Do Offline Metrics Predict Online Performance in Recommender Systems?

Karl Krauth*[1], Sarah Dean[1,2], Alex Zhao[1,2], Wenshuo Guo[1,2], Mihaela Curmei[1,2], Benjamin Recht[1], and Michael I. Jordan[1]

[1]University of California, Berkeley
[2]Equal contribution

November 8, 2020

Abstract

Recommender systems operate in an inherently dynamical setting. Past recommendations influence future behavior, including which data points are observed and how user preferences change. However, experimenting in production systems with real user dynamics is often infeasible, and existing simulation-based approaches have limited scale. As a result, many state-of-the-art algorithms are designed to solve supervised learning problems, and progress is judged only by offline metrics. In this work we investigate the extent to which offline metrics predict online performance by evaluating eleven recommenders across six controlled simulated environments. We observe that offline metrics are correlated with online performance over a range of environments. However, improvements in offline metrics lead to diminishing returns in online performance. Furthermore, we observe that the ranking of recommenders varies depending on the amount of initial offline data available. We study the impact of adding exploration strategies, and observe that their effectiveness, when compared to greedy recommendation, is highly dependent on the recommendation algorithm. We provide the environments and recommenders described in this paper as `RecLab`: an extensible ready-to-use simulation framework at this URL: `https://github.com/berkeley-reclab/RecLab`.

1 Introduction

Recommender systems operate in dynamical settings. The recommendations given during one round of user interaction will affect the observations used to make recommendations in the future. The sequential nature of the problem is complicated by the fact that a deployed recommender system contends with changing preferences, due to external causes or induced by the recommendations themselves. From well-known effects like popularity bias in item recommendations to contested phenomena like polarization and radicalization among users; myopic optimization of offline metrics can cause unintended consequences Dandekar et al. [2013], Abdollahpouri et al. [2017], Faddoul et al. [2020]. Foreseeing the effects of deployed recommender systems is a complex socio-technical problem, depending on human psychology and behavioral economics. But even the basic questions of reliability and reproducibility for recommendation algorithm design remain unanswered.

[*] karlk@berkeley.edu

In this paper, we focus on the evaluation of recommender performance in dynamical settings. Despite the fact that the recommender systems community is well aware of the challenges posed by dynamical interactions Kouki and Said [2019], most recommendation algorithms and preference models are primarily designed and evaluated in an offline setting Lee et al. [2016], Sedhain et al. [2015], Zheng et al. [2016], Wang et al. [2006], Rendle [2012], Steck [2019], Ning and Karypis [2011]. The typical offline-first evaluation methodology involves the following four steps:

1. **Dataset Creation -** An organization or research group creates a dataset by collecting user interactions with a set of items hosted on an internet platform. Two prominent examples are the Netflix Prize and MovieLens datasets.

2. **Offline Evaluation -** Algorithm developers use the datasets from step (1) to evaluate their recommender systems. The developers train their algorithm on a train split of the datasets, tune the algorithm's parameters on a validation split, and evaluate the algorithm on a test split using various offline metrics. Common metrics include root mean squared error (RMSE) and normalized discounted cumulative gain (nDCG).

3. **Comparison with Baselines -** The developers compare their results on offline metrics with other algorithm's results, either by running the other algorithms themselves or by referring to previously published work. If their algorithm compares favorably the developers may publish their work, or if they can, proceed to the next step.

4. **Online Evaluation -** The algorithm is deployed on a real platform and its performance is evaluated using online metrics, which are usually defined to capture some notion of utility. Examples of popular online metrics include click-through rate (CTR) and watch time.

Offline evaluations make sense given the difficulty of evaluating algorithms online. Most researchers do not have access to a platform on which to perform online evaluations, and even those that do may not be able to perform large-scale evaluations due to the engineering effort required or the potential for lost revenue. However, offline evaluation comes with its own set of challenges. The data collected in step (1) is influenced by the recommender that is deployed at the time, which often leads to selection bias Schnabel et al. [2016]. Dacrema et al. [2019] demonstrate that step (2) of this evaluation procedure is often performed incorrectly, leading to non-reproducible results due to practices like inconsistent dataset splitting. Furthermore, they show that the baselines in step (3) are often incorrectly tuned, leading to a false sense of progress, a finding corroborated by Rendle et al. [2019]. The difficulty of properly evaluating algorithms offline brings into question the relationship between steps (1) to (3) and step (4).

In this work, we study the relationship between offline and online evaluation in controlled simulation environments to see the impact of recommender feedback effects, user dynamics, exploration, and low data. By using large-scale simulations, our results isolate these effects devoid of any confounding factors. We evaluate eleven recommendation algorithms across six simulated environments.

The recommenders we evaluate encompass simple baselines, neighborhood-based models, kernel-based models, linear models, factorization models, and neural models. The simulated environments on which we evaluate represent a diverse set of scenarios including two settings implemented in prior work and one setting that is initialized using the MovieLens dataset Harper and Konstan [2015].

We first show that offline metrics can act as a good proxy for online performance by replicating the offline-first evaluation methodology in a controlled setting. We compute RMSE and nDCG on an offline dataset for each recommender, simulate the interactive process of recommendation, compute the average user ratings of recommended items, and then compare the offline and online

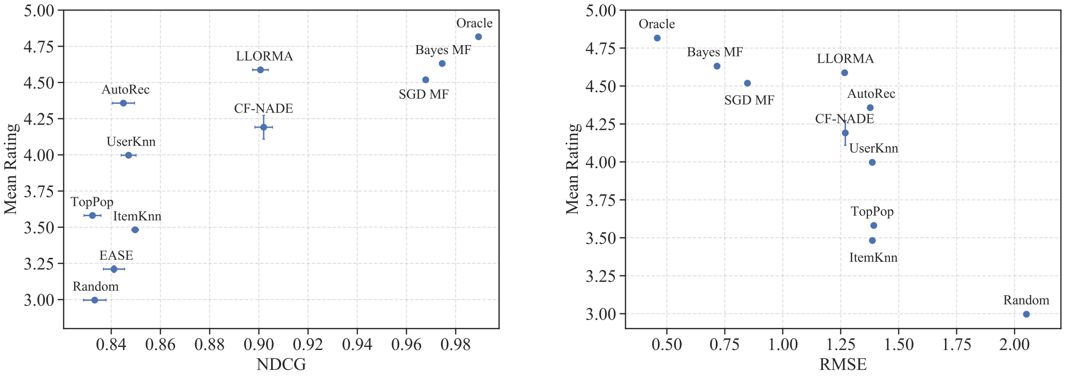

Figure 1: Left: The nDCG@20 plotted against the mean user ratings of all recommended items on the `topics-dynamic` environment. Right: The RMSE plotted against the mean user ratings. nDCG and RMSE are averaged across 5 folds on the offline dataset associated with the environment, user ratings are averaged across 10 trials. Each point represents a single model evaluation with error bars representing 95% confidence intervals.

metrics. Figure 1 shows such a comparison on the `topics-dynamic` environment. While there is a strong correlation between nDCG and mean user rating, we note that improvements in nDCG past a certain point suffer from diminishing improvement in mean user ratings. We examine these effects in more details in Section 5.

We then consider low-data regimes, in which recommenders do not have access to a large offline dataset (Section 6). In this case, augmenting existing recommenders with exploration techniques leads to better prediction accuracy over the *full population* of users and items. However, we show that the performance benefit of such augmentations, as measured by the average user rating of the recommended items, varies depending on the recommender algorithm and the underlying environment. Finally, we look at the relationship between a recommender's item coverage and online performance. We observe that the correlation between the two metrics depends on the environment, indicating that further investigation is necessary.

Taken together, our large-scale simulation results suggest that offline metrics can be a useful tool when online evaluation is not possible. However, they also bring into question the value of chasing small improvements in predictive performance, since those results might only lead to very small improvements in the online setting. This is especially true when data is plentiful and state-of-the-art recommendation algorithms can predict near-optimally. Instead, researchers should focus on a holistic evaluation of their algorithms; taking into account metrics that look beyond predictive accuracy and considering issues of measurement and sampling in the construction of the datasets they use.

Since there are many interesting research questions that can be studied through simulation, we open-source our simulation package. Our package is designed with large-scale evaluation in mind, and reproduces a number of popular and state-of-the-art recommenders. We also make available many environments, while making it easy to extend the package with new environments. It is our hope that this package can be integrated into recommender algorithm development and evaluations.

2 Related Work

Other simulations There has recently been increased interest in studying recommenders through simulation. Chaney et al. [2018] propose an environment in which users have limited knowledge of their utility, and show that recommendation algorithms homogenize user behavior within their simulations. Schmit and Riquelme [2018] evaluate the performance of ridge regression and matrix factorization in a two timestep simulated dynamical setting. Ie et al. [2019a] and Rohde et al. [2018] both propose simulation frameworks that are focused on evaluating reinforcement-learning based recommenders. Mansoury et al. [2020] and Jiang et al. [2019] use simulation to study the negative effects of feedback loops in recommender systems. Our work distinguishes itself from prior simulation studies by being the first to investigate a wide range of recommendation algorithms across a large number of environments. Furthermore, we are the first to investigate simulations at the scale of common benchmark datasets, while still running for many timesteps.

Online recommendation When in production, recommender systems interact iteratively with a changing environment where the set of users and items is not constant and user preferences evolve alongside the recommender. Several empirical studies on deployed recommender systems identify inconsistencies in online and offline performance Beel and Langer [2015], Mogenet et al. [2019], Rossetti et al. [2016], while others show how richer sets of offline metrics can be used to predict online performance Maksai et al. [2015]. Our work brings a systematic lens to this problem to understand more broadly the validity of practices around algorithm design and evaluation.

Many recommendation algorithms have been designed to address the difficulty of online recommendation in dynamical environments. Classically, time-aware models exploit the sequential nature of recommendation by incorporating temporal context Koren [2009], Vinagre et al. [2015], Campos et al. [2015]. More recently, a body of work treats online recommendation as a causal inference problem where the recommender model must de-bias logged training data Schnabel et al. [2016], Matuszyk et al. [2015], Saito et al. [2020], Sinha et al. [2016]. Lastly, others seek to improve recommender systems by directly addressing the online problem either through exploration strategies or reinforcement learning algorithms Li et al. [2010], Kawale et al. [2015], Ie et al. [2019b], Chen et al. [2019].

Alternative metrics Our work complements research on the societal effects of recommender systems, which considers alternative metrics including diversity, utility, serendipity and fairness [Adomavicius et al., 2013, Nguyen et al., 2014, Fleder and Hosanagar, 2009, Singh and Joachims, 2018, Yao and Huang, 2017]. Many authors have examined the limitations of accuracy as the sole metric for evaluating recommenders and have sought to define alternate metrics. Herlocker et al. [2004] proposed a variety of metrics for assessing a recommender's coverage, diversity, novelty, and serendipity, while Kaminskas and Bridge [2016] provide a more recent survey of the approaches for training recommenders with respect to these alternative metrics. Recently, Dean et al. [2020] introduced a measure of reachability which combines ideas of coverage with user agency in interactive systems.

3 Reproduced Recommenders

We evaluate eleven recommenders including baseline models, neighborhood models, factorization machine models, and several recent deep models. We choose to investigate recommenders mentioned by Rendle et al. [2019] as these models have all been run on the MovieLens10M dataset, giving us a starting point of comparison. The recommendation algorithms that we consider choose items to recommend based on a *relevance prediction*. Except when specified, recommendations are *greedy*, meaning that the item with the highest relevance prediction will be recommended. Therefore, the main difference between the following algorithms is their prediction component. All the models we reproduce only make use of ratings when making predictions, in future work we wish to also evaluated content-based and temporal models. We focus on the settings of rating prediction, where models are tuned so that relevance predictions match ratings (e.g. RMSE).

- `TopPop` - The TopPop algorithm recommends the most popular items to every user without personalization. The popularity of each item is measured by its average rating.
- `ItemKNN` - The ItemKNN algorithm is a collaborative filtering method using k-nearest neighborhood and item similarities Wang et al. [2006]. We implement ItemKNN in the same way as Hug [2017].
- `UserKnn` - The UserKnn algorithm is identical to ItemKnn, except that it uses user features instead of item features Wang et al. [2006].
- `Oracle` - The oracle recommender has access to the internals of the simulated environment, and will recommend the item with the highest true rating at each time step. Since this oracle baseline is still greedy, it does not plan for environment dynamics. Additionally, since actual ratings are generated with some noise, the RMSE of the oracle baseline is not zero.
- `Random` - This baseline predicts ratings uniformly at random.
- `SGD MF` - A factorization machine implemented in LibFM Rendle [2012]. The model is trained using SGD.
- `Bayes MF` - Another variant of factorization machines implemented in LibFM. The model is trained and the hyperparameters are automatically tuned using MCMC.
- `AutoRec` - An autoencoder framework for collaborative filtering Sedhain et al. [2015]. We train the algorithm with RMSProp and use the item-based version `I-AutoRec`. Our implementation makes use of the source code provided by the authors.[1]
- `CF-NADE` - A neural autoregressive architecture for collaborative filtering Zheng et al. [2016] trained with Adam. We adapt a publicly available implementation for our experiments.[2]
- `LLORMA` - LLORMA Lee et al. [2016] is a generalization of low rank matrix factorization techniques. LLORMA approximates the rating matrix as a weighted sum of low-rank matrices. We adapt a publicly available implementation for our experiments.[3]
- `EASE` - EASE Steck [2019] is a linear model designed for sparse data, especially implicit feedback data in recommenders. We do not include this recommender when computing RMSE as it outputs non-normalized relevance scores.

[1] `https://github.com/mesuvash/NNRec`

[2] `https://github.com/JoonyoungYi/CFNADE-keras`

[3] `https://github.com/JoonyoungYi/LLORMA-tensorflow`

4 The RecLab Simulated Environments

In this section we summarize the environments that we provide through our simulation framework. We consider both environments where users must consume the single item that is recommended to them and environments where users can choose from a slate of items. Unless mentioned otherwise we set our environments to have 1000 users and 1700 items, which is similar to the MovieLens 100K dataset. All the environment hyperparameters values we used are available in the experiments directory of the provided code.[4]

`topics-static` In the `topics-static` environment, each item is assigned to one of K topics and users prefer certain topics, this is similar to the simulation presented by Ie et al. [2019b]. The preference of user u for items i of topic k_i is initialized as $\pi(u, k_i) \sim Unif(0.5, 5.5)$, while the topic k of item i is chosen randomly from the set of all topics. When user u is recommended item i it will rate the item as $r_t(u, i) = \mathsf{clip}(\pi(u, k_i) + \epsilon)$ where $\epsilon \sim \mathcal{N}(0, \sigma^2)$ represents exogenous noise not modeled by the simulation, and *clip* truncates values to be between 1 and 5.

`topics-dynamic` In `topics-dynamic` items are rated in the same way as `topics-static`. In this setting however, user preferences can change as a result of the items they consume. Since past work has shown that users might become more interested in a topic through repeated exposure Ge et al. [2020], Mansoury et al. [2020], we incorporate this phenomenon into our model. If item i is recommended to user u then their preferences are updated as

$$\begin{aligned} \pi_{t+1}(u, k) &\leftarrow \mathsf{clip}(\pi_t(u, k) + a) \quad k = k_i, \\ \pi_{t+1}(u, k') &\leftarrow \mathsf{clip}\left(\pi_t(u, k') - \frac{a}{K-1}\right) \quad \forall k' \neq k_i, \end{aligned}$$

where a is a fixed affinity parameter. Another well-studied phenomenon is the fact that users get bored from being recommended the same topic in a short period of time Kapoor et al. [2015], Warlop et al. [2018]. We model this as:

$$\begin{aligned} r_t(u, i) = \mathsf{clip}(\pi_t(u, k_i) - \lambda \mathbf{1}\{&\text{topic } k_i \text{ observed} \geq \tau \text{ times within} \\ &m \text{ previous timesteps}\} + \epsilon). \end{aligned}$$

The effect of boredom arises from three parameters: memory length m, boredom threshold τ, and boredom penalty λ. If a user observes the same topic more than τ times within the last m timesteps then their ratings is penalized by λ.

`latent-static` In the `latent-static` environment, we represent users and items with d-dimensional latent feature vectors. This is a common assumption when developing factor models Koren [2008], Bell et al. [2007], Koren et al. [2009], and allows us to investigate a different user-item representation than the topics-based simulations. The rating of user u on item i is computed using these latent vectors as well as bias terms: $r(u, i) = \mathsf{clip}(\mu_0 + c_u + b_i + \mathbf{p}_u^\top \mathbf{q}_i + \epsilon)$, where $\mu_0 = 3$ is a global bias, $c_u \sim \mathcal{N}(0, 0.25)$ is the bias of user u, $b_i \sim \mathcal{N}(0, 0.25)$ is the bias of item i, $\mathbf{p}_u \sim \mathcal{N}(0, \sqrt{0.5/d})$ is the latent factor of user u, $\mathbf{q}_i \sim \mathcal{N}(0, \sqrt{0.5/d})$ is the latent factor of item i, and $\epsilon \sim \mathcal{N}(0, \sigma^2)$. `RecLab` also includes a `latent-dynamic` environment with a similar concept of boredom and affinity change as `topics-dynamic`. However, none of our experimental results use `latent-dynamic`.

[4] Experiment code for the paper can be downloaded at https://github.com/berkeley-reclab/RecLab

`ML-100K` The `ML-100K` environment is identical to the `latent-static` environment, except that the parameters are generated based on the MovieLens 100K (ML_100K) dataset Harper and Konstan [2015]. We train a LibFM factorization model on the ML_100K dataset, with hyperparameters tuned to achieve low RMSE through cross validation. We then extract the model's biases and latent factors to initialize the environment. We use `ML-100K` to confirm that our experiments generalize to situations where the simulator's parameters are initialized using real user interaction data.

`latent-score` The `latent-score` environment was first proposed in Section 5.2.3 of Schmit and Riquelme [2018]. The difference between `latent-score` and `latent-static` is that users have partial knowledge of an item's value. They use this partial information, along with the recommender's predicted score, to select an item from a slate of recommended items. We evaluate this environment with 170 users and 100 items due to computational limitations.

`beta-rank` This environment was introduced by Chaney et al. [2018]. It is similar to `latent-score`: users know part of the value for each item and users/items are represented by latent vectors. In `beta-rank` the rating of a user i on an item j is given by $r(u,i) \sim Beta(\mathbf{p}_u^\top \mathbf{q}_i, \sigma^2)$, where $\mathbf{p}_u$ is the latent vector for user i, $\mathbf{q}_i$ is the latent vector for item i, and the Beta distribution is parametrized according to its mean and variance. In this setting users chose from a slate of items based upon their observed utility and the recommender's ranking. We evaluate this environment with 170 users and 100 items due to computational limitations.

5 The Relationship Between Offline Metrics and Online Performance

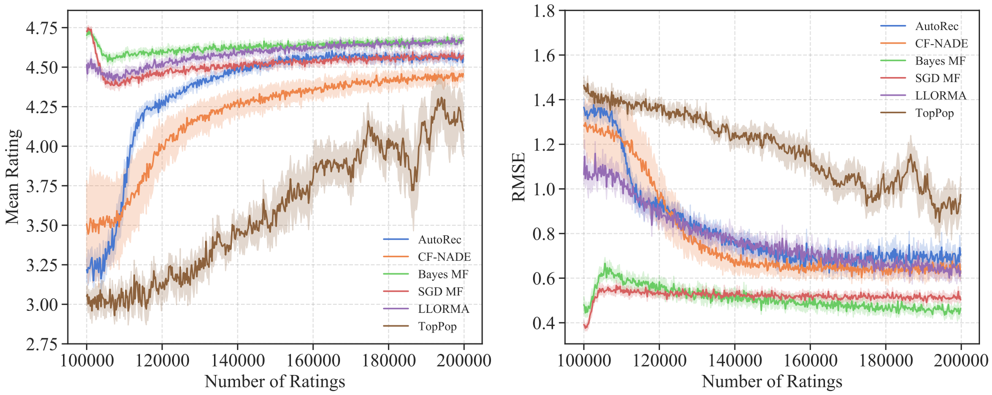

Figure 2: The performance of select recommenders over time on the `topics-dynamic` environment. The left plot shows the mean rating of the items that are recommended at each timestep. The right plot shows the RMSE between the predicted and true ratings of the recommended items at each timestep. Both plots are created by averaging over 10 experiment trials. The shaded areas represent 95% confidence intervals.

In this section we explore the relationship between offline metrics and online performance across all the environments described in Section 4. Given an environment, we evaluate each recommender by following these steps:

1. We create an offline dataset by sampling user-item pairs without replacement and get their ratings from the environment. We sample pairs uniformly, which removes the sampling bias introduced by data collected when a recommendation algorithm is already deployed. This allows us to focus on the effects of environment and recommender dynamics.

2. We tune the recommenders on the offline dataset using grid search on the hyperparameters. Our search aims to minimize the mean RMSE evaluated using 5-fold cross-validation. We use these hyperparameters throughout the experiment.

3. We begin by training each recommender using our offline dataset. At each timestep, online users are uniformly sampled from the set of all users. We recommend items to the sampled users and observe their ratings. Lastly, we retrain the recommenders on all the acquired data so far, and repeat until the end of the experiment. We repeat for 10 trials for every recommender on each environment.

This method of experimentation closely mimics the offline-first evaluation method mentioned in Section 1. Notice that we control for factors which may impact online performance, including the initial sampling distribution and the distribution of online users, so that our results illustrate exclusively the effect of environment and recommender dynamics. Furthermore, we consider simple online deployment, retraining from scratch whenever data is added and using the same hyperparameters throughout. While it is possible to improve these methods, for example by performing just a few gradient steps, we wish to avoid any confounding factors whose effect on performance aren't well understood.

Throughout this section, we consider greedy recommendation: we always recommend the item with the highest predicted relevance that hasn't already been recommended to each user .

5.1 Dynamic Environment

In this section we discuss results on `topics-dynamic`, an environment where user's preferences dynamically change over time. We sample 100k initial ratings on which to tune the recommenders, at every timestep we sample 200 users to recommend items to, and we run the simulation until we observe 200k ratings. As shown in Figure 1, nDCG@20[5] and RMSE are predictive of mean user rating. While it is well known that no offline metric can perfectly track online performance, the reason for these discrepancies has not been extensively studied.

We explain some of these discrepancies by exploring the full timeseries behavior of `topics-dynamic` for a select group of recommenders. The left plot in Figure 2 shows the average rating over the recommended items at each timestep, while the right plot shows the RMSE between the predicted and true ratings of the recommended items at each timestep. We emphasize that the RMSE shown in Figure 2 is only computed with respect to the recommended items at each timestep, and hence is not a measure of accuracy on the whole population of ratings. We are particularly interested in reasons why two recommenders might have: similar offline nDCG/RMSE but dissimilar mean user rating, similar mean user rating but dissimilar offline nDCG/RMSE, or dissimilar nDCG and RMSE. With these goals in mind we make the following observations:

- **Recommenders can affect user preferences in ways not captured by offline metrics.** `TopPop` improves its performance significantly as time progresses. There is a large difference in mean rating between it and `Random`, despite the fact that both recommenders have very close nDCG. As we discuss in Section 5.2, `TopPop`'s improvements are due to the environment's dynamics.

[5] We tried many different values for nDCG@k and got near identical behavior for all k.

- **The size of the initial dataset affects the offline ranking of recommenders.** `LLORMA` performs well starting at the first timestep. This seems to contradict its relatively low NDCG on the offline dataset. However, further investigation reveals that LLORMA's NDCG increases to 0.948 when trained with $100,000$ datapoints instead of the $80,000$ available during 5-fold cross-validation.

- **Recommender dynamics can affect their performance in ways not captured by offline metrics.** `AutoRec` performs much better than its offline nDCG and RMSE indicate. Unlike `LLORMA`, the initial performance of `AutoRec` is bad. At the first timestep, it barely performs better than `Random`. However, once it observes data that is sampled from its own recommendations, it is able to quickly match the performance of the best recommenders.

- **The same user dynamics can have a positive effect on one recommender, while simultaneously having a negative effect on another.** `CF-NADE` improves much more slowly than `LLORMA`, explaining the difference in rating between both algorithms despite the similar nDCG.[6] As we show in Section 5.2, this is primarily due to the negative effect of the user dynamics on `CF-NADE`.

- **A low RMSE is sufficient for selecting high-value items, but it is not necessary.** `SGD MF` and `Bayes MF` both perform on-par with `LLORMA`, despite the fact that `LLORMA` has a worse offline RMSE and per-timestep RMSE. This is because RMSE captures a recommender's ability to predict ratings, whereas mean user rating captures a recommender's ability to identify high-value items.

5.2 Static Environment

In this section, we investigate the performance of recommenders on the `topics-static` environment. We focus on comparing these results with those obtained on `topics-dynamic` to disentangle phenomena caused by the environment dynamics and those caused by the recommender dynamics. We initialized `topics-static` with the same user preferences and item topics as `topics-dynamic`. Furthermore, we ensure the offline dataset is the same for both environments. As a result each recommender's hyperparameters are the same as in the `topics-dynamic` setting.

Figure 3 compares the nDCG@20 of each recommender on the offline dataset with the average user rating across all timesteps. In this setting, nDCG is still positively correlated with mean rating, and while most of the observations made in Section 5.1 are still a concern even when there are no environment dynamics at play, we identify two notable differences. First, `TopPop` performs on par with `Random`. Without environment dynamics, the underlying preferences remain uniformly distributed, so popularity is not predictive. This is in contrast to `topics-dynamic` where the average preference for each topic is also initialized to be roughly 3, but by the end of the trial with `TopPop`, two of the topics have average affinities higher than 4.5. `TopPop` pushes user preferences toward certain topics, inducing item popularity effects in the data. Furthermore, `CF-NADE` performs significantly better, showing that the same user dynamics can have positive or negative effects on performance depending on the recommendation algorithm. This emphasizes the importance of evaluating general-purpose recommenders across a diverse range of datasets and environments. We observed similar results when comparing RMSE to mean rating in the `topics-static` environment.

[6] This observation also holds for `UserKnn` and `ItemKnn` although we do not show all these recommenders in Figure 2 to reduce clutter.

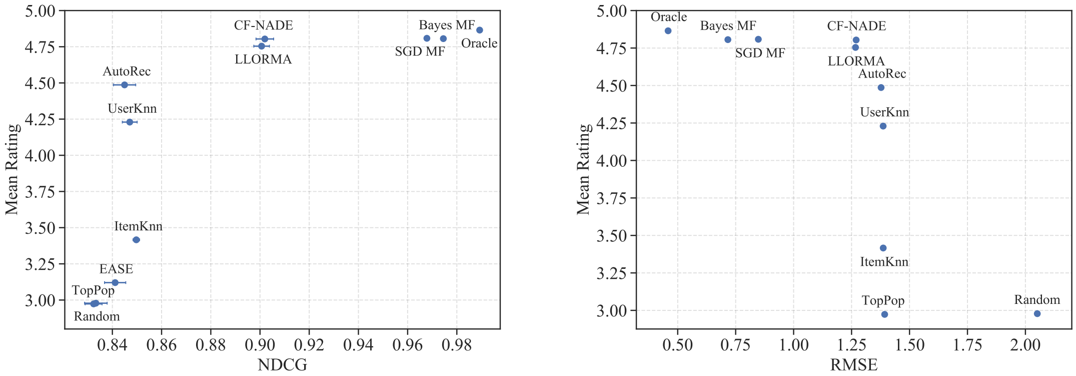

Figure 3: Left: The nDCG@20 plotted against the mean user ratings of all recommended items on the `topics-static` environment. Right: The RMSE plotted against the mean user ratings. NDCG and RMSE are averaged across 5 folds on the offline dataset associated with the environment, user ratings are averaged across 10 trials. Each point represents a single model with error bars representing 95% confidence intervals.

5.3 Other Environments

So far we have only demonstrated that RMSE and nDCG are predictive of online performance for two environments: `topics-static` and `topics-dynamic`. To ensure that this observation is robust, we benchmark the recommendation models across all environments from Section 4. The left plot of Figure 4 shows the Spearman rank correlations spe [2008] between the nDCG@20 and the mean rating, while the right plot shows the correlations between the RMSE and the mean rating. We see that both nDCG and RMSE are predictive of online performance across all environments. For `latent-score` and `beta-rank` we sample 1000 initial ratings and run the simulation until we have 2000 ratings. For all other environments, we sample 100k ratings and run the simulation until we have 200k ratings.

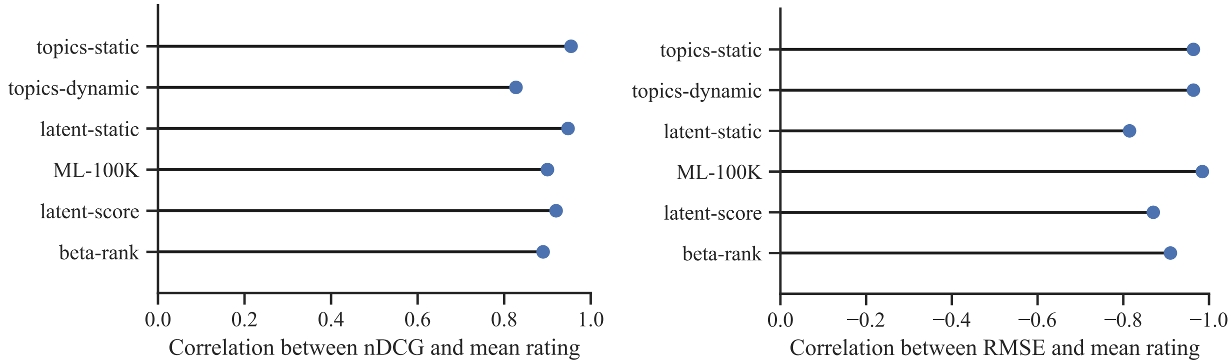

Figure 4: Left: The Spearman correlation between the nDCG@20 and the mean user ratings of all recommended items across all environments. Right: The Spearman correlation between the RMSE and the mean user ratings of all recommended items across all environments.

6 The Effects of Exploration

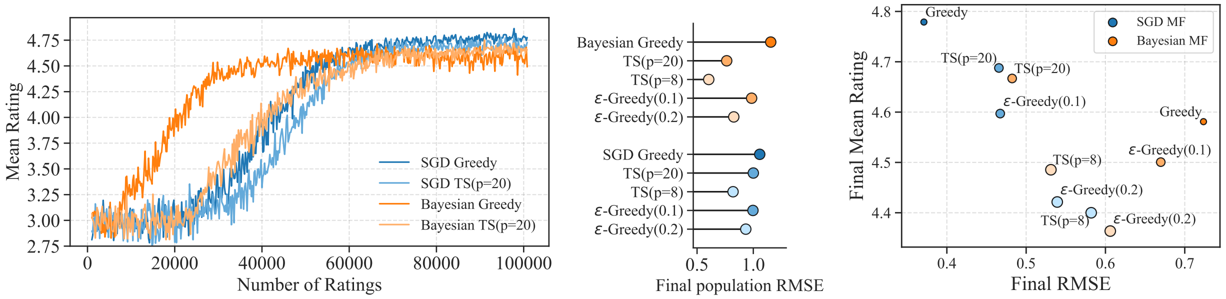

Figure 5: Exploration strategies on `topics-static-lowdata`. Left: The mean rating starts low and increases as the recommenders get more data. Middle: The population RMSE at the end of the experiment measures the overall identification. Right: The final RMSE and mean rating indicate the online performance. Final metrics are computed as the average of the 1,000 final ratings.

Previously we considered a controlled and high-data setting. We now consider the challenges posed by a low-data setting and whether these challenges can be mitigated by exploration. In particular, we investigate whether simple modifications to recommenders designed and tuned in offline settings can further improve their performance in online settings. The results so far consider the repeated application of a greedy selection rule. Common wisdom argues that a purely exploitative strategy will be suboptimal in a sequential decision making setting.

We therefore consider probabilistic selection rules, which have been suggested by the literature on bandit strategies in recommendation systems Kawale et al. [2015]. Prior work suggests that employing non-deterministic selection rules improves regret bounds over the horizon of recommendations Li et al. [2010]. We specifically consider two widely used non-deterministic selection strategies:

1. **ϵ-Greedy**: The probability of choosing item i for user u among a set of available items $\mathcal{I}_u$ is:

$$\mathbb{P}\{i\} = \begin{cases} 1-\epsilon & \text{if } i = \arg\max_{i\in\mathcal{I}_u} \hat{r}(u,i) \\ \frac{\epsilon}{|\mathcal{I}_u|} & \text{otherwise.} \end{cases}$$

2. **Thompson Sampling**: The probability of choosing an item is proportional to an increasing function $\phi(\cdot)$ of the predicted ratings:

$$\mathbb{P}\{i\} \sim \phi(\hat{r}(u,i)).$$

   We use the function $\phi(r) = r^p$, where p is a parameter which controls the spread of the sampling distribution.

To measure the effects of exploration, we focus on a low data setting, where recommenders initially have access to one rating per user on average. We consider the `topics-static-lowdata` environment, a version of `topics-static` in which 1,000 randomly selected ratings are initially revealed. We consider two of the highest performing algorithms: `SGD MF` and `Bayes MF`. The recommenders are augmented with the stochastic selection rules described above. We consider four settings of exploration: ϵ-Greedy with $\epsilon = 0.1$ and $\epsilon = 0.2$ and Thompson Sampling (TS) with $p = 20$ (less spread) and $p = 8$ (more spread). We follow the experimental procedure outlined in Section 5, with `num_trials=1`. Since the initial dataset is too small for tuning, each recommender uses the same hyperparameter tuning as for our `topics-static` experiments.

6.1 Performance Effects

We first consider the performance on `topics-static-lowdata`. The leftmost panel in Figure 5 plots the average rating of recommended items over time, demonstrating how the performance evolves for various recommendation strategies. It is immediately notable that greedy `Bayes MF` achieves a worse final performance than greedy `SGD MF`, despite its superior performance on full size environments and its more rapid initial increase. Additionally, while Thompson sampling improves the performance of `Bayes MF`, we do not see the same benefit for `SGD MF`. The rightmost panel in Figure 5 summarizes this finding: the vertical position of each point indicates the final mean rating, which is computed as the average of the 1,000 final ratings. Comparing the eight recommendation strategies, for `SGD MF`, exploration strategies with higher amounts of randomness achieve lower performance.

To further understand the quality of the recommenders, we examine the RMSE of the recommended items, as well as the *population* RMSE, which serves as a measure of overall prediction accuracy by considering every user and item pair in the system. The middle panel in Figure 5 shows the final population RMSE. Evaluated by this metric, all exploration strategies have a positive effect, with more randomness leading to a larger decrease in population RMSE. While useful for understanding the effects of exploration strategies, this measure is unrealistic, since deployed systems can only observe the ratings of recommended items. The right panel in Figure 5 plots the *observed* final RMSE along the horizontal axis, which is computed over the final 1,000 ratings. For `SGD MF`, greedy performs the best, while exploration strategies with less randomness perform better than those with more. For `Bayes MF`, the pattern is reversed.

These experiments lend insight to two important issues: the performance of algorithms in low data settings, and how properties of the underlying rating distribution influence the effect of biased sampling. Even though `Bayes MF` was the highest performing algorithm when it began with access to 100,000 ratings, its performance suffers in this low data setting. The difference is likely related to the biased distribution that arises from online recommendations. `Bayes MF` integrates hyperparameter tuning into its inference process, and therefore the regularization values are automatically determined. On the other hand, we do not re-tune hyperparameters for `SGD MF` online, which might lead to additional exploration and more robustness towards the bias in the online data. Furthermore, we ran an expanded set of exploration-based experiments on the `latent-static` environment. We found that the observed differences between results in `topics-static` and `latent-static` indicate that sampling biases affect performance in a way that depends on the underlying rating distribution. These observations highlight the fact that online deployment of recommender algorithms comes with setting-dependent caveats and challenges.

7 Discussion

Our experiments led to encouraging results regarding the way recommender systems are currently evaluated. In performing these evaluations, we surfaced observations that paint a more multifaceted view of a recommender's performance. In this section, we touch on three major areas we hope future algorithm developers will keep in mind when designing new recommender systems.

7.1 The Role of Simulation in Evaluation

Our work makes heavy use of simulation to investigate recommender systems. It is worthwhile to view such simulations with a critical eye, as they have clear limitations. We justify each design choice in our environments by referring to data-backed prior work, and we also run experiments on existing environments that are vetted by the research community. However, none of these environments are a perfect recreation of the real world, which would involve the immensely complex task of reproducing human behavior. Although this is a limitation of our work, there is significant value in studying algorithms and metrics in a simplified setting. The use of simulation to study simplified settings is more accessible and thus reproducible. It also lower the stakes, leading to faster iteration when designing new algorithms. Crucially, the ability to control for many factors allows for a more mechanistic understanding of observed phenomena. This is a widely accepted fact in theory-oriented work Chu et al. [2011], Wang et al. [2013], and is starting to be adopted within more empirical settings. For example, simulation has been widely accepted within the field of reinforcement learning as a tool to benchmark algorithms. These simulations vary from semi-realistic physical models Todorov et al. [2012] to video-games with little grounding in reality Mnih et al. [2013], Berner et al. [2019]. Furthermore, the fields of computer vision and natural language processing have also started to assess their widescale evaluation practices by experimenting on simplified tasks that act as a proxy to real-world tasks Recht et al. [2019], Miller et al. [2020].

A recommender system or metric performing well in simulation should not be interpreted as a carte blanche to claim such a system/metric would perform well in real-world settings. On the other hand, academic recommender systems are often developed as generalized rating prediction engines, with no specific platform in mind. Hence, a recommender system failing to perform well in simulation should be taken as strong negative evidence that such a system would fail to perform well in the real world and should probably be reworked; just as a supervised learning algorithm that fails to fit a simple toy problem would be seen with suspicion.

We hope that our simulation framework can be incorporated in the evaluation pipeline of recommender systems since simulations can catch many issues that would otherwise only surface when a recommender system is deployed. When combined with offline evaluation, simulation represents a powerful tool to identify good candidates for deployment. Furthermore, simulation is a useful lens through which to study recommendation algorithms. Our work uncovered many interesting recommender system phenomena that could not have been surfaced through offline evaluation only. It could not have been surfaced in real-world online evaluation either, since it would have been impossible to e.g. switch on and off the user dynamics while keeping everything else fixed. We believe that simulation studies can uncover many more such phenomena and lead to a more complete understanding of the complex dynamics in recommendations. Especially as we develop simulations that better encapsulate user behaviors, the simulated results provide a window into potential real-world scenarios. For example, Section 6 suggests a significant impact on a recommender's performance from biased sampling, however further work remains to be done to characterize this phenomenon in full generality.

7.2 Diminishing Returns

Our results provide strong evidence that offline metrics are predictive of online performance. However, this comes with a number of caveats. Increases in offline metrics lead to diminishing returns in terms of online performance, as shown in Figure 1. The hesitance of technology companies to adopt expressive yet computationally expensive models Amatriain and Basilico [2012] indicates that we may already operate in this regime of diminished returns for real-world recommendation tasks. Furthermore, recommendation tasks customarily involve an abundance of data with which most current algorithms might already be predicting near optimally. Instead, large accuracy gains in these high-data settings could be obtained through the measurement of new user and item features rather than algorithmic innovation.

This issue is compounded by another source of diminishing returns: as predictive models achieve

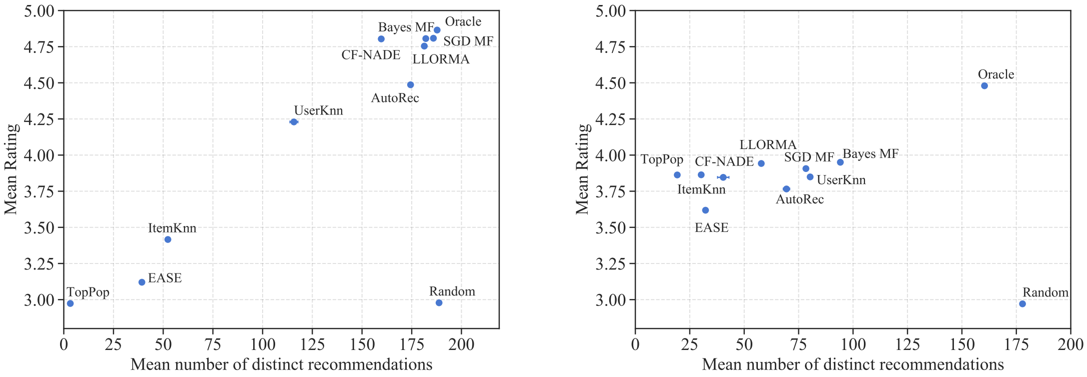

Figure 6: Left: Mean number of distinct items recommended at each timestep plotted against the mean user ratings of all recommended items on the `topics-static` environment. Right: Mean number of distinct items recommended at each timestep plotted against the mean user ratings of all recommended items on the `latent-static` environment.

higher-and-higher scores on offline benchmarks, their complexity grows exponentially Amodei and Hernandez [2018]. This is not a feasible solution for deployed recommenders where models must handle up to billions of users and items Covington et al. [2016]. Even our simulations, which are modest in size when compared to a production system, took many thousands of compute hours due to the computational complexity of state-of-the-art models. These issues indicate that further research effort would be better spent gaining deeper understanding of existing algorithms and datasets to guide the focus of algorithmic improvements.

7.3 Metrics Beyond Accuracy

Though our main motivation was to evaluate algorithmic performance for models trained to maximize accuracy, the data we gathered allows for an analysis of additional metrics. Here, we briefly describe some observations pertaining to alternate metrics to motivate future in depth analyses, Figure 6 shows that the relationship between coverage Kaminskas and Bridge [2016] and mean user rating is dependent on the environment.[7] The average number of distinct items within a timestep is positively correlated with mean ratings in the `topics-static` environment, however in `latent-static` coverage does not correlate with RMSE or mean ratings. We hypothesize that the low correlation in `latent-static` is due to the presence of item biases, making it easier for recommenders to learn items with high biases rather than exploring long-tail items. Future work may explore the effect of item biases and user behaviors on coverage and diversity.

We hope that these observations will motivate future work to consider how metrics such as fairness, coverage, novelty, and diversity can be used to design better recommendation systems. In data-rich settings where the performance gaps between recommenders are minimal, metrics beyond accuracy can guide decision-making about model choice and tuning strategies. Furthermore, varied metrics may be a useful proxy for accounting for real-world user behavior; while capturing the exact dynamics of evolving user preferences is a challenging task, ideas of novelty and diversity may help bridge the gap in bettering user experiences in the long run.

[7] We also experimented with the Gini coefficient, a measure of aggregate recomendation diversity, and observed the same results.

References

Spearman Rank Correlation Coefficient, pages 502–505. Springer New York, New York, NY, 2008. ISBN 978-0-387-32833-1. doi: 10.1007/978-0-387-32833-1_379. URL https://doi.org/10.1007/978-0-387-32833-1_379.

H. Abdollahpouri, R. Burke, and B. Mobasher. Controlling popularity bias in learning-to-rank recommendation. In *RecSys '17: Proceedings of the eleventh ACM conference on recommender systems*, pages 42–46, 2017.

G. Adomavicius, J. Bockstedt, S. Curley, and J. Zhang. Do recommender systems manipulate consumer preferences? a study of anchoring effects. *Information systems research*, 24(4):956–975, 2013.

X. Amatriain and J. Basilico. Netflix recommendations: Beyond the 5 stars, 2012. URL https://netflixtechblog.com/netflix-recommendations-beyond-the-5-stars-part-1-55838468f429.

D. Amodei and D. Hernandez. AI and compute, 2018. URL https://openai.com/blog/ai-and-compute/.

J. Beel and S. Langer. A comparison of offline evaluations, online evaluations, and user studies in the context of research-paper recommender systems. In *International conference on theory and practice of digital libraries*, pages 153–168. Springer, 2015.

R. Bell, Y. Koren, and C. Volinsky. Modeling relationships at multiple scales to improve accuracy of large recommender systems. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 95–104, 2007.

C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

P. Campos, F. Díez, and I. Cantador. Time-aware recommender systems: A comprehensive survey and analysis of existing evaluation protocols. *Wiley interdisciplinary reviews: Data mining and knowledge discovery*, 5(5):195–215, 2015.

A. J. Chaney, B. M. Stewart, and B. E. Engelhardt. How algorithmic confounding in recommendation systems increases homogeneity and decreases utility. In *Proceedings of the 12th ACM Conference on Recommender Systems*, pages 224–232, 2018.

M. Chen, A. Beutel, P. Covington, S. Jain, F. Belletti, and E. H. Chi. Top-k off-policy correction for a reinforce recommender system. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 456–464, 2019.

W. Chu, L. Li, L. Reyzin, and R. Schapire. Contextual bandits with linear payoff functions. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 208–214, 2011.

P. Covington, J. Adams, and E. Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, pages 191–198, 2016.

M. F. Dacrema, P. Cremonesi, and D. Jannach. Are we really making much progress? a worrying analysis of recent neural recommendation approaches. In *Proceedings of the 13th ACM Conference on Recommender Systems*, pages 101–109, 2019.

P. Dandekar, A. Goel, and D. Lee. Biased assimilation, homophily, and the dynamics of polarization. In *Proceedings of the National Academy of Sciences*, pages 5791–5796, 2013.

S. Dean, S. Rich, and B. Recht. Recommendations and user agency: the reachability of collaboratively-filtered information. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, pages 436–445, 2020.

M. Faddoul, G. Chaslot, and H. Farid. A longitudinal analysis of youtube's promotion of conspiracy videos. *arXiv preprint arXiv:2003.03318*, 2020.

D. Fleder and K. Hosanagar. Blockbuster culture's next rise or fall: The impact of recommender systems on sales diversity. *Management science*, 55(5):697–712, 2009.

Y. Ge, S. Zhao, H. Zhou, C. Pei, F. Sun, W. Ou, and Y. 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.

F. M. Harper and J. A. Konstan. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4):1–19, 2015.

J. Herlocker, J. Konstan, L. Terveen, and J. Riedl. Evaluating collaborative filtering recommender systems. *ACM transactions on information systems*, 22(1):5–53, 2004.

N. Hug. Surprise, a Python library for recommender systems. `http://surpriselib.com`, 2017.

E. Ie, C.-w. Hsu, M. Mladenov, V. Jain, S. Narvekar, J. Wang, R. Wu, and C. Boutilier. Recsim: A configurable simulation platform for recommender systems. *arXiv preprint arXiv:1909.04847*, 2019a.

E. Ie, V. Jain, J. Wang, S. Navrekar, R. Agarwal, R. Wu, H.-T. Cheng, M. Lustman, V. Gatto, P. Covington, et al. Reinforcement learning for slate-based recommender systems: A tractable decomposition and practical methodology. *arXiv preprint arXiv:1905.12767*, 2019b.

R. Jiang, S. Chiappa, T. Lattimore, A. György, and P. Kohli. Degenerate feedback loops in recommender systems. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, pages 383–390, 2019.

M. Kaminskas and D. Bridge. Diversity, serendipity, novelty, and coverage: A survey and empirical analysis of beyond-accuracy objectives in recommender systems. *ACM transactions on interactive intelligent systems*, 7(1), 2016.

K. Kapoor, K. Subbian, J. Srivastava, and P. Schrater. Just in time recommendations: Modeling the dynamics of boredom in activity streams. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, pages 233–242, 2015.

J. Kawale, H. H. Bui, B. Kveton, L. Tran-Thanh, and S. Chawla. Efficient thompson sampling for online matrix-factorization recommendation. In *Advances in neural information processing systems*, pages 1297–1305, 2015.

Y. 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*, pages 426–434, 2008.

Y. Koren. Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 447–456, 2009.

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

A. B. Kouki and A. Said. Offline and online evaluation of recommendations. In *Collaborative Recommendations: Algorithms, Practical Challenges and Applications*, pages 295–328. World Scientific Publishing Company, 2019.

J. Lee, S. Kim, G. Lebanon, Y. Singer, and S. Bengio. Llorma: Local low-rank matrix approximation. *The Journal of Machine Learning Research*, 17(1):442–465, 2016.

L. Li, W. Chu, J. Langford, and R. E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 661–670, 2010.

A. Maksai, F. Garcin, and B. Faltings. Predicting online performance of news recommender systems through richer evaluation metrics. In *Proceedings of the 9th ACM Conference on Recommender Systems*, pages 179–186, 2015.

M. Mansoury, H. Abdollahpouri, M. Pechenizkiy, B. Mobasher, and R. Burke. Feedback loop and bias amplification in recommender systems. *arXiv preprint arXiv:2007.13019*, 2020.

P. Matuszyk, J. Vinagre, M. Spiliopoulou, A. M. Jorge, and J. Gama. Forgetting methods for incremental matrix factorization in recommender systems. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 947–953, 2015.

J. Miller, K. Krauth, B. Recht, and L. Schmidt. The effect of natural distribution shift on question answering models. *arXiv preprint arXiv:2004.14444*, 2020.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

A. Mogenet, T. A. N. Pham, M. Kazama, and J. Kong. Predicting online performance of job recommender systems with offline evaluation. In *Proceedings of the 13th ACM Conference on Recommender Systems*, pages 477–480, 2019.

T. T. Nguyen, P.-M. Hui, F. M. Harper, L. Terveen, and J. A. Konstan. Exploring the filter bubble: the effect of using recommender systems on content diversity. In *Proceedings of the 23rd international conference on World wide web*, pages 677–686, 2014.

X. Ning and G. Karypis. Slim: Sparse linear methods for top-n recommender systems. In *2011 IEEE 11th International Conference on Data Mining*, pages 497–506. IEEE, 2011.

B. Recht, R. Roelofs, L. Schmidt, and V. Shankar. Do imagenet classifiers generalize to imagenet? *arXiv preprint arXiv:1902.10811*, 2019.

S. Rendle. Factorization machines with libFM. *ACM Trans. Intell. Syst. Technol.*, 3(3):57:1–57:22, May 2012. ISSN 2157-6904.

S. Rendle, L. Zhang, and Y. Koren. On the difficulty of evaluating baselines: A study on recommender systems. *arXiv preprint arXiv:1905.01395*, 2019.

D. Rohde, S. Bonner, T. Dunlop, F. Vasile, and A. Karatzoglou. Recogym: A reinforcement learning environment for the problem of product recommendation in online advertising. *arXiv preprint arXiv:1808.00720*, 2018.

M. Rossetti, F. Stella, and M. Zanker. Contrasting offline and online results when evaluating recommendation algorithms. In *Proceedings of the 10th ACM conference on recommender systems*, pages 31–34, 2016.

Y. Saito, S. Yaginuma, Y. Nishino, H. Sakata, and K. Nakata. Unbiased recommender learning from missing-not-at-random implicit feedback. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, pages 501–509, 2020.

S. Schmit and C. Riquelme. Human interaction with recommendation systems. In *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, pages 862–870, 2018.

T. Schnabel, A. Swaminathan, A. Singh, N. Chandak, and T. Joachims. Recommendations as treatments: debiasing learning and evaluation. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning-Volume 48*, pages 1670–1679, 2016.

S. Sedhain, A. K. Menon, S. Sanner, and L. Xie. Autorec: Autoencoders meet collaborative filtering. In *Proceedings of the 24th international conference on World Wide Web*, pages 111–112, 2015.

A. Singh and T. Joachims. Fairness of exposure in rankings. In *24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2219–2228, 2018.

A. Sinha, D. F. Gleich, and K. Ramani. Deconvolving feedback loops in recommender systems. In *Advances in neural information processing systems*, pages 3243–3251, 2016.

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

E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

J. Vinagre, A. M. Jorge, and J. Gama. An overview on the exploitation of time in collaborative filtering. *Wiley interdisciplinary reviews: Data mining and knowledge discovery*, 5(5):195–215, 2015.

J. Wang, A. P. De Vries, and M. J. Reinders. Unifying user-based and item-based collaborative filtering approaches by similarity fusion. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 501–508, 2006.

Y. Wang, L. Wang, Y. Li, D. He, and T.-Y. Liu. A theoretical analysis of ndcg type ranking measures. In *Conference on Learning Theory*, pages 25–54, 2013.

R. Warlop, A. Lazaric, and J. Mary. Fighting boredom in recommender systems with linear reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1757–1768, 2018.

S. Yao and B. Huang. Beyond parity: Fairness objectives for collaborative filtering. In *31st Conference on Neural Information Processing Systems*, 2017.

Y. Zheng, B. Tang, W. Ding, and H. Zhou. A neural autoregressive approach to collaborative filtering. *arXiv preprint arXiv:1605.09477*, 2016.

Appendix A The `RecLab` Framework

Figure 7 shows the process of evaluating a recommendation system within RecLab.

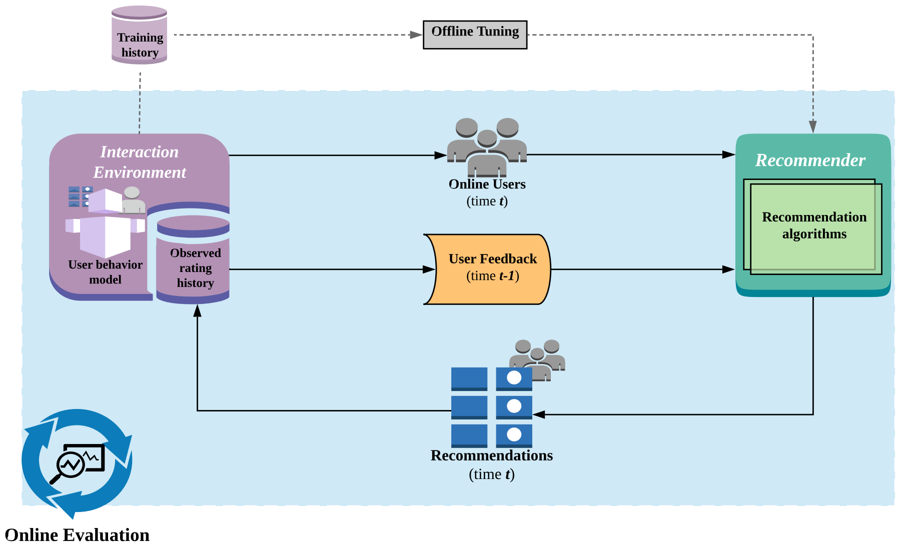

Figure 7: Illustration of the offline-first evaluation and online evaluation pipeline

Appendix B RMSE on `Topics-static`

Figure 2 shows the RMSE plotted against the mean rating of all recommended items on the `latent-static` environment.

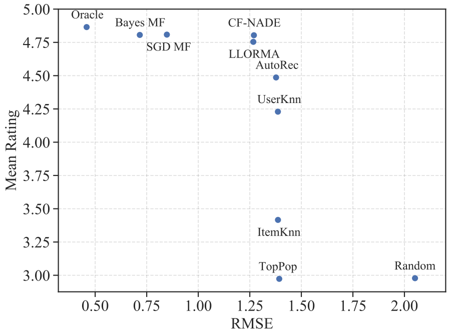

Figure 8: The RMSE plotted against the mean user ratings of all recommended items on the `topics-static` environment. RMSE is averaged across 5 folds on the offline dataset associated with the environment, user ratings are averaged across 10 trials. Each point represents a single model evaluation with error bars representing 95% confidence intervals.

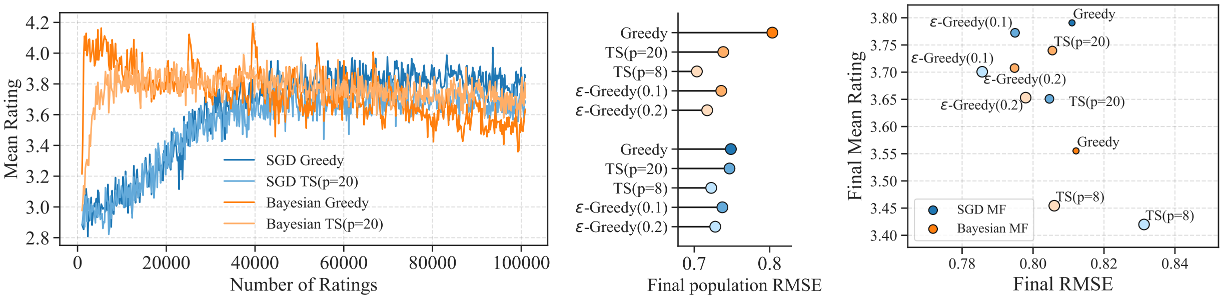

Figure 9: Exploration strategies on `latent-static-lowdata`. Left: The mean rating over time. Middle: The population RMSE at the end of the experiment measures the overall identification. Right: The final RMSE and mean rating indicate the online performance. Final metrics are computed as the average of the 1,000 final ratings.

Appendix C Exploration on `Latent-static`

Figure 9 shows the effect of exploration on `latent-static-lowdata`. The observations from the exploration experiments on `latent-static-topics` are generally corroborated by the results on `latent-static-lowdata`. However, there are a few key differences. First, notice that `Bayes MF` is able to immediately achieve high mean ratings followed by a decreasing performance. The `latent-static` environment generates ratings with an item bias term, in contrast to `topics-static`, which does not have an inherent quality or popularity structure among the items. We therefore hypothesize that `Bayes MF` is exploiting high popularity items at the expense of learning personalized preferences. We also see a difference in the relative performance of `SGD MF` strategies: the right panel in Figure 9 shows that greedy does not achieve the best final RMSE.