On the Generalizability and Predictability of Recommender Systems

Duncan McElfresh[*1] **Sujay Khandagale**[*1] **Jonathan Valverde**[*2] **John P. Dickerson**[2] **Colin White**[1]

[1]Abacus.AI [2]University of Maryland

Abstract While other areas of machine learning have seen more and more automation, designing a high-performing recommender system still requires a high level of human effort. Furthermore, recent work has shown that modern recommender system algorithms do not always improve over well-tuned baselines. A natural follow-up question is, "how do we choose the right algorithm for a new dataset and performance metric?" In this work, we start by giving the first large-scale study of recommender system approaches by comparing 18 algorithms and 100 sets of hyperparameters across 85 datasets and 315 metrics. We find that the best algorithms and hyperparameters are highly dependent on the dataset and performance metric, however, there are also strong correlations between the performance of each algorithm and various meta-features of the datasets. Motivated by these findings, we create RecZilla, a meta-learning approach to recommender systems that uses a model to predict the best algorithm and hyperparameters for new, unseen datasets. By using far more meta-training data than prior work, RecZilla is able to substantially reduce the level of human involvement when faced with a new recommender system application. We not only release our code and pretrained RecZilla models, but also all of our raw experimental results, so that practitioners can train a RecZilla model for their desired performance metric: `https://github.com/naszilla/reczilla`.

1 Introduction

While some areas of machine learning have benefitted greatly from repurposing existing computation through pretrained models [20, 50, 32, 21, 44], recommender system (rec-sys) research has followed a different trajectory: despite their widespread usage across many e-commerce, social media, and entertainment companies such as Amazon, YouTube, and Netfix [11, 26, 52], there is far less work in reusing models. Many rec-sys techniques are designed and optimized with just a *single* dataset in mind [26, 31, 11, 40, 55]. Intuitively, this might be because each rec-sys application is highly unique based on the dataset and the target metric. For example, a typical user session looks very different among e.g. YouTube, Home Depot, and AirBnB [11, 40, 31]. However, this intuition has not been formally established. Furthermore, recent work has shown that neural recommender system algorithms do not always improve over well-tuned baselines such as k-nearest neighbor and matrix factorization [18]. A natural question is then, "how do we choose the right algorithm for a new dataset and performance metric?"

In this work, we show that the best algorithm and hyperparameters are highly dependent on the dataset and user-defined performance metric. Specifically, we run the first large-scale study of rec-sys approaches by comparing 18 algorithms across 85 datasets and 315 metrics. For each dataset and algorithm pair, we test up to 100 hyperparameters (given a 10 hour time limit per pair). The codebase that we release, which includes a unified API for a large, diverse set of algorithms, datasets, and metrics, may be of independent interest. We show that the algorithms do not *generalize* – the

[*]Equal contribution. This work was extended to a full-length paper here: `https://arxiv.org/abs/2206.11886`.

 © 2022 the authors, released under CC BY 4.0

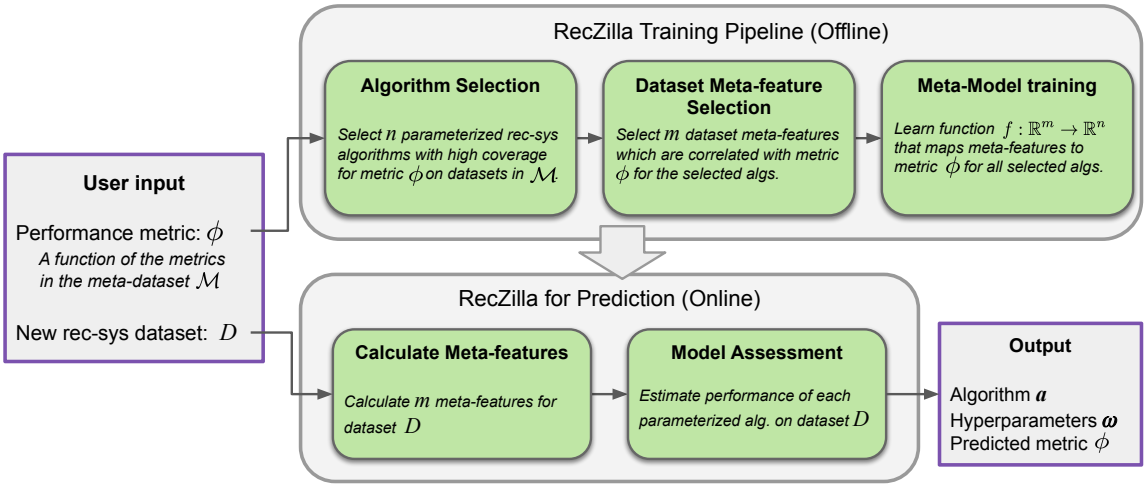

Figure 1: RecZilla recommends a parameterized rec-sys algorithm for a user-provided dataset and performance metric. The RecZilla pipeline is built using a meta-dataset $\mathcal{M}$ that includes many different performance metrics evaluated on many different rec-sys algorihtms on many different datasets; we estimate algorithm performance using dataset meta-features.

performance of algorithms changes substantially across datasets and across performance metrics. Furthermore, the best hyperparameters on one dataset often perform significantly worse than the best hyperparameters on a different dataset. On the other hand, we *do* show that various meta-features of the dataset can be used to *predict* the performance of rec-sys algorithms.

Motivated by these findings, we introduce RecZilla, a meta-learning-based algorithm selection approach (see Figure 1) inspired by SATzilla [57]. At the core of RecZilla is a model that, given a user-defined performance metric, predicts the best rec-sys algorithm and hyperparameters for a new dataset based on meta dataset features such as number of users and items, and spectral properties of the interaction matrix. We show that RecZilla quickly finds high-performing algorithms on datasets it has never seen before. While there has been prior work on meta-learning for recommender systems [16, 17], no prior work is metric-independent, searches for hyperparameters as well as algorithms, or considers more than nine dataset families. By running an ablation study on the number of meta-training datasets, we show that more dataset families are crucial to the success of RecZilla. We release ready-to-use, pretrained RecZilla models for common test metrics, and we release the raw results from our large-scale study, along with code so that practitioners can easily train a new RecZilla model for their specific performance metric of interest.

Our contributions. We summarize our main contributions below.

- We run a large-scale study of recommender systems, showing that the best algorithm and hyperparameters are highly dependent on the dataset and user-defined performance metric. We also show that dataset meta-features are predictive of the performance of algorithms.
- We create RecZilla, an algorithm selection approach which, given a performance metric, efficiently predicts the best algorithm and set of hyperparameters on new datasets.
- We release a public repository containing 85 datasets and 18 rec-sys algorithms, accessed through a unified API. Furthermore, we release both pretrained RecZilla models, and raw data so that users can train a new RecZilla model on their desired metric.

Related work. Recommender systems are a widely studied area of research [8]. Common approaches include k-nearest neighbors [1], matrix factorization [39, 43], and deep learning approaches [11, 26, 52]. For a survey on recommender systems, see [8, 4]. A recent meta-study showed that of the

12 neural rec-sys approaches published at top conferences between 2015 and 2018, 11 performed worse than well-tuned baselines (e.g. nearest neighbor search or linear models) [18].

Algorithm selection for recommender systems was first studied in 2011 [34] by using a graph representation of item ratings. Follow-up work used dataset meta-features to select the best nearest neighbor and matrix factorization algorithms [23, 3, 28]. Subsequent work focused on improving the model and framework [17] including studying 74 meta-features systematically [13]. More recent approaches from 2018 run meta-learning for recommender systems by casting the meta-problem itself as a collaborative filtering problem. Performance is then estimated with subsampling landmarkers [14, 16, 15]. No prior work in algorithm selection for rec-sys includes open-source Python code. There is also work on automated machine learning (AutoML) for recommender systems, without meta-learning [56, 6, 29, 30]. To the best of our knowledge, no meta-learning or AutoML rec-sys papers have run experiments on more than nine dataset families or four test metrics, and no prior work predicts hyperparameters in addition to algorithms.

2 Analysis of Recommender Systems

In this section, we present a large-scale empirical study of rec-sys algorithms across a diverse set of datasets and metrics. We assess the following two research questions.

1. **Generalizability**. If a rec-sys algorithm or set of hyperparameters performs well on one dataset and metric, will it perform well on other datasets or on other metrics?
2. **Predictability**. Given a metric, can various dataset meta-features be used to predict the performance of rec-sys algorithms?

Experimental design. We run 18 rec-sys algorithms, including clustering-based, matrix factorization, linear, and baseline methods. We run these algorithms on 85 datasets from 19 dataset "families": a family refers to an original dataset (such as Movielens), while "dataset" refers to a single train-test split drawn from the original dataset, which may be a small subset of the original. We use 21 different base metrics (such as precision, recall, NDCG) computed at 15 different cutoff values. For full details of the algorithms, datasets, and metrics, see Appendix A.

For each dataset, we compute a train and test split based on leave-last-k-out (and our repository also includes splits based on global timestamp). For eac algorithm, we expose several hyperparameters and give ranges based on common values. For each dataset, we run each algorithm on a random sample of up to 100 hyperparameter sets. Each algorithm is allocated a 10 hour limit for each dataset split; we train and test the algorithm with at most 100 hyperparameter sets on an `n1-highmem-2` CPU, until the time limit is reached. Each algorithm is trained on the train split, and the performance metrics are computed on the test split. By running 18 algorithms, up to 100 hyperparameters, and 85 datasets, this resulted in 84 769 successful experiments, and by computing 315 metrics, our final meta-dataset of results includes more than 26 million evaluations.

Generalizability. Our first observation is that *all* algorithms perform *well* on some datasets, and poorly on others. First we identify the best-performing hyperparameter set for each (algorithm, dataset) pair—to simulate hyperparameter optimization using our meta-dataset. We then rank all algorithms for each dataset, according to several performance metrics. If we focus on a single metric, then most algorithms are ranked first according to this metric on at least one dataset.

Average performance is more varied: some algorithms tend to perform better than others. Table 7 shows the mean, min (best) and max (worst) ranking of all 18 algorithms over all dataset and all accuracy and hit-rate metrics. Nearly all algorithms are ranked first for at least one metric, on at least one dataset; the only exception is Random, which has a minimum rank 2. Many algorithms perform very well on average; interestingly, the three algorithms with the highest average ranking each come from different algorithm families: Item-KNN is a similarity-based metric, SLIM-BPR is based on linear models, and SVD is a matrix factorization method. Furthermore, most algorithms perform very poorly in some cases: the maximum rank is at least 15 (out of 18) for all algorithms.

Predictability. We calculate 383 different meta-features to characterize each dataset. These meta-features include statistics on the rating matrix—including basic statistics, the distribution-based features of Cunha et al. [13], and landmark features [14]—which measure the performance of simple rec-sys algorithms on a subset of the training dataset. Since these meta-features are used for algorithm selection in Section 3, they are calculated using only the training split of each dataset. For more details on the dataset meta-features, see Appendix A.4.

We find that some meta-features are highly correlated with the performance of algorithms. For example, "mean of item rating count distribution" has a correlation of 0.941 with SlopeOne, and "median of item rating count distribution" has a correlation of 0.933 with CoClustering. See Table 8 for more details. This experiment motivates the design of RecZilla in the next section, which trains a model using dataset meta-features to predict the performance of algorithms on new datasets.

In Appendix A, we train three different meta-learner functions (XGBoost, KNN, and linear regression) using our meta-dataset, to predict performance metric PREC@10 for 10 rec-sys algorithms with high average performance. MAE decreases as more dataset families are added, suggesting that it is possible to estimate rec-sys algorithm performance using dataset meta-features.

3 RecZilla: Automated Algorithm Selection

In the previous section, we found that *(1)* the best algorithm and hyperparameters strongly depend on the dataset and user-chosen performance metric, and *(2)* the performance of algorithms can be predicted from dataset meta-features. Points *(1)* and *(2)* naturally motivate an algorithm selection approach to rec-sys powered by meta-learning.

In this section, we present *RecZilla*, which is motivated by a practical challenge: given a performance metric and a new rec-sys dataset, quickly identify an algorithm and hyperparameters that perform well on this dataset. This challenge arises in many settings—e.g., when selecting good baseline algorithms for academic research, or when developing high-performing rec-sys algorithms for a commercial application. We begin with an overview and then formally present our approach.

Overview. As alluded to earlier, *RecZilla* is an algorithm selection approach powered by meta-learning. We use the results from the previous section as the meta-training dataset. Given a user-specified performance metric, we train a meta-model that predicts the performance of each of a set of algorithms and hyperparameters on a dataset, by using meta-features of the dataset. Given a new, unseen dataset, we compute the meta-features of the dataset, and then use the meta-model to predict the performance of each algorithm, returning the best algorithm according to the user-selected performance metric. See Figure 1, and see Appendix B for the full details of RecZilla.

Experimental setup. Focusing on the performance metric PREC@10, we build a meta-dataset $\mathcal{M}$ using all rec-sys datasets, algorithms, and meta-features described in Section 2. All meta-learners are evaluated using leave-one-dataset-out evaluation: we iteratively select each dataset *family* as the meta-test dataset, and run the full RecZilla pipeline using the remaining datasets as the meta-training data. Splitting on dataset families rather than datasets ensures that there is no test data leakage. Then for each dataset D in the test set, we compare the performance metric of the predicted best parameterized algorithm to the performance metric of the ground-truth best algorithm using `%Diff`: the percentage difference of PREC@10 on the predicted best algorithm vs. the ground-truth best algorithm.

Comparisons to existing methods. We compare RecZilla to polynomial SVM with 74 meta-features (the best approach from a 2018 analysis [17]) and CF4CF-META [15], which combines CF4CF [16] with earlier meta-learning approaches. Due to their basis on all prior work in the area, these two methods can be seen as representative of all prior work on algorithm selection for recommender systems. We refer to them by `cunha2018` and `cf4cf-meta`. Note that `cunha2018` has no open-source code, and `cf4c4-meta` only has code in R. Furthermore, in order to give a more fair empirical study, we implement both approaches directly within our codebase. Each model uses the same meta-training datasets, algorithm selection procedure, and base algorithms. Since a main novelty

Table 1: Comparison between RecZilla and two representative prior algorithm selection approaches. We report the mean and standard deviation across 50 trials for 19 test sets, for 950 total trials. The runtime is the average time it takes to output predictions on the meta-test dataset.

Approach	Runtime (sec)	%Diff (↓)	PREC@10 of best pred. (↑)
`cunha2018` [17]	**0.39**	52.9 ± 23.0	0.00813 ± 0.0113
`cf4cf-meta` [15]	6.68	43.5 ± 21.8	0.00808 ± 0.00773
RecZilla	6.69	**35.1± 24.1**	**0.00884± 0.00848**

of RecZilla is predicting hyperparameters as well as algorithms, the other two approaches are only given the algorithms with the default hyperparameters.

We compare RecZilla (with an XGBoost model) to `cunha2018` and `cf4c4-meta`. The algorithms are given all 18 dataset families not in the test set, to use as training data. We run 50 trials for all 19 possible test sets in the leave-one-dataset-out evaluation, for a total of 950 trials. See Table 1. RecZilla outperforms the other two approaches in both %Diff and in terms of the PREC@10 value of the rec-sys algorithm outputted by each meta-learning algorithm.

In Appendix B, we give an ablation study on the number of training meta-datapoints and meta-features used by RecZilla, as well as the meta-model of RecZilla.

4 Conclusions, Limitations, and Broader Impact

In this work, we conducted the first large-scale study of rec-sys approaches: we compared 18 algorithms and 100 sets of hyperparameters across 85 datasets and 315 metrics. We showed that for a given performance metric, the best algorithm and hyperparameters highly depend on the dataset. We also find that various meta-features of the datasets are predictive of algorithmic performance and runtimes. Motivated by these findings, we created RecZilla, the first metric-independent, hyperparameter-aware algorithm selection approach to recommender systems. Through empirical evaluation, we show that given a user-defined metric, RecZilla effectively predicts high-performing algorithms and hyperparameters for new, unseen datasets, substantially reducing the need for human involvement. We release our code and pretrained RecZilla models, as well as raw experimental results so that users can train new RecZilla models on their own test metrics of interest.

Limitations. While our work progresses prior work along several axes, there are still avenues for improvement. First, the meta-learning problem in RecZilla is low-data. Although we added nearly all common rec-sys research datasets into RecZilla, the result is still only 85 meta-datapoints (datasets). While we guarded against over-fitting to the training data in numerous ways, RecZilla can still be improved by more training data. Therefore, as new recommender system datasets are released in the future, our hope is to add them to our API, so that RecZilla continuously improves over time. Furthermore, the magnitude of our evaluation (78 929 rec-sys models trained) leaves our meta-data susceptible to biases based on experiment success/failures. Therefore, RecZilla may have higher uncertainty for the datasets and algorithms that are more likely to fail.

Broader impact. Our work is "meta-research": there is not one specific application that we target, but our work makes it substantially easier for researchers and practitioners to quickly train recommender system models when given a new dataset. On the research side, this is a net positive because researchers can much more easily include baselines, comparisons, and run experiments on large numbers of datasets, all of which lead to more principled empirical comparisons. On the applied side, our day-to-day lives are becoming more and more influenced by recommendations generated from machine learning models, which comes with pros and cons. These recommendations connect users with needed items that they would have had to spend time searching for [36]. Although these recommendations may lead to harmful effects such as echo chambers [24, 37], techniques to identify and mitigate harms are improving [27, 45].

5 Reproducibility Checklist

1. For all authors…

   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] [The main claims in the abstract and introduction reflect the paper's contributions and scope.]

   (b) Did you describe the limitations of your work? [Yes] [See Section 4.]

   (c) Did you discuss any potential negative societal impacts of your work? [Yes] [See Section 4.]

   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes] [We read the ethics review guidelines and ensured our paper conforms to them.]

2. If you are including theoretical results…

   (a) Did you state the full set of assumptions of all theoretical results? [N/A] [We did not include theoretical results.]

   (b) Did you include complete proofs of all theoretical results? [N/A] [We did not include theoretical results.]

3. If you ran experiments…

   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results, including all requirements (e.g., `requirements.txt` with explicit version), an instructive `README` with installation, and execution commands (either in the supplemental material or as a URL)? [Yes] [We include the code, data, and instructions to reproduce the results here: `https://github.com/naszilla/reczilla`.]

   (b) Did you include the raw results of running the given instructions on the given code and data? [Yes] [We include our raw results; see `https://github.com/naszilla/reczilla`.]

   (c) Did you include scripts and commands that can be used to generate the figures and tables in your paper based on the raw results of the code, data, and instructions given? [Yes] [We include scripts to generate our exact results. See the `scripts` folder in `https://github.com/naszilla/reczilla`.]

   (d) Did you ensure sufficient code quality such that your code can be safely executed and the code is properly documented? [Yes] [We included multipe documentation files, and put in a reasonable effort to make our code as easy to use as possible.]

   (e) Did you specify all the training details (e.g., data splits, pre-processing, search spaces, fixed hyperparameter settings, and how they were chosen)? [Yes] [See Sections 2 and 3.]

   (f) Did you ensure that you compared different methods (including your own) exactly on the same benchmarks, including the same datasets, search space, code for training and hyperparameters for that code? [Yes] [Yes, see Section 3.]

   (g) Did you run ablation studies to assess the impact of different components of your approach? [Yes] [We gave an ablation study in Section B.]

   (h) Did you use the same evaluation protocol for the methods being compared? [Yes] [In our ablation, the same evaluation protocol was used.]

   (i) Did you compare performance over time? [Yes] [See Section 3.]

   (j) Did you perform multiple runs of your experiments and report random seeds? [Yes] [We ran 50 trials of our experiments in Section 3.]

(k) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] [All of our experiments have error bars.]

(l) Did you use tabular or surrogate benchmarks for in-depth evaluations? [N/A] [There do not exist tabular or surrogate benchmarks for recommender systems.]

(m) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] [We include this information in Section 3].

(n) Did you report how you tuned hyperparameters, and what time and resources this required (if they were not automatically tuned by your AutoML method, e.g. in a NAS approach; and also hyperparameters of your own method)? [Yes] [We explained hyperparameter tuning in Section 3.]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets…

(a) If your work uses existing assets, did you cite the creators? [Yes] [See Section A.]

(b) Did you mention the license of the assets? [N/A] Our experiments were conducted only on publicly available datasets.

(c) Did you include any new assets either in the supplemental material or as a URL? [N/A] We did not include new assets.

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] Our experiments were conducted only on publicly available datasets.

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] Our experiments were conducted only on publicly available datasets.

5. If you used crowdsourcing or conducted research with human subjects…

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A] [We did not conduct research with human subjects.]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A] [We did not conduct research with human subjects.]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A] [We did not conduct research with human subjects.]

Acknowledgments

Dickerson was supported in part by NSF CAREER Award IIS-1846237, NIST MSE Award #20126334, DARPA GARD #HR00112020007, and DoD WHS Award #HQ003420F0035.

References

[1] David Adedayo Adeniyi, Zhaoqiang Wei, and Y Yongquan. Automated web usage data mining and recommendation system using k-nearest neighbor (knn) classification method. *Applied Computing and Informatics*, 12(1):90–108, 2016.

[2] Gediminas Adomavicius and YoungOk Kwon. Improving aggregate recommendation diversity using ranking-based techniques. *IEEE Transactions on Knowledge and Data Engineering*, 24(5):896–911, 2011.

[3] Gediminas Adomavicius and Jingjing Zhang. Impact of data characteristics on recommender systems performance. *ACM Transactions on Management Information Systems (TMIS)*, 3(1):1–17, 2012.

[4] Charu C Aggarwal et al. *Recommender systems*, volume 1. Springer, 2016.

[5] 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, page 273–280, New York, NY, USA, 2013. Association for Computing Machinery.

[6] Rohan Anand and Joeran Beel. *Auto-Surprise: An Automated Recommender-System (AutoRecSys) Library with Tree of Parzens Estimator (TPE) Optimization*, page 585–587. Association for Computing Machinery, New York, NY, USA, 2020.

[7] Krisztian Balog, Filip Radlinski, and Shushan Arakelyan. Transparent, scrutable and explainable user models for personalized recommendation. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR'19, page 265–274, New York, NY, USA, 2019. Association for Computing Machinery.

[8] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez. Recommender systems survey. *Knowledge-Based Systems*, 46:109–132, 2013.

[9] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[10] 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*, pages 811–816, 2014.

[11] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, pages 191–198, 2016.

[12] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 39–46, 2010.

[13] Tiago Cunha, Carlos Soares, and André CPLF de Carvalho. Selecting collaborative filtering algorithms using metalearning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 393–409. Springer, 2016.

[14] Tiago Cunha, Carlos Soares, and André CPLF de Carvalho. Recommending collaborative filtering algorithms using subsampling landmarkers. In *International Conference on Discovery Science*, pages 189–203. Springer, 2017.

[15] Tiago Cunha, Carlos Soares, and André CPLF de Carvalho. Cf4cf-meta: Hybrid collaborative filtering algorithm selection framework. In *International Conference on Discovery Science*, pages 114–128. Springer, 2018.

[16] Tiago Cunha, Carlos Soares, and André CPLF de Carvalho. Cf4cf: recommending collaborative filtering algorithms using collaborative filtering. In *Proceedings of the 12th ACM Conference on Recommender Systems*, pages 357–361, 2018.

[17] Tiago Cunha, Carlos Soares, and André CPLF de Carvalho. Metalearning and recommender systems: A literature review and empirical study on the algorithm selection problem for collaborative filtering. *Information Sciences*, 423:128–144, 2018.

[18] 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 (TOIS)*, 39(2):1–49, 2021.

[19] Maurizio Ferrari Dacrema, Paolo Cremonesi, and Dietmar Jannach. Are we really making much progress? A worrying analysis of recent neural recommendation approaches. In Toine Bogers, Alan Said, Peter Brusilovsky, and Domonkos Tikk, editors, *Proceedings of the 13th ACM Conference on Recommender Systems, RecSys 2019, Copenhagen, Denmark, September 16-20, 2019*, pages 101–109. ACM, 2019.

[20] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[21] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

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

[23] Michael Ekstrand and John Riedl. When recommenders fail: predicting recommender failure for algorithm selection and combination. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 233–236, 2012.

[24] Yingqiang Ge, Shuya Zhao, Honglu Zhou, Changhua Pei, Fei Sun, Wenwu Ou, and Yongfeng Zhang. Understanding echo chambers in e-commerce recommender systems. In *Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval*, pages 2261–2270, 2020.

[25] Thomas George and Srujana Merugu. A scalable collaborative filtering framework based on co-clustering. In *Fifth IEEE International Conference on Data Mining (ICDM'05)*, pages 4–pp. IEEE, 2005.

[26] Carlos A Gomez-Uribe and Neil Hunt. The netflix recommender system: Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems (TMIS)*, 6(4):1–19, 2015.

[27] Pietro Gravino, Bernardo Monechi, and Vittorio Loreto. Towards novelty-driven recommender systems. *Comptes Rendus Physique*, 20(4):371–379, 2019.

[28] Josephine Griffith, Colm O'Riordan, and Humphrey Sorensen. Investigations into user rating information and predictive accuracy in a collaborative filtering domain. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, pages 937–942, 2012.

[29] Garima Gupta and Rahul Katarya. Enpso: An automl technique for generating ensemble recommender system. *Arabian Journal for Science and Engineering*, 46(9):8677–8695, 2021.

[30] Srijan Gupta. Auto-caserec: A novel automated recommender system framework. 2020.

[31] Malay Haldar, Mustafa Abdool, Prashant Ramanathan, Tao Xu, Shulin Yang, Huizhong Duan, Qing Zhang, Nick Barrow-Williams, Bradley C Turnbull, Brendan M Collins, et al. Applying deep learning to airbnb search. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1927–1935, 2019.

[32] Kaiming He, Ross Girshick, and Piotr Dollár. Rethinking imagenet pre-training. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4918–4927, 2019.

[33] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE international conference on data mining*, pages 263–272. Ieee, 2008.

[34] Zan Huang and Daniel Dajun Zeng. Why does collaborative filtering work? transaction-based recommendation model validation and selection by analyzing bipartite random graphs. *INFORMS Journal on Computing*, 23(1):138–152, 2011.

[35] Nicolas Hug. Surprise: A python library for recommender systems. *Journal of Open Source Software*, 5(52):2174, 2020.

[36] Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich. *Recommender systems: an introduction.* Cambridge University Press, 2010.

[37] Ray Jiang, Silvia Chiappa, Tor Lattimore, András György, and Pushmeet Kohli. Degenerate feedback loops in recommender systems. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, pages 383–390, 2019.

[38] Yehuda Koren. Factorization meets the neighborhood: A multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, page 426–434, New York, NY, USA, 2008. Association for Computing Machinery.

[39] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[40] Pigi Kouki, Ilias Fountalis, Nikolaos Vasiloglou, Xiquan Cui, Edo Liberty, and Khalifeh Al Jadda. From the lab to production: A case study of session-based recommendations in the home-improvement domain. In *Fourteenth ACM conference on recommender systems*, pages 140–149, 2020.

[41] Daniel Lemire and Anna Maclachlan. Slope one predictors for online rating-based collaborative filtering. In *Proceedings of the 2005 SIAM International Conference on Data Mining*, pages 471–475. SIAM, 2005.

[42] Mark Levy and Kris Jack. Efficient top-n recommendation by linear regression. 2013.

[43] Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. *Advances in neural information processing systems*, 27, 2014.

[44] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

[45] Virginia Morini, Laura Pollacci, and Giulio Rossetti. Toward a standard approach for echo chamber detection: Reddit case study. *Applied Sciences*, 11(12):5390, 2021.

[46] Allan H. Murphy. The finley affair: A signal event in the history of forecast verification. *Weather and Forecasting*, 11(1):3 – 20, 1996.

[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):1–34, 2016.

[48] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618*, 2012.

[49] 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, page 175–186, New York, NY, USA, 1994. Association for Computing Machinery.

[50] Tal Ridnik, Emanuel Ben-Baruch, Asaf Noy, and Lihi Zelnik-Manor. Imagenet-21k pretraining for the masses. *arXiv preprint arXiv:2104.10972*, 2021.

[51] 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, page 285–295, New York, NY, USA, 2001. Association for Computing Machinery.

[52] Brent Smith and Greg Linden. Two decades of recommender systems at amazon. com. *Ieee internet computing*, 21(3):12–18, 2017.

[53] Harald Steck. Embarrassingly shallow autoencoders for sparse data. In *The World Wide Web Conference*, pages 3251–3257, 2019.

[54] Amos Tversky. Features of similarity. *Psychological review*, 84(4):327, 1977.

[55] Fan Wang, Xiaomin Fang, Lihang Liu, Yaxue Chen, Jiucheng Tao, Zhiming Peng, Cihang Jin, and Hao Tian. Sequential evaluation and generation framework for combinatorial recommender system. *arXiv preprint arXiv:1902.00245*, 2019.

[56] Ting-Hsiang Wang, Xia Hu, Haifeng Jin, Qingquan Song, Xiaotian Han, and Zirui Liu. *AutoRec: An Automated Recommender System*, page 582–584. Association for Computing Machinery, New York, NY, USA, 2020.

[57] Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. SATzilla: portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 32:565–606, June 2008.

[58] Tao Zhou, Zoltán Kuscsik, Jian-Guo Liu, Matúš Medo, Joseph Rushton Wakeling, and Yi-Cheng Zhang. Solving the apparent diversity-accuracy dilemma of recommender systems. *Proceedings of the National Academy of Sciences*, 107(10):4511–4515, 2010.