method on one dataset and better except for one measurement on the other.
Leveraging Meta-path based Context for *top-n* Recommendation with a Neural Co-attention Model [26]	KDD ’18	On the single dataset where the results could be reliably reproduced, the method is outperformed both by traditional nearest-neighbor approaches and by existing non-neural models.
Collaborative Memory Network for Recommendation Systems [16]	SIGIR ’18	The proposed method was competitive against personalized baselines on one of the datasets, but outperformed by a non-personalized one. On two other datasets, the proposed method falls behind the baselines.

Spectral Collaborative Filtering [70]	RecSys '18	A major issue regarding the evaluation was spotted. After correction, the proposed method does not work as expected, leading to non-competitive performance.

4.1 Collaborative Deep Learning for Recommender Systems (CDL)

Method. CDL is the earliest method in our analysis, published at KDD '15 [62]. CDL is a hybrid method which applies deep learning to jointly learn a deep representation of content information and collaborative information. Technically, it is a probabilistic feed-forward model for joint learning of a stacked denoising autoencoder and collaborative filtering.

Datasets. The evaluation in the original paper is based on three datasets. Two of them are data samples that were previously collected from the *CiteULike* platform and which were already used in earlier research. One dataset is based on rating data from the Netflix Prize competition, augmented with content information by the authors. In our evaluation, we considered only the CiteULike datasets, because the content information used in combination with the Netflix dataset is not publicly available. Two versions of the CiteULike dataset were considered, a dense version *CiteULike-a* and a sparser one *CiteULike-t.* Both datasets are relatively small (135k and 205k interactions, respectively). For each of these datasets, experiments were made in two sparsity configurations. These configurations are described by a parameter P, which defines how many interactions per user are left in the training set (with the rest going to the test set). For parameter P, values 1 and 10 were reported, which correspond to 5.5k and 55.5k training interactions, respectively. Note that with $P = 1$ there is only one training interaction per user in the training dataset.

Evaluation. Several baseline techniques were explored, among them a number of hybrid matrix factorization approaches, a content-based deep learning technique designed for music recommendation, as well as Collaborative Topic Regression, a method combining Latent Dirichlet Allocation on the content and collaborative filtering on the interactions. For evaluation purposes, P interactions for each user were randomly sampled to be part of the training set as mentioned above. The average results of five evaluation runs are reported. The authors report Recall for comparably long list lengths (50 to 300), and Mean Average Precision for list length 300.

Results and Discussion. The authors found their method to outperform all baselines on all measures. We could reproduce their results based on the provided code and dataset.[11] To optimize the baselines in our own evaluation, we used 20% of the training set as a validation set.[12] After optimization, our results show that CDL—in three out of four configurations (CiteULike-a with *P=10* and CiteUlike-t with *P=1* and *P=10*)—was consistently outperformed by our simple hybrid technique (ItemKNN CFCBF) and, in many cases, also by the pure content-based method (ItemKNN CBF). Only when removing all but one user interaction from the CiteULike-a dataset (with *P=1*) CDL was, by a large extent, better than any of our baselines. In particular, in the settings where *P=1*, pure collaborative filtering techniques were, as expected, not competitive.

[11] In the source code the authors provided it is reported that the original evaluation contained an error such that the absolute values of the evaluation metrics was higher than the correct one, although the relative performance ordering of the algorithms remained unaltered. Once this error is fixed we can reproduce their results.

[12] Information about the validation set size was not provided in the original paper. In the evaluation scenario where *P=1*, due to the presence of only 1 training instance per user, any sampling would result in cold users. Therefore, in this scenario, the validation data is also contained in the train data. In the evaluation scenario where *P=10*, training and validation data are disjoint.

Table 4 shows exemplary results for the CiteULike-a dataset (*P=10*), with about 55k interactions in the training dataset. Detailed results for all datasets can be found in the online appendix (Section 3). In the table, we highlight in bold those entries where a baseline outperformed CDL. We can observe that, for shorter and much more typical list lengths, even the simplest collaborative filtering approaches outperform CDL. The iALS method based on matrix factorization for implicit feedback data was better with respect to CDL in all measurements and cutoff lengths. Finally, the best results were achieved with the pure content-based method that uses only item features to recommend similar items (ItenKNN CBF).

Table 4. Selected Results for the CDL Method on the CiteULike-a dataset with *P=10*.

	a 10					
	REC@50	REC@100	REC@150	REC@200	REC@250	REC@300
TopPopular	0.0040	0.0078	0.0103	0.0204	0.0230	0.0258
UserKNN CF jaccard	**0.0818**	**0.1206**	0.1471	0.1700	0.1885	0.2023
ItemKNN CF cosine	**0.0989**	**0.1435**	**0.1751**	0.1976	0.2147	0.2289
P$^3\alpha$	**0.0910**	**0.1340**	**0.1633**	0.1867	0.2054	0.2200
RP$^3\beta$[13]	**0.0958**	**0.1398**	**0.1682**	0.1904	0.2088	0.2231
EASE^R	**0.0835**	**0.1242**	0.1528	0.1771	0.1956	0.2100
SLIM	**0.0869**	**0.1280**	**0.1558**	0.1783	0.1963	0.2107
PureSVD	**0.0716**	0.1079	0.1313	0.1491	0.1636	0.1760
iALS	**0.0781**	**0.1388**	**0.1835**	**0.2186**	**0.2473**	**0.2706**
ItemKNN CBF cosine	**0.2235**	**0.3180**	**0.3829**	**0.4283**	**0.4651**	**0.4950**
ItemKNN CFCBF TFIDF cosine	**0.1858**	**0.2816**	**0.3445**	**0.3930**	**0.4335**	**0.4642**
CollaborativeDL	0.0580	0.1108	0.1546	0.1946	0.2314	0.2640

From a methodological perspective, there is no indication of why comparably long list lengths were used for evaluation in the paper and why no measurements were reported for list lengths below 50, which is commonly the case in the literature.

4.2 Collaborative Variational Autoencoder (CVAE)

Method. Like CDL, the *CVAE* method [31] is a hybrid technique that relies both on content information and collaborative features for recommending. The work was published at KDD '17. Technically, the model learns deep latent representations from content data in an unsupervised manner and also considers implicit relationships between items and users from both content and rating. Unlike previous works with denoising criteria, CVAE learns a latent distribution for content in the latent space instead of the observation space through an inference network.

Datasets and Evaluation. The CVAE method is evaluated in the same way as the CDL approach, i.e., two datasets from *CiteULIke* are used and different sparsity configurations are evaluated. Likewise, the authors of CVAE measure Recall at different (long) list lengths. As an additional baseline, the authors include the CDL [62] method described in the previous section. The hyperparameters for all baseline methods were optimized using a validation set which is, however, not described.

[13] We report RP$^3\beta$ [47] for completeness although the DL algorithm we evaluate here predates its publication.

Results and Evaluation. We could reproduce the results for CVAE.[14] Table 5 shows the results of our experiments for the CiteULike-a dataset with *P=10*, again using the same evaluation measures and protocol as used in the original paper.

Table 5. Experimental results for the CVAE method for the CiteULike-a with *P=10.*

	REC@50	REC@100	REC@150	REC@200	REC@250	REC@300
TopPopular	0.0040	0.0078	0.0103	0.0204	0.0230	0.0258
UserKNN CF jaccard	**0.0818**	0.1206	0.1471	0.1700	0.1885	0.2023
ItemKNN CF cosine	**0.0989**	0.1435	0.1751	0.1976	0.2147	0.2289
$P^3\alpha$	**0.0910**	0.1340	0.1633	0.1867	0.2054	0.2200
$RP^3\beta$[15]	**0.0958**	0.1398	0.1682	0.1904	0.2088	0.2231
$EASE^R$	**0.0835**	0.1242	0.1528	0.1771	0.1956	0.2100
SLIM	**0.0869**	0.1280	0.1558	0.1783	0.1963	0.2107
PureSVD	0.0716	0.1079	0.1313	0.1491	0.1636	0.1760
iALS	0.0781	0.1388	0.1835	0.2186	0.2473	0.2706
ItemKNN CBF TFIDF cosine	**0.2235**	**0.3180**	**0.3829**	**0.4283**	**0.4651**	**0.4950**
ItemKNN CFCBF TFIDF cosine	**0.1858**	**0.2816**	**0.3445**	**0.3930**	**0.4335**	**0.4642**
CollaborativeDL	0.0580	0.1108	0.1546	0.1946	0.2314	0.2640
CollaborativeVAE	0.0805	0.1569	0.2232	0.2760	0.3250	0.3687

The baseline results shown in Table 5 are identical to those of Table 4—as they were done on the same dataset and with the same evaluation protocol—except that Table 5 has an additional row for the results for the CVAE method. Again, the simple hybrid baselines outperform the more complex CVAE method on all measures on this dataset. We can, however, observe that CVAE is indeed consistently better than the CDL method, which is the main baseline method in [31]. For the other dataset and sparsity configurations, our results are similar to what was reported in the previous section on CDL.

Overall, the authors of CVAE could show an advance with respect to CDL, but our results indicate that CDL did not represent a strong baseline method. In the remainder of this paper, we will observe the following phenomenon several times: a neural method is introduced as improving the state-of-the-art, and subsequent works only focus on outperforming this new neural method, without considering alternative baselines.

4.3 Neural Collaborative Filtering (NCF)

Method. The *Neural network-based Collaborative Filtering* (*NCF*) [23] framework was presented at WWW '17 and rapidly became very influential, being used as a baseline for most later neural recommendation approaches, as shown in Figure 1. The framework generalizes matrix factorization in a way that the commonly used inner product is replaced by a neural architecture, which can learn different types of functions from the data and therefore can also model non-linearities. Different variants are considered in the paper: Generalized Matrix Factorization, Multi-Layer Perceptron, and Neural Matrix Factorization, where the last one, called NeuMF is an ensemble of the other two. In our evaluation, we only consider Neural Matrix Factorization, because this method led to the best results.

[14] In the source code the authors provided it is reported that the original evaluation contained an error such that the absolute values of the evaluation metrics was higher than the correct one, although the relative performance ordering of the algorithms remained unaltered. Once this error is fixed we can reproduce their results.

[15] We report $RP^3\beta$ [47] for completeness although the DL algorithm we evaluate here predates its publication.

Datasets. Two datasets were used for evaluating the method, one rating dataset from MovieLens (*MovieLens1M*) and one dataset with implicit feedbacks from *Pinterest.* The Pinterest dataset was pre-processed by removing all users with less than 20 interactions. After pre-processing, the dataset contained 1.5 million interactions. For the MovieLens rating dataset, all 1 million ratings were transformed to 1 to mimic an implicit-feedback dataset, with missing entries transformed to 0.

Evaluation. The authors use a leave-last-out procedure to evaluate their method. For each user, the last interaction (based on its timestamp) is put into the test set. The resulting data splits used in the experiments are shared by the authors. To avoid to compute scores for all recommendable items, which is considered too time-consuming by the authors even for datasets of modest size, the performance of the algorithms is measured by determining how the last hidden item is ranked within 100 randomly sampled other items. Hit Rate and NDCG at list length 10 are used as performance metrics.

As personalized baselines, the authors include Matrix Factorization with Bayesian Personalized Ranking (BPR matrix factorization), the eALS method from 2016 and the ItemKNN method. The original hyperparameter optimization is done on a validation set obtained by randomly selecting one interaction per user. For the ItemKNN method, the number of neighbors was varied, but no other configurations were tested by the authors (e.g., shrink term or normalization). According to the reported experiments, the NCF method, and in particular the NeuMF variant, outperforms all baselines on all dataset on all performance measures.

Results and Discussion. We could reproduce the reported results. However, the analysis of the provided source code shows that the number of training epochs was chosen by maximising the Hit Rate on the test data. Since the number of epochs is a parameter like any other, it must be fixed before testing, e.g., through early stopping on a validation set. In our experiments, we therefore report the performance measure for the number of epochs that was considered optimal based on the validation set.

Table 6 and Table 7 report our results for both the MovieLens and Pinterest datasets. We report the results for list length 10, as in the original paper. Since the authors in [23] also plot the results at different list lengths from 1 to 10, we also include measurements at list lengths 1 and 5 for comparison purposes.

Table 6. Experimental results for NCF (MovieLens 1M)

	HR@1	NDCG@1	HR@5	NDCG@5	HR@10	NDCG@10
TopPopular	0.1051	0.1051	0.3048	0.2064	0.4535	0.2543
UserKNN CF asymmetric	0.1925	0.1925	0.5081	0.3548	0.6781	0.4102
ItemKNN CF asymmetric	0.1765	0.1765	0.4906	0.3369	0.6608	0.3917
$P^3\alpha$	0.1791	0.1791	0.4846	0.3352	0.6460	0.3876
$RP^3\beta$[16]	0.1836	0.1836	0.4935	0.3419	0.6758	0.4011
$EASE^R$	**0.2119**	**0.2119**	**0.5502**	**0.3857**	**0.7098**	**0.4374**
SLIM	**0.2207**	**0.2207**	**0.5576**	**0.3953**	**0.7162**	**0.4468**
PureSVD	**0.2132**	**0.2132**	0.5339	0.3783	0.6937	0.4303
iALS	**0.2106**	**0.2106**	**0.5507**	**0.3863**	**0.7111**	**0.4383**
NCF (NeuMF variant)	0.2088	0.2088	0.5411	0.3803	0.7093	0.4349

[16] We report $RP^3\beta$ [47] for completeness although the DL algorithm we evaluate here predates its publication.

Table 7. Experimental results for NCF (Pinterest)

	HR@1	NDCG@1	HR@5	NDCG@5	HR@10	NDCG@10
TopPopular	0.0468	0.0468	0.1665	0.1064	0.2740	0.1409
UserKNN CF jaccard	**0.2898**	**0.2898**	0.7038	**0.5056**	0.8655	**0.5583**
ItemKNN CF asymmetric	**0.2920**	**0.2920**	**0.7113**	**0.5102**	0.8765	**0.5641**
$P^3\alpha$	**0.2853**	**0.2853**	0.7022	0.5024	0.8700	0.5571
$RP^3\beta$[16]	**0.2966**	**0.2966**	**0.7151**	**0.5149**	**0.8796**	**0.5685**
$EASE^R$	**0.2889**	**0.2889**	0.7053	**0.5057**	0.8682	**0.5589**
SLIM	**0.2913**	**0.2913**	0.7059	**0.5072**	0.8679	**0.5601**
PureSVD	0.2630	0.2630	0.6628	0.4706	0.8268	0.5241
iALS	**0.2811**	**0.2811**	**0.7144**	**0.5061**	0.8762	**0.5590**
NCF (NeuMF variant)	0.2801	0.2801	0.7101	0.5029	0.8777	0.5576

On the well-known MovieLens dataset (Table 6), NeuMF was competitive against the simple baselines, however was outperformed by all but one non-neural machine learning methods. On the Pinterest dataset (Table 7), NeuMF could only outperform PureSVD, which is not optimized for implicit feedback datasets. Most non-neural machine learning techniques, were often either similar or better than NeuMF.[17]

As a side observation, we can see that machine learning methods were clearly favorable over simple techniques for the MovieLens dataset. For the Pinterest dataset, however, it turns out that this advantage diminishes—at least in this experiment—and that a well-tuned ItemKNN method led to similar and sometimes better performance than machine learning techniques.

4.4 Deep Matrix Factorization (DMF)

Method. Deep Matrix Factorization Models (*DMF*) were proposed at IJCAI '17 [66]. As an input to their model, the authors first build a user-item matrix from explicit ratings and implicit feedback, which is then used by a deep *structure learning* architecture. One key aspect here is that a common low-dimensional space for representing users and items is used. Furthermore, the authors develop a new loss function based on cross entropy that considers both implicit feedback and explicit ratings.

Datasets. Experiments were made on four public datasets: the two smallest MovieLens datasets (*100k* and *1M*), and two publicly available datasets collected from Amazon.com (for the *Movie* and *Music* domains). All datasets contain ratings on a 1 to 5 scale. The datasets were pre-processed (if needed) so that there were at least 20 ratings for each user. Furthermore, for the Amazon datasets only, items were considered for which more than 5 ratings existed.

The Amazon Music data set that resulted from the pre-processing step was shared by the authors. It however contains users with less than 20 interactions and items with less than 5 ratings. Therefore, it remains unclear how exactly the filtering was done. In order to keep the results presented in this paper consistent across datasets, we have pre-processed all datasets—including Amazon Music—as

[17] It shall be noted here that after the first publication of our results [18], the authors of NeuMF provided us with an alternative configuration of their method, which included new hyperparameter values taken from alternative hyperparameter ranges, and requiring other slight changes in the training procedure. While this new configuration led to slightly improved results for their method, the results of our analysis were confirmed. In this context we would like to clarify that for all neural methods investigated here better configurations than those reported in the original papers may indeed exist. Finding such configurations, e.g., in the form of better hyperparameter ranges or alternative network structures, is however not the goal of our work. Instead, our goals are to assess the reproducibility of existing works and to compare the best reported results against existing baseline techniques.

Table 8. Experimental results for DMF for the MovieLens1M (left) and MovieLens100k (right) datasets.

	HR@10	NDCG@10
TopPopular	0.4418	0.2475
UserKNN CF asymmetric	0.6626	0.3975
ItemKNN CF cosine	0.6520	0.3851
$P^3\alpha$	0.6097	0.3639
$RP^3\beta$[19]	0.6304	0.3726
EASE^R	0.6691	**0.4093**
SLIM	**0.6825**	**0.4209**
PureSVD	0.6570	0.4015
iALS	**0.6947**	**0.4257**
DMF *nce*	0.6782	0.4063
DMF *bc*	0.6731	0.4033

	HR@10	NDCG@10
TopPopular	0.4145	0.2342
UserKNN CF asymmetric	0.5898	0.3387
ItemKNN CF tversky	0.6026	0.3506
$P^3\alpha$	0.5717	0.3421
$RP^3\beta$[19]	0.5685	0.3270
EASE^R	0.6111	0.3591
SLIM	**0.6238**	**0.3765**
PureSVD	0.5877	0.3555
iALS	**0.6142**	**0.3691**
DMF *nce*	0.6111	0.3637
DMF *bc*	0.6026	0.3623

described in the original paper and we have not used the Amazon Music dataset shared by the authors. We also run our experiments on the Amazon Music dataset shared by the authors: the relative performance between baselines and NCF (not reported here) do not change with respect to the results reported here.

Evaluation. The evaluation procedure is *leave-last-out* similar to the one used for the NCF method. For each user, the last interaction (based on its timestamp) is held out and ranked together with 99 negative (non-interacted) random items.[18] The Hit Rate and NDCG at list length 10 are used as metrics. The data splits that were used in the experiments were not shared by the authors, therefore we created data splits based on the information in the paper.

As personalized baselines, the authors consider NCF [23] as well as the baselines reported in that article, i.e., eALS and ItemKNN. However, the authors used NCF with binarized feedback, while DMF used explicit feedback. Hyperparameters for the machine learning methods were tuned on a validation set built from the training set by randomly sampling one interaction per user, and the authors report that eALS and NCF were tuned as in the original papers. For the ItemKNN method, no details about neighborhood sizes or the used similarity function are provided.

Results and Discussion. We reproduced the experiments reported by the authors based on the code that was provided to us upon request. We ran DMF with both loss functions that were also evaluated in the paper and could confirm that the normalized version *nce* often leads to tiny accuracy improvements over the binary version *bc*. Our results however revealed that for three out of four datasets one of the simple baselines outperformed DMF on both measures.

Table 8 shows the results for the MovieLens datasets, which are among the most often used ones in the literature. The results obtained by DMF are better than traditional nearest-neighbor baselines on the MovieLens1M dataset, but slightly worse than those obtained with the iALS and SLIM. For the smaller MovieLens100k dataset, the observed ranking is generally similar and again iALS and SLIM outperform DMF on both measures.

The detailed results for the Amazon datasets are shown in Table 9. The results for EASE^R are missing for the Amazon Music and Movies datasets, as the author-provided Python implementation of the method needed too much memory on these datasets. For the Amazon Music dataset it is

[18] The paper reports that 100 negative items are used, as described for NCF. However, the source code provided by the authors uses 99 negative items. In our experiments we have used 99 negative items.

[19] We report $RP^3\beta$ [47] for completeness although the DL algorithm we evaluate here predates its publication.

Table 9. Experimental results for DMF for the Amazon Music (left) and Amazon Movies (right) datasets. EASE^R results are missing because the code required too much memory on these datasets.

	HR@10	NDCG@10
TopPopular	0.5201	0.3007
UserKNN CF cosine	**0.6754**	**0.4976**
ItemKNN CF cosine	0.6647	**0.4884**
P$^3\alpha$	0.6576	**0.4816**
RP$^3\beta^{19}$	**0.6742**	**0.4909**
EASE^R	-	-
SLIM	0.6469	0.4746
PureSVD	0.5912	0.4189
iALS	0.6600	**0.4879**
DMF nce	0.6718	0.4815
DMF bc	0.6659	0.4815

	HR@10	NDCG@10
TopPopular	0.5799	0.3490
UserKNN CF cosine	0.7214	0.5020
ItemKNN CF asymmetric	0.6983	0.4913
P$^3\alpha$	0.6971	0.5029
RP$^3\beta^{19}$	0.7103	0.5077
EASE^R	-	-
SLIM	0.6980	0.5005
PureSVD	0.6021	0.4156
iALS	0.7352	0.5230
DMF *nce*	**0.7864**	**0.5447**
DMF *bc*	0.7818	0.5417

interesting to observe that the simple UserKNN and RP$^3\beta$ methods work better here than other machine learning models, and also better than DMF. For the Amazon Movies dataset, the DMF method was actually much better than all other methods on both measures. In particular the gains in terms of the Hit Rate are substantial and much higher than the second best method iALS.

Like for the case of CDL and the CVAE methods described in Sections 4.1 and 4.2, a better performance could only be observed for one of the datasets. In the case of CDL and CVAE better results were obtained for very sparse datasets with only one training interaction per user. Looking at the Amazon Movies dataset characteristics in the context of the DMF method, we can see that it is extremely sparse after the pre-processing step, in which 80% of the interactions were removed. In the end, there are 878k remaining interactions for over 80k movies. The Amazon Music dataset is even sparser. After the preprocessing step, which removes more than 94% of the interactions, it has only 46k interactions for 18k items. Further investigations are necessary to better understand why DMF works so well in this case, which could help us design algorithms that also work well on other datasets with similar characteristics.

Regarding methodological aspects, we found that the authors reported the best Hit Rate and NDCG results across different epochs. We therefore report the numbers here that were obtained after determining a suitable number of epochs on the validation set. In that context, the provided code shows that the authors sample different negative items to be used for testing in each training epoch. This seems questionable as well, in particular when considered in combination of the practice of reporting the best value for each metric across epochs. In our experiments we use the same negative item set for all evaluations.

From a conceptual perspective, the authors argue that they combine implicit feedback and explicit feedback in their approach. While this might be true in some interpretation, the authors mainly rely on the explicit ratings and add zeros to the empty matrix cells. Furthermore, when comparing their method with NCF, they only fed the binarized data to NCF, even though this method could deal with explicit rating data as well.

4.5 Variational Autoencoders for Collaborative Filtering (Mult-VAE)

Method. In [32], the authors propose a collaborative filtering method for recommendation based on implicit feedback using variational autoencoders. The method is called *Mult-VAE* and was presented at WWW '18. Technically, the paper introduces a generative model with multinomial likelihood and a different regularization parameter for the learning objective and uses Bayesian inference for parameter estimation. The authors furthermore show that there is an efficient way to tune the parameter using annealing.

Datasets. The authors use three datasets for evaluation. The first two datasets contain explicit feedback in the form of movie ratings (*MovieLens20M* and *Netflix*). The third dataset contains play counts for musical tracks. All datasets are binarized. For the movie datasets, ratings higher than three are considered positive signals and only users with more than five interactions are retained. For the music dataset, users with more than 20 interactions are retained; tracks that were listed less than 200 times are filtered out. After preprocessing, the datasets are still relatively large, having between 10 and almost 57 millions interactions.[20]

Evaluation. Four machine learning models are used as baselines, iALS, SLIM, NCF [23] (see Section 4.3) and the Collaborative Denoising Autoencoder (CDAE) method proposed in [64] in 2016. For evaluation purposes, the datasets are split into training, validation and test splits by holding-out users. For instance, for the MovieLens20M dataset (136k users overall), 10k users are removed for validation and 10k users are removed for testing. For each hold-out user, 80% of the interactions are used as user profile, and the remaining 20% are used as ground truth to measure the performance metrics. The models are optimized for NDCG@100 on the validation set. Performance results for Recall@20 and Recall@50 are reported as well.

Note that in order to be able to use matrix factorization baselines on cold users we built the cold users' latent factors based upon both their user profile and the latent factors of the warm items. In particular, we added a hyperparameter to the matrix factorization models to select how those cold user's latent factors are estimated, either via an item based similarity or an item embeddings average, see [14]. In the first case an ItemKNN model is created by defining the similarity matrix as the product of the items' latent factor matrix by its transpose. In the second case the latent factors of a user are the product of the user profile and the items' latent factors, resulting in the average of the embeddings of the items the user interacted with. Usually, generating an ItemKNN similarity matrix proved to be the most effective solution.

Results and Discussion. Using the code, the data splits and information about the seed for the random number generator that the authors provided, we could reproduce the results from the paper. In the original paper NDCG@100 is used as an optimization goal, however no reason is provided for this, as well as why Recall@20 and Recall@50 are used as additional measures, but not, e.g., Recall@100. To obtain a more comprehensive picture, we made additional measurements at the corresponding but missing cut-off values: Recall@100, NDCG@20 and NDCG@50. Table 10 shows our results for the MovieLens20M dataset and Table 11 those for the Netflix data.

For the MovieLens dataset, we observed a positive result and could confirm the claims made by the authors of Mult-VAE. On all measurements, both the original and the additional ones, Mult-VAE leads to better performance results than all baseline methods. SLIM is the second best method in this evaluation, with performance results that are around 1% to 2% lower in terms of the NDCG.

For the Netflix dataset, the claims of the authors could not be confirmed to the full extent. In terms of NDCG, which is the optimization criterion, SLIM outperforms Mult-VAE at all list lengths. Mult-VAE is however better in terms of Recall.

Overall, with Mult-VAE a method was found which was easy to reproduce, thanks to all needed material being made by the authors publicly available. Furthermore, as our results indicated, the

[20] We did not run experiments for the music dataset as the original paper did not contain sufficient information to guarantee we used the dataset in the exact same way as the authors.

Table 10. Results for Mult-VAE for the MovieLens20M dataset. UserKNN could not be applied because of the evaluation protocol (hold-out of users).

	REC@20	NDCG@20	REC@50	NDCG@50	REC@100	NDCG@100
TopPopular	0.1441	0.1201	0.2320	0.1569	0.3296	0.1901
UserKNN	-	-	-	-	-	-
ItemKNN CF asymmetric	0.2937	0.2444	0.4486	0.3087	0.5709	0.3527
$P^3\alpha$	0.2620	0.2168	0.4047	0.2742	0.5287	0.3182
$RP^3\beta$	0.3006	0.2501	0.4540	0.3133	0.5797	0.3583
$EASE^R$	0.3100	0.2639	0.4608	0.3267	0.5860	0.3711
SLIM	0.3356	0.2920	0.4893	0.3576	0.6110	0.4017
PureSVD	0.1620	0.1137	0.2778	0.1593	0.3974	0.1995
iALS	0.2030	0.1340	0.3628	0.1954	0.4976	0.2418
Mult-VAE	**0.3541**	**0.2988**	**0.5222**	**0.3690**	**0.6517**	**0.4158**

Table 11. Results for Mult-VAE for the Netflix dataset. UserKNN could not be applied because of the evaluation protocol (hold-out of users).

	REC@20	NDCG@20	REC@50	NDCG@50	REC@100	NDCG@100
TopPopular	0.0786	0.0762	0.1643	0.1159	0.2717	0.1570
UserKNN	-	-	-	-	-	-
ItemKNN CF cosine	0.2091	0.1970	0.3387	0.2592	0.4598	0.3092
$P^3\alpha$	0.1960	0.1759	0.3325	0.2412	0.4633	0.2962
$RP^3\beta$	0.2210	0.2053	0.3633	0.2739	0.4932	0.3281
$EASE^R$	0.2393	0.2288	0.3801	0.2978	0.5072	0.3510
SLIM	0.2555	**0.2479**	0.4002	**0.3203**	0.5299	**0.3752**
PureSVD	0.1177	0.0908	0.2193	0.1357	0.3247	0.1765
iALS	0.1397	0.1014	0.2675	0.1570	0.3930	0.2066
Mult-VAE	**0.2615**	0.2423	**0.4127**	0.3167	**0.5456**	0.3730

method consistently outperformed existing methods at least on one well-known and comparably large dataset.

4.6 NeuRec: On Nonlinear Transformation for Personalized Ranking

Method. NeuRec [69] was presented at IJCAI ’18. The work aims at learning user-item relationships from implicit feedback and combines latent factor models with neural networks in order to capture both linear and non-linear dependencies in the data. Technically, the user-item interaction matrix is first mapped into a low-dimensional space with multi-layered networks. Recommendations are then generated by computing the inner product of item and user latent factors. A user-based and an item-based variant are proposed.

Datasets. The authors use four public datasets for their evaluations (*MovieLens1M*, *HetRec*, *FilmTrust*, *Frappe*). Three of them contain movie ratings, which are binarized by converting all ratings to 1 and the missing entries to 0. The largest dataset (MovieLens1M) comprises 1 million interactions. The fourth dataset (Frappe) is from the domain of mobile app recommendation and contains about 20k interactions after pre-processing, which consisted of the removal of multiple interactions between the same user-item pairs. Data splits were not provided online but could be reproduced based on the information in the paper.

Evaluation. The authors use five random training-test splits (80%/20%) for evaluation and report the average results. As performance metrics, the authors use Precision and Recall at list lengths 5 and 10, as well as Mean Average Precision, MRR, and the NDCG, at list length 10.

As non-trivial baselines, the authors consider SLIM, BPR matrix factorization, NeuMF and the GMF model, which is part of NCF [23]. Information about hyperparameter tuning for the baselines is not provided except for GMF and NeuMF, which are said to use the "default" configuration as described in the original article. Hyperparameters for NeuRec were determined through grid search and the finally used values are reported in the paper in detail. The number of training epochs is not reported in the paper.[21] As usual we selected the number of epochs via early-stopping on a validation split.

Results and Discussion. Even though the authors published a runnable implementation of their method and provided detailed information on the hyperparameters, we could *not* obtain the results reported in the original paper. We contacted the authors but we were not able to reconstruct an experimental pipeline (from pre-processing to hyperparameter optimization) that led to results that were comparable to the ones reported in the original paper. In the end, the reason for this discrepancy could not be clarified. The outcome of our evaluation is that NeuRec is outperformed on any data set and almost on any measure by at least one, but usually several, of the baselines in our comparison.

Since the detailed results are comprehensive, given the number of datasets and evaluation measures, we only provide in Table 12 the results for the most commonly used MovieLens1M dataset and for list lengths 5 and 10. All other results can again be found in the online appendix (Section 3).

Looking at the results, we observe that on the MovieLens dataset even the simplest baselines are better than NeuRec and that the performance of the best baselines is better by a large margin. For the HetRec and FilmTrust datasets (not shown in detail here), the picture is mostly the same. Finally, for the small and rarely used Frappe dataset, NeuRec actually leads to the best results for Precision@5, but is outperformed, e.g., by $RP^3\beta$ on all other measures.[22]

Regarding methodological aspects, we found again that researchers optimized the number of epochs on the test set and apparently reported the best results of NeuRec for different measures at, potentially, different training epochs. Furthermore, it is unclear from the paper if hyperparameter optimization was done for the baselines. For the NCF method, the authors state that they used the "configuration" proposed in the original paper, but it is unclear if this refers to the network structure, the hyperparameters, or both.

4.7 CoupledCF: Learning Explicit and Implicit User-item Couplings

Method. CoupledCF [68] was also presented at IJCAI '18. The approach is based on the observation that users and items in real-world datasets are not independent and identically distributed. The proposed method therefore aims to learn implicit and explicit couplings between users and items and to thereby leverage available side information (e.g., user demographics, item features) more effectively. Technically, a complex architecture is used, involving a CNN for learning the couplings

[21] According to an exchange of emails with the authors, the training was done for a large number of epochs and the best performance values on the test set were reported.

[22] We point out that the Frappe dataset is very small and exhibits very unstable results depending on the random split. In particular, compared to the results reported in the original paper, our TopPop algorithm exhibits results that are four times higher; also NeuRec's values are two times higher.

Table 12. Experimental results for NeuRec for the MovieLens1M dataset.

			@5					@10		
	PREC	REC	MAP	NDCG	MRR	PREC	REC	MAP	NDCG	MRR
TopPopular	0.2105	0.0402	0.1531	0.0689	0.3621	0.1832	0.0685	0.1168	0.0939	0.3793
UserKNN CF asymmetric	**0.4223**	**0.1068**	**0.3456**	**0.1678**	**0.6406**	**0.3631**	**0.1733**	**0.2787**	**0.2237**	**0.6517**
ItemKNN CF asymmetric	**0.3995**	**0.0984**	**0.3244**	**0.1563**	**0.6179**	**0.3452**	**0.1590**	**0.2618**	**0.2084**	**0.6293**
$P^3\alpha$	**0.4041**	**0.1007**	**0.3286**	**0.1596**	**0.6250**	**0.3456**	**0.1627**	**0.2627**	**0.2121**	**0.6362**
$RP^3\beta$	**0.4080**	**0.1007**	**0.3325**	**0.1602**	**0.6260**	**0.3508**	**0.1639**	**0.2676**	**0.2137**	**0.6374**
$EASE^R$	**0.4360**	**0.1073**	**0.3608**	**0.1697**	**0.6475**	**0.3745**	**0.1731**	**0.2923**	**0.2259**	**0.6585**
SLIM	**0.4437**	**0.1106**	**0.3692**	**0.1749**	**0.6578**	**0.3813**	**0.1770**	**0.3003**	**0.2321**	**0.6679**
PureSVD	**0.4123**	**0.0987**	**0.3371**	**0.1586**	**0.6266**	**0.3575**	**0.1624**	**0.2722**	**0.2132**	**0.6380**
iALS	**0.4164**	**0.1036**	**0.3373**	**0.1635**	**0.6327**	**0.3628**	**0.1702**	**0.2743**	**0.2200**	**0.6443**
INeuRec	0.3280	0.0663	0.2554	0.1110	0.5003	0.2839	0.1094	0.2027	0.1500	0.5129
UNeuRec	0.2098	0.0395	0.1560	0.0684	0.3663	0.1856	0.0688	0.1199	0.0944	0.3852

based on the side information, and a deep CF model that considers explicit and implicit interactions between users and items.

Datasets. Experiments were made on two public datasets, the *MovieLens1M* rating dataset and a dataset called *Tafeng* containing grocery store transactions. The Tafeng dataset has about 750k transactions (i.e., less than the MovieLens1M dataset), but is much more sparse as it contains many more users and items. The explicit ratings in the MovieLens1M dataset are transformed into binary ratings, where each rating is considered as a positive interaction. Both datasets contain side information about users and items that is used by the CoupledCF algorithm. Therefore, we have included item-based and user-based content techniques among the baselines. The authors of the paper provided us with the train-test splits, including the sampled test negative items they had used during the evaluation.

Evaluation. A leave-one-out procedure is used, where for each user one random interaction is put in the test set. For evaluating the performance, 99 items are sampled for which there was no interaction for the given user. The 100 items are then ranked by the algorithm and the Hit Rate and the NDCG are used to evaluate the performance. Cut-off list lengths between 1 and 10 were considered.

The hyperparameters of the proposed model were systematically fine-tuned by the authors. Information about hyperparameter tuning for the baselines is not provided. The considered baselines include NCF [23], and Google's Wide&Deep method.

Results and Discussion. We could not fully reproduce the results by the authors based on the provided code and data splits. Different variations of the proposed model were tested in the original paper. We used the best-performing one (CoupledCF) in our experiments as well as their simplest variant (DeepCF), where the latter is the only one for which we could fully reproduce the results. Furthermore, we observe there were some apparent issues regarding the way the authors sampled the negative items. For both Movielens and Tafeng the negative item data contains duplicates leading to some users having as few as 93 unique negative items. For the Tafeng dataset, cumulatively, almost 3,000 negative items also appeared as train items or test items for that same user. Even more alarmingly, 8% of the users in the Tafeng dataset have inconsistent test data, being either associated to a test item but no negative items or vice versa. Due to the evaluation methodology, in the first of those cases even a random recommender would exhibit a perfect score on all metrics.

Our experimental results are shown in Table 13 (MovieLens) and Table 14 (Tafeng).

Table 13. Experimental results for CoupledCF for the MovieLens1M dataset.

	HR@1	NDCG@1	HR@5	NDCG@5	HR@10	NDCG@10
TopPopular	0.1086	0.1086	0.3224	0.2174	0.4738	0.2661
UserKNN CF asymmetric	0.2003	0.2003	0.5210	0.3678	0.6902	0.4228
ItemKNN CF cosine	0.1810	0.1810	0.5111	0.3514	0.6844	0.4077
$P^3\alpha$	0.1829	0.1829	0.4965	0.3451	0.6591	0.3979
$RP^3\beta$	0.1801	0.1801	0.5083	0.3498	0.6626	0.3997
$EASE^R$	**0.2187**	**0.2187**	**0.5675**	**0.3988**	**0.7260**	**0.4502**
SLIM	**0.2258**	**0.2258**	**0.5778**	**0.4073**	**0.7281**	**0.4561**
PureSVD	**0.2167**	**0.2167**	**0.5540**	**0.3916**	0.7055	**0.4408**
iALS	**0.2220**	**0.2220**	**0.5684**	**0.4017**	**0.7232**	**0.4518**
ItemKNN CBF asymmetric	0.0636	0.0636	0.2164	0.1412	0.3406	0.1812
UserKNN CBF tversky	0.1200	0.1200	0.3381	0.2300	0.5012	0.2826
ItemKNN CFCBF cosine	0.1818	0.1818	0.5159	0.3526	0.6833	0.4068
UserKNN CFCBF jaccard	0.1982	0.1982	0.5169	0.3644	0.6805	0.4174
DeepCF	0.1959	0.1959	0.5522	0.3795	0.7171	0.4330
CoupledCF	0.2071	0.2071	0.5465	0.3817	0.7079	0.4342

Table 14. Experimental results for CoupledCF for the Tafeng dataset.

	HR@1	NDCG@1	HR@5	NDCG@5	HR@10	NDCG@10
TopPopular	0.2805	0.2805	0.5284	0.4090	0.6608	0.4519
UserKNN CF cosine	**0.3509**	**0.3509**	**0.5738**	**0.4681**	**0.6708**	**0.4995**
ItemKNN CF cosine	**0.3570**	**0.3570**	**0.5722**	**0.4705**	**0.6643**	**0.5003**
$P^3\alpha$	**0.3520**	**0.3520**	**0.5840**	**0.4746**	**0.6769**	**0.5047**
$RP^3\beta$	**0.3396**	**0.3396**	0.5134	**0.4317**	0.5895	0.4563
$EASE^R$	**0.3499**	**0.3499**	**0.5715**	**0.4663**	**0.6691**	**0.4978**
SLIM	**0.3493**	**0.3493**	**0.5717**	**0.4662**	**0.6719**	**0.4987**
PureSVD	0.2748	0.2748	0.5219	0.4028	0.6532	0.4454
iALS	**0.3290**	**0.3290**	**0.5628**	**0.4523**	0.6618	**0.4844**
ItemKNN CBF asymmetric	0.0547	0.0547	0.0945	0.0741	0.1425	0.0894
UserKNN CBF asymmetric	0.2633	0.2633	0.4766	0.3744	0.5880	0.4103
ItemKNN CFCBF asymmetric	**0.3610**	**0.3610**	**0.5735**	**0.4736**	0.6561	**0.5003**
UserKNN CFCBF asymmetric	**0.3581**	**0.3581**	**0.5911**	**0.4811**	**0.6905**	**0.5133**
DeepCF	0.2860	0.2860	0.5358	0.4161	0.6625	0.4571
CoupledCF	0.2767	0.2767	0.5272	0.4065	0.6597	0.4494

For the MovieLens datasets we can observe that CoupledCF is almost consistently able to outperform the simple neighborhood-based methods and the hybrids (except for very short list lengths). However, relatively simple non-neural methods like iALS and $EASE^R$ are consistently better than CoupledCF. The differences between the CoupledCF and the DeepCF variant are tiny and the simpler DeepCF is sometimes even better. This stands in contrast to the results in the original article, where the differences were large.

For the Tafeng dataset, even the nearest-neighbor methods outperform CoupledCF by far. Only the pure content-based baselines do not reach the performance level of CoupledCF. Generally, on this dataset, the performance of the proposed method is at the level of the TopPopular baseline. The simpler DeepCF method also leads to better accuracy results than the CoupledCF variant.

Looking at methodological aspects, it seems that the baselines were not properly optimized and default hyperparameters were used. Furthermore, from the provided code it seems that the number of epochs was determined on the test set, as was done in other papers examined in this work. A specific problem in this work also lies in the creation of the train and test splits which are inconsistent with the description reported in the article and are likely the result of an erroneous splitting procedure.

4.8 DELF: A Dual-Embedding based Deep Latent Factor Model for Recommendation

Method. The *DELF* model, presented at IJCAI '18, was designed for *top-n* recommendation tasks given implicit feedback data. Inspired by previous work (NSVD) [46], the authors propose to learn *dual* embeddings to capture certain interactions in the data. Instead of using only the common user embedding, the authors propose to learn an additional item-based user embedding and vice versa for item embeddings. The embeddings are then combined to model non-linear interactions between users and items within a deep learning architecture. Through this approach the authors generalize ideas of NSVD and Neural Collaborative Filtering (NCF). Two variants of the approach, *DELF-MLP* and *DELF-EF* were investigated in the original paper.

Datasets. Two public rating datasets are used for the evaluation. One is the well-known *MovieLens1M* dataset and the other one the *Amazon Music* dataset. The rating datasets were binarized by transforming each non-zero rating to 1. Pre-processing was applied so that for both datasets only users were retained for which more than 20 interactions were observed. Through this preprocessing, the Amazon Music dataset was reduced to less than one tenth of its original size in terms of interactions (only 76k interactions for 40k items remain in the Amazon Music dataset).

Evaluation. The evaluation procedure was similar to the one used for NCF as discussed in Section 4.3: a leave-last-out procedure was applied using the interaction timestamp, the hidden element was ranked within 99 randomly sampled negative items, and the Hit Rate and the NDCG at a cut-off length of 10 were used as performance measures. The non-trival baselines include BPR matrix-factorization, iALS, DMF as discussed in Section 4.4, and two variants of the NCF model [23] described in Section 4.3.

The hyperparameters for the proposed model were systematically optimized on a validation set built with the second most recent interaction. No information is provided regarding the hyperparameter optimization of the baselines.

Results and Discussion. We reproduced the results using a validation set which we constructed in the same way as reported by the authors. NDCG@10 was used as an optimization criterion. The results are shown in Table 15 (Amazon Music) and Table 16 (MovieLens1M).

Across both datasets and across all measurements, the proposed model was never the best-performing one. On the Amazon dataset, iALS was better than all other methods. Furthermore, all neighborhood models and SLIM outperformed the new methods in all but one measurement (NDCG@20). A consistent "win" for DELF method was only observed over the PureSVD method. For the MovieLens1M data, iALS, EASE^R, and SLIM outperformed DELF on all measures. The UserKNN method was also consistently better in terms of the NDCG (the optimization criterion).

[23] The results for EASE^R are missing due to the algorithm memory requirements exceeding the instance capacity. Note that we are relying on a simple implementation in Python which does not optimize memory requirements.

Table 15. Experimental results for DELF for the Amazon Music Data.

	HR@5	NDCG@5	HR@10	NDCG@10	HR@20	NDCG@20
TopPopular	0.2474	0.1730	0.3041	0.1913	0.3738	0.2090
UserKNN CF cosine	**0.3150**	**0.2495**	**0.3471**	**0.2600**	0.3738	**0.2668**
ItemKNN CF asymmetric	**0.3090**	**0.2506**	**0.3401**	**0.2609**	0.3717	**0.2689**
P$^3\alpha$	**0.3074**	**0.2465**	**0.3373**	**0.2564**	0.3689	**0.2644**
RP$^3\beta$	**0.3046**	**0.2434**	**0.3379**	**0.2543**	0.3651	**0.2611**
EASE^R[23]	-	-	-	-	-	-
SLIM	**0.3101**	**0.2526**	**0.3411**	**0.2625**	0.3711	**0.2701**
PureSVD	0.2627	0.2141	0.3084	0.2290	0.3542	0.2406
iALS	**0.3319**	**0.2604**	**0.3706**	**0.2729**	**0.4109**	**0.2831**
DELF MLP	0.2883	0.2239	0.3335	0.2386	0.3760	0.2494
DELF EF	0.2856	0.2159	0.3330	0.2315	0.3809	0.2437

Table 16. Experimental results for DELF on the MovieLens dataset.

	HR@5	NDCG@5	HR@10	NDCG@10	HR@20	NDCG@20
TopPopular	0.3300	0.2228	0.4698	0.2674	0.6576	0.3148
UserKNN CF asymmetric	0.5188	**0.3628**	0.6865	**0.4171**	0.8334	**0.4544**
ItemKNN CF cosine	0.4978	0.3418	0.6814	0.4011	0.8311	0.4390
P$^3\alpha$	0.4945	0.3438	0.6574	0.3965	0.7950	0.4313
RP$^3\beta$	0.5138	0.3559	0.6809	0.4102	0.8276	0.4475
EASE^R	**0.5609**	**0.3954**	**0.7248**	**0.4486**	**0.8559**	**0.4818**
SLIM	**0.5706**	**0.4038**	**0.7306**	**0.4557**	**0.8586**	**0.4882**
PureSVD	**0.5513**	**0.3891**	**0.7021**	**0.4382**	0.8303	**0.4708**
iALS	**0.5652**	**0.3979**	**0.7268**	**0.4503**	**0.8465**	**0.4807**
DELF MLP	0.5234	0.3592	0.6892	0.4132	0.8356	0.4503
DELF EF	0.4718	0.3210	0.6423	0.3762	0.7942	0.4146

Looking at methodological aspects, like in other works discussed here, the authors did not optimize the number of epochs on the validation set but took the best values using the test. This is a methodological issue leading to information leakage from the test data.

4.9 Outer Product-based Neural Collaborative Filtering (ConvNCF)

Method. The *ConvNCF* method [22] was presented at IJCAI '18. Its main idea is to explicitly model the pairwise correlations between the dimensions of the embedding using an outer product. With this technique, the authors aim to create an *interaction map*, which is more expressive than existing methods that use simple concatenations of embeddings or element-wise products.

Datasets. The proposed method is evaluated on two public implicit-feedback datasets, *Gowalla* and *Yelp*. Both datasets contain multiple implicit interactions at different timestamps for the same user-item pair. These interactions are merged by keeping only the earliest for each user-item pair.[24] Both dataset have been filtered by removing items with less than 10 interactions and users with less than 2 (Gowalla) or 10 (Yelp) interactions. Both datasets and train/test splits are provided by

[24] For the Gowalla dataset the authors use the first interaction appearing in the file as the *earlier* interaction.

the authors. The Yelp dataset has about 69k interactions after processing. The Gowalla dataset is sparser and with more interactions (1.2M). Both filtered datasets with their train/test splits have been provided by the authors.

Evaluation. Each dataset is split into training and test data according to a leave-last-out protocol. Evaluation is done by randomly selecting 999 negative items, and the algorithm has to rank these items together with the hidden item. Different methodological issues were observed, as will be discussed below. The Hit Rate and the NDCG at different list lengths are used as evaluation measures. Hyperparameter optimization is done via a validation set, except for the embedding size, which is kept constant for all methods.

Results and Discussion. We could reproduce the results using the code and the data provided by the authors. The results for the Yelp dataset are shown in Table 17. Those for the larger Gowalla dataset are given in Table 18.

Table 17. Experimental results for ConvNCF for the Yelp dataset.

	HR@5	NDCG@5	HR@10	NDCG@10	HR@20	NDCG@20
TopPopular	0.0817	0.0538	0.1200	0.0661	0.1751	0.0799
UserKNN CF asymmetric	**0.2131**	**0.1400**	**0.3209**	**0.1747**	**0.4482**	**0.2068**
ItemKNN CF cosine	**0.2521**	**0.1686**	**0.3669**	**0.2056**	**0.4974**	**0.2385**
P$^3\alpha$	**0.2146**	**0.1395**	**0.3211**	**0.1737**	0.4442	**0.2049**
RP$^3\beta$	**0.2202**	**0.1431**	**0.3323**	**0.1793**	**0.4667**	**0.2132**
EASE^{R}[23]	-	-	-	-	-	-
SLIM	**0.2330**	**0.1535**	**0.3475**	**0.1904**	**0.4799**	**0.2238**
PureSVD	**0.2011**	**0.1307**	0.3002	**0.1626**	0.4238	0.1938
iALS	**0.2048**	**0.1348**	**0.3080**	**0.1680**	0.4319	**0.1993**
ConvNCF	0.1947	0.1250	0.3059	0.1608	0.4446	0.1957

Table 18. Experimental results for ConvNCF for the Gowalla dataset.

	HR@5	NDCG@5	HR@10	NDCG@10	HR@20	NDCG@20
TopPopular	0.2188	0.1652	0.2910	0.1884	0.3803	0.2110
UserKNN CF cosine	**0.7131**	**0.5879**	**0.7939**	**0.6142**	0.8532	**0.6293**
ItemKNN CF tversky	**0.7047**	**0.5864**	0.7790	**0.6105**	0.8331	**0.6244**
P$^3\alpha$	**0.6926**	**0.5703**	0.7674	**0.5948**	0.8158	**0.6071**
RP$^3\beta$	**0.6836**	**0.5525**	0.7723	**0.5814**	0.8361	**0.5976**
EASE^{R}[23]	-	-	-	-	-	-
SLIM	0.6365	**0.5284**	0.7083	0.5517	0.7608	0.5651
PureSVD	0.5653	0.4482	0.6627	0.4798	0.7393	0.4993
iALS	0.6460	0.5081	0.7554	0.5436	0.8356	0.5641
ConvNCF	0.6702	0.5233	0.7799	0.5590	**0.8623**	0.5799

For the Yelp dataset, ConvNCF is consistently outperformed by the traditional nearest-neighbor methods, RP$^3\beta$, and SLIM. The other baselines outperform ConvNCF as well in most cases. For the Gowalla case, ConvNCF is slightly more competitive. In all but one measurement, however, it is outperformed by the UserKNN method. Interestingly, the simple machine learning methods do not work better on this dataset than the simple baselines.

A number of methodological issues were observed with this paper. First, the number of epochs, as in other papers, was apparently determined on the test data. Furthermore, the authors decided to set the embedding size to the constant value of 64 for all baselines. However, the embedding size is a hyperparameter to be tuned for each dataset and for each embedding-based algorithm, as different models with different objective functions or training procedures will likely require different values for it. There were also issues with the provided test splits. Negative test samples contained duplicates and partially overlapped with the train data, resulting in a number of unique negative items as low as 961 for some users. [25]

4.10 Leveraging Meta-path based Context (MCRec)

Method. The *MCRec* [26] method was published at KDD '18. It is a hybrid method that uses side information about the recommendable items in the recommendation process. The side information is represented as a network structure, and meta-paths are relation sequences that connect objects in this graph. Technically, the authors use a priority-based sampling technique to select more informative paths instances and a novel co-attention mechanism to improve the representations of meta-path based context, users and items.

Datasets. Three datasets are used for the evaluation. The historical *MovieLens100k* dataset, another small data set containing listening logs from *Last.fm*, and a dataset containing user feedback from *Yelp*. With almost 200k ratings, the Yelp dataset is the largest one. The available meta-data includes genres for movies, artists for tracks, and city and category for businesses. Only for the MovieLens100k dataset the used data splits are publicly available. The rating information in the datasets is transformed to binary data.

Evaluation. The datasets are split into 80% training and 20% test splits. Ten percent of the training data are used for validation. For each positive item in the test set, 50 negative test interactions are randomly selected in such a way that the negative interactions in train and test data are disjoint.[26] Precision, Recall, and the NDCG are then used as evaluation measures at a cut-off length of 10. The averaged results of ten randomized runs are reported in the paper. Since the NDCG measure was implemented in an unusual way by computing the "ideal" NDCG only based on the successfully recommended items, the values reported in the paper are much higher than the ones obtained by us using a standard implementation of the NDCG. Looking at the evaluation code provided by the authors, we furthermore observed that the authors reported the best results for each metric across different epochs. Since this is inappropriate, we report the values of the metrics after the optimal number of epochs is chosen with early stopping on the validation set.

As baselines, the authors use ItemKNN, two collaborative MF methods (one based on BPR loss, the other based on cross entropy loss), two hybrid MF methods, two methods designed for metadata networks, and a number of variants of their own method. For their own method, the final hyperparameters are reported, which are apparently the same for both datasets. For the MF and

[25] We report that at the time of our experiments the published version of the algorithm contained a bug that caused information leakage from the test data. Items not present in the train data were divided in two categories, those *not* present in the test data were sampled as negative items during the training of the model, while those present in the test data were not. Items unobserved in the train data should all be potential negative items to be sampled during training, regardless of the test data, otherwise test items will be advantaged over other items. This bug was later fixed on the public Github repository. In our experiments, during the training we only relied upon train data and never used any information from the test data.

[26] This evaluation protocol is different from more traditional protocols, as different users are tested against a different number of negative samples.

the NeuMF method, hyperparameters and configuration are taken from the original papers. Other baselines are reported to be systematically optimized on a validation set.

Results and Discussion. The meta-path information is hard-coded for the MovieLens dataset in the provided code and the preprocessed version of the data is publicly available. No code is available for the other two datasets. Furthermore, since the code for pre-processing the Yelp and Last.fm is not available, we only did an evaluation for the MovieLens dataset.[27] The results are reported in Table 19. They show that MCRec is outperformed both by the traditional neighborhood-based methods, the more complex learning-based methods and one of our simple hybrids. Our pure content-based baseline led to generally weak results (worse than TopPopular) and was also outperformed by MCRec.

Table 19. Experimental results for MCRec for the MovieLens dataset.

	PREC@10	REC@10	NDCG@10
TopPopular	0.1907	0.1180	0.1361
UserKNN CF dice	**0.3439**	**0.2236**	**0.2693**
ItemKNN CF asymmetric	**0.3354**	**0.2226**	**0.2638**
$P^3\alpha$	**0.3305**	0.2081	**0.2554**
$RP^3\beta$	**0.3435**	**0.2191**	**0.2588**
$EASE^R$	**0.3640**	**0.2318**	**0.2815**
SLIM	**0.3770**	**0.2441**	**0.2957**
PureSVD	**0.3542**	**0.2247**	**0.2724**
iALS	**0.3548**	**0.2287**	**0.2748**
ItemKNN CBF cosine	0.0455	0.0185	0.0254
ItemKNN CFCBF cosine	**0.3398**	**0.2239**	**0.2646**
MCRec	0.3110	0.2113	0.2466

From a methodological perspective, it is worth mentioning that the used datasets are very small, at least compared to the usual MovieLens and Netflix datasets, where we have millions of recorded interactions. We will discuss aspects of scalability later in this paper.

4.11 Collaborative Memory Network for Recommendation System (CMN)

Method. Collaborative Memory Networks (CMF) [16] was presented at SIGIR '18, and it represents a collaborative-filtering approach based on memory networks and neural attention mechanisms. The underlying idea of the approach is to combine latent factor with neighborhood (memory-based) approaches in a non-linear fashion.

Datasets. The method was evaluated on three public datasets. One containing data from *Epinions*, a *CiteULike* dataset used in previous works (*CiteULike-a*), and a dataset from *Pinterest*. The last dataset is the largest one and has about 1.5M interactions. The smallest Epinions dataset contains ratings on scale from one to five, which are binarized (all non-zero ratings are set to one). The train/test splits for CiteULike and Pinterest are provided by the authors.

[27] The preprocessing code for Movielens was made available later by the authors on their GitHub repository.

Evaluation. The authors use a leave-one-out methodology similar to the one used in previous works. The test set contains one positive item per user, the train set all other user-item pairs. If the user rated only one item, this interaction is kept in the training set. To evaluate the algorithm, for each user 100 unobserved (negative) items are sampled and ranked together with the positive item. The Hit Rate and the NDCG at a cut-off length of 10 are used as evaluation metrics.

As baselines, the authors include both simple, non-neural, and neural methods in there experiments: ItemKNN, BPR matrix-factorization, SVD++, two variants of NCF [23], and the Collaborative Denoising Auto Encoder.

Hyperparameters are tuned on a validation set. Details for the hyperparameter optimization process are provided for the proposed model, but not for the baselines. The number of training epochs is not mentioned in the paper. In our evaluation, we applied early stopping as described previously.

Results and Discussion. We could reproduce the results of the paper using the data splits provided by the authors for two datasets, CiteULike and Pinterest. Since the split used for Epinions was not made available we had to recreate it based on the information provided in the paper, in that case, however, we could not reproduce the original results. In Table 20, we show the results for the Pinterest dataset, the largest one. For the CMN method, we report the results for the *CMN-3* variant, which led to the best results.

Table 20. Experimental results for CMN for the Pinterest dataset.

	HR@5	NDCG@5	HR@10	NDCG@10
TopPopular	0.1665	0.1064	0.2740	0.1409
UserKNN CF asymmetric	**0.7043**	**0.5058**	0.8664	**0.5586**
ItemKNN CF cosine	**0.7122**	**0.5115**	**0.8775**	**0.5654**
$P^3\alpha$	0.6990	**0.5034**	0.8596	**0.5559**
$RP^3\beta$	**0.7147**	**0.5150**	**0.8772**	**0.5680**
$EASE^R$	**0.7050**	**0.5106**	0.8559	**0.5599**
SLIM	**0.7084**	**0.5107**	**0.8683**	**0.5628**
PureSVD	0.6619	0.4721	0.8146	0.5219
iALS	**0.7219**	**0.5175**	**0.8677**	**0.5652**
CMN	0.7013	0.5005	0.8674	0.5547

The results show that CMN led to competitive results, but is actually slightly outperformed on all measures by algorithms from all families, including traditional nearest-neighbor methods. On the CiteULike dataset, the main observations are the same. On this smaller dataset, however, the performance of CMN is often much lower than the one achieved by nearest-neighbor methods and machine learning techniques.

A quite surprising result is found for the Epinions dataset. Here, the trivial TopPopular method consistently led to the best values across all measurements. The difference to all other methods is often huge. This indicates that the popularity distribution of the items in this dataset is very skewed, which makes it difficult to make personalized recommendations that are better in terms of information retrieval measures than the TopPopular method. The detailed results are available in the online material (Section 3).

4.12 Spectral Collaborative Filtering (SpectralCF)

Method. SpectralCF, presented at RecSys '18 [70] is a graph-based approach. Users and items are represented as a bipartite graph. The novelty of this method is a convolution approach which operates on the *spectral domain*. The method considers both proximity and connectivity information in the graph, which is assumed to be particularly helpful for cold-start problems.

Datasets. Three public datasets are used for the evaluation, *MovieLens1M*, another movie rating dataset (*HetRec*), and an *Amazon* dataset (*InstantVideo*). Explicit ratings are binarized. Further pre-processing is applied to, e.g., remove users associated to less than 5 interactions. After pre-processing, the MovieLens1M dataset is shrinked to one fifth of the original size (226k interactions). The other datasets are even smaller (71k and 22k interactions, respectively). The data split for the MovieLens1M dataset is provided online by the authors. Due to an apparent problem on how the splits were generated—see our discussions below—we created our own splits based on the information provided in the paper.

Evaluation. Two evaluation scenarios are tested, a regular one and a cold-start setup. We evaluated both scenarios. For the main scenario, 80% of the interactions of each user are randomly put into the training set and the rest is used for evaluation. The random process is repeated five times and averaged results are reported. Recall and MAP at different list lengths are used as metrics for this scenario.

In the cold-start scenario, the training set is built with different degrees of sparsity by varying the number P of interactions associated with each user, where P is varied from one to five. The remaining items associated with each users are used as test set. Recall@20 and MAP@20 are used in this scenario for evaluation.

As baselines, the authors consider ItemKNN, BPR matrix factorization, iALS, NCF [23] and two graph-based methods, GNMF and GCMC, originally designed for explicit feedback datasets. The hyperparameters are tuned on a validation set and the parameter ranges are reported in the paper. For NCF the authors stated that they used the configuration reported in the original paper.

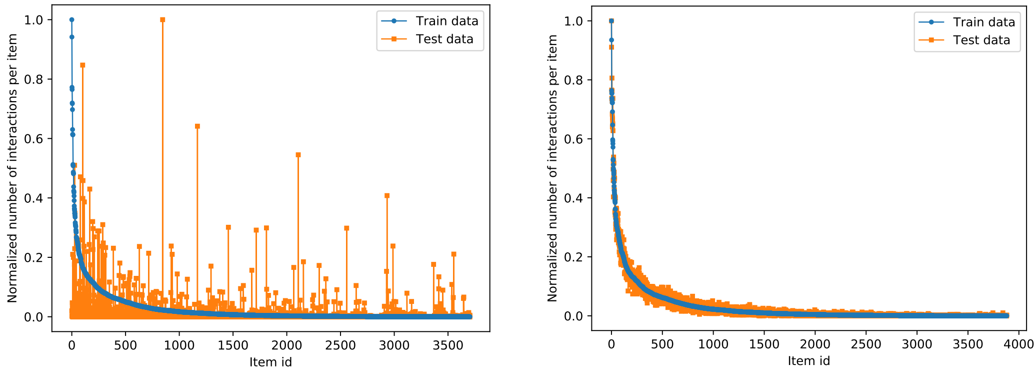
Results and Discussion. We reproduced the results obtained by the authors based on the provided code and following the information in the paper. Our first set of experiments showed that on HetRec and Amazon Video datasets the performance of SpectralCF was relatively weak, whereas it performed very well and better than all baselines when using the provided data splits for the MovieLens dataset. We therefore investigated the provided data splits and found that these splits were unlikely the result of the splitting procedure described in the paper, see the discussions below. For this reason, we created new data splits ourselves following the described procedure.

The results obtained for the MovieLens1M dataset are shown in Table 21. The results show that SpectralCF is outperformed by all baselines in our comparison often exhibiting a recommendation quality equal to the TopPopular method. The observations for the other datasets are similar, the full results are available in the online material (Section 3).

To illustrate the data splitting problem, we compared the popularity distribution of the items in the training and the test split in the provided data, see Figure 2b. The figure plots the normalized popularity (i.e., the popularity of an item over the popularity of the most popular item) for both training and test items. Items are ranked based on their popularity in the training set, with the most popular training items being on the left. In case of a true random split, the normalized popularity values of the items in the training and the test split should be relatively close. However, the figure shows the split provided by the authors has some major deviations. Figure 2b shows the popularity distributions of our random split, which are almost identical between training and test sets.

Table 21. Experimental results for SpectralCF for the MovieLens dataset

	REC@20	MAP@20	REC@40	MAP@40	REC@60	MAP@60	REC@80	MAP@80	REC@100	MAP@100
TopPopular	**0.1892**	**0.0584**	**0.2788**	**0.0636**	**0.3356**	**0.0666**	**0.3834**	**0.0687**	0.4226	**0.0702**
UserKNN CF jaccard	**0.3001**	**0.1201**	**0.4133**	**0.1285**	**0.4900**	**0.1335**	**0.5459**	**0.1367**	**0.5884**	**0.1388**
ItemKNN CF asymmetric	**0.2876**	**0.1134**	**0.4000**	**0.1213**	**0.4768**	**0.1263**	**0.5367**	**0.1295**	**0.5820**	**0.1317**
P3alpha	**0.2939**	**0.1141**	**0.4150**	**0.1233**	**0.4900**	**0.1285**	**0.5462**	**0.1318**	**0.5905**	**0.1342**
RP3beta	**0.2737**	**0.1044**	**0.3879**	**0.1124**	**0.4664**	**0.1173**	**0.5234**	**0.1206**	**0.5726**	**0.1230**
EASE^R	**0.2967**	**0.1176**	**0.4118**	**0.1261**	**0.4855**	**0.1312**	**0.5402**	**0.1345**	**0.5854**	**0.1367**
SLIM	**0.3069**	**0.1265**	**0.4246**	**0.1356**	**0.5010**	**0.1410**	**0.5564**	**0.1443**	**0.6001**	**0.1466**
PureSVD	**0.2593**	**0.1023**	**0.3654**	**0.1100**	**0.4387**	**0.1148**	**0.4925**	**0.1179**	**0.5352**	**0.1200**
iALS	**0.3033**	**0.1183**	**0.4201**	**0.1273**	**0.4933**	**0.1326**	**0.5493**	**0.1360**	**0.5925**	**0.1383**
SpectralCF	0.1813	0.0533	0.2643	0.0581	0.3274	0.0613	0.3823	0.0635	0.4261	0.0651



(a) Normalized popularity distributions of the train and test splits provided by the original authors.

(b) Normalized popularity distributions of the train and test splits generated by us.

Fig. 2. Normalized popularity distributions of the train and test splits for SpectralCF, the value 1 corresponds to the most popular item in that split. For a random split, as can be seen in Figure 2b, the normalized values of both splits are, on average, similar. In the split provided by the original authors, however, as can be seen in Figure 2a, train and test data have quite different distributions.

Besides the visual analysis, we also computed numerical statistics like the Gini index and Shannon entropy. In a true random split, the Gini index should be close to the value obtained on the unsplitted data (in this case 0.78) for both the training and test splits. However, the Gini index of the provided test split is much higher (0.92). This indicates that the provided test split has a much higher popularity bias than we would expect. A similar consideration applies for the Shannon entropy: the entropy of the original dataset is close to 10 and is similar to the entropy of our random train/test split. However, the provided test split has a lower entropy (8.5), i.e., it is easier to predict.

Regarding other methodological aspects, we found that the authors only report one set of hyperparameters, whereas one would expect hyperparameter settings for each dataset. In our evaluation, we therefore optimized the hyperparameters for all baselines and all datasets individually.

5 DISCUSSION

Our work indicates that rather limited progress was made through deep learning approaches applied to the classic *top-n* recommendation problem, at least when considering the papers that were analyzed in present work and which were published at top-level scientific conferences. While many papers claiming to make advances over the state of the art were published, these mostly seemed to amount to *phantom progress*. In this section, we will review the possible causes for the apparent stagnation in this field. We will structure our discussions along three dimensions: reproducibility, methodology, and fundamental considerations.

5.1 Reproducibility

The reproducibility statistics for the conferences we analysed in this study are reported in Table 1. Using our specific criteria, we found that 12 of 26 papers could be reproduced based on the provided code and publicly available data sets. In recent years, we could observe an increased trend for researchers to share the source code of their algorithms and experiments, probably due to the increased importance of reproducibility as a criterion in the academic peer-reviewing process. Nonetheless, we found that for more than half of the papers the source code was either not provided or it was lacking important details to the extent that a reliable reproduction of the results was made difficult or impossible.

The reasons for not sharing the code, not even in private with other researchers, sometimes remain obscure. Technically, a scientific paper should contain all necessary information for others to reproduce the work. Nonetheless, providing the artifacts (code, data sets) that were used in the experiments is important, as details may sometimes be missing from the papers, e.g., for space reasons, that could have a significant impact on the observed results. Generally, however, establishing reproducibility in the context of algorithmic proposals, as discussed in our paper, is relatively easy compared to other disciplines. Nowadays, applied research in deep learning is mostly based on public libraries and freely accessible software tools. Virtualization technology can also be leveraged effectively to make the entire software environment used for the experiments available to the community. Furthermore, while many algorithms have some non-deterministic elements, the overall performance is usually quite stable over several runs, which should allow other researchers to obtain at least comparable results when using the artifacts of the original authors.

Sometimes, researchers argue that they cannot provide the code because of the rules of the organization employing them. This, however, seems contradictory. Once a method is proposed and published in a scientific paper, others could in principle use it as-is, without restrictions. The competitive gain of organizations therefore seems limited. In some cases, the data used in the experiments is also not made available to the community, which means that no one will ever have the chance to verify, and possibly contest or falsify, the claimed results under the exact same conditions.

In our study, we contacted the authors of papers where the artifacts were not publicly available. While some of the authors responded to our inquiries and shared their code with us, we found that in the great majority of cases (10 out of 14 for non reproducible papers) we received no reply. We can only speculate about the reasons for these phenomena. Clearly, providing code for others can lead to a substantial amount of extra work for the researcher. Furthermore, if reviewers do not put much emphasis on this, there might be not enough incentives for the researcher to go the extra mile. At the same time, publishing all artifacts can make certain assumptions or limitations of a method more obvious, e.g., when the scalability of a method is limited. In general, however, it should be in the interest of the community, and hence of researchers themselves, that others can reproduce, confirm or falsify their results, as this is a cornerstone of scientific research in the first place.

5.2 Methodological Issues

Throughout the paper, we reported a number of methodological issues that can contribute to the limited or non-existing progress in this field. Here, we summarize our main observations.

- *Choice of baselines:* Algorithms for *top-n* recommendation problems have been investigated for decades, and now it is less than clear what represents the state-of-the-art. Determining the state-of-the-art can be difficult or even impossible due to the fact that there exist no general "best" algorithm. In fact, performance rankings of algorithms depend on a variety of factors, including the dataset characteristics or the evaluation procedure and measure. Another observation is that often only complex machine learning methods are considered as baselines. Comparably simple methods like $P^3\alpha$ and $RP^3\beta$ are not widely known in the community, even though they were published at top-level venues and often lead to strong results.
- *Propagation of weak baselines:* Nowadays, methods like NCF are often considered as competitive state-of-the-art baselines, even though our analysis showed that they are often not better than relatively simple and well-known techniques.
- *Lack of proper tuning of baselines:* This is probably the most striking observation of our analysis and is not specifically tied to deep learning approaches [49] or to recommendation problems [33]. Researchers apparently invest significant efforts in optimizing their own new method but do not pay the same attention to their baselines. Sometimes, authors simply pick the hyperparameter settings reported to be optimal from a previous paper, even though those may refer to a different dataset or experimental procedure. Probably, this behavior might be the result of a *confirmation bias*, i.e., the tendency to search for results that affirm rather than refute prior research hypotheses.

Regarding the choice of the baselines, we found that in some cases researchers "re-use" the experimental design that was used in previous studies, i.e., they use the same datasets, evaluation protocol and metrics, to demonstrate progress. This is in principle very meaningful as it helps the comparison of results. However, this also means that the experimental setup is not questioned anymore, because it has been used by many researchers before. Some papers in our present study are based on two quite small CiteULike datasets that were used in some early deep learning papers. These datasets were reused by other authors later on, without questioning if these quite small datasets were representative for a larger set of real-world recommendation problems or whether they were useful at all to analyze certain phenomena. In our analysis, we even found that the Epinions dataset had such characteristics that the non-personalized recommendation of the most popular items was favorable over any personalized technique.

In general, the reuse of experimental designs is still the exception rather than the norm. In our analysis, we found that researchers use a large variety of evaluation protocols, datasets, and performance measures in their experiments. This is expected in articles having different goals and aimed at different scenarios. However, in most cases, there is no particular argumentation on why a certain metric is used or why particular datasets from a certain domain serve as a basis for the research. To illustrate these phenomena, we show in Table 22 which datasets were used for evaluation in the reproducible papers.

Besides the fact that 12 reproducible articles used 18 different datasets, the authors also relied on quite a number of different data splitting procedures, including *leave-one-out*, *leave-last-out*, *80/20 training/test split*, *hold-out of users* or *retain only 1 or 10 interactions per user*. When evaluating, researchers somewhat arbitrarily used 50, 100, or 1000 negative samples per positive sample for ranking. Moreover, the metrics–including the Hit Rate, Precision, Recall, MAP or NDCG—were measured on a variety of cut-off thresholds between 1 and 300. In addition, several pre-processing strategies were applied, for example, retaining only users or items with an arbitrary minimum number of interactions. In most cases, no justification is provided for why a particular choice of pre-processing, metrics and cut-off was selected and which relevant scenario aims to represent.

Table 22. Datasets used by reproducible papers.

Dataset	Paper	Dataset	Paper
Amazon Movie	[66]	Amazon Music	[66] [11]
CiteULike-a	[16] [31] [62]	CiteULike-t	[31] [62]
Epinions	[16]	FilmTrust	[69]
Frappe	[69]	Gowalla	[22]
LastFM	[70] [26]	Movielens Hetrec	[69]
Movielens100K	[26] [66]	MovieLens1M	[70] [23] [69] [68] [11]
MovieLens20M	[32]	MSD	[32]
Netflix Prize	[62] [32]	Pinterest	[16] [23]
Tafeng	[68]	Yelp	[70] [26] [22]

Regarding the issue of baselines often not being properly tuned, one problematic factor with deep learning methods is that most of them are computationally complex, and some of them have a large number of hyperparameters to tune. Furthermore, it is usually also possible to vary the structure of the network in terms of the layers and nodes in each layer. An exhaustive search through the hyperparameter space for a large number of network architectures is therefore often not possible in reasonable time.

Regarding absolute running times, let us consider a few examples.[28]

- The often-cited NCF (NeuMF) method (Section 4.3) needs *four hours* to train on one of our machines for the popular MovieLens1M dataset. Training the best-performing SLIM method on the same machine and dataset requires only *4 minutes*. Interestingly, the more recent EASE^R method has better accuracy than NeuMF, but only needs about *12 seconds* for training on the same machine. Training NeuMF on the slightly larger Pinterest dataset (1.5M interactions) took more than *two days*, compared to the training time of SLIM (*12 minutes*) or EASE^R (*2 minutes*).
- Similar observations can be made for the early CDL method (Section 4.1). On the larger CiteULike-a dataset with about 55k interactions, CDL needs almost two hours for training, the EASE^R method 2 minutes, and our well-performing ItemKNN CFCBF hybrid less than 10 seconds to pre-compute the neighborhoods.
- The Mult-VAE method (Section 4.2), which significantly outperformed our baselines on one dataset, seems to be also favorable over both baselines and other neural methods in terms of training times. For the relatively large MovieLens20M dataset, training needs about 21 minutes on our machine. This is much faster than the SLIM algorithm, which needs almost two hours and was competitive with Mult-VAE on the other dataset. EASE^R, which is on a par with SLIM, is again favorable here, requiring only about 3 minutes for training.
- The algorithm with the longest training time on our machine is DMF. Its best variant, based on normalized cross entropy, requires almost *5 days* of training on the MovieLens1M dataset, while the simple iALS baseline, able to outperform DMF an all measures, requires only *4 minutes* of training.

[28] All reported measurements were made on the same hardware: one AWS instance p3.2xlarge, with 8 vCPU, 30GB RAM, and one Tesla V100-SXM2-16GB GPU. All deep-learning algorithms (with the exception of CDL) were trained using the GPU. All of our baselines were trained only on the CPUs. The detailed measurements are available in the online material (Section 3)

Limited scalability is not a methodological issue per se. However, the enormous costs for tuning the baselines can lead to researchers rather take hyperparameter settings from previous papers—even though they were determined for different evaluation setups—and reuse them in their own experiments.

Besides problems related to the baselines, we observed a number of other technical issues. These include uncommon implementations of ranking measures, non-randomized data splits, reporting best results across different epochs, and determining the best number of epochs on the test set.

5.3 Fundamental Issues

Probably the most important factors that contribute to the apparent stagnation are today's incentive mechanisms in the academic system and our established research practices, see also [35]. Regarding the incentive system, researchers are more and more evaluated based on the citations their works receive. This might lead to the phenomenon that researchers develop a tendency to investigate problems that are popular and (easily) "publishable" [61]. With the ongoing "neural hype" [33], papers that do not propose or use deep learning approaches might get criticized by reviewers for not using state-of-the-art technology. It therefore might appear much easier to publish a neural approach than other works, even though the true value of these complex models is not always fully clear. Furthermore, with the thinness of the reviewer pool, the "mathiness" of many papers [35] and the sheer amount of machine learning papers submitted every year, it becomes more and more difficult to identify those works that truly move the field forward.

One particularly seductive aspect in the context of algorithmic works on *top-n* recommendation is that there is an implicit general agreement about how research in this area should be done. Unlike in other types of recommender systems research, e.g., research that involves user studies, the experimental offline evaluation design approach is generally pre-determined and is not questioned. Generally, to have a paper accepted at one of the prestigious conferences considered in this paper, one has to (at least) propose a new technical approach that outperforms some state-of-the-art methods on one evaluation protocol and on at least a couple of established metrics and publicly available datasets. Which dataset is used often seems arbitrary, and today's papers in most cases do not motivate their work based on domain-specific considerations or an underlying theory.[29] Researchers are also flexible in terms of their evaluation method. As discussed above, various protocol variants and accuracy measures are used today, and in most papers the selection of measures is not explained. Finally, as discussed throughout the paper, it is difficult to know what represents the state-of-the-art in a certain area or for a specific subproblem.

All these degrees of freedom make it very tempting for researchers to focus on accuracy improvements for *top-n* recommendation, where it is in some sense easy to "prove" progress and where no special research design has to be developed and argued for. However, the established research model, as discussed, allows for a lot of arbitrariness.

For example, a novel algorithm may be compared against a baseline using a number of accuracy metrics at different cut-off lengths on multiple datasets. The probability of finding some combinations of measures and datasets by which the novel algorithm seems to outperform the baseline increases with the number of cases examined [21].

Online machine learning competitions as hosted, e.g., on Kaggle[30], represent the other extreme. On Kaggle and also in specific recommender systems challenges, the dataset and the evaluation method are pre-defined by the organization running the competition. Furthermore, participants do not see the test set and usually do not measure the recommendation quality by themselves, which

[29] See also [29] for a satirical discussion of this general problem in machine learning.

[30] https:///www.kaggle.com

avoids, as we have discovered in our study, a potential source of mistakes. One typical criticism of such online competitions is that they result in a "leaderboard chasing" culture, which can lead to limited innovation.

All in all, current research practice can easily create an illusion of progress in terms of accuracy improvements. But even when this progress is real, the tragedy lies in us not even knowing with certainty whether these accuracy improvements will lead to better recommendations in practice, neither in terms of business value nor in terms of the quality perception by users. In fact, a number of research works indicate that algorithms with higher offline accuracy do not necessarily lead to better perceived recommendation quality [4, 13, 20, 40, 51]. For this reason, recommender systems research is different from other areas like image recognition or natural language processing (e.g., automated translation), where accuracy improvements can often directly lead to better systems.

This problem of focusing solely on accuracy optimization has actually been well known in the recommender systems community for many years [42], and the obsessive hunt for the "best model" is also common to other areas of applied machine learning research [61]. As shown in comparative evaluations like [37], however, there actually *is no best model*, and the ranking of algorithms depends on many factors like dataset characteristics or evaluation approach. While there is no doubt that being better able to predict the relevance of items for individual users is something useful, considering only abstract accuracy measures appears to be a strong oversimplification of the problem.

6 CONCLUSION

Our work reveals that despite the large number of new methods that are published on the topic of *top-n* recommendation every year, the progress in this field seems actually limited or non-existent. Our analysis shows that sometimes even relatively simple methods lead to performance levels that are similar or even better than the most recent complex neural methods. In other cases, the computationally expensive neural methods did not outperform well-established matrix factorization approaches or linear models. A number of issues contribute to the observed phenomena. We not only identified different methodological problems but also found that the choice of the baselines and often the lack of a proper optimization of these baselines represent key issues that hamper our progress. These phenomena however are not specific to the domain of recommender systems or to neural approaches.

Increasing the reproducibility of published research was identified as one of possible strategies to mitigate some of the observed problems. In the information retrieval research community, questions of replicability and reproducibility have recently received more attention, probably thanks in part to the surprising results from [33, 67], and [3]. An increased awareness and corresponding initiatives are also advisable for the recommender systems community. However, even with better reproducibility, fundamental problems of algorithms-based recommender systems research remain. The reason is that, unlike in some IR tasks, better retrieval or prediction performance does not necessarily lead to recommendations that are better in terms of the users' quality perception or in terms of the business value for providers.

In the end, these problems lead to a certain stagnation of the research field. A large number of researchers hunt for the "best" model, even though there are many indications that no such model exists, as the performance ranking of algorithms depends on many factors. Unfortunately, as discussed in [25] for the domain of theoretical physics[31], there are no strong signs of a crisis, where more and more researchers critically reflect on what has been claimed and what has been

[31] See also http://backreaction.blogspot.com/2018/11/the-present-phase-of-stagnation-in.html

actually achieved in the past decades. The ongoing wave of machine learning research, in contrast, seems to add to the stagnation.

A HYPERPARAMETER RANGES

Table 23. Hyperparameter list, value ranges and distributions for the baselines reported in this paper.

Algorithm	Hyperparameter	Range	Type	Distribution
UserKNN, ItemKNN cosine	topK	5 - 1000	Integer	uniform
	shrink	0 - 1000	Integer	uniform
	normalize[a]	True, False	Categorical	
	feature weighting	none, TF-IDF, BM25	Categorical	
UserKNN, ItemKNN dice	topK	5 - 1000	Integer	uniform
	shrink	0 - 1000	Integer	uniform
	normalize[a]	True, False	Categorical	
UserKNN, ItemKNN jaccard	topK	5 - 1000	Integer	uniform
	shrink	0 - 1000	Integer	uniform
	normalize[a]	True, False	Categorical	
UserKNN, ItemKNN asymmetric	topK	5 - 1000	Integer	uniform
	shrink	0 - 1000	Integer	uniform
	normalize[a]	True	Categorical	
	asymmetric alpha	0 - 2	Real	uniform
	feature weighting	none, TF-IDF, BM25	Categorical	
UserKNN, ItemKNN tversky	topK	5 - 1000	Integer	uniform
	shrink	0 - 1000	Integer	uniform
	normalize[a]	True	Categorical	
	tversky alpha	0 - 2	Real	uniform
	tversky beta	0 - 2	Real	uniform
P3alpha	topK	5 - 1000	Integer	uniform
	alpha	0 - 2	Real	uniform
	normalize similarity[b]	True, False	Categorical	
RP3beta	topK	5 - 1000	Integer	uniform
	alpha	0 - 2	Real	uniform
	beta	0 - 2	Real	uniform
	normalize similarity[b]	True, False	Categorical	
SLIMElasticNet	topK	5 - 1000	Integer	uniform
	l1 ratio	10^{-5} - 10^{0}	Real	log-uniform
	alpha	10^{-3} - 10^{0}	Real	uniform
PureSVD	num factors	1 - 350	Integer	uniform
iALS	num factors	1 - 200 [c]	Integer	uniform
	epochs	1 - 500	Integer	early-stopping
	confidence scaling	linear, log	Categorical	
	alpha	10^{-3} - $5 \cdot 10^{+1}$ [d]	Real	log-uniform
	epsilon	10^{-3} - 10^{+1} [d]	Real	log-uniform
	reg	10^{-5} - 10^{-2}	Real	log-uniform
EASE^{R}	l2 norm	10^{0} - 10^{+7}	Real	log-uniform

[a] The *normalize* hyperparameter in KNNs refers to the use of the denominator when computing the similarity.

[b] The *normalize similarity* hyperparameter refers to applying L1 regularisation on the rows of the similarity matrix

[c] The number of factors is lower than PureSVD due to iALS being slower.

[d] The maximum value of this hyperparameter had been suggested in the article proposing the algorithm.

REFERENCES

[1] Fabio Aiolli. Efficient top-n recommendation for very large scale binary rated datasets. In *Proceedings of the 7th ACM Conference on Recommender Systems (RecSys '13)*, pages 273–280. ACM, 2013.

[2] Sebastiano Antenucci, Simone Boglio, Emanuele Chioso, Ervin Dervishaj, Shuwen Kang, Tommaso Scarlatti, and Maurizio Ferrari Dacrema. Artist-driven layering and user's behaviour impact on recommendations in a playlist continuation scenario. In *Recommender Systems Challenge Workshop at the 12th ACM Conference on Recommender Systems (RecSys '18)*, pages 4:1–4:6, 2018.

[3] Timothy G. Armstrong, Alistair Moffat, William Webber, and Justin Zobel. Improvements that don't add up: Ad-hoc retrieval results since 1998. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management (CIKM '09)*, pages 601–610, 2009.

[4] Jöran Beel and Stefan Langer. A comparison of offline evaluations, online evaluations, and user studies in the context of research-paper recommender systems. In *Proceedings 19th International Conference on Theory and Practice of Digital Libraries (TPDL '15)*, pages 153–168, 2015.

[5] Robert M Bell and Yehuda Koren. Improved neighborhood-based collaborative filtering. In *KDD Cup and Workshop at the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '07)*, pages 7–14, 2007.

[6] Shlomo Berkovsky, Ivan Cantador, and Domonkos Tikk, editors. *Collaborative Recommendations – Algorithms, Practical Challenges and Applications*. World Scientific, 2019.

[7] Homanga Bharadhwaj, Homin Park, and Brian Y. Lim. Recgan: Recurrent generative adversarial networks for recommendation systems. In *Proceedings of the 12th ACM Conference on Recommender Systems (RecSys '18)*, pages 372–376, New York, NY, USA, 2018. ACM.

[8] Daniel Billsus and Michael J. Pazzani. Learning collaborative information filters. In *Proceedings of the 15th International Conference on Machine Learning (ICML '98)*, pages 46–54, 1998.

[9] John S. Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI '98)*, pages 43–52, 1998.

[10] Jingyuan Chen, Hanwang Zhang, Xiangnan He, Liqiang Nie, Wei Liu, and Tat-Seng Chua. Attentive collaborative filtering: Multimedia recommendation with item-and component-level attention. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '17)*, pages 335–344. ACM, 2017.

[11] Weiyu Cheng, Yanyan Shen, Yanmin Zhu, and Linpeng Huang. Delf: A dual-embedding based deep latent factor model for recommendation. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence, (IJCAI '18)*, pages 3329–3335, 2018.

[12] Colin Cooper, Sang Hyuk Lee, Tomasz Radzik, and Yiannis Siantos. Random walks in recommender systems: exact computation and simulations. In *Proceedings of the 23rd International Conference on World Wide Web (WWW '14)*, pages 811–816, 2014.

[13] Paolo Cremonesi, Franca Garzotto, and Roberto Turrin. Investigating the persuasion potential of recommender systems from a quality perspective: An empirical study. *Transactions on Interactive Intelligent Systems*, 2(2):1–41, 2012.

[14] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the 4th ACM Conference on Recommender Systems (RecSys '10)*, pages 39–46, 2010.

[15] Lee R. Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302, 1945.

[16] Travis Ebesu, Bin Shen, and Yi Fang. Collaborative memory network for recommendation systems. *Proceedings of the 41st International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '18)*, 2018.

[17] Ali Mamdouh Elkahky, Yang Song, and Xiaodong He. A multi-view deep learning approach for cross domain user modeling in recommendation systems. In *Proceedings of the 24th International Conference on World Wide Web (WWW '15)*, pages 278–288. International World Wide Web Conferences Steering Committee, 2015.

[18] Maurizio Ferrari Dacrema, Paolo Cremonesi, and Dietmar Jannach. Are we really making much progress? a worrying analysis of recent neural recommendation approaches. *Proceedings of the 13th ACM Conference on Recommender Systems (RecSys '19)*, 2019. Source: https://github.com/MaurizioFD/RecSys2019_DeepLearning_Evaluation.

[19] Antonino Freno, Martin Saveski, Rodolphe Jenatton, and Cédric Archambeau. One-pass ranking models for low-latency product recommendations. In *Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '15)*, pages 1789–1798, New York, NY, USA, 2015. ACM.

[20] Florent Garcin, Boi Faltings, Olivier Donatsch, Ayar Alazzawi, Christophe Bruttin, and Amr Huber. Offline and online evaluation of news recommender systems at Swissinfo.Ch. In *Proceedings of the 18th ACM Conference on Recommender Systems (RecSys '14)*, pages 169–176, 2014.

[21] Asela Gunawardana and Guy Shani. Evaluating recommender systems. In *Recommender systems handbook*, pages 265–308. Springer, 2015.

[22] Xiangnan He, Xiaoyu Du, Xiang Wang, Feng Tian, Jinhui Tang, and Tat-Seng Chua. Outer product-based neural collaborative filtering. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence, (IJCAI '18)*,

pages 2227–2233, 2018.

[23] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web (WWW '17)*, pages 173–182, 2017.

[24] José Miguel Hernández-Lobato, Matthew W Hoffman, and Zoubin Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 918–926. Curran Associates, Inc., 2014.

[25] Sabine Hossenfelder. *Lost in Math: How Beauty Leads Physics Astray.* Basic Books, 2018.

[26] Binbin Hu, Chuan Shi, Wayne Xin Zhao, and Philip S Yu. Leveraging meta-path based context for top-n recommendation with a neural co-attention model. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '18)*, pages 1531–1540. ACM, 2018.

[27] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM '08)*, volume 8, pages 263–272. Citeseer, 2008.

[28] Donghyun Kim, Chanyoung Park, Jinoh Oh, Sungyoung Lee, and Hwanjo Yu. Convolutional matrix factorization for document context-aware recommendation. In *Proceedings of the 10th ACM Conference on Recommender Systems (RecSys '16)*, pages 233–240, New York, NY, USA, 2016. ACM.

[29] D. LaLoudouana and M. Bonouliqui Tarare. http://www.jmlg.org/papers/laloudouana03.pdf. Data set selection. Online http://www.jmlg.org/papers/laloudouana03.pdf, 2002.

[30] Mark Levy and Kris Jack. Efficient top-n recommendation by linear regression. In *RecSys Large Scale Recommender Systems Workshop*, 2013.

[31] Xiaopeng Li and James She. Collaborative variational autoencoder for recommender systems. In *Proceedings of the 23th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '17)*, pages 305–314. ACM, 2017.

[32] Dawen Liang, Rahul G Krishnan, Matthew D Hoffman, and Tony Jebara. Variational autoencoders for collaborative filtering. In *Proceedings of the 27th International Conference on World Wide Web (WWW '18)*, pages 689–698. International World Wide Web Conferences Steering Committee, 2018.

[33] Jimmy Lin. The neural hype and comparisons against weak baselines. *SIGIR Forum*, 52(2):40–51, January 2019.

[34] G. Linden, B. Smith, and J. York. Amazon.com recommendations: item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, 2003.

[35] Zachary C. Lipton and Jacob Steinhardt. Troubling trends in machine learning scholarship, 2018.

[36] Pasquale Lops, Marco De Gemmis, and Giovanni Semeraro. Content-based recommender systems: State of the art and trends. In *Recommender systems handbook*, pages 73–105. Springer, 2011.

[37] Malte Ludewig and Dietmar Jannach. Evaluation of session-based recommendation algorithms. *User-Modeling and User-Adapted Interaction*, 28(4–5):331–390, 2018.

[38] Malte Ludewig, Noemi Mauro, Sara Latifi, and Dietmar Jannach. Performance comparison of neural and non-neural approaches to session-based recommendation. In *Proceedings of the 13th ACM Conference on Recommender Systems (RecSys '19)*, Copenhagen, 2019.

[39] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. Statistical and machine learning forecasting methods: Concerns and ways forward. *PloS one*, 13(3), 2018.

[40] Andrii Maksai, Florent Garcin, and Boi Faltings. Predicting online performance of news recommender systems through richer evaluation metrics. In *Proceedings of the 9th ACM Conference on Recommender Systems (RecSys '15)*, pages 179–186, 2015.

[41] Jarana Manotumruksa, Craig Macdonald, and Iadh Ounis. A contextual attention recurrent architecture for context-aware venue recommendation. In *Proceedings of the 41st International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '18)*, pages 555–564. ACM, 2018.

[42] Sean M. McNee, John Riedl, and Joseph A. Konstan. Being accurate is not enough: How accuracy metrics have hurt recommender systems. In *CHI '06 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '06, pages 1097–1101, New York, NY, USA, 2006. ACM.

[43] Bamshad Mobasher, Xin Jin, and Yanzan Zhou. Semantically enhanced collaborative filtering on the web. In Bettina Berendt, Andreas Hotho, Dunja Mladenič, Maarten van Someren, Myra Spiliopoulou, and Gerd Stumme, editors, *Web Mining: From Web to Semantic Web*, pages 57–76, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[44] ThaiBinh Nguyen and Atsuhiro Takasu. Npe: Neural personalized embedding for collaborative filtering. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI '18)*, pages 1583–1589. AAAI Press, 2018.

[45] Xia Ning and George Karypis. SLIM: Sparse linear methods for top-n recommender systems. In *Proceedings of the 11th IEEE International Conference on Data Mining (ICDM '11)*, pages 497–506, 2011.

[46] Arkadiusz Paterek. Improving regularized singular value decomposition for collaborative filtering. In *Proceedings KDD Cup and Workshop*, pages 39–42, 2007.

[47] Bibek Paudel, Fabian Christoffel, Chris Newell, and Abraham Bernstein. Updatable, accurate, diverse, and scalable recommendations for interactive applications. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 7(1):1, 2017.

[48] Ali Mustafa Qamar, Éric Gaussier, Jean-Pierre Chevallet, and Joo-Hwee Lim. Similarity learning for nearest neighbor classification. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM '08)*, pages 983–988, 2008.

[49] Steffen Rendle, Li Zhang, and Yehuda Koren. On the difficulty of evaluating baselines: A study on recommender systems. *CoRR*, abs/1905.01395, 2019.

[50] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM Conference on Computer-Supported Cooperative Work (CSCW '94)*, pages 175–186, 1994.

[51] Marco Rossetti, Fabio Stella, and Markus Zanker. Contrasting offline and online results when evaluating recommendation algorithms. In *Proceedings of the 10th ACM Conference on Recommender Systems (RecSys '16)*, pages 31–34, 2016.

[52] Noveen Sachdeva, Kartik Gupta, and Vikram Pudi. Attentive neural architecture incorporating song features for music recommendation. In *Proceedings of the 12th ACM Conference on Recommender Systems (RecSys '18)*, pages 417–421, New York, NY, USA, 2018. ACM.

[53] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web (WWW '01)*, pages 285–295, 2001.

[54] Harald Steck. Embarrassingly shallow autoencoders for sparse data. In *Proceedings of the 28th International Conference on World Wide Web (WWW '19)*, TheWebConf 2019, pages 3251–3257, 2019.

[55] Zhu Sun, Jie Yang, Jie Zhang, Alessandro Bozzon, Long-Kai Huang, and Chi Xu. Recurrent knowledge graph embedding for effective recommendation. In *Proceedings of the 12th ACM Conference on Recommender Systems (RecSys '18)*, pages 297–305, New York, NY, USA, 2018. ACM.

[56] Yi Tay, Luu Anh Tuan, and Siu Cheung Hui. Latent relational metric learning via memory-based attention for collaborative ranking. In *Proceedings of the 27th International Conference on World Wide Web (WWW '18)*, pages 729–739. International World Wide Web Conferences Steering Committee, 2018.

[57] Yi Tay, Luu Anh Tuan, and Siu Cheung Hui. Multi-pointer co-attention networks for recommendation. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '18)*, 2018.

[58] Trinh Xuan Tuan and Tu Minh Phuong. 3d convolutional networks for session-based recommendation with content features. In *Proceedings of the 11th ACM Conference on Recommender Systems (RecSys '17)*, pages 138–146, New York, NY, USA, 2017. ACM.

[59] Amos Tversky. Features of similarity. *Psychological Review*, 84(4):327–352, 1977.

[60] Flavian Vasile, Elena Smirnova, and Alexis Conneau. Meta-prod2vec: Product embeddings using side-information for recommendation. In *Proceedings of the 10th ACM Conference on Recommender Systems (RecSys '16)*, pages 225–232, New York, NY, USA, 2016. ACM.

[61] Kiri Wagstaff. Machine learning that matters. In *Proceedings of the 29th International Conference on Machine Learning (ICML '12)*, pages 529–536, 2012.

[62] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. Collaborative deep learning for recommender systems. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '15)*, pages 1235–1244, 2015.

[63] Jun Wang, Stephen Robertson, Arjen P de Vries, and Marcel JT Reinders. Probabilistic relevance ranking for collaborative filtering. *Information Retrieval*, 11(6):477–497, 2008.

[64] Yao Wu, Christopher DuBois, Alice X Zheng, and Martin Ester. Collaborative denoising auto-encoders for top-n recommender systems. In *Proceedings of the 9th ACM International Conference on Web Search and Data Mining (WSDM '16)*, pages 153–162. ACM, 2016.

[65] Zhenghua Xu, Thomas Lukasiewicz, Cheng Chen, Yishu Miao, and Xiangwu Meng. Tag-aware personalized recommendation using a hybrid deep model. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence, (IJCAI '17)*, pages 3196–3202, 2017.

[66] Hong-Jian Xue, Xinyu Dai, Jianbing Zhang, Shujian Huang, and Jiajun Chen. Deep matrix factorization models for recommender systems. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence, (IJCAI '18)*, pages 3203–3209, 2017.

[67] Wei Yang, Kuang Lu, Peilin Yang, and Jimmy Lin. Critically examining the neural hype: Weak baselines and the additivity of effectiveness gains from neural ranking models. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '19)*, pages 1129–1132, 2019.

[68] Quangui Zhang, Longbing Cao, Chengzhang Zhu, Zhiqiang Li, and Jinguang Sun. Coupledcf: Learning explicit and implicit user-item couplings in recommendation for deep collaborative filtering. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence, (IJCAI '18)*, pages 3662–3668, 7 2018.

[69] Shuai Zhang, Lina Yao, Aixin Sun, Sen Wang, Guodong Long, and Manqing Dong. Neurec: On nonlinear transformation for personalized ranking. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence, (IJCAI '18)*, pages 3669–3675, 2018.

[70] Lei Zheng, Chun-Ta Lu, Fei Jiang, Jiawei Zhang, and Philip S. Yu. Spectral collaborative filtering. In *Proceedings of the 12th ACM Conference on Recommender Systems (RecSys '18)*, pages 311–319, 2018.