A Generic Top-N Recommendation Framework For Trading-off Accuracy, Novelty, and Coverage

Zainab Zolaktaf #1, Reza Babanezhad #2, Rachel Pottinger #3

# *Department of Computer Science, University of British Columbia, Vancouver, B.C, Canada*
1 zolaktaf@cs.ubc.ca, 2 rezababa@cs.ubc.ca, 3 rap@cs.ubc.ca

***Abstract*—Standard collaborative filtering approaches for top-N recommendation are biased toward popular items. As a result, they recommend items that users are likely aware of and underrepresent *long-tail* items. This is inadequate, both for consumers who prefer novel items and because concentrating on popular items poorly covers the item space, whereas high item space coverage increases providers' revenue.**

We present an approach that relies on historical rating data to learn user long-tail novelty preferences. We integrate these preferences into a *generic* re-ranking framework that customizes balance between accuracy and coverage. We empirically validate that our proposed framework increases the novelty of recommendations. Furthermore, by promoting long-tail items to the right group of users, we significantly increase the system's coverage while scalably maintaining accuracy. Our framework also enables personalization of existing non-personalized algorithms, making them competitive with existing personalized algorithms in key performance metrics, including accuracy and coverage.

I. Introduction

The goal in top-N recommendation is to recommend to each consumer a small set of N items from a large collection of items [1]. For example, Netflix may want to recommend N appealing movies to each consumer. Collaborative Filtering (CF) [2], [3] is a common top-N recommendation method. CF infers user interests by analyzing partially observed user-item interaction data, such as user ratings on movies or historical purchase logs [4]. The main assumption in CF is that users with similar interaction patterns have similar interests.

Standard CF methods for top-N recommendation focus on making suggestions that accurately reflect the user's preference history. However, as observed in previous work, CF recommendations are generally biased toward popular items, leading to a rich get richer effect [5], [6]. The major reasons for this are *popularity bias* and *sparsity* of CF interaction data (detailed in Section VI). In a nutshell, to maintain accuracy, recommendations are generated from the dense regions of the data, where the popular items lie.

However, accurately suggesting popular items, may not be satisfactory for the consumers. For example, in Netflix, an accuracy-focused movie recommender may recommend "Star Wars: The Force Awakens" to users who have seen "Star Wars: Rogue One". But, those users are probably already aware of "The Force Awakens". Considering additional factors, such as novelty of recommendations, can lead to more effective suggestions [1], [7], [8], [9], [10].

Focusing on popular items also adversely affects the satisfaction of the providers of the items. This is because accuracy-focused models typically achieve a low overall item-space coverage across their recommendations, whereas high item-space coverage helps providers of the items increase their revenue [5], [7], [11], [12], [13], [14].

In contrast to the relatively small number of popular items, there are copious *long-tail* items that have fewer observations (e.g., ratings) available. More precisely, using the Pareto principle (i.e., the $80/20$ rule), long-tail items can be defined as items that generate the lower 20% of observations [13]. Experimentally we found that these items correspond to almost 85% of the items in several datasets (Sections II-A and IV).

As previously shown, one way to improve the novelty of top-N sets is to recommend interesting long-tail items [1], [15]. The intuition is that since they have fewer observations available, they are more likely to be unseen [16]. Moreover, long-tail item promotion also results in higher overall coverage of the item space [5], [7], [8], [10], [11], [12], [13], [17]. Because long-tail promotion reduces accuracy [6], there are trade-offs to be explored.

This work studies three aspects of top-N recommendation: accuracy, novelty, and item-space coverage, and examines their trade-offs. In most previous work, predictions of a base recommendation system are *re-ranked* to handle these trade-offs [14], [17], [18], [19]. The re-ranking models are computationally efficient but suffer from two drawbacks. First, due to performance considerations, parameters that balance the trade-off between novelty and accuracy are not customized per user. Instead they are cross-validated at a global level. This can be detrimental since users have varying preferences for objectives such as long-tail novelty. Second, the re-ranking methods are often limited to a specific base recommender that may be sensitive to dataset density. As a result, the datasets are pruned and the problem is studied in dense settings [14], [20]. But real world scenarios are often sparse [4], [21].

We address these limitations by directly inferring user preference for long-tail novelty from interaction data. This allows us to customize the re-ranking per user, and design a *generic* framework, which resolves the second problem. In particular, since the long-tail novelty preferences are estimated independently of any base recommender model, we can plug-in an appropriate base recommender w.r.t. the dataset sparsity.

Modelling user preference for long-tail novelty using only item popularity statistics, e.g., the average popularity of rated items as in [22], disregards additional information like whether the user found the item interesting and the long-tail preferences of other users of the items. We propose an optimization approach that incorporates this information and directly learns, from interaction data, the users' long-tail novelty preferences.

Next, we integrate these learned preferences in a generic top-N recommendation framework to provide customized balance between accuracy and coverage. Using our framework, we design a novel algorithm, *Ordered Sampling-based Locally Greedy (OSLG)*, that relies on the learned long-tail novelty preferences to scalably correct for popularity bias. Our work does not rely on any additional contextual data, although such data, if available, can help promote newly-added long-tail items [23], [24]. In summary:

- We examine various measures for estimating user long-tail novelty preference in Section II and propose an optimization framework to directly learn users' preferences for long-tail items from interaction data in Section II-C.
- We integrate the user preference estimates into a generic re-ranking framework to provide customized balance between accuracy and coverage (Section III). We introduce OSLG, a scalable algorithm that relies on user long-tail preferences to maintain the coverage and accuracy trade-off (Section III-C).
- We conduct an extensive empirical study and evaluate performance from accuracy, novelty, and coverage perspectives (Section IV). We use five datasets with varying density and difficulty levels. In contrast to most related work, our evaluation considers realistic settings that include a large number of infrequent items and users.
- Our empirical results confirm that the performance of re-ranking models is impacted by the base recommender and the dataset density. Our generic approach enables us to easily incorporate a suitable base recommender to devise an effective solution for both dense and sparse settings. In dense settings, we use the same base recommender as existing re-ranking approaches, and we outperform them in accuracy and coverage metrics. For sparse settings, we plug-in a more suitable base recommender, and devise an effective solution that is competitive with existing top-N recommendation models in accuracy and novelty.

Section VI describes related work. Section VII concludes.

II. Long-tail novelty preference

We begin this section by introducing our notation. We then describe various models for measuring user long-tail novelty preference (Section II-B).

A. Notation and data model

Let $\mathcal{U}$ denote the set of all consumers or users and $\mathcal{I}$ the set of all items. We reserve u, v for indexing users, and i, j for indexing items. Our dataset $\mathcal{D}$, is a set of ratings of users on various items, i.e., $\mathcal{D} = \{r_{ui} : u \in \mathcal{U}, i \in \mathcal{I}\}$. Since every user rates only a small subset of items, $\mathcal{D}$ is a small subset of a complete rating matrix $\mathbf{R}$, i.e., $\mathcal{D} \subset \mathbf{R} \in \mathbb{R}^{|\mathcal{U}| \times |\mathcal{I}|}$.

Parameter	**Symbol**
Dataset	$\mathcal{D}$
Train dataset	$\mathcal{R}$
Test dataset	$\mathcal{T}$
Set of users	$\mathcal{U}$
Set of items	$\mathcal{I}$
Set of long tail items in $\mathcal{R}$	$\mathcal{L}$
Specific user	u
Specific item	i
Set of items of u in $\mathcal{R}$	$\mathcal{I}_u^{\mathcal{R}}$
Set of items of u in $\mathcal{T}$	$\mathcal{I}_u^{\mathcal{T}}$
Set of users of i in $\mathcal{R}$	$\mathcal{U}_i^{\mathcal{R}}$
Set of users of i in $\mathcal{T}$	$\mathcal{U}_i^{\mathcal{T}}$
Rating of user u on item i	r_{ui}
Size of top-N set	N
Top-N set of u	$\mathcal{P}_u$
Collection of top-N sets for all users	$\mathcal{P}$
Long-tail novelty preference of user u acc. model m	θ_u^m
Accuracy function	$a(.)$
Coverage function	$c(.)$
Value function of user u	$v_u(.)$

TABLE I: Notation.

We split $\mathcal{D}$ into a train set $\mathcal{R}$ and test set $\mathcal{T}$. Let $\mathcal{I}^{\mathcal{R}}$ denote items in train, with $\mathcal{I}_u^{\mathcal{R}}$ the items rated by a single user u in the train set. Similarly, $\mathcal{I}^{\mathcal{T}}$ denote the test items, with $\mathcal{I}_u^{\mathcal{T}}$ denoting the test items of a single user u. Let $\mathcal{U}_i^{\mathcal{R}}$ denote users that rated item i in the train set, and $\mathcal{U}_i^{\mathcal{T}}$ users that rated the item in test. For each user, we generate a top-N set by ranking all unseen train items ($\mathcal{I}^{\mathcal{R}} \setminus \mathcal{I}_u^{\mathcal{R}}$).

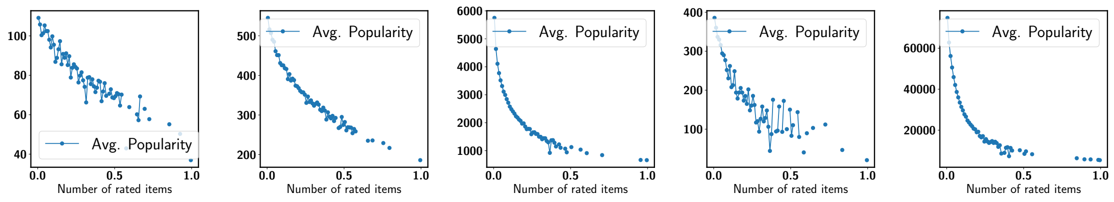
We denote the frequency of item i in a given set $\mathcal{A}$ with $f_i^{\mathcal{A}}$. Following [14], the popularity of an item i is its frequency in the train set, i.e., $f_i^{\mathcal{R}} = |\mathcal{U}_i^{\mathcal{R}}|$. Based on the Pareto principle [13], or the $80/20$ rule, we determine long-tail items, $\mathcal{L}$, as those that generate the lower 20% of the total ratings in the train set, $\mathcal{L} \subset \mathcal{I}^{\mathcal{R}}$ (i.e, items are sorted in decreasing popularity). In our work, we use $x_i = \frac{x_i - \min(\mathbf{x})}{\max(\mathbf{x}) - \min(\mathbf{x})}$ for normalizing a generic vector $\mathbf{x}$.

Table I summarizes our notation. We typeset the sets (e.g., $\mathcal{A}$), use upper case bold letters for matrices (e.g., $\mathbf{A}$), lower-case bold letters for vectors (e.g., $\mathbf{a}$), and lower case letters for scalar variables (e.g., a).

B. Simple long-tail novelty preference models

Users have different preferences for discovering long-tail items. Given the train set $\mathcal{R}$, we need a measure of the user's willingness to explore less popular items. Let θ_u^m denote user preference for long-tail novelty as measured by a model m.

Figure 1 plots the average popularity of rated items vs. the number of rated items (or *user activity*) for our datasets. As user activity increases, the average popularity of rated items decreases. This motivates an *Activity* measure $\theta_u^A = |\mathcal{I}_u^{\mathcal{R}}|$. But, most users only rate a few items, and θ_u^A does not indicate whether those items were long-tail or popular.



(a) ML-100K (b) ML-1M (c) ML-10M (d) MT-200K (e) Netflix

Fig. 1: For each user u, we consider the set of items rated by u in train $\mathcal{I}_u^{\mathcal{R}}$, and compute its average popularity $\bar{a} = \frac{1}{|\mathcal{I}_u^{\mathcal{R}}|}\sum_{i\in\mathcal{I}_u^{\mathcal{R}}} f_i^{\mathcal{R}}$. The x-axis shows the binned normalized $|\mathcal{I}_u^{\mathcal{R}}|$, while the y-axis plots the mean of the corresponding $\bar{a}$ values. The average popularity of items rated decreases as the number of items rated increases.

Instead, we can define a *Normalized Long-tail* measure

$$\theta_u^N = \frac{|\mathcal{I}_u^{\mathcal{R}} \cap \mathcal{L}|}{|\mathcal{I}_u^{\mathcal{R}}|} \tag{II.1}$$

which is the fraction of long-tail items in the user's rated items. The higher this fraction, the higher her preference for long-tail items. However, θ_u^N does not capture the user's interest (e.g, rating) in the items, and does not distinguish between the various long-tail items.

To resolve both problems, we can use similar notions as in TFIDF [25]. The rating an item receives from a particular user reflects its importance for that user. To capture the discriminative power of the item among the set of users and control for the fact that some items are generally more popular, we also incorporate an inverse popularity factor ($|\mathcal{U}_i^{\mathcal{R}}|$) that is logarithmically scaled. We define *TFIDF-based* measure using

$$\theta_u^T = \frac{1}{|\mathcal{I}_u^{\mathcal{R}}|}\sum_{i\in\mathcal{I}_u^{\mathcal{R}}} r_{ui}\log\left(\frac{|\mathcal{U}|}{|\mathcal{U}_i^{\mathcal{R}}|}\right) \tag{II.2}$$

This measure increases proportionally to the user rating r_{ui}, but is also counterbalanced by the popularity of the item. A higher θ_u^T shows more preference for less popular items. Although θ_u^T considers both the user interest r_{ui}, and the item popularity $|\mathcal{U}_i^{\mathcal{R}}|$, it has no indication about the preferences of the users in $\mathcal{U}_i^{\mathcal{R}}$. To address this limitation, observe Eq. II.2 can be re-written as

$$\theta_u^T = \frac{1}{|\mathcal{I}_u^{\mathcal{R}}|}\sum_{i\in\mathcal{I}_u^{\mathcal{R}}} \theta_{ui} = \frac{\sum_{i\in\mathcal{I}_u} w_i\theta_{ui}}{\sum_{i\in\mathcal{I}_u} w_i} \tag{II.3}$$

where $\theta_{ui} = r_{ui}\log\left(\frac{|\mathcal{U}|}{|\mathcal{U}_i^{\mathcal{R}}|}\right)$ is a *per-user-item* long-tail preference value, and $w_i = 1$ for all items. Basically, θ_u^T gives equal importance to all items and is a crude average of θ_{ui}.

We can generalize the idea in Eq. II.3. Specifically, for each user, we consider a *generalized long-tail novelty* preference estimate, θ_u^*. We assume θ_u^* is a weighted average of θ_{ui}. However, rather than imposing equal weights, we define w_i to indicate an *item importance* weight. Our second assumption is that an item is important when its users do not regard it as a mediocre item; when their preference for that item deviates from their generalized preference. In other words, an item i is important when $\sum_{u\in\mathcal{U}_i^{\mathcal{R}}}(\theta_{ui}-\theta_u^*)^2$ is large. Since w_i and θ_u^* influence each other, below we describe how to learn these variables in a joint optimization objective.

C. Learning generalized long-tail novelty preference

We define $\epsilon_i = \left[\sum_{u\in\mathcal{U}_i^{\mathcal{R}}} 1 - \left(\theta_{ui} - \theta_u^*\right)^2\right]$ as the *item mediocrity*. Assuming $|\theta_{ui} - \theta_u^*| \le 1$ (explained later), the maximum of ϵ_i is obtained when $\theta_{ui} = \theta_u^*$. We formulate our objective as:

$$O(\mathbf{w}, \boldsymbol{\theta}^*) = \sum_{i\in\mathcal{I}^{\mathcal{R}}} w_i\Big[\sum_{u\in\mathcal{U}_i^{\mathcal{R}}} 1 - \left(\theta_{ui} - \theta_u^*\right)^2\Big] = \sum_{i\in\mathcal{I}^{\mathcal{R}}} w_i\epsilon_i$$

which is the total weighted mediocrity. Here, $\mathcal{I}^{\mathcal{R}}$ are the items in train, $\mathcal{U}_i^{\mathcal{R}}$ denotes users that rated item i in the train set (Section II-A), and θ_{ui} is the per-user-item preference value, computed from rating data. Our objective has two unknown variables: $\boldsymbol{\theta}^* \in \mathbb{R}^{|\mathcal{U}|}$, $\mathbf{w} \in \mathbb{R}^{|\mathcal{I}|}$. We use an alternating optimization approach for optimizing $\mathbf{w}$ and $\boldsymbol{\theta}^*$. When optimizing w.r.t. $\mathbf{w}$, we must minimize the objective function in accordance with our intuition about w_i. In particular, for larger mediocrity coefficient, we need smaller weights. On the other hand, when optimizing w.r.t. $\boldsymbol{\theta}^*$, we need to increase the closeness between θ_u^* and all θ_{ui}'s, which is aligned with our intuition about θ_u^*. So, we have to maximize the objective function w.r.t. $\boldsymbol{\theta}^*$. Our final objective is a minimax problem:

$$\min_{\mathbf{w}}\max_{\boldsymbol{\theta}^*} \quad O(\mathbf{w}, \boldsymbol{\theta}^*) - \lambda_1\sum_{i\in\mathcal{I}^{\mathcal{R}}}\log w_i \tag{II.4}$$

where we have added a regularizor term $\sum_{i\in\mathcal{I}^{\mathcal{R}}}\log w_i$ to prevent w_i from approaching 0 [26].

When θ^* is fixed, we need to solve a minimization problem involving only $\mathbf{w}$. By taking the derivative w.r.t. w_i we have

$$w_i = \frac{\lambda_1}{\sum_{u\in\mathcal{U}_i^{\mathcal{R}}} 1 - \left(\theta_{ui} - \theta_u^*\right)^2} = \frac{\lambda_1}{\epsilon_i} \tag{II.5}$$

When $\mathbf{w}$ is fixed, we need to solve a maximization problem involving $\boldsymbol{\theta}^*$. Taking the derivative w.r.t. θ_u^* we derive

$$\theta_u^* = \frac{\sum_{i\in\mathcal{I}_u^{\mathcal{R}}} w_i\theta_{ui}}{\sum_{i\in\mathcal{I}_u^{\mathcal{R}}} w_i} \tag{II.6}$$

Essentially, Eq. II.5 characterizes an item's weight, based on the item mediocrity. The higher the mediocrity, the lower the weight. Moreover, for every user u, θ_u^* is a weighted average of all θ_{ui}. Note, in Eq. II.6, $\theta^*_u = \theta^T_u$ when $w_i = 1$ for all items. Our generalized θ^*_u incorporates both the user interest and popularity of items (via θ_{ui}), and the preferences of other users of the item (via w_i). Furthermore, since we need $|\theta_{ui} - \theta^*_u| \leq 1$, and prefer $\theta^*_u \in [0,1]$, we project all θ_{ui} to the $[0,1]$ interval before applying the algorithm. We also set $\lambda_1 = 1$.

III. Generic Re-ranking Framework

We define user satisfaction based on the accuracy of the top-N set and its coverage of the item space, so as to introduce novelty and serendipity into recommendation sets. We consider three components: 1) an *accuracy recommender* that is responsible for suggesting accurate top-N sets. The function $a(.)$ measures the score of a set of items according to the accuracy recommender. 2) a *coverage recommender* that is responsible for suggesting top-N sets that maximize coverage across the item space, and consequently promote long-tail items. The function $c(.)$ measures the score of a set according to the coverage recommender. 3) the user preference for long-tail novelty $\theta_u \in [0,1]$. We define individual user value functions for a top-N set $\mathcal{P}_u$ as

$$v_u(\mathcal{P}_u) = (1-\theta_u)a(\mathcal{P}_u) + \theta_u c(\mathcal{P}_u) \tag{III.1}$$

With slight abuse of notation, let $a(i)$ and $c(i)$ denote the accuracy score and coverage score of a single item i. We ensure $a(i), c(i) \in [0,1]$ to have the same scale. We define $a(\mathcal{P}_u) = \sum_{i \in \mathcal{P}_u} a(i)$ and $c(\mathcal{P}_u) = \sum_{i \in \mathcal{P}_u} c(i)$.

The user value function in Eq. III.1 positively rewards sets that increase coverage. Similar intuitions for encouraging solutions with desirable properties, e.g., diverse solutions, have been used in related work [27]. However, their trade-off parameters are typically obtained via parameter tuning or cross validation. By contrast, we impose personalization via the user preference estimate, θ_u, that is learned based on historical rating data. Next, we list the various base recommender models integrated in our framework.

A. Accuracy Recommender

The accuracy recommender provides an accuracy score, $a(i)$, for each item i. We experiment with three models (Section IV-A provides details and setup configurations):

- **Most popular (Pop)** [1] is non-personalized and recommends the most popular unseen items. Pop produces accurate recommendations, but low coverage and novelty [1]. Since it does not score items, we define $a(i) = 1$ if item i is in the top-N set suggested by pop, otherwise $a(i) = 0$.
- **Regularized SVD (R-SVD)** [28] learns latent factors for users and items by analysing user-item interaction data (e.g., ratings). The factors are then used to predict the values of unobserved ratings. We use R-SVD to compute a predicted rating matrix $\hat{\mathbf{R}} \in \mathbb{R}^{|\mathcal{U}|\times|\mathcal{I}|}$. We normalize the predicted rating vectors of all users to ensure $\hat{r}_{ui} \in [0,1]$, and define $a(i) = \hat{r}_{ui}$.
- **PureSVD (P-SVD)** [1] is also a latent factor model. We follow the same procedure as R-SVD, using P-SVD factors [1]. Note, P-SVD scores correspond to associations between users and items.

B. Coverage Recommender

The coverage recommender provides a coverage score, $c(i)$, for each item i. We use three coverage recommenders:

- **Random (Rand)** recommends N unseen items randomly. It has high coverage, but low accuracy [5]. We define $c(i) \sim$unif$(0,1)$.
- **Static coverage (Stat)** focuses exclusively on promoting less popular items. We define $c(i)$ to be a monotone decreasing function of the popularity of i in the train set, i.e., $f^{\mathcal{R}}_i$. We use $c(i) = \frac{1}{\sqrt{f^{\mathcal{R}}_i + 1}}$ in our work. Note the gain of recommending an item is constant.
- **Dynamic coverage (Dyn)** To better correct for the popularity bias in recommendations, rather than the train set $\mathcal{R}$, we can define $c(i)$ based on the set of recommendations made so far. In particular, let $\mathcal{P} = \{\mathcal{P}_u\}^{|\mathcal{U}|}_{u=1}$ denote the collection of top-N sets assigned to all users, and $\mathcal{A} \subset \mathcal{P}$, a partial collection where a subset of users have been assigned top-N sets. We measure the long-tail appeal of an item using a monotonically decreasing function of the popularity of i in $\mathcal{A}$, i.e., $f^{\mathcal{A}}_i$. We use $c(i) = \frac{1}{\sqrt{f^{\mathcal{A}}_i + 1}}$ in our work. The main intuition is that recommending an item has a diminishing returns property: the more the item is recommended, the less the gain of the item in coverage, i.e., $c(i) = 1$ when $\mathcal{A} = \emptyset$, but decreases as the item is more frequently suggested.

C. Optimization Algorithm

The overall goal of the framework is to find an optimal top-N collection $\mathcal{P} = \{\mathcal{P}_u\}^{|\mathcal{U}|}_{u=1}$ that maximizes the aggregate of the user value functions:

$$\max_{\mathcal{P}} \; v(\mathcal{P}) = \sum_{u \in \mathcal{U}} v_u(\mathcal{P}_u) \tag{III.2}$$

The combination of Random and Static coverage with the accuracy recommenders in Section III-A, result in value functions that can be optimized greedily and independently, for each user. Using the dynamic recommender, however, creates a dependency between the optimization of user value functions, where the items suggested to one user depend on those suggested to previous users. Therefore, user value functions can no longer be optimized independently.

Optimization algorithm for dynamic coverage. Since Dynamic coverage is monotonically decreasing in $f^{\mathcal{A}}_i$, where $\mathcal{A} \subset \mathcal{P}$, when used in Eq.III.1, the overall objective in Eq. III.2 becomes submodular across users. Maximizing a submodular function is NP-hard [29]. However, a key observation is that the constraint of recommending N items to each user, corresponds to a partition matroid over the users. Finding a top-N collection $\mathcal{P}$ that maximizes Eq. III.2 is therefore an instance of maximizing a submodular function subject to a matroid constraint (see Appendix B). A *Locally Greedy* heuristic, due to Fisher et al. [30], can be applied: consider the users separately and in arbitrary order. At each step, select a user u arbitrarily, and greedily construct $\mathcal{P}_u$ for that user.

Algorithm 1: OSLG

```
   Input: $N, S, \mathcal{U}, \mathcal{I}, \mathcal{R}$
   Output: $\mathcal{P}$
1  $\mathbf{f} \leftarrow 0, \mathcal{P} \leftarrow \emptyset, \boldsymbol{\theta} \leftarrow$ Estimate long-tail preference from $\mathcal{R}$;
2  $\mathcal{S} \leftarrow$ Sample $S$ users from $\mathcal{U}$ acc. to KDE($\boldsymbol{\theta}$);
3  Sort $\mathcal{S}$ acc. to $\boldsymbol{\theta}$ increasingly ;
   // Performed sequentially for users in sample
4  for $u \in \mathcal{S}$ do
5      Update coverage function $c(.)$ based on new frequencies $\mathbf{f}$ ;
6      $\mathcal{P}_u$ = $\mathcal{P}_u \cup \text{argmax}_{i \in \mathcal{I}}\ v_u(\mathcal{P}_u \cup i)$ - $v_u(\mathcal{P}_u)$;
7      foreach $i \in \mathcal{P}_u$ do $\mathbf{f}_i = \mathbf{f}_i + 1$;
8      F($\theta_u$) $\leftarrow \mathbf{f}$;
9      $\mathcal{P} = \mathcal{P} \cup \mathcal{P}_u$;
10 end
   // Performed in parallel for users not in sample
11 foreach $u \in \mathcal{U} \setminus \mathcal{S}$ do
12     $\hat{\mathbf{f}} \leftarrow F(\text{argmin}(\theta_s - \theta_u))$ for $s \in \mathcal{S}$ ;
13     Update coverage function $c(.)$ based on new frequencies $\hat{\mathbf{f}}$;
14     $\mathcal{P}_u$ = $\mathcal{P}_u \cup \text{argmax}_{i \in \mathcal{I}}\ v_u(\mathcal{P}_u \cup i)$ - $v_u$ $\mathcal{P} = \mathcal{P} \cup \mathcal{P}_u$;
15 end
16 return $\mathcal{P}$
```

Proceed until all users have been assigned top-N sets. Locally Greedy produces a solution at least half the optimal value for maximizing a submodular monotone function subject to a matroid constraint [30].

However, locally greedy is sequential and has a computational complexity of $O(|\mathcal{U}|.|\mathcal{I}|.N)$ which is not scalable. Instead, we introduce a heuristic we call *Ordered Sampling-based Locally Greedy* (OSLG). Essentially, we make two modifications based on the user long-tail preferences: first, proportionate to the distribution of user long-tail preferences θ, we sample a subset of users, and run the sequential algorithm on this sample only. Second, to allow the system to recommend more established or popular products to users with lower long-tail preference, instead of arbitrary order, we modify locally greedy to consider users in increasing long-tail preference.

Algorithm 1 shows our framework for Dynamic coverage: We use $\mathbf{f}_i$ as a shorthand for the Dynamic coverage function argument $f_i^{\mathcal{A}}$, where $\mathcal{A} \subset \mathcal{P}$. First, $\mathbf{f}$ is initialized and user preferences θ are estimated (line 1). Next, we use Kernel Density Estimation (KDE) [31] to approximate the probability density function (pdf) of θ, and use the pdf to draw a sample $\mathcal{S}$ of size S from θ and find the corresponding users in $\mathcal{U}$ (line 2). The sampled users are then sorted in increasing long-tail preference (line 3), and the algorithm iterates over the users. In each iteration, it updates the dynamic coverage function (line 5) and assigns a top-N set to the current user by maximizing her value function (line 6). The coverage function argument $\mathbf{f}$ is then updated w.r.t. the recently assigned top-N set (line 7). Moreover, $\mathbf{f}$ is associated with the current long-tail preference estimate θ_u, and stored (line 8). The algorithm then proceeds to the next user. Since the Dynamic coverage function (Eq. III-B) is monotonically decreasing in $\mathbf{f}_i$, frequently recommended items are weighted down by the value function of subsequent users. Consequently, as we reach users who prefer long-tail items and discovery, their value functions prefer relatively unpopular items that have not been recommended before. Thus, the induced user ordering, results in the promotion of long-tail items to the right group of users, such that we obtain better coverage while maintaining user satisfaction.

For each user not included in the sample set, $u \notin \mathcal{S}$, we find the most similar user $u_s \in \mathcal{S}$, where similarity is defined as $|\theta_u - \theta_s|$ (line 12), and use $F(\theta_s)$ to compute the coverage score (line 13), and assign a top-N set. Observe, the value functions of $u \in \mathcal{U} \setminus \mathcal{S}$, are independent of each other, and lines 12-14 can be performed in parallel. The computational complexity of the sequential part drops to $O(|\mathcal{S}|.|\mathcal{I}|.N)$ at the cost of $O(|\mathcal{S}|.|\mathcal{I}|)$ extra memory.

IV. Empirical Evaluation

A. Experimental setup

Datasets and data split. Table II describes our datasets. We use MovieLens 100K (ML-100K), 1 Million (Ml-1M), 10 Million (ML-10M) ratings [32], Netflix, and MovieTweetings 200K (MT-200K) [33]. In the MovieLens datasets, every consumer has rated at least 20 movies ($\tau = 20$), with $r_{ui} \in \{1, \ldots, 5\}$ (ML-10M has half-star increments). MT-200K contains voluntary movie ratings posted on twitter, with $r_{ui} \in \{0, \ldots, 10\}$. Following [34], we preprocessed this dataset to map the ratings to the interval $[1, 5]$. Due to the extreme sparsity of this dataset and to ensure every user has some data to learn from, we filtered the users to keep those with at least 5 ratings ($\tau = 5$).

Our selected datasets have varying density levels. Additionally, MT-200K and Netflix include a large number of difficult infrequent users, i.e., in MT-200K, 47.42% (3.37% in Netflix) of the users have rated fewer than 10 items, with the minimum being 4. We chose these datasets to study performance in settings where users provide few feedback [4], [21].

Next, we randomly split each dataset into train and test sets by keeping a fixed ratio κ of each user's ratings in the train set and moving the rest to the test set [35]. This way, when $\kappa = 0.8$, an infrequent user with 5 ratings will have 4 train and 1 test rating, while a user with 100 ratings, will have 80 train ratings and the rest in test. For ML-1M and ML-10M, we set $\kappa = 0.5$. For MT-200K, we set $\kappa = 0.8$. For Netflix, we use their probe set as our test set, and remove the corresponding ratings from train. We remove users in the probe set who do not appear in train set, and vice versa.

Test ranking protocol and performance metrics. For testing, we adopt the "All unrated items test ranking protocol" [36],

Dataset	$\lvert\mathcal{D}\rvert$	$\lvert\mathcal{U}\rvert$	$\lvert\mathcal{I}\rvert$	d%	$\mathcal{L}\%$	κ	τ
ML-100K	100K	943	1682	6.30	66.98	0.5	20
ML-1M	1M	6,040	3,706	4.47	67.58	0.5	20
ML-10M	10M	69,878	10,677	1.34	84.31	0.5	20
MT-200k	172,506	7,969	13,864	0.16	86.84	0.8	5
Netflix	98,754,394	459,497	17,770	1.21	88.27	-	-

TABLE II: Datasets description. $|\mathcal{D}|$ is # ratings in dataset. Density is d% $= (|\mathcal{D}|/|\mathcal{U}| * |\mathcal{I}|) \times 100\%$. Long-tail percentage is $\mathcal{L}\% = (|\mathcal{L}|/|\mathcal{I}^{\mathcal{R}}|) \times 100\%$. The train-test split ratio per user is κ, and τ is the minimum # ratings per user.

Local Ranking Accuracy Metrics	Precision@N $= \frac{1}{N\lvert\mathcal{U}\rvert} \sum_{u \in \mathcal{U}} \lvert\mathcal{I}_u^{\mathcal{T}+} \cap \mathcal{P}_u\rvert$
	Recall@N $= \frac{1}{\lvert\mathcal{U}\rvert} \sum_{u \in \mathcal{U}} \frac{\lvert\mathcal{I}_u^{\mathcal{T}+} \cap \mathcal{P}_u\rvert}{\lvert\mathcal{I}_u^{\mathcal{T}+}\rvert}$
	F-measure@N $= \frac{\text{Precision@N.Recall@N}}{\text{Precision@N+Recall@N}}$
Longtail Promotion	LTAccuracy@N $= \frac{1}{N\lvert\mathcal{U}\rvert} \sum_{u \in \mathcal{U}} \lvert\mathcal{L} \cap \mathcal{P}_u\rvert$
	StratRecall@N $= \frac{\sum_{u \in \mathcal{U}} \sum_{i \in \mathcal{I}_u^{\mathcal{T}+} \cap \mathcal{P}_u} s_i}{\sum_{u \in \mathcal{U}} \sum_{i \in \mathcal{I}_u^{\mathcal{T}+}} s_i}$, $s_i = \left(\frac{1}{f_i^{\mathcal{R}}}\right)^{\beta}$
Coverage Metrics	Coverage@N $= \frac{\lvert\cup_{u \in \mathcal{U}} \mathcal{P}_u\rvert}{\lvert\mathcal{I}\rvert}$
	Gini@N $= \frac{1}{\lvert\mathcal{I}\rvert}(\lvert\mathcal{I}\rvert + 1 - 2\frac{\sum_{j=1}^{\lvert\mathcal{I}\rvert}(\lvert\mathcal{I}\rvert+1-j)f[j]}{\sum_{j=1}^{\lvert\mathcal{I}\rvert} f[j]})$

TABLE III: Performance Metrics. Notation is in Section II-A. For gini, the vector $\mathbf{f}$ is sorted in non-decreasing order of recommendation frequency of items, i.e., $f[j] \leq f[j+1]$.

[5] where for each user, we generate the top-N set by ranking all items that do not appear in the train set of that user (details in Appendix C).

Table III summarizes the performance metrics. To measure how accurately an algorithm can rank items for each user, we use local rank-based precision and recall [37], [5], [36], where each metric is computed per user and then averaged across all users. Precision is the proportion of relevant test items in the top-N set, and recall is the proportion of relevant test items retrieved from among a user's relevant test items. As commonly done in the literature [37], [38], for each user u, we define her relevant test items as those that she rated highly, i.e., $\mathcal{I}_u^{\mathcal{T}+} = \{i : i \in \mathcal{I}_u^{\mathcal{T}}, r_{ui} \geq 4\}$. Note, because the collected datasets have many missing ratings, the hypothesis that only the observed test ratings are relevant, underestimates the true precision and recall [36]. But, this holds for all algorithms, and the measurements are known to reflect performance in real-world settings [36]. F-measure is the harmonic mean of precision and recall. For algorithms that re-rank rating-prediction models, we also report the prediction metric (Prediction@N) that is known to be correlated with precision [11]. It is the average predicted rating value of the top-N set, according to the underlying rating-prediction model. It measures the deviation of any top-N from the default greedy strategy of a rating prediction model [11], [20].

We use Long-Tail Accuracy (LTAccuracy@N) [20] to measure the novelty of recommendation lists. It computes the proportion of the recommended items that are unlikely to be seen by the user. Moreover, we use Stratified Recall (Stratified Recall$N$) [36] which measures the ability of a model to compensate for the popularity bias of items w.r.t train set. Similar to [36], we set $\beta = 0.5$. Note, LTAccuracy emphasizes a combination of novelty and coverage, while Stratified Recall emphasizes a combination of novelty and accuracy.

For coverage we use two metrics: Coverage@N is the ratio of the total number of distinct recommended items to the total number of items [20], [5]. A maximum value of 1 indicates each item in $\mathcal{I}$ has been recommended at least once. Gini [39], measures the inequality among values of a frequency distribution $\mathbf{f}$. It lies in $[0, 1]$, with 0 representing perfect equality, and larger values representing skewed distributions. In Table III, $\mathbf{f}$ is the recommendation frequency of items, and is sorted in non-decreasing order, i.e., $f[j] \leq f[j+1]$.

Other algorithms and their configuration. We compare against, or integrate the following methods in our framework.

- **Pop** [1]: is a non-personalized algorithm For ranking tasks, it obtains high accuracy [1], [5], since it takes advantage of the popularity bias of the data. However, as stated in [1], Pop makes trivial recommendations that lack novelty.
- **Rand**: is non-personalized and randomly suggests N unseen items from among all items. It obtains high coverage and novelty, but low accuracy [5].
- **R-SVD** [40]: are latent-factor models for rating prediction. We used `LIBMF` for a Regularized SVD (R-SVD) model, with L2-Norm as the loss function, and L2-regularization, and Stochastic Gradient Descent (SGD) for optimization. We also tested the same model with non-negative constraints (R-SVDN) [28], but did not find significant performance difference. We omit R-SVDN from our results. We performed 10-fold cross validation and tested: number of latent factors $g \in \{8, 20, 40, 50, 80, 100\}$, L2-reqularization coefficients $\lambda \in \{0.001, 0.005, 0.01, 0.05, 0.1\}$, learning rate $\eta \in \{0.002, 0.003, 0.01, 0.03\}$. Details of chosen algorithm parameters, in different datasets, is in Appendix A.
- **CoFiRank** [41]: is a ranking prediction model that can optimize directly the Normalized Discounted Cumulative Gain (NDCG) ranking measure [42]. For CofiRank, we experimented with both regression (squared) loss (Cofi^R) and NDCG loss (Cofi^N). We used the source code from [41], with parameters set according to [41]: 100 dimensions and $\lambda = 10$, and default values for other parameters (source code). Similar to [43], [44], we found Cofi^R to perform consistently better than Cofi^N in our experiments on ML-1M and ML-100K. We only report results for Cofi^R.
- **P-SVD** [1]: is a latent factor model, known for achieving high accuracy and novelty in [1]. In P-SVD, missing values are imputed by zeros and conventional SVD is performed. For P-SVD we use Python's `sparsesvd` module and tested number of latent factors $g \in \{10, 20, 40, 60, 100, 150, 200, 300\}$. As an accuracy rec-

ommender, we tested two configurations of P-SVD: one with few factors 10 and one with larger factors 100.

- **RBTPOP** and **RBTAVG** [14]: are Ranking-Based Techniques (RBT) that maximize coverage by re-ranking the output of rating prediction models. We implemented both models. RBTPOP re-ranks a few of the items in the head according to their popularity, while RBTAVG re-ranks according to the average ratings of the items. As in [14], $T_{max} = 5$ in all datasets. The parameter T_R controls the extent of re-ranking. We tested $T_R \in [4, 4.2, 4.5]$, and found $T_R = 4.5$ to yield more accurate results. Furthermore, because our datasets contain a wider range of users compared to [14], we set $T_H = 1$ on all datasets, except ML-10M and Netflix, where we set $T_H = 0$. tested $T_R \in \{4, 4.5\}$.
- **Resource allocation** (**5D**, **5DRR**, **5DA**,**5DARR**) [20]: is a resource allocation approach for re-ranking the output of a rating prediction model. The method has two phases: 1) resources are allocated to items according to the received ratings, and 2) the resources are distributed according to the relative preferences of the users, and top-N sets are generated by assigning a a 5D score (for accuracy, balance, coverage, quality, and quantity of long-tail items) to every user-item pair. We use the variants proposed in [20], which are combinations of the scoring function (5D) with the rank by rankings (RR) and accuracy filtering (A) algorithms (Section 3.2.2 in [20]). We implemented and ran all four variants with default values set according to [20]: $k = 3.|\mathcal{I}|$ and $q = 1$.
- **PRA** [22]: is a generic Personalized Ranking Adaptation (PRA) re-ranking framework, that first estimates user tendency for various criteria like diversity and novelty, then iteratively and greedily re-ranks items in the head of the recommendations to match the top-N set with the user tendencies. We compare with the novelty-based variant of this framework, which relies on item popularity statistics to measure user novelty tendencies. We use the the mean-and-deviation based heuristic, that is measured using the popularity of rated items, and was shown to provide comparable results with other heuristics in [22]. For the configurable parameters, we followed [22]: Sample set size $S_u \in \min(|\mathcal{I}_u^{\mathcal{R}}|, 10)$, the exchangeable set size $|X_u| \in \{10, 20, 50\}$, we used "optimal swap" strategy with $maxSteps = 20$.

Our framework. To refer to variants of our framework, we use $\texttt{Preference Model}^{\texttt{Cov. Recommender}}_{\texttt{Acc. Recommender}}$. We include sample size for dynamic coverage, e.g., $\theta^{*}{}^{Dyn300}_{Pop}$ uses 300 samples. We run algorithms that involve randomness (e.g., sampling-based variants) 10 times, and report the average.

B. Distribution of long-tail novelty preferences

Figure 2 plots the histogram of various long-tail preference models

. We observe θ^A is skewed to the right. This is due to the sparsity problem, where the majority of users rate a few items. θ^N is also skewed to the right across all datasets, due to both the popularity bias and sparsity problems [37], [45]. On the other hand, θ^* is normally distributed, with a larger mean and a larger variance, on all datasets. In the experiments in Section IV-C, we study the effect of these preference models on performance.

C. Performance of OSLG

Effect of sample size. The sample size S is a system-wide hyper-parameter in the OSLG algorithm, and should be determined w.r.t. preference for accuracy or coverage, and the accuracy recommender. For tuning S, we run experiments on ML-1M, and assess its effect on F-measure and coverage. As shown in Figure 4, increasing S, increases coverage and decreases the F-measure of most accuracy recommenders. Regarding R-SVD, the scores output by this model lead to the initial decrease and later increase of F-measure. Since we want to maintain accuracy, we fix $S = 300$ in the rest of our experiments, although a larger S can be used for R-SVD.

Effect of the user preference model, the accuracy recommender, and their interplay. To examine the effect of the user long-tail novelty preference model, we run OSLG with the following long-tail preference models:

- **Random** θ^R randomly initializes θ_u^R (10 times per user).
- **Constant** θ^C assigns the same constant value C to all users. We report results for $C = 0.5$.
- **Normalized Long-tail** θ^L (Eq. II.1) measures the proportion of long-tail items the user has rated in the past.
- **Tfidf-based** θ^T (Eq. II.2) incorporates user interest and popularity of items.
- **Generalized** θ^* (Eq. II.6) incorporates user interest, popularity of items, and the preferences of other users.

Since OSLG with $S = 300$ involves sampling, we run each variant 10 times (with random shuffling for θ^C) and average over the runs. We run the experiments on ML-1M.

Figure 5 compares the combination of dynamic coverage with different long-tail novelty preference models, and different accuracy recommenders. As expected, the base accuracy recommender typically obtains the best F-measure. Moreover, θ^R and θ^C often have the lowest F-measure. Different variants of our framework, are generally in the middle, which shows that the long-tail preference estimates are better than θ^R and θ^C in terms of accuracy. For Stratified Recall, similar trends as accuracy are observed in the ranking of the algorithms.

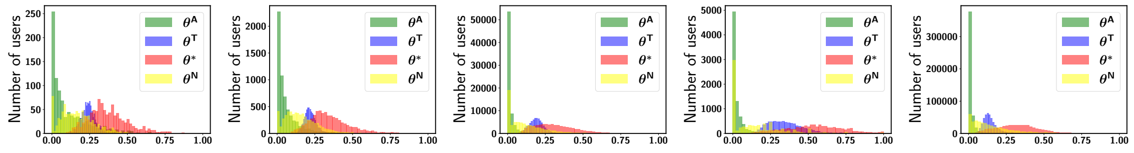
The trends are approximately the same as we vary the accuracy recommender in Figures 5.b, 5.c, and 5.d.

V. Comparison with other recommendation models

We conduct two rounds of experiments: first, we evaluate re-ranking frameworks that post-process rating-prediction models, then we study general top-N recommendation algorithms.

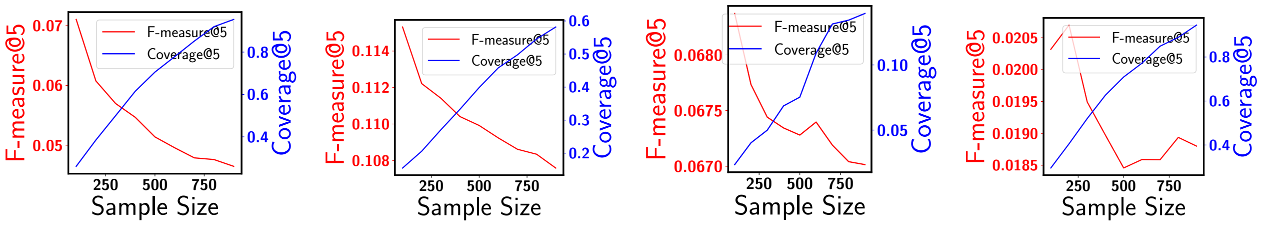
A. Comparison with re-ranking methods for rating-prediction

Standard re-ranking frameworks, typically use a rating prediction model as their accuracy recommender. In this section,



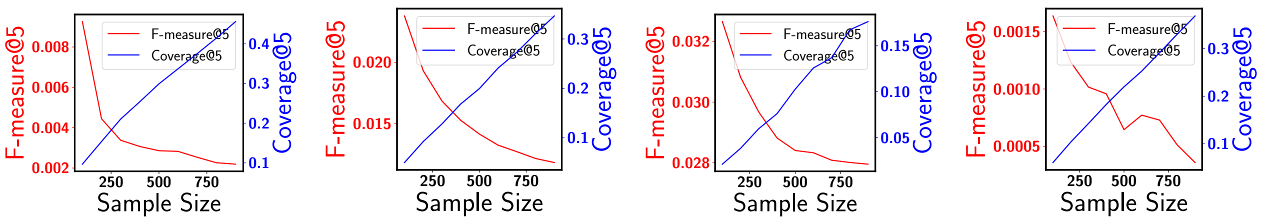
(a) ML-100K (b) ML-1M (c) ML-10M (d) MT-200K (e) Netflix

Fig. 2: Histogram of long-tail novelty preference models. Observe θ^A is skewed toward smaller values because of sparsity, i.e., the majority of users rate a few items. θ^N is also biased toward smaller values, due to a combination of popularity bias and sparsity. θ^T and θ^* are less biased and more normally distributed and alleviate both problems.



(a) P-SVD with 100 factors (b) P-SVD with 10 factors (c) Pop (d) R-SVD

Fig. 3: Top-5 recommendation performance of OSLG $\theta^{*DYN(S)}_{AccuracyRecommender}$, as sample size (S) is varied, across different accuracy recommenders. The accuracy recommender is indicated in each sub-figure. Dataset is ML-1M.



(a) P-SVD with 100 factors (b) P-SVD with 10 factors (c) Pop (d) R-SVD

Fig. 4: Top-5 recommendation performance of OSLG $\theta^{*DYN(S)}_{AccuracyRecommender}$, as sample size (S) is varied, across different accuracy recommenders. The accuracy recommender is indicated in each sub-figure. Dataset is MT-200K.

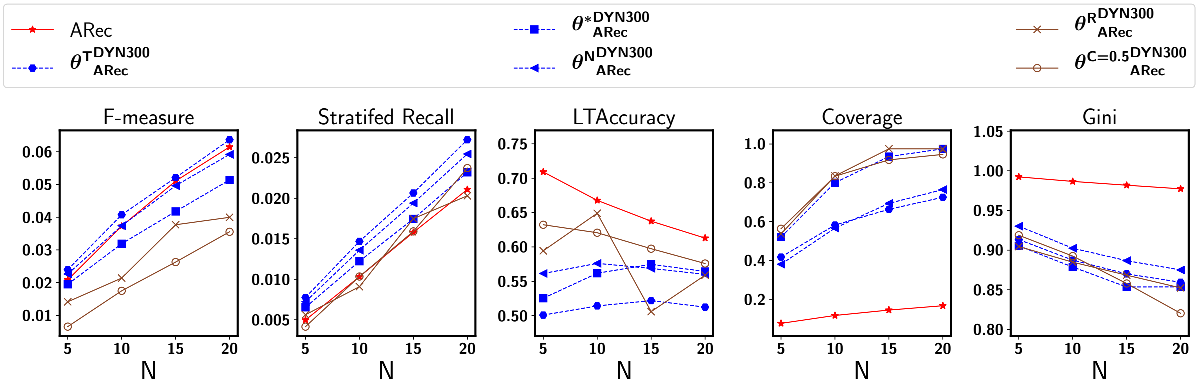
we use R-SVD as the underlying rating prediction model for all re-ranking methods, and analyze performance across datasets with varying density levels. Table IV shows top-5 recommendation performance. [1] We compared the standard greedy ranking strategy of R-SVD (denoted as R-SVD), RBTPOP, RBTAVG, PRA (two variants with $|X_u| = 10$, and $|X_u| = 20$), Resource Allocation (5D, 5DA, 5DRR, 5DARR), , $\theta^{*DYN300}_{R-SVD}$, and $\theta^{*DYN300}_{R-SVD}$. We report results for $N = 5$, since users rarely look past the items at the top of recommendation sets.

We have omitted ML-100K, due to space limitations, although results are generally similar to ML-1M. Regarding R-SVD, the model has high overall novelty (LTAccuracy), but under-performs all other models in coverage and gini. Essentially, R-SVD picks a small subset of the items, including long-tail items, and recommends them to all the users. After all, R-SVD model is trained by optimizing Root Mean Square Error (RMSE) accuracy measure, which is defined w.r.t. available data and not the complete data [46]. Therefore, when the model is used to choose a few items (N) from among *all* available items, as is required in top-N recommendation problems, it does not obtain good overall performance.

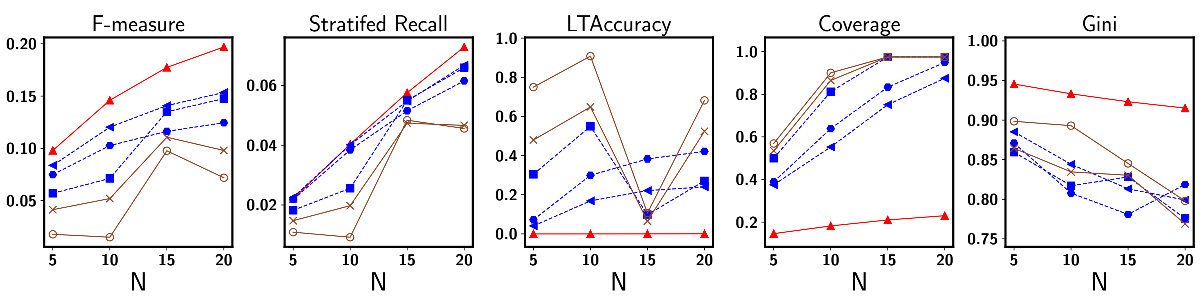
Regarding our two models, in dense settings (ML-1M and ML-100K), our methods outperform all other models in all metrics except LTAccuracy. In sparse settings, except on ML-10M, we have at least one method in the top 2 methods w.r.t f-measure. In both sparse and dense settings, except on ML-10M, we rank first w.r.t. stratified recall. Observe, other methods like 5D focus exclusively on LTAccuracy and significantly reduce F-measure and stratified recall.

In summary, the performance of R-SVD depends on the dataset density. On the sparse datasets, it does not provide accurate suggestions w.r.t. f-measure, and subsequent re-ranking models also fail at making accurate suggestions. w.r.t precision and recall. Therefore, re-ranking a rating prediction model like

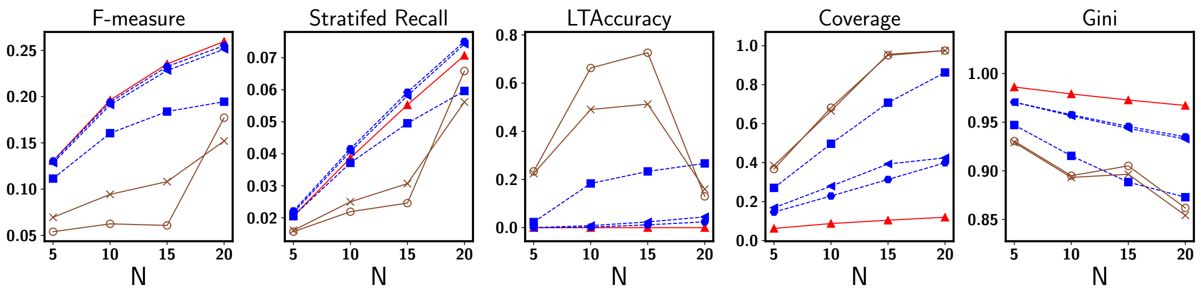
[1] The configuration parameters of R-SVD for each dataset are reported in Appendix A.



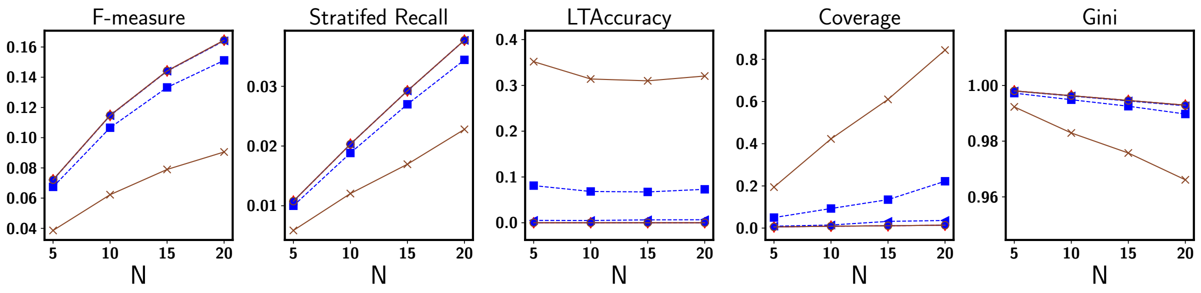
(a) Accuracy recommender (ARec) is R-SVD



(b) Accuracy recommender (ARec) is P-SVD100



(c) Accuracy recommender (ARec) is P-SVD10



(d) Accuracy recommender (ARec) is Pop

Fig. 5: Performance of OSLG with different accuracy recommenders and different long-tail preference models (Sample Size=|300|). Dataset is ML-1M. The trends are approximately the same as we vary the accuracy recommender in Figures 5.a, 5.b, 5.c, and 5.d. Overall, in each row, the accuracy model (ARec) typically achieves the highest F-measure, but obtains the lowest coverage and gini. Variants of our framework that use θ^{*}, θ^{T}, θ^{N}, obtain higher f-measure levels compared to those that use θ^{R} and θ^{C}. They also consistently improve stratified recall, independent of the accuracy recommender. Stratified Recall emphasizes novelty and accuracy.

R-SVD, is only effective for dense settings. However, our framework is generic, and enables us to plug-in a different accuracy recommender. We show the results for this in the next section.

In addition to precision and recall, we also report the prediction accuracy metric, that is commonly used to evaluate the re-ranking of rating prediction models [11], [20]. As shown in Table IV, on all datasets, the R-SVD model obtains the highest prediction scores, and the relative ordering of the algorithms in terms of prediction, is consistent across all datasets. Furthermore, although this metric seems correlated with precision and recall on the dense datasets ML-100K, ML-1M, and ML-10M, where users have at least 10 ratings in the train set, on Netflix and MT-200K, that include a wider range of infrequent users and items, the precision and recall values are approximately zero, and it is more difficult to asses performance w.r.t the prediction metric.

Moreover, all re-ranking techniques increase coverage, but reduce accuracy. 5D obtains the highest novelty among all models, but reduces precision and recall significantly. On most datasets, our model ${\theta^*}_{RSVD}^{DYN300}$, significantly increases coverage and decreases gini, while maintaining reasonable levels of accuracy.

B. Comparison with top-N item recommendation models

As shown previously, in sparse settings, re-ranking frameworks that rely on rating prediction models do not generate accurate solutions. In this section we plug-in a different accuracy recommender w.r.t. dataset density. On MT-200K, we plug-in Pop. On all other datasets we plug-in P-SVD100 as the accuracy recommender. For all generic frameworks we use the same accuracy recommender.

For our framework, we use three variants which differ in their coverage recommender: ${\theta^*}_{ARec}^{DYN300}$,${\theta^*}_{ARec}^{STAT}$ and ${\theta^*}_{ARec}^{RAND}$. We also compare to the generic re-ranking framework, PRA (denoted PRA$_{ARec}$). We compared with standard top-N recommendation algorithms: Rand, Pop, P-SVD with 10 factors (P-SVD10), P-SVD with 100 factors (P-SVD100), and CofiRank (Cofi$^R, 100$).

Figure 6 compares accuracy, coverage and novelty trade-offs. On all datasets, Rand achieves the best coverage and gini, but the lowest accuracy. Similar to [1], [5], we find the non-personalized method Pop, which takes advantage of popularity bias of the data, is a strong contender in accuracy, but under-performs in coverage and novelty.

Regarding our models, Figure 6 shows the best improvements in coverage are obtained when we use either Random or Dynamic recommenders. Static is generally not a strong coverage recommender. For example, on ML-10M, ${\theta^*}_{ARec}^{STAT}$ obtains high novelty (LTAccuracy), but does not lead to significant improvements in coverage, and reduces accuracy. This is because the Static coverage recommender has constant gain for long-tail promotion and focuses exclusively on recommending (a subset of the) long-tail items.

Our main model is ${\theta^*}_{ARec}^{DYN300}$. An interesting observation is that on MT-200K, ${\theta^*}_{ARec}^{DYN-300}$ and ${\theta^*}_{ARec}^{RAND}$, which rely on the non-personalized accuracy recommender, Pop, show competitive performance with personalized algorithms like P-SVD100 or Cofi$^R, 100$.

VI. Related Work

We provide an overview of related work here. A detailed description of baselines and methods integrated in our framework is provided in Section IV.

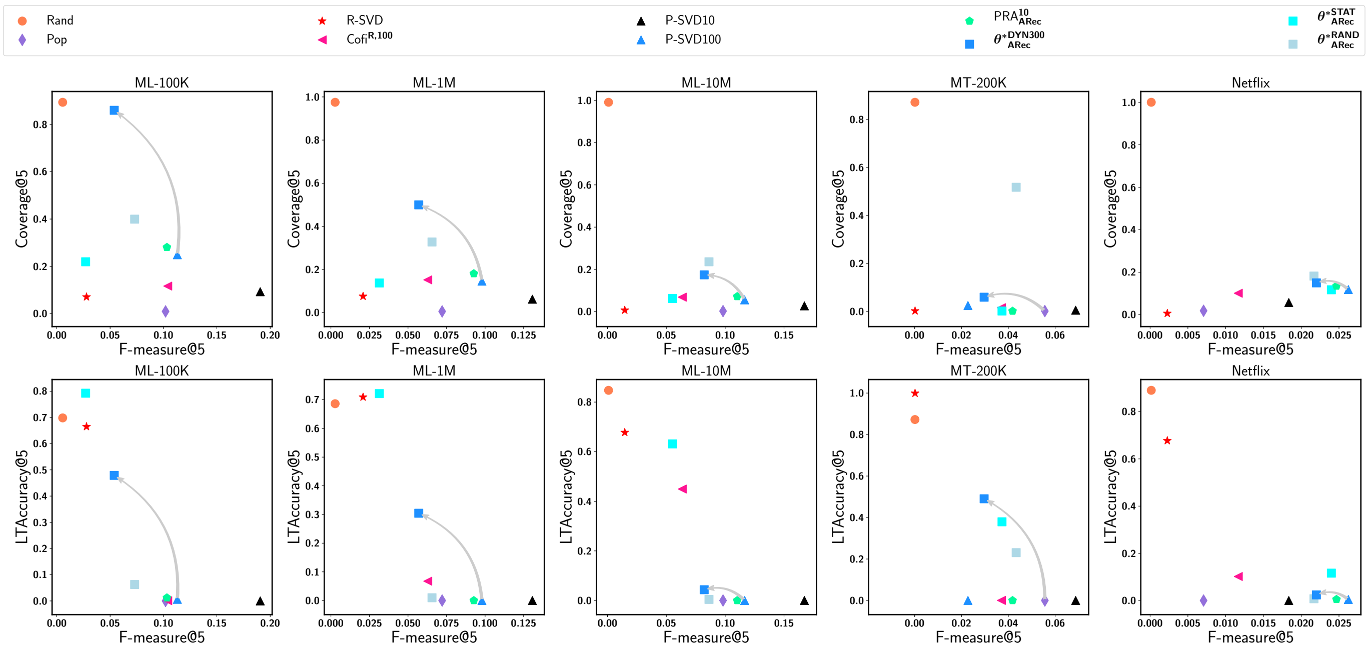
Accuracy-focused rating prediction CF models aim to accurately predict unobserved user ratings [47], [3]. Accuracy is measured using error metrics such as Root Mean Square Error or Mean Absolute Error that are defined w.r.t. the observed user feedback. Generally, these methods are grouped into memory-based or model-based [47]. Earlier memory-based or neighbourhood models used the rating data directly to compute similarities between users [48] or items [49]. However, these models are not scalable on large-scale datasets like Netflix and MovieLens 10M. Model-based methods instead train a parametric model on the user-item matrix. Matrix Factorization models [47], [40], a.k.a. Singular Value Decomposition (SVD) models, belong to this group and are known for both scalablility and accuracy [1]. These models factorize the user-item matrix into a product of two matrices; one maps the users ($\mathcal{U}$), and the other maps items ($\mathcal{I}$), into a latent factor space of dimension $g \ll \min(|\mathcal{U}|, |\mathcal{I}|)$. Here, each user u, and each item i, are represented by a factor vector, $\mathbf{p}_u \in \mathbb{R}^g$, and $\mathbf{q}_i \in \mathbb{R}^g$, respectively. Ratings are estimated using $\hat{r}_{ui} = \mathbf{q}_i^T \mathbf{p}_u$. Due to the large number of missing values in the user-item matrix, the regularized squared error on the observed ratings is minimized. The resulting objective function is non-convex, and iterative methods such as Stochastic Gradient Descent (SGD) or Alternating Least Squares (ALS) [40], [50], [51] can be used to find a local minimum. We use a Regularized SVD (R-SVD) model and the same model with Non-negative constraints (R-SVDN) [40], [28], in Section IV.

Accuracy-focused ranking prediction CF models focus on accurately compiling ranked lists. The intuition is that since predicted rating values are not shown to the user, the focus should be on accurately selecting and ranking items. Accordingly, accuracy is measured using ranking metrics, like recall and precision, that can be measured either on the observed user feedback, or on *all* items [1], [36]. For ranking tasks, Most Popular (Pop) obtains high precision and recall [1], [5]. PureSVD (P-SVD) [1] and CofiRank (Cofi) [41] are well-known latent factor models for ranking, with others in [43], [35]. We use Pop, P-SVD, and Cofi in our experiments.

Multi-objective methods devise new models that optimize several objectives, like coverage and novelty, in addition to accuracy [5], [6], [13], [38], [52]. In [38], items are assumed to be similar if they significantly co-occur with the same items. This leads to better representations for long-tail items, and increases their chances of being recommended. A new performance measure that combines accuracy and popularity bias is proposed in [6]. This measure can be gradually tuned

	Alg.	F@5	S@5	L@5	C@5	G@5	Score
ML-100K	R-SVD	0.0279 (2)	0.0098 (4)	0.6649 (4)	0.0707 (8)	0.9886 (9)	5.4 (4)
	5D	0.0013 (9)	0.0014 (9)	**0.9260** (1)	0.2586 (3)	0.9000 (3)	5.0 (2)
	5DARR	0.0071 (8)	0.0035 (8)	0.7669 (2)	0.1171 (7)	0.9783 (8)	6.6 (6)
	RBTPOP$^{1}_{4.5}$	0.0136 (7)	0.0043 (7)	0.7362 (3)	0.1284 (6)	0.9759 (7)	6.0 (5)
	RBTAVG$^{1}_{4.5}$	0.0201 (6)	0.0075 (6)	0.6271 (5)	0.1938 (4)	0.9583 (5)	5.2 (3)
	PRA$^{10}_{R-SVD}$	0.0252 (4)	0.0103 (3)	0.5642 (7)	0.1171 (7)	0.9674 (6)	5.4 (4)
	PRA$^{20}_{R-SVD}$	0.0255 (3)	0.0095 (5)	0.5544 (8)	0.1379 (5)	0.9581 (4)	5.0 (2)
	$\theta^{*DYN300}_{R-SVD}$	0.0243 (5)	0.0106 (2)	0.5751 (6)	**0.7925** (1)	**0.7200** (1)	3.0 (1)
	$\theta^{T\,DYN300}_{R-SVD}$	**0.0314** (1)	**0.0134** (1)	0.4969 (9)	0.5262 (2)	0.8208 (2)	3.0 (1)
ML-1M	R-SVD	0.0208 (2)	0.0050 (6)	0.7091 (3)	0.0758 (9)	0.9923 (9)	5.8 (8)
	5D	0.0008 (9)	0.0006 (9)	**0.9579** (1)	0.1927 (4)	0.9468 (3)	5.2 (5)
	5DARR	0.0167 (6)	0.0052 (5)	0.6649 (5)	0.1360 (6)	0.9752 (6)	5.6 (7)
	RBTPOP$^{1}_{4.5}$	0.0091 (8)	0.0022 (8)	0.8019 (2)	0.1125 (8)	0.9872 (8)	6.8 (9)
	RBTAVG$^{1}_{4.5}$	0.0155 (7)	0.0044 (7)	0.6816 (4)	0.2261 (3)	0.9704 (4)	5.0 (4)
	PRA$^{10}_{R-SVD}$	0.0207 (3)	0.0053 (4)	0.6268 (6)	0.1171 (7)	0.9800 (7)	5.4 (6)
	PRA$^{20}_{R-SVD}$	0.0205 (4)	0.0055 (3)	0.5976 (7)	0.1436 (5)	0.9714 (5)	4.8 (3)
	$\theta^{*DYN300}_{R-SVD}$	0.0195 (5)	0.0065 (2)	0.5254 (8)	**0.5205** (1)	**0.9054** (1)	3.4 (2)
	$\theta^{T\,DYN300}_{R-SVD}$	**0.0240** (1)	**0.0078** (1)	0.5009 (9)	0.4174 (2)	0.9133 (2)	3.0 (1)
ML-10M	R-SVD	**0.0147** (1)	**0.0021** (1)	0.6775 (5)	0.0066 (9)	0.9992 (9)	5.0 (3)
	5D	0.0000 (9)	0.0000 (8)	**1.0000** (1)	0.1248 (3)	**0.9609** (1)	4.4 (2)
	5DARR	0.0024 (8)	0.0007 (7)	0.9421 (2)	0.0489 (5)	0.9968 (5)	5.4 (4)
	RBTPOP$^{1}_{4.5}$	0.0086 (6)	0.0012 (6)	0.8062 (3)	0.0210 (6)	0.9973 (7)	5.6 (5)
	RBTAVG$^{1}_{4.5}$	0.0087 (5)	0.0013 (5)	0.8039 (4)	0.0614 (4)	0.9945 (4)	4.4 (2)
	PRA$^{10}_{R-SVD}$	0.0116 (2)	0.0020 (2)	0.5888 (9)	0.0085 (8)	0.9978 (8)	5.8 (6)
	PRA$^{20}_{R-SVD}$	0.0110 (3)	0.0020 (2)	0.5992 (7)	0.0115 (7)	0.9972 (6)	5.0 (3)
	$\theta^{*DYN300}_{R-SVD}$	0.0061 (7)	0.0014 (4)	0.5946 (8)	**0.1630** (1)	0.9930 (2)	4.4 (2)
	$\theta^{T\,DYN300}_{R-SVD}$	0.0094 (4)	0.0017 (3)	0.6013 (6)	0.1462 (2)	0.9931 (3)	3.6 (1)
MT-200K	R-SVD	0.0002 (6)	0.0004 (4)	0.9991 (2)	0.0029 (9)	0.9995 (9)	6.0 (7)
	5D	0.0000 (7)	0.0000 (5)	**0.9996** (1)	0.0597 (3)	**0.9789** (1)	3.4 (2)
	5DARR	0.0002 (6)	0.0005 (3)	0.9433 (5)	0.0206 (5)	0.9970 (6)	5.0 (4)
	RBTPOP$^{1}_{4.5}$	0.0002 (6)	0.0005 (3)	0.9988 (3)	0.0154 (6)	0.9968 (5)	4.6 (3)
	RBTAVG$^{1}_{4.5}$	0.0005 (3)	0.0008 (2)	0.9701 (4)	0.0273 (4)	0.9957 (4)	3.4 (2)
	PRA$^{10}_{R-SVD}$	0.0003 (5)	0.0008 (2)	0.9202 (6)	0.0058 (8)	0.9985 (8)	5.8 (6)
	PRA$^{20}_{R-SVD}$	0.0004 (4)	0.0008 (2)	0.8038 (7)	0.0082 (7)	0.9974 (7)	5.4 (5)
	$\theta^{*DYN300}_{R-SVD}$	**0.0010** (1)	**0.0010** (1)	0.7402 (9)	**0.1427** (1)	0.9852 (2)	2.8 (1)
	$\theta^{T\,DYN300}_{R-SVD}$	0.0009 (2)	0.0008 (2)	0.7913 (8)	0.1405 (2)	0.9859 (3)	3.4 (2)
Netflix	R-SVD	**0.0023** (1)	0.0019 (2)	0.6772 (6)	0.0062 (8)	0.9997 (8)	5.0 (4)
	5D	0.0000 (8)	0.0001 (7)	**0.9968** (1)	**0.3523** (1)	**0.9463** (1)	3.6 (1)
	5DARR	0.0012 (5)	0.0011 (5)	0.7862 (4)	0.1854 (2)	0.9928 (2)	3.6 (1)
	RBTPOP$^{1}_{4.5}$	0.0010 (7)	0.0010 (6)	0.8199 (2)	0.0044 (9)	0.9991 (7)	6.2 (6)
	RBTAVG$^{1}_{4.5}$	0.0011 (6)	0.0010 (6)	0.8054 (3)	0.0290 (5)	0.9978 (5)	5.0 (4)
	PRA$^{10}_{R-SVD}$	0.0020 (2)	0.0017 (3)	0.6697 (8)	0.0115 (7)	0.9991 (7)	5.4 (5)
	PRA$^{20}_{R-SVD}$	0.0018 (3)	0.0017 (3)	0.6722 (7)	0.0158 (6)	0.9987 (6)	5.0 (4)
	$\theta^{*DYN300}_{R-SVD}$	0.0013 (4)	0.0015 (4)	0.7176 (5)	0.0995 (3)	0.9966 (3)	3.8 (2)
	$\theta^{T\,DYN300}_{R-SVD}$	0.0020 (2)	**0.0020** (1)	0.6168 (9)	0.0692 (4)	0.9975 (4)	4.0 (3)

TABLE IV: Top-5 recommendation performance for re-ranking a rating prediction model, R-SVD. The metrics are (F)measure@5, (S)tratified Recall@5, (L)TAccuracy@5, (C)overage@5, and (G)ini@5. Bolded entries show the best value for each metric, with relative rank of each algorithm on each metric inside parenthesis. For all models, improving trade-offs is better on dense datasets. Regarding our two models $\theta^{*DYN300}_{R-SVD}$ and $\theta^{T\,DYN300}_{R-SVD}$, in dense settings (ML-1M and ML-100K), our methods outperform others in all metrics except LTAccuracy. On all datasets, our models obtain the lowest average rank across all metrics (last column). Overall, the results suggest a different accuracy recommender should be used in sparse settings.



(a)

Fig. 6: Accuracy vs Coverage vs Novelty. For our main algorithm $\theta^{*DYN300}_{ARec}$, we use arrows. The head of the arrow shows our model, while the bottom shows the underlying accuracy recommender (ARec). On MT-200K, ARec is P-SVD100. On MT-200K ARec is Pop. Note, the R-SVD model, is consistantly dominated by all other models in F-measure and coverage.

towards recommending long-tail items. More recently, the idea of recommending users to items as a means of improving sales diversity, and implicitly, recommendation novelty, while retaining precision, has been explored in [5]. In comparison to both [6], [38], we focus on targeted promotion of long-tail items. While [5] focuses on neighbourhood models for top-N recommendation, our framework is generic and independent of the recommender models. Graph-based approaches for long-tail item promotion are studied in [13], [52]. They construct a bipartite user-item graph and use a random walk to trade-off between popular and long-tail items. Rather than devise new multi-objective models, we post-process existing models.

Re-ranking methods post-process the output of a standard model to account for additional objectives like coverage and diversity rather than devising a new model. These algorithms are very efficient. [9] explores re-ranking to maximize diversity within individual top-N. It shows that users preferred diversified lists despite their lower accuracy. However, diversifying individual top-N sets does not necessarily increase coverage [17], [38]. Re-ranking techniques that directly maximize coverage and promote long-tail items are explored in [11], [14], [20], [22]. In contrast to [11], [14], [20] that re-rank rating prediction models, our framework is generic and is independent of the base recommendation model. Furthermore, our long-tail personalization is independently learned from interaction data. PRA [22] is also generic framework for re-ranking, although we differ in our long-tail novelty prefernce modelling. We use RBT [14], Resource allocation [20], PRA [22] as baselines since we share similar objectives.

Modelling user novelty preference is studied in [17], [22], [53], [54], [55]. As explained in [53], an item can be novel in three ways: 1) it is new to the system and is unlikely to be seen by most users (cold-start), 2) it existed in the system but is new to a single user, 3) it existed in the system, was previously known by the user, but is a forgotten item. [53], [55] focus on definitions 2 and 3 of novelty, which are useful in settings where seen items can be recommended again, e.g., music recommendation. For defining users' novelty preferences, tag and temporal data are used in [53], [55]. A logistic regression model is used to predict user novelty preferences in [53], while [55] learn a curiosity model for each user based on her access history and item tags. In [54], gross movie earnings are assumed to reflect movie popularity, and are used to define a personal popularity tendency (PPT) for each user. The idea is to match the PPT of each user with the PPT of recommended items [54]. The major difference between our work and [53], [54], [55] is that we focus on the cold-start definition of novelty (definition 1), we do not use contextual or temporal information to define users' tendencies for novelty, and we consider domains where each item is accessed at most once (seen items cannot be recommended).

In [17], users are characterized based on their tendency towards disputed items, defined as items with high average rating and high variance. These items are claimed to be in the long-tail. We differ in terms of long-tail novelty definition, and consequently our preference estimates. Furthermore, [17] independently solve a constrained convex optimization problem for each user, with the user's disputed item tendency as a constraint. [18], [19] use a user risk indicator to decide between a personalized and a non-personalized model; both focus on accuracy. In contrast, we combine accuracy and coverage models. Furthermore, while their risk indicators are optimized via cross validation, we learn the users' long-tail preferences.

PRA [22] models user preference for long-tail novelty using on item popularity statistics. In contrast, we consider additional information, like if the user found the item interesting, and the long-tail preferences of other users of the item.

CF interaction data properties and test ranking protocol are two important aspects to consider in recommendation setting. CF interaction data suffers from popularity bias [37]. In the movie rating domain, for instance, users are more likely to rate movies they know and like [1], [6], [36], [37], [46]. As a result, the partially observed interaction data is not a random subset of the (unavailable) complete interaction data. Furthermore, many real-world CF interaction datasets [33], [4], [21] are sparse, and the majority of items and users have few observations available. Due to the popularity bias and sparsity of datasets, many accuracy-focused CF models are also biased toward recommending popular items.

Moreover, some accuracy evaluation protocols are also biased and tend to reward popularity biased algorithms [1], [5], [36]. In [36], the main evaluation protocols are assessed in detail, and the "All unrated items test ranking protocol" is described to be closer to the accuracy the user experience in real-world recommendation setting, where performance is measured using the complete data rather than available data [36]. Following [36], [5], and w.r.t. the additional experiments we conducted in Appendix C , we chose the "All unrated items test ranking protocol" for experiments in Section IV.

VII. Conclusion

This paper presents a recommendation framework for bringing long-tail items to the attention of the right group of users. To achieve this, we profile the users according to their willingness to explore novel long-tail items. We learn the user long-tail novelty preferences in an optimization framework. We then integrate the learned preference in a generic framework, that targets both accuracy and coverage. Extensive experiments on several datasets confirm that there are trade-offs between accuracy, coverage, and novelty. Almost all re-ranking models increase coverage and novelty at the cost of accuracy. However, existing re-ranking models typically rely on rating prediction models, and are hence more effective on dense settings. Using a generic approach, we are able to easily incorporate a suitable base accuracy recommender to devise an effective solution for both sparse and dense settings. Although we integrated the long-tail novelty preference estimates into a re-ranking framework, their use-case is not limited to these frameworks. We also intend to explore the temporal and topical dynamics of long-tail novelty preference, particularly in settings where contextual information is available.

References

[1] P. Cremonesi, Y. Koren, and R. Turrin, "Performance of recommender algorithms on top-n recommendation tasks," in *RecSys*, 2010.

[2] J. Herlocker, J. A. Konstan, and J. Riedl, "An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms," *Information retrieval*, vol. 5, no. 4, 2002.

[3] J. Lee, M. Sun, and G. Lebanon, "A comparative study of collaborative filtering algorithms," *arXiv preprint arXiv:1205.3193*, 2012.

[4] B. Kanagal, A. Ahmed, S. Pandey, V. Josifovski, J. Yuan, and L. Garcia-Pueyo, "Supercharging recommender systems using taxonomies for learning user purchase behavior," *VLDB*, 2012.

[5] S. Vargas and P. Castells, "Improving sales diversity by recommending users to items," in *RecSys*, 2014.

[6] H. Steck, "Item popularity and recommendation accuracy," in *RecSys*, 2011.

[7] P. Castells, N. J. Hurley, and S. Vargas, *Recommender Systems Handbook*. Springer US, 2015, ch. Novelty and Diversity in Recommender Systems.

[8] M. Zhang and N. Hurley, "Avoiding monotony: improving the diversity of recommendation lists," in *RecSys*, 2008.

[9] C.-N. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen, "Improving recommendation lists through topic diversification," in *WWW*, 2005.

[10] Y. C. Zhang, D. Ó. Séaghdha, D. Quercia, and T. Jambor, "Auralist: introducing serendipity into music recommendation," in *WSDM*, 2012.

[11] G. Adomavicius and Y. Kwon, "Maximizing aggregate recommendation diversity: A graph-theoretic approach," in *International Workshop on Novelty and Diversity in Recommender Systems (DiveRS 2011)*, 2011.

[12] C. Anderson, *The Long Tail: Why the Future of Business Is Selling Less of More*. Hyperion, 2006.

[13] H. Yin, B. Cui, J. Li, J. Yao, and C. Chen, "Challenging the long tail recommendation," *PVLDB*, vol. 5, no. 9, pp. 896–907, 2012.

[14] G. Adomavicius and Y. Kwon, "Improving aggregate recommendation diversity using ranking-based techniques," *TKDE*, 2012.

[15] M. Ge, C. Delgado-Battenfeld, and D. Jannach, "Beyond accuracy: evaluating recommender systems by coverage and serendipity," in *RecSys*, 2010.

[16] M. Kaminskas and D. Bridge, "Diversity, serendipity, novelty, and coverage: A survey and empirical analysis of beyond-accuracy objectives in recommender systems," *ACM Trans. Interact. Intell. Syst.*, 2016.

[17] T. Jambor and J. Wang, "Optimizing multiple objectives in collaborative filtering," in *RecSys*, 2010.

[18] W. Zhang, J. Wang, B. Chen, and X. Zhao, "To personalize or not: a risk management perspective," in *RecSys*, 2013, pp. 229–236.

[19] J. Wang and J. Zhu, "Portfolio theory of information retrieval," in *SIGIR*, 2009, pp. 115–122.

[20] Y.-C. Ho, Y.-T. Chiang, and J. Y.-J. Hsu, "Who likes it more?: mining worth-recommending items from long tails by modeling relative preference," in *WSDM*, 2014, pp. 253–262.

[21] Y. Liu, T.-A. N. Pham, G. Cong, and Q. Yuan, "An experimental evaluation of point-of-interest recommendation in location-based social networks," *Proceedings of the VLDB Endowment*, 2017.

[22] M. Jugovac, D. Jannach, and L. Lerche, "Efficient optimization of multiple recommendation quality factors according to individual user tendencies," *Expert Systems with Applications*, vol. 81, 2017.

[23] D. Agarwal and B.-C. Chen, "Regression-based latent factor models," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2009, pp. 19–28.

[24] M. Saveski and A. Mantrach, "Item cold-start recommendations: Learning local collective embeddings," in *RecSys*, 2014, pp. 89–96.

[25] G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," *Information processing & management*, 1988.

[26] Y. Li, Q. Li, J. Gao, L. Su, B. Zhao, W. Fan, and J. Han, "On the discovery of evolving truth," in *KDD*. ACM, 2015.

[27] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong, "Diversifying search results," in *WSDM*, 2009.

[28] Y. Zhuang, W.-S. Chin, Y.-C. Juan, and C.-J. Lin, "A fast parallel sgd for matrix factorization in shared memory systems," in *RecSys*, 2013.

[29] S. Khuller, A. Moss, and J. S. Naor, "The budgeted maximum coverage problem," *Information Processing Letters*, vol. 70, no. 1, 1999.

[30] M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey, *An analysis of approximations for maximizing submodular set functions-II*. Springer, 1978.

[31] S. J. Sheather and M. C. Jones, "A reliable data-based bandwidth selection method for kernel density estimation," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 683–690, 1991.
[32] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *ACM Trans. on Interactive Intelligent Systems (TiiS)*, 2016.
[33] S. Dooms, T. De Pessemier, and L. Martens, "Movietweetings: a movie rating dataset collected from twitter," in *CrowdRec at RecSys*, 2013.
[34] J. M. Hernandez-lobato, N. Houlsby, and Z. Ghahramani, "Probabilistic matrix factorization with non-random missing data," in *ICML*, 2014.
[35] J. Lee, S. Bengio, S. Kim, G. Lebanon, and Y. Singer, "Local collaborative ranking," in *WWW*, 2014, pp. 85–96.
[36] H. Steck, "Evaluation of recommendations: rating-prediction and ranking," in *Proceedings of the 7th ACM conference on Recommender systems*. ACM, 2013, pp. 213–220.
[37] D. K. Agarwal and B.-C. Chen, *Statistical Methods for Recommender Systems*. Cambridge University Press, 2016.
[38] K. Niemann and M. Wolpers, "A new collaborative filtering approach for increasing the aggregate diversity of recommender systems," in *SIGKDD*, 2013, pp. 955–963.
[39] M. O. Lorenz, "Methods of measuring the concentration of wealth," *Publications of the American Statistical Association*, 1905.
[40] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, no. 8, pp. 30–37, 2009.
[41] M. Weimer, A. Karatzoglou, Q. V. Le, and A. Smola, "Maximum margin matrix factorization for collaborative ranking," *Advances in neural information processing systems*, pp. 1–8, 2007.
[42] E. M. Voorhees, "Overview of trec 2001," in *Trec*, 2001.
[43] S. Balakrishnan and S. Chopra, "Collaborative ranking," in *WSDM*, 2012.
[44] M. Volkovs and R. S. Zemel, "Collaborative ranking with 17 parameters," in *Advances in Neural Information Processing Systems*, 2012, pp. 2294–2302.
[45] B. M. Marlin, R. S. Zemel, S. Roweis, and M. Slaney, "Collaborative filtering and the missing at random assumption," in *UAI*, 2007.
[46] H. Steck, "Training and testing of recommender systems on data missing not at random," in *SIGKDD*, 2010.
[47] Y. Koren, "Factorization meets the neighborhood: a multifaceted collaborative filtering model," in *SIGKDD*, 2008.
[48] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl, "An algorithmic framework for performing collaborative filtering," in *SIGIR*, 1999.
[49] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *WWW*, 2001.
[50] A. Mnih and R. Salakhutdinov, "Probabilistic matrix factorization," in *Advances in neural information processing systems*, 2007.
[51] S. Rendle, "Factorization machines with libFM," *ACM Trans. on Intelligent Systems and Technology*, vol. 3, no. 3, pp. 1–22, May 2012.
[52] L. Shi, "Trading-off among accuracy, similarity, diversity, and long-tail: A graph-based recommendation approach," in *RecSys*, 2013, pp. 57–64.
[53] K. Kapoor, V. Kumar, L. Terveen, J. A. Konstan, and P. Schrater, "I like to explore sometimes: Adapting to dynamic user novelty preferences," in *Proceedings of the 9th ACM Conference on Recommender Systems*. ACM, 2015, pp. 19–26.
[54] J. Oh, S. Park, H. Yu, M. Song, and S.-T. Park, "Novel recommendation based on personal popularity tendency," in *Data Mining (ICDM), 2011 IEEE 11th International Conference on*. IEEE, 2011, pp. 507–516.
[55] P. Zhao and D. L. Lee, "How much novelty is relevant?: It depends on your curiosity," in *SIGIR*. ACM, 2016.
[56] A. Krause and D. Golovin, "Submodular function maximization."

Appendix

A. Regularized SVD configuration

Table V provides details for the setup of Regularized SVD (R-SVD) and the same model with non-negative constraints (R-SVDN). We use LIBMF [28] with L2-Norm as the loss function, and L2-regularization, and SGD for optimization. We performed 10-fold cross validation and tested $g \in \{10, 20, 40, 80, 100, 300\}$, $\lambda_Q \in \{0.01, 0.05, 0.1\}$, $\eta \in \{0.002, 0.003, 0.01, 0.03\}$.

B. Analysis of Fully Sequential Dynamic Coverage

In this section, we show the problem of finding a top-N collection $\mathcal{P} = \{\mathcal{P}_u\}_{u=1}^{|\mathcal{U}|}$ that maximizes Eq. III.2, is an instance of maximizing a submodular function subject to a matroid constraint.

	R-SVD				R-SVDN			
Dataset	η	λ	g	RMSE	η	λ	g	RMSE
ML-100K	0.03	0.05	100	0.935	0.03	0.05	100	0.935
ML-1M	0.03	0.05	100	0.868	0.03	0.05	100	0.875
ML-10M	0.003	0.005	20	0.872	0.003	0.005	20	0.872
MT-200k	0.01	0.01	40	0.761	0.01	0.01	40	0.761
Netflix	0.002	0.05	100	0.979	0.002	0.05	100	0.979

TABLE V: R-SVD and R-SVDN parameters on different datasets. g is the number of latent factors, η is the learning rate, $\lambda = \lambda_Q = \lambda_P$ is the L2-reqularization coefficients.

Submodularity and Monotonicity Let $\mathcal{I}$ denote a ground set of items. Given a set function $f : 2^{\mathcal{I}} \rightarrow \mathbb{R}$, $\delta(i|\mathcal{A}) := f(\mathcal{A} \cup \{i\}) - f(\mathcal{A})$ is the marginal gain of f at $\mathcal{A}$ with regard to item i. Furthermore, f is submodular if and only if $\delta(i|\mathcal{A}) \geq \delta(i|\mathcal{B}), \forall \mathcal{A} \subseteq \mathcal{B} \subseteq \mathcal{I}, \forall i \in \mathcal{I} \backslash \mathcal{B}$. It is modular if $f(\mathcal{A} \cup i) = f(\mathcal{A}) + f(i)$, $\forall \mathcal{A} \subset \mathcal{I}, i \in \mathcal{I} \setminus \mathcal{A}$. Furthermore, f is monotone increasing if $f(\mathcal{A}) \leq f(\mathcal{B}), \forall \mathcal{A} \subseteq \mathcal{B} \subseteq \mathcal{I}$. Equivalently, a function is monotone increasing if and only if $\forall \mathcal{A} \subseteq \mathcal{I}$ and $i \in \mathcal{I}$, $\delta(i|\mathcal{A}) \geq 0$ [56]. Submodular functions have the following concave composition property:

Theorem A.1. *Let $g : \mathbb{R} \rightarrow \mathbb{R}$ be continuous valued non-decreasing concave function, and $f : 2^{\mathcal{I}} \rightarrow \mathbb{R}$ a non-decreasing submodular set function. The composition $h = g \circ f : 2^{\mathcal{I}} \rightarrow \mathbb{R}$ is non-decreasing submodular set function.*

Theorem A.2. *The objective function $v(.)$ in Eq.III.2 is submodular monotone increasing across the users.*

Proof: Let $\mathcal{P} = \{\mathcal{P}_u\}_{u=1}^{|\mathcal{U}|}$ denote the collection of top-N sets assigned to all users. Let $\mathcal{A}$ and $\mathcal{B}$ denote partial collections, where only a subset of users have been assigned some items, with the property $\mathcal{A} \subset \mathcal{B} \subset \mathcal{P}$.

Next, consider the specific user u, and let $\mathcal{P}_u$ denote her current partial top-N set, where $|\mathcal{P}_u| < N$. Consider adding a new item $i \in \mathcal{I} \setminus \mathcal{P}_u$. For every such item, we have $f_i^{\mathcal{A}} \leq f_i^{\mathcal{B}}$, therefore $\frac{1}{1+f_i^{\mathcal{A}}} \geq \frac{1}{1+f_i^{\mathcal{B}}}$. Composing with $g(x) = \sqrt{x}$, we have $\frac{1}{\sqrt{1+f_i^{\mathcal{A}}}} \geq \frac{1}{\sqrt{1+f_i^{\mathcal{B}}}}$. Therefore the coverage function $c(.)$ is submodular. Furthermore, the accuracy function $a(.)$ is a modular function. Therefore, the overall value function $v(.)$ is submodular, since it is a linear combination of a modular accuracy function and submodular coverage function.

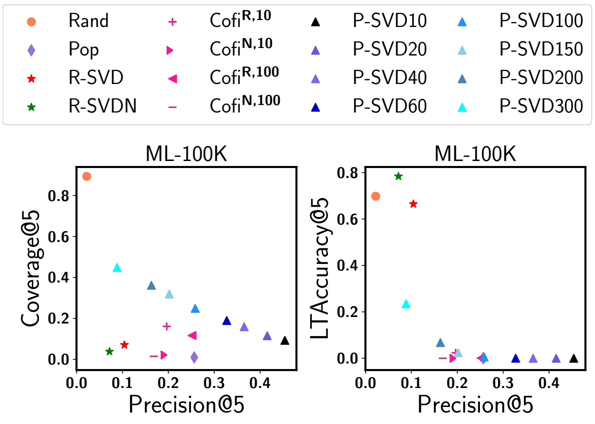
Furthermore, both $a(.)$ and $c(.)$ map a set of items $\mathcal{P}_u$ to the $[0, 1]$ range. Both functions are additive in terms of the number of items and hence monotonically increasing, i.e., adding a new element $i \in \mathcal{I} \setminus \mathcal{P}_u$ to the set $\mathcal{P}_u$ can only increase their value. Since θ_u is also in $[0, 1]$, $v_u(.)$ is monotonically increasing.

■

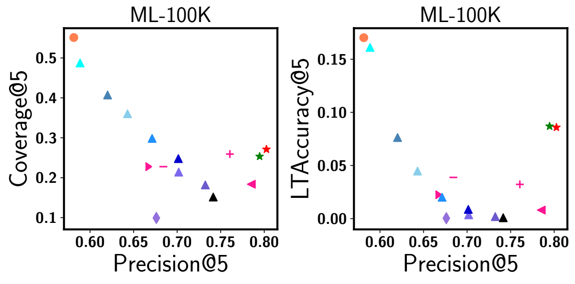
Matroids A set system is a pair $(\mathcal{I}, \mathcal{F})$, where $\mathcal{I}$ denotes a ground set of elements and $\mathcal{F} = \{\mathcal{A} : \mathcal{A} \subseteq \mathcal{I}\}$, is a collection of subsets of $\mathcal{I}$. A set system is an independence system if it satisfies 1) $\emptyset \in \mathcal{F}$, 2) $\mathcal{A} \subseteq \mathcal{B} \in \mathcal{F}$ then $\mathcal{A} \in \mathcal{F}$. A *matroid* is an independence system that also satisfies the property $\mathcal{A}, \mathcal{B} \in \mathcal{F}$ and $|\mathcal{A}| > |\mathcal{B}|$ then $\exists i \in \mathcal{A} \setminus \mathcal{B}$ with $\mathcal{B} \cup \{i\} \in \mathcal{F}$. A *uniform matroid*, is a special class of matroids that satisfies $\mathcal{F} = \{\mathcal{A} : \mathcal{A} \subseteq \mathcal{I}, |\mathcal{A}| \leq N\}$, that is all basis are maximal. A *partition matroid* satisfies $\mathcal{F} = \{\mathcal{A} : \mathcal{A} = \cup_{i=1}^{N} \mathcal{A}_i, \mathcal{A}_i \subseteq \mathcal{I}_i, |\mathcal{A}_i| \leq l_i, \cup \mathcal{I}_i = \mathcal{I}\}$.

Lemma A.1. *The constraint of recommending N items to each user, corresponds to a partition matroid over the users.*

Proof: Define a new ground set $\mathcal{N} = \{(u, i) : u \in \mathcal{U}, i \in \mathcal{I}\}$. Define $\mathcal{N}_u = \{(u, i) : i \in \mathcal{I}\}, u \in \mathcal{U}$ and let $l_u = N, \forall u \in \mathcal{U}$. Let $\mathcal{M} = (\mathcal{U}, \mathcal{F})$ where $\mathcal{F} = \{\mathcal{A} : \mathcal{A} = \cup_{u \in \mathcal{U}} \mathcal{A}_u, \mathcal{A}_u \subseteq \mathcal{N}_u, |\mathcal{A}_u| \leq l_u, \forall u \in \mathcal{U}\}$, i.e., $\mathcal{A}$ form independent sets of a partition matroid. ■

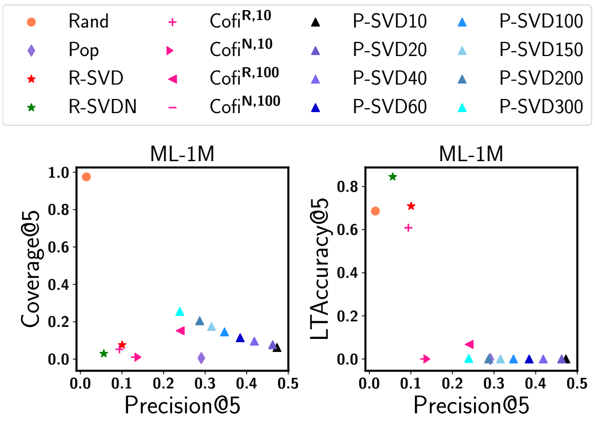


(a) All unrated items ranking protocol

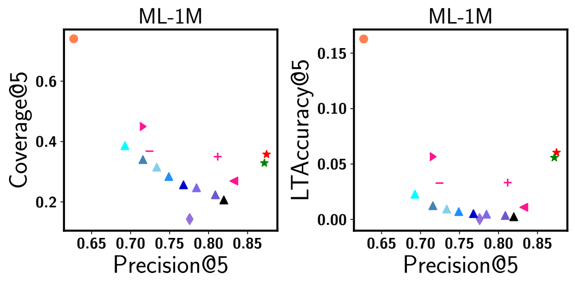


(b) Rated test-items ranking protocol

Fig. 8: Comparing the trade-offs for top-5 recommendation, when using different ranking protocols.



(a) All unrated items ranking protocol



(b) Rated test-items ranking protocol

Fig. 7: Comparing the trade-offs for top-5 recommendation, when using different ranking protocols.

C. *Effect of test ranking protocol on performance metrics*

In our empirical study, we studied off-line recommendation performance of top-N recommendation algorithms from three perspectives: accuracy, novelty, and coverage (Section IV-A). The choice of *test ranking protocol* [36] is also an important aspect in off-line evaluations. The test ranking protocol describes which items in the test set are ranked for each user [36]. We use the definitions in [36]:

- **Rated test-items ranking protocol**: for each user, we only rank the observed items in the test set of that user.
- **All unrated items ranking protocol**: for each user, we rank all items that do not appear in the train set of that user.

In [36], it was shown that the choice of test ranking protocol can affect accuracy measurements considerably. In particular, accuracy is measured either by error metrics (e.g., RMSE and MAE) or ranking metrics (e.g., recall and precision) in recommendation settings [36]. Error metrics are defined w.r.t the observed user feedback only. Rating prediction models that optimize these metrics in the train phase, e.g., R-SVD, typically adopt the rated test-items ranking protocol in the test phase. However, error metrics are not precise indicators of accuracy, since they only consider the subset of items that have feedback whereas in real-world applications, the system must find and rank a few items (N) from among *all* items [1], [36].

Ranking metrics can be measured on the observed user feedback, or on *all* items. However, if measured on the observed user feedback, these metrics can be strongly biased, due to the popularity bias of datasets [37], [1], [6], [5], [36]. Therefore, in top-N recommendation settings, these metrics are measured using the all-items ranking protocol, to better reflect accuracy as experienced by users in real-world applications [36], [5].

We extend the empirical study of [36] by evaluating the effect of the test ranking protocol on accuracy, coverage, and novelty. In our study, we used standard accuracy-focused CF models (introduced in Section VI). We set $N = 5$, and ran experiments on ML-100K and Ml-1M datasets, shown in Figures 8 and 7, respectively.

Figure 7 shows results for the ML-1M dataset. The first observation is that all algorithms obtain higher precision scores using the rated test-item ranking protocol. In particular, precision lies in $[0.0, 0.6]$ for the all unrated items ranking protocol, while it lies in $[0.6, 0.9]$ for the rated test-items ranking protocol.

As a specific example, consider Rand which randomly suggests items according to the ranking protocol. As expected, random suggestion from among all items results in low precision. However, random suggestion from among the test items of each user, results in an average precision of almost 0.6. This demonstrates the bias of the rated test-items ranking protocol. Our results also confirm the findings of [1], Pop is a strong contender in accuracy metrics, using both test protocols [1]. Recent work in [21] also confirmed that Pop outperformed more sophisticated algorithms for tourists datasets, that are sparse and where the users have few and irregular interests (only visit popular locations). In addition, although R-SVD and R-SVDN achieve low precision with the all unrated items ranking protocol, they obtain the highest precision scores using the rated test-items ranking protocol. Since these models are optimized w.r.t. the observed user feedback, the the rated test-items ranking protocol which only includes observed user feedback, is to their advantage.

Regarding the effect of the ranking protocol on LTAccuracy, we observe that LTAccuracy lies in $[0, 1]$ for the all unrated test items ranking protocol, but drops to $[0, 0.2]$ for the rated test-items ranking protocol. By definition, long-tail items are those that receive fewer ratings, therefore, they receive fewer ratings in the test set, and since the observed test-items are ranked per user, LTAccuracy scores are generally lower.

Regarding the effect of ranking protocol on coverage, overall the rated test-items ranking protocol results in better coverage for all algorithms except Random. This is because random suggestion from among a user's test items, is a more constrained algorithm compared to random from among all train items.

Regarding the trade-off between metrics, irrespective of the ranking protocol, we observe that Pop makes precise yet trivial recommendations that lack novelty, as indicated by the low long tail accuracy and coverage in Figure 7. For the PureSVD models (P-SVD), increasing the number of factors results, decreases precision, and results in an increase in coverage and long-tail accuracy. Among all baselines , CofiRank with Regression loss (Cofi^R) has reasonable coverage, precision, and long-tail accuracy.

Overall, our experiments in this section confirm the findings of prior work [1], [36], [37]: due to the popularity bias of recommendation datasets, rank-based precision using the rated test-items ranking protocol is strongly biased, as demonstrated by the results of Pop (which takes advantage of the popularity bias). Using the rated test-items protocol, Pop achieves a precision of approximately 0.7 and 0.8, on ML-100K and ML-1M, respectively. It outperforms personalized models like CofiRank. However, using the all-items ranking protocol, the performance of these models aligns better with expectation. Following prior research on top-N recommendation [36], [5], we conducted the experiments in Section IV using the all-items ranking protocol.