On the Generalizability and Predictability of Recommender Systems

Anonymous Author(s)
Affiliation
Address
email

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 is also a strong correlation 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://anonymous.4open.science/r/anon-reczilla-51FC.

1 Introduction

Due to the large computational resources for training machine learning models, researchers have found many ways to repurpose existing computation. For example, it is common to start with a pretrained ImageNet classification model for computer vision tasks [30, 72, 48], or a pretrained BERT model for natural language tasks [31, 63]. These approaches work well because the core problems are largely homogeneous; for example, any computer vision model at its core must be able to distinguish edges, colors, and shapes. Even a task-specific architecture can be found automatically through neural architecture search [35], since the building blocks such as convolutional layers stay the same.

On the other hand, 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 [19, 39, 74], there is far less work in reusing models or automating the process of selecting models. Many rec-sys techniques are designed and optimized with just a *single* dataset in mind [39, 46, 19, 56, 78]. 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 [19, 56, 46]. 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

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

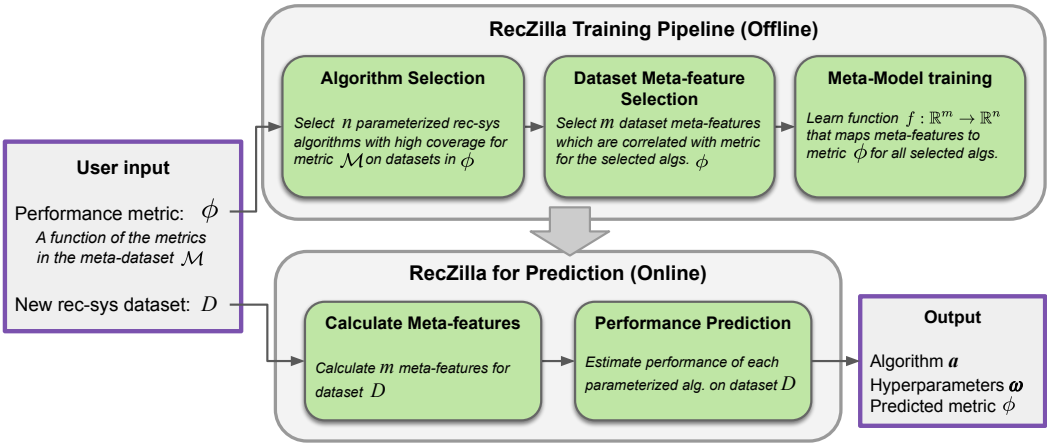

Figure 1: RecZilla recommends a parameterized rec-sys algorithm for a user-provided dataset and performance metric. The RecZilla pipeline is built using a meta-dataset $\mathcal{M}$ that includes many different performance metrics evaluated on many different rec-sys algorihtms on many different datasets; we estimate algorithm performance using dataset meta-features. To avoid over-fitting and reduce runtime, users can limit the number of algorithms and meta-features to constants n and m.

matrix factorization [26]. 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 set of algorithms which perform well changes substantially across dataset and across performance metrics. Furthermore, the best hyperparameters of a rec-sys algorithm on one dataset often perform significantly worse than the best hyperparameters on a different dataset. Although we show that there are no universal algorithms that work well on most datasets, we *do* show that various meta-features of the dataset can be used to *predict* the performance of rec-sys algorithms. In fact, the same meta-features are also predictive of the runtime of rec-sys algorithms as well as the "dataset hardness" – how challenging it is to find a high-performing model on a particular dataset.

Motivated by these findings, we introduce RecZilla, a meta-learning-based algorithm selection approach (see Figure 1) inspired by SATzilla [80]. 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 [24, 25], 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, a meta-learning-based 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 [10]. Common approaches include k-nearest neighbors [1], matrix factorization [55, 61], and deep learning approaches [19, 39, 74]. For a survey on recommender systems, see [10, 4]. A recent meta-study showed that of the 12 published 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) [26].

Algorithm selection for recommender systems was first studied in 2011 [50] 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 [34, 3, 41]. Subsequent work focused on improving the model and framework [25] including studying 74 meta-features systematically [21]. 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 [22, 24, 23]. 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 [79, 6, 44, 45]. Finally, we note that meta-learning across rec-sys datasets is not to be confused with the body of work on meta-learning user preferences *within a single* rec-sys dataset [15, 16, 17]. To the best of our knowledge, no meta-learning or AutoML rec-sys paper has 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 large, 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?

Algorithms, datasets, and metrics implemented. We run experiments with 18 rec-sys algorithms, including methods from recent literature and common baselines. This includes five similarity and clustering-based methods: User-KNN [71], Item-KNN [73], P3Alpha [18], RP3Beta [69], and Co-Clustering [37]; six Matrix-Factorization (MF) methods: MF-FunkSVD, MF-AsySVD [54], MF-BPR [70], IALS [49], Pure-SVD, Non-negative matrix factorization (NMF) [20]; five methods based on linear models: Global-Effects, SLIM-BPR [7], SLIM-Elastic-Net [60], EASE-R [75], and SlopeOne [59]; and two simple baselines: Random and Top-Pop. These algorithms were chosen due to their high performance, popularity, and speed. For many algorithms, we used the implementations from the codebase of Dacrema et al. [26]. For full details of the algorithms, see Appendix A.

We run the algorithms on 85 datasets from 19 dataset "families": Amazon [68], Anime[1], BookCrossing [84], CiaoDVD [57, 43], Dating (Libimseti.cz) [57, 58], Epinions [65, 64], FilmTrust [42], Frappe [8], Gowalla [14], Jester2 [38], LastFM [11], MarketBias-Electronics and MarketBias-ModCloth [77], MovieTweetings [33], Movielens [47], NetflixPrize [9], Here, a "dataset 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 implemented the majority of rec-sys datasets commonly used for research, and to the best of our knowledge, this is the largest number of rec-sys datasets accessible in a single open-source repository.

We use 23 different "base" metrics: ARHR, Average Popularity, Diversity Similarity, F1 Score, Gini Index, Herfindahl Index, Hit Rate, Item Coverage, Item-hit Coverage, Precision (PREC), Precision-Recall Min Denominator, Recall, MAP, MAP Min Denominator, Mean Inter-List Diversity, MRR, NDCG, Novelty, Shannon Entropy, User Coverage, User-hit Coverage. All of these metrics are computed at cutoffs $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 30, 40, 50\}$, for a total of 315 different metrics.

[1] https://www.kaggle.com/datasets/CooperUnion/anime-recommendations-database

Table 1: The relative performance of each rec-sys algorithm depends on the dataset and metric. This table shows the mean, min (best) and max (worst) rank achieved by all algorithms over all 85 datasets, over 10 accuracy and hit-rate metrics at all cutoffs tested. This includes metrics NDCG, precision, recall, Prec.-Rec.-Min-density, hit-rate, F1, MAP, MAP-Min-density, ARHR, and MRR.

Rank	Item-KNN	SLIM-BPR	SVD	P3alpha	iALS	User-KNN	NMF	RP3beta	EASE-R	MF-FunkSVD	SLIM-ElasticNet	MF-AsySVD	MF-BPR	TopPop	CoClustering	GlobalEffects	SlopeOne	Random
mean	3.4	5.0	5.7	5.7	6.0	6.2	6.4	6.7	6.7	7.5	7.8	8.2	8.6	9.9	11.8	12.1	13.2	14.2
min	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2
max	17	14	16	17	17	16	15	16	17	16	16	15	16	18	17	18	18	18

See Appendix A.6 for more details. These metrics include the most popular from the literature, and we use the Dacrema et al. [26] implementations. We focus mainly on metrics related to rating- and ranking-based metrics, such as NDCG and Precision.

Experimental design. 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). Given each of the above algorithms, 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. We refer to each combination of (algorithm, hyperparameter set, dataset) as an *experiment*.

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. We give a more detailed look at the breakdown of experiments in Appendix A, and we discuss any potential biases in the resulting dataset in Section 4.

2.1 On the generalizability of rec-sys algorithms

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?

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. For example, for metric PREC@10, 14 out or 18 algorithms are ranked first on at least one dataset; the exceptions are CoClustering, SlopeOne, GlobalEffects, and Random; the same is true for many other metrics and cutoffs (see Table 5 in Appendix C).

Average performance is more varied: some algorithms tend to perform better than others. Table 1 shows the mean, min (best) and max (worst) ranking of all 18 algorithms over all dataset and all accuracy and hit-rate metrics. This includes metrics NDCG, precision, recall, Prec.-Rec.-Min-density, hit-rate, F1, MAP, MAP-Min-density, ARHR, and MRR (see Appendix A.6 for descriptions of these 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.

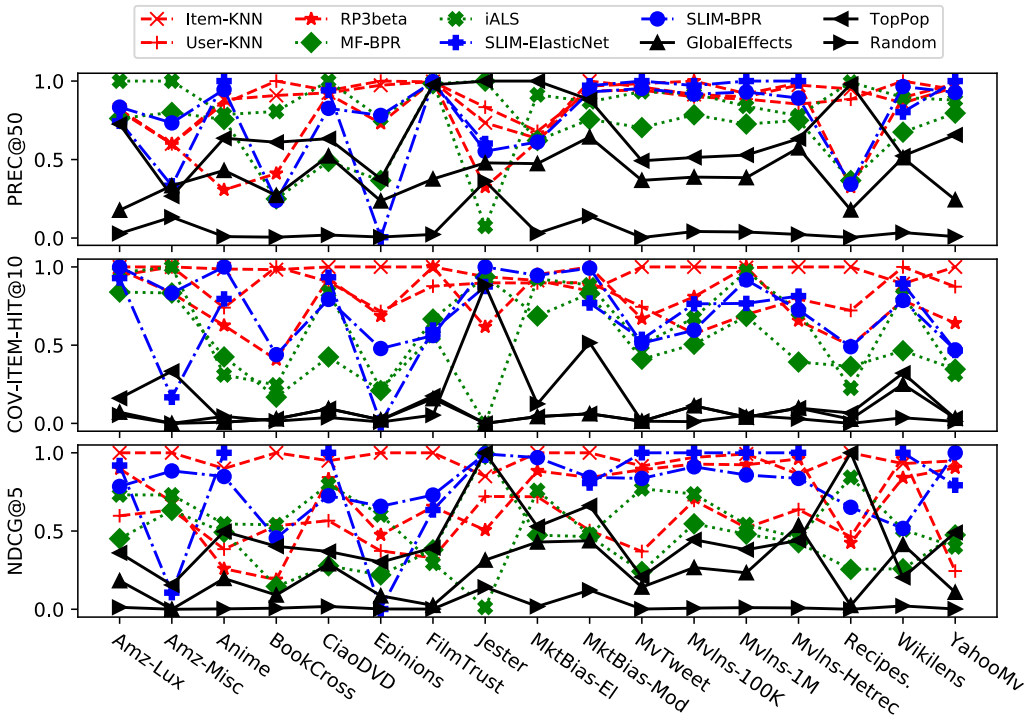

Figure 2: Relative algorithm performance depends on both the dataset and metric: no algorithm dominates across all dataests or metrics. Each plot shows a different metric, normalized to $[0, 1]$ for each dataset; the horizontal axis shows different dataset, ordered alphabetically. Each series corresponds to a different algorithm: similarity-based methods are red, matrix factorization methods are green, and baseline methods are black.

To illustrate the changes in algorithm performance across datasets, Figure 2 shows the normalized metric values for eight algorithms across 17 dataset splits. Some algorithms tend to perform well (Item-KNN and SLIM-BPR) and others poorly (Random, TopPop), but no algorithm clearly dominates for all metrics and datasets. This is a primary motivation for our meta-learning pipeline descirbed in Section 3: different algorithms perform well for different datasets on different metrics, so it is important to identify appropriate algorithms for each setting.

2.2 On the predictability of rec-sys algorithms

Can attributes of the rec-sys dataset be used to predict the performance of rec-sys algorithms?

Dataset meta-features. 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. [21], and landmark features [22]—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, they are calculated using only the training split of each dataset. For more details on the dataset meta-features, see Appendix A.4.

Algorithm performance prediction. Table 2 shows the meta-features that are most highly-correlated with the performance (PREC@10) of each algorithm, using their default parameters. Several meta-features aare highly-correlated with algorithm performance; one of the simplest metrics—the mean of the item rating count distribution—is highly correlated with performance of two rec-sys algorithms. 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.

Table 2: Highest absolute correlations computed across 85 datasets and weighed inversely proportional to dataset family frequency, over all pairs of algorithm families and meta-features, for the PREC@10 performance metric and the default algorithm hyperparameters.

Abs. Correlation	Algorithm Family	Meta-feature
0.941	SlopeOne	Mean of item rating count distribution
0.933	CoClustering	Median of item rating count distribution
0.887	MF-BPR	Sparsity of rating matrix
0.855	RP3beta	Mean of item rating count distribution
0.846	UserKNN	Landmarker, Pure SVD, mAP@5

As a toy-model version of RecZilla, 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 (see Appendix A for details). We use leave-one-out leave-one-out evaluation for each meta-learner: one dataset family is held out for testing, while m are used for training. Figure 3 shows the mean absolute error (MAE) of each meta-learner; these results are aggregated over 200 random samples of randomly-selected training dataset families. MAE decreases as more dataset families are added, suggesting that it is possible to estimate rec-sys algorithm performance using dataset meta-features.

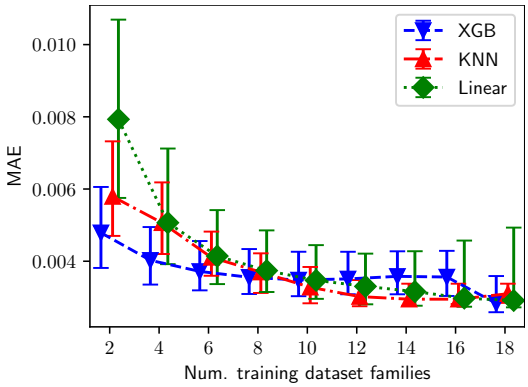

Figure 3: Three basic meta-learners (KNN, linear regression, and XGB) are trained randomly-selected dataset families to predict performance metric PREC@10. As more dataset families are added, the meta-learners are better able to predict rec-sys algorithm performance, suggesting that our dataset meta-features are useful for predicting rec-sys algorithm performance. Vertical axis shows mean absolute error (MAE), over all folds of leave-one-out validation, and 200 random trials; in each trial a different set of training datasets are chosen. Error bars show the 40th and 60th percentile.

We also find that performance metrics are not the only values that can be predicted with dataset meta-features. In particular, we find that the *runtime* of rec-sys algorithms is also highly correlated with different meta-features. Furthermore, we compute a simple measure of *dataset hardness*, which we compute as, given a performance metric, the maximum value achieved for that dataset across all algorithms. For example, if all 18 algorithms do not perform well on the MovieTweetings dataset, then we can expect that the MovieTweetings dataset is "hard". Once again, we find that certain dataset meta-features are highly correlated with dataset hardness. For more details on meta-feature correlation with algorithm runtimes and dataset hardness, see Appendix A.

The fact that dataset meta-features are correlated with algorithm performance, algorithm runtimes, and dataset hardness is a strong positive signal that meta-learning is worthwhile and useful in the context of recommender systems. We explore this direction further in the next section.

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.

Preliminaries. We start with notation for the rec-sys problem. Let D denote a rec-sys dataset, consisting of a set of user-item interactions, split into a training and validation set. Let a denote a rec-sys algorithm parameterized by a set of $k(a)$ hyperparameters $\boldsymbol{\omega} \in H(a) \subseteq \mathbb{R}^{k(a)}$, which is algorithm-dependent. Suppose we train algorithm a on the training split of dataset D, using hyperparameters $\boldsymbol{\omega}$; we denote the *performance* of algorithm a with hyperparameters $\boldsymbol{\omega}$ on dataset D as $\phi(a, \boldsymbol{\omega}, D) \in \mathbb{R}$. The function $\phi(\cdot, \cdot, \cdot)$ represents a *performance metric* for the recommender system that is selected by the user; throughout this paper we refer to this function and its numerical value simply as *performance*. In this paper, larger values of $\phi(\cdot, \cdot, \cdot)$ indicates better performance. We report the *normalized* performance defined as

$$\overline{\phi}(a, \boldsymbol{\omega}, D) \equiv 100 \times \frac{\phi(a, \boldsymbol{\omega}, D) - P_D^{min}}{P_D^{max} - P_D^{min}},$$

where P_D^{max} (resp. P_D^{min}) are the max (resp. min) performance of any algorithm on dataset D.

Next we define notation for the meta-learning problem. Given a fixed performance function ϕ, let $\mathcal{M} = \{(D_i, a_i, \boldsymbol{\omega}_i, y_i)\}_{i=1}^{M}$ denote a *meta-dataset* consisting of M tuples. Each tuple consists of a dataset D, algorithm a parameterized by $\boldsymbol{\omega}$, and performance y, where $y_i \equiv \phi(a_i, \boldsymbol{\omega}_i, D_i)$. Since many algorithms have a wide range of hyperparameters, we refer to the set of *parameterized algorithms* in the meta-dataset as $\mathcal{S} = \{(a_i, \boldsymbol{\omega}_i)\}_{i=1}^{N}$. We represent each dataset using vector $\boldsymbol{d} \in \mathbb{R}^m$, where each element of $\boldsymbol{d}$ is a meta-feature of the dataset. In this setting, the meta-learning task is to identify a function that estimates the performance of parameterized algorithm $(a, \boldsymbol{\omega})$ on dataset D.

3.1 The RecZilla Algorithm Selection Pipeline

The RecZilla algorithm selection pipeline takes as input a meta-dataset $\mathcal{M}$ built with a user-chosen performance function for the recommendation task $\phi(\cdot, \cdot, \cdot)$. RecZilla can accommodate any performance function that is a computable function of the performance metrics present in the meta-dataset. The RecZilla pipeline proposed here returns both an algorithm $a \in \mathcal{A}$ *and* a set of hyperparameters $\boldsymbol{\omega} \in H(a)$, so that RecZilla can be used without additional hyperparameter optimization.

Since the meta-learning problem is in the low-data regime, to guard against overfitting, we select a subset of the algorithms that have good coverage over the training dataset. We similarly select only the meta-features which are most predictive of the user-selected performance metric for the selected algorithms. Below we outline each of the steps used to build the proposed RecZilla pipeline.

1. **Algorithm subset selection:** We select a subset $\mathcal{S}' \subseteq \mathcal{S}$ of n parameterized algorithms that collectively perform well on all datasets in the meta-training dataset $\mathcal{M}$, according to performance function ϕ. We aim to select an algorithm subset with high *coverage* over the set of known datasets, where coverage of a subset $\mathcal{S}'$ is defined as

$$C(\mathcal{S}') = \frac{1}{|\mathcal{D}|} \sum_{D \in \mathcal{D}} \max_{(a, \boldsymbol{\omega}) \in \mathcal{S}'} \overline{\phi}(a, \boldsymbol{\omega}, D). \tag{1}$$

   In other words, the coverage of subset $\mathcal{S}'$ is the normalized performance metric of the *best* performing algorithm in $\mathcal{S}'$, averaged over all datasets $\mathcal{D}$. Selecting a subset with maximum coverage is itself a difficult problem; we use a greedy heuristic as follows. We begin with $\mathcal{S}' = \{\}$ and iteratively add the parameterized algorithm $(a^*, \boldsymbol{\omega}^*) \in \arg\max_{(a, \boldsymbol{\omega}) \in \mathcal{S}} C(\mathcal{S}' \cup \{(a, \boldsymbol{\omega})\})$ to $\mathcal{S}'$ until $|\mathcal{S}'| = n$. That is, we greedily ensure that the coverage is maximized at each step.

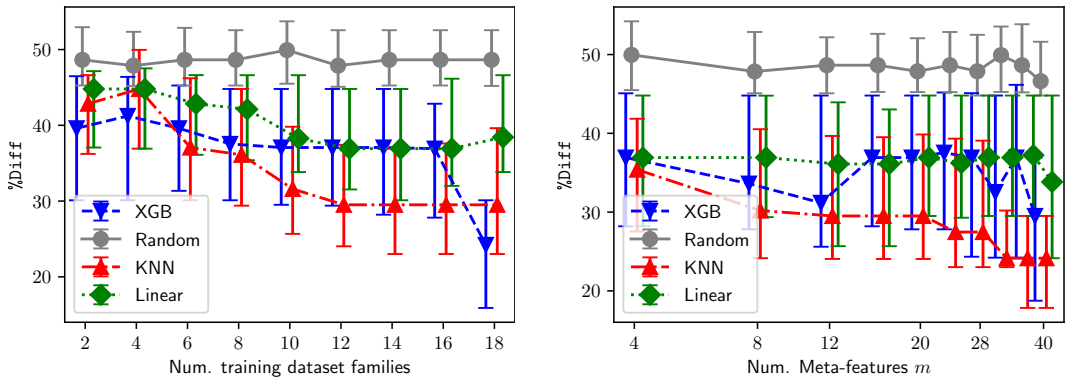

Figure 4: Performance of the RecZilla pipeline improves as we add more training meta-datapoints, and more meta-features m. Subsets of training meta-datapoints and meta-features are selected randomly, over 50 random trials. Points show the median %`Diff`, and error bars show the 40th and 60th percentile over all folds and random trials. (Left) $m = 10$ meta-features, while the number of training dataset families varies. (Right) All training data is used, while m varies.

2. **Meta-feature selection:** Similar to the previous point, we select a subset of meta-features with good coverage over the meta-training dataset $\mathcal{M}$, where here coverage is defined in terms of the correlation between algorithm performance and each meta-feature (see Appendix A.4 for details). Let $M : \mathcal{D} \rightarrow \mathbb{R}^m$ denote the resulting function that maps a dataset to a vector of meta-features.
3. **Meta-learning:** We learn a function $f : \mathbb{R}^m \rightarrow \mathbb{R}^n$, where $f(\boldsymbol{d}) = \hat{\boldsymbol{y}}$ is a vector of the estimated performance metric of all parameterized algorithms in $\mathcal{S}$ on dataset meta-features $\boldsymbol{d}$.

Note that the RecZilla pipeline has two hyperparameters: n, the number of parameterized algorithms in $\mathcal{S}'$; and m, the number of dataset meta-features used by the meta-learner. In our experiments, we run an ablation study with both n and m, as well as different functions f for the meta-learning model.

Using RecZilla for Algorithm Selection. After developing the RecZilla pipeline, we use the following steps to select an algorithm for a new dataset D':

1. Calculate m meta-features of the dataset $\boldsymbol{d}' \leftarrow M(D')$.
2. Estimate the performance of all parameterized algorithms: $\boldsymbol{y}' \leftarrow f(\boldsymbol{d}')$.
3. Return the parameterized algorithm in $\mathcal{S}'$ with the best estimated performance.

3.2 Experiments

In this section, we evaluate the end-to-end RecZilla pipeline. We start by describing the specific versions of RecZilla used in our experiments. We use four different meta-learning functions within RecZilla: XGBoost [12], linear regression, k-nearest neighbors, and uniform random. For KNN, we set $k = 5$ and use the L_2 distance from the selected meta-features.

Experimental setup. Focusing only on performance metric to PREC@10, we build a meta-dataset $\mathcal{M}$ using all rec-sys datasets, algorithms, and meta-features described in Section 2. We first use the algorithm selection and meta-feature selection procedures described above to select $n = m = 100$ parameterized algorithms and meta-features for PREC@10. For all experiments we use the best 10 parameterized algorithms selected during this process. We vary both the number of training meta-datapoints and meta-features used by RecZilla; the datapoints and features are randomly selected over 50 random trials. All RecZilla 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 $(a', \boldsymbol{\omega}')$ to the performance metric of the ground-truth best algorithm y^*, using the percentage-difference-from-best: $\%\texttt{Diff} = 100 \times (y^* - \phi(D, a', \boldsymbol{\omega}'))/y^*$. %`Diff` is between 0 and 100, and smaller values indicate better performance.

Results and discussion. Figure 4 (Left) shows `%Diff` vs. the size of the meta-training set, and Figure 4 (Right) shows the results of an ablation study on the number of selected meta-features m; all results are aggregated over all leave-one-out folds and 50 random trials. Generally, XGBoost and KNN outperform the linear model and the random baseline, with XGBoost achieving top performance when using 10 meta-features and the maximum number of training datasets. Furthermore, the number of datasets in the meta-training set matters more than the meta-learning model itself. For example, the improvement of XGBoost from 4 to 10 and to 18 training datasets is larger than the difference in performance between XGBoost and KNN at 4 and 10 training datasets, respectively. Finally, we find that the optimal number of meta-features for XGBoost and KNN peaks between 10 and 40.

Pre-trained RecZilla models. We release pre-trained RecZilla models for PREC@10, NDCG@10, and Item-hit Coverage@10, trained with XGBoost on all 18 datasets, with algorithms $n = 10$ and meta-features $m = 10$. We also include a RecZilla model that predicts the Pareto-front of PREC@10 and model training time, so that users can select their desired trade-off between performance and runtime. Finally, we include a pipeline so that users can choose a metric from the list of 315 (or any desired combination of the 315 metrics) and train the resulting RecZilla model.

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 not only release our code and pretrained RecZilla models, but we also release the raw experimental results so that users can train new RecZilla models on their own test metrics of interest. This codebase, which includes a unified API for 85 datasets and 18 algorithms, may be of independent 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. Another possible solution to increase the meta-training dataset is to create subsets of existing rec-sys datasets. Another limitation is that RecZilla does not directly predict the performance of hyperparameters for algorithms on a given dataset. Although care must be taken to not overfit, modifying RecZilla to predict the performance of an algorithm together with a set of hyperparameters is an interesting avenue for future work. Finally, the magnitude of our evaluation (78 929 rec-sys models trained) leaves our meta-data susceptible to biases based on experiment success/failures. While we fixed many common errors such as out-of-memory errors, it was infeasible to give each experiment specific attention. Therefore, RecZilla may have higher uncertainty for the datasets and algorithms that are more likely to fail. An interesting future improvement to RecZilla would be to predict the likelihood that a new dataset will successfully train on a new dataset.

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 [52]. Although these recommendations may lead to harmful effects such as echo chambers [36, 53], techniques to identify and mitigate harms are improving [40, 66].

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] Linas Baltrunas, Karen Church, Alexandros Karatzoglou, and Nuria Oliver. Frappe: Understanding the usage and perception of mobile app recommendations in-the-wild.

[9] James Bennett, Stan Lanning, et al. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35. Citeseer, 2007.

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

[11] Iván Cantador, Peter Brusilovsky, and Tsvi Kuflik. 2nd workshop on information heterogeneity and fusion in recommender systems (hetrec 2011). In *Proceedings of the 5th ACM conference on Recommender systems*, RecSys 2011, New York, NY, USA, 2011. ACM.

[12] 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.

[13] Weiyu Cheng, Yanyan Shen, Yanmin Zhu, and Linpeng Huang. Delf: A dual-embedding based deep latent factor model for recommendation. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 3329–3335. International Joint Conferences on Artificial Intelligence Organization, 7 2018.

[14] Eunjoon Cho, Seth A. Myers, and Jure Leskovec. Friendship and mobility: User movement in location-based social networks. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, page 1082–1090, New York, NY, USA, 2011. Association for Computing Machinery.

[15] Andrew Collins and Joeran Beel. Meta-learned per-instance algorithm selection in scholarly recommender systems. *arXiv preprint arXiv:1912.08694*, 2019.

[16] Andrew Collins, Dominika Tkaczyk, and Joeran Beel. A novel approach to recommendation algorithm selection using meta-learning. In *AICS*, pages 210–219, 2018.

[17] Andrew Collins, Dominika Tkaczyk, and Joeran Beel. One-at-a-time: A meta-learning recommender-system for recommendation-algorithm selection on micro level. *arXiv preprint arXiv:1805.12118*, 2018.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] Maurizio Ferrari Dacrema, Paolo Cremonesi, and Dietmar Jannach. Methodological issues in recommender systems research (extended abstract). In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 4706–4710. ijcai.org, 2020.

[29] Maurizio Ferrari Dacrema, Federico Parroni, Paolo Cremonesi, and Dietmar Jannach. Critically examining the claimed value of convolutions over user-item embedding maps for recommender systems. In Mathieu d'Aquin, Stefan Dietze, Claudia Hauff, Edward Curry, and Philippe Cudré-Mauroux, editors, *CIKM '20: The 29th ACM International Conference on Information and Knowledge Management, Virtual Event, Ireland, October 19-23, 2020*, pages 355–363. ACM, 2020.

[30] 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.

[31] 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.

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

[33] Simon Dooms, Toon De Pessemier, and Luc Martens. Movietweetings: a movie rating dataset collected from twitter. In *Workshop on Crowdsourcing and Human Computation for Recommender Systems, CrowdRec at RecSys 2013*, 2013.

[34] 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.

[35] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. In *JMLR*, 2019.

[36] 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.

[37] 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.

[38] Ken Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval*, 4(2):133–151, 2001.

[39] 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.

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

[41] 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.

[42] G. Guo, J. Zhang, and N. Yorke-Smith. A novel bayesian similarity measure for recommender systems. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2619–2625, 2013.

[43] Guibing Guo, Jie Zhang, Daniel Thalmann, and Neil Yorke-Smith. Etaf: An extended trust antecedents framework for trust prediction. In *2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2014)*, pages 540–547, 2014.

[44] 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.

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

[46] 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.

[47] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4), dec 2015.

[48] 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.

[49] 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.

[50] 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.

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

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

[53] 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.

[54] 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.

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

[56] 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.

[57] Jérôme Kunegis. Konect: The koblenz network collection. In *Proceedings of the 22nd International Conference on World Wide Web*, WWW '13 Companion, page 1343–1350, New York, NY, USA, 2013. Association for Computing Machinery.

[58] Jérôme Kunegis, Gerd Gröner, and Thomas Gottron. Online dating recommender systems: The split-complex number approach. In *Proceedings of the 4th ACM RecSys Workshop on Recommender Systems and the Social Web*, RSWeb '12, page 37–44, New York, NY, USA, 2012. Association for Computing Machinery.

[59] 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.

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

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

[62] Dawen Liang, Rahul G. Krishnan, Matthew D. Hoffman, and Tony Jebara. Variational autoencoders for collaborative filtering. In *Proceedings of the 2018 World Wide Web Conference*, WWW '18, page 689–698, Republic and Canton of Geneva, CHE, 2018. International World Wide Web Conferences Steering Committee.

[63] 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.

[64] Paolo Massa and Paolo Avesani. Trust-aware recommender systems. In *Proceedings of the 2007 ACM Conference on Recommender Systems*, RecSys '07, page 17–24, New York, NY, USA, 2007. Association for Computing Machinery.

[65] Paolo Massa, Kasper Souren, Martino Salvetti, and Danilo Tomasoni. Trustlet, open research on trust metrics. *Scalable Computing: Practice and Experience*, 9(4), 2008.

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

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

[68] Jianmo Ni, Jiacheng Li, and Julian McAuley. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 188–197, Hong Kong, China, November 2019. Association for Computational Linguistics.

[69] 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.

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

[71] 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.

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

[73] 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.

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

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

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

[77] Mengting Wan, Jianmo Ni, Rishabh Misra, and Julian McAuley. *Addressing Marketing Bias in Product Recommendations*, page 618–626. Association for Computing Machinery, New York, NY, USA, 2020.

[78] 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.

[79] 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.

[80] 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.

[81] Shuai Zhang, Lina Yao, Aixin Sun, Sen Wang, Guodong Long, and Manqing Dong. Neurec: On nonlinear transformation for personalized ranking. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 3669–3675. International Joint Conferences on Artificial Intelligence Organization, 7 2018.

[82] 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, page 311–319, New York, NY, USA, 2018. Association for Computing Machinery.

[83] 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.

[84] Cai-Nicolas Ziegler, Sean M. McNee, Joseph A. Konstan, and Georg Lausen. Improving recommendation lists through topic diversification. In *Proceedings of the 14th International Conference on World Wide Web*, WWW '05, page 22–32, New York, NY, USA, 2005. Association for Computing Machinery.

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 (either in the supplemental material or as a URL)? [Yes] [We include the code, data, and instructions to reproduce the results here: `https://anonymous.4open.science/r/anon-reczilla-51FC`.]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] [See Section 2 and Appendix A.]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] [See Section 3.]
   (d) 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 2].
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 2 and Appendix 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.]