CPR: Collaborative Pairwise Ranking for Online List Recommendations

Saurabh Gupta
Amazon
Seattle, USA
gsaur@amazon.com

Bharathan Balaji
Amazon
Seattle, USA

Runfei Luo
Amazon
Seattle, USA

ABSTRACT

Classical approaches to recommendation systems like collaborative filtering learn a static model given the user historic interaction data. These approaches do not perform well in dynamic environments where the sets of users and items are continually changing. Users convey their preferences implicitly by providing feedback in the form of clicks, views and ratings, as they interact with the system. Utilizing this feedback in an online manner is crucial for building a good user experience. Contextual bandit algorithms provide a suitable framework for learning user preferences online by balancing the explore-exploit trade-off. Much of the bandit literature focuses on choosing one item, we extend these algorithms to recommend a list of actions by assuming a cascade click model. We provide an empirical study across different scenarios to showcase the benefits of collaborative online learning and exploration. Finally, we propose a novel algorithm – Collaborative Pairwise Ranking (CPR), that uses pairwise differentiable gradient descent to perform online ranking collaboratively. We showcase that this approach outperforms state-of-the-art collaborative bandit approaches, especially in the presence of noisy feedback common in practical scenarios.

CCS CONCEPTS

• **Information systems** → **Recommender systems**; **Personalization**; **Learning to rank**.

KEYWORDS

collaborative pairwise ranking, contextual bandits, online learning to rank, ranking, recommender systems

Reference Format:
Saurabh Gupta, Bharathan Balaji, and Runfei Luo. 2020. CPR: Collaborative Pairwise Ranking for Online List Recommendations. In *3rd Workshop on Online Recommender Systems and User Modeling (ORSUM 2020), in conjunction with the 14th ACM Conference on Recommender Systems, September 25th, 2020, Virtual Event, Brazil.*

1 INTRODUCTION

Recommendation systems are essential for modern online websites and mobile applications, as they are known to promote sales and service use substantially. Eighty percent of movies watched on Netflix came from recommendations [8], sixty percent of video clicks came from home page recommendation in YouTube [5]. Many recommender systems are formulated in a one-way fashion: given sufficient historical data, a supervised learning model (such as linear regression [7] or factorization machines [19]), is trained to capture the underlying preferences of users over items. In dynamic recommendation domains, such as news, ads and videos, active users and the set of items to recommend change frequently, hence classical collaborative filtering type methods [20], such as matrix factorization, break down [2].

Such dynamic scenarios can be modeled as interactive learning systems where the model can quickly adapt to user preferences on new content through interactions like clicks, views and purchases. However, interactive learning systems pose several challenges. *First*, since the system gets feedback only for the items that it recommends, it needs to decide how to balance the exploration and exploitation - whether to explore user preferences by recommending different items or to exploit what has been learned so far. *Second*, what type of predictive model should be used (e.g., linear, decision tree, neural networks, etc.). Some algorithms, such as LinUCB [15], constrain the predictive model to be linear. One also needs to decide whether the latent features based on interaction data of users and items should be learned alongside using the observable features. *Third*, in many practical cases a list of items is to be recommended. This becomes an online ranking problem, which brings its own challenges of learning the optimal ordering of items using a ranking loss.

Contextual bandits are a popular choice in interactive recommendation systems. Li et al. [15] designed the *LinUCB* algorithm to learn an item selection strategy based on user clicks to minimize the long term regret using the principle of optimism under uncertainty. *hLinUCB* [23] extended the algorithm to learn latent interaction features of users and items online. Online Learning to Rank (OLTR) algorithms provide an alternative formulation, where the ranking model provides a list of items to the user at each impression, and then immediately learns from observed user interactions and updates its behavior accordingly [18]. However, OLTR methods cannot do collaborative learning across users.

Inspired by the recent works which pose recommendation as an interactive learning problem [18, 23, 26], we explore contextual bandit and OLTR frameworks by applying the algorithms for list recommendations in the presence of noisy feedback, and extend them to overcome their respective shortcomings. Our contributions are:

- We introduce the cascading collaborative bandits algorithm that extends hLinUCB from top-1 to top-k recommendations using a cascade click model [4].
- We propose Collaborative Pairwise Ranking (CPR), an OLTR algorithm that learns the latent features of users and items.

ORSUM@ACM RecSys 2020, September 25th, 2020, Virtual Event, Brazil

CPR is a pairwise approach that optimizes the ranking directly, whereas cascading bandits are point-wise.
- We evaluate CPR with varying feedback noise levels in dynamic environments, where users and items arrive over time. CPR outperforms state-of-the-art algorithms in our experiments.

To our knowledge, CPR is the first latent factor based OLTR method and this work is the first study making comparisons of OLTR and bandit algorithms in a dynamic list recommendation setting.

2 RELATED WORK

Contextual bandits have been widely used to model interactive recommendations [1, 15, 28]. They learn the policy based on the estimated reward of each action using contextual information. LinUCB [15] selects an action with the highest upper confidence bound and achieves optimal regret. KernelUCB [14] extends the linear reward function to kernel functions. These approaches recommend a single item only. Cascading bandits [28] extend LinUCB and LinTS [1] to do top-K recommendations. *hLinUCB* [23] combines the benefits of online latent factor learning with the efficient exploration strategies of bandits to learn hidden features of users and items. Several other works perform online matrix factorization with bandit exploration [12, 17, 27] for single item recommendation. In contrast to prior bandit approaches, CPR uses pairwise ranking for top-K recommendations.

OLTR approaches learn user preferences by approaching optimization as a dueling bandit problem [26]. They estimate the gradient of the model w.r.t. user satisfaction by comparing the current model to sampled variations of the model. Several works have used Dueling Bandit Gradient Descent (DBGD) as a basis and extended upon it. Notably, Hofmann et al. [11] have proposed a method that guides exploration by only sampling variations that seem promising from historical interaction data. DBGD uses interleaving to determine the gradient direction from the resulting set of models. Multileave Gradient Descent (MGD) [21] replaces the interleaving of DBGD with multileaving methods. Pairwise Differentiable Gradient Descent (PDGD) [18] improves DBGD and MGD in terms of unbiased estimation using pairwise item preferences and allows for any differentiable ranking model. In addition, PDGD does not rely on sampling models for exploration, but instead models rankings as Plackett-Luce distributions over items. This allows PDGD to be explorative in cases where it is uncertain about specific items. However, PDGD cannot do collaborative learning across users. CPR extends the PDGD algorithm to collaboratively learn latent user/items features.

3 PROBLEM FORMULATION

Suppose we have a finite set $\boldsymbol{D} = \{1, \ldots, D\}$ of D total items. Let $\Pi^k = \{(a_1, \ldots, a_k) : a_1, \ldots, a_k \in \boldsymbol{D}, a_i \neq a_j \text{ for any } \{i \neq j\}$ be the set of all k-rankings of distinct items from $\boldsymbol{D}$, where k is a fixed number representing the size of ranked list. The learning agent takes a feature representation $\boldsymbol{d}$ of an item i as input and outputs a score using a function $f_{\boldsymbol{\theta}_u}(\boldsymbol{d}_i)$. Here, u represents the user index. At time t, C candidate item vectors $\boldsymbol{C} = \{\boldsymbol{d}_i | i \in \boldsymbol{D}\}$ are revealed to the learning agent. The learning agent scores each of these items and produces a ranked list of k items $\boldsymbol{A_t} = (a_1, \ldots, a_k) \in \Pi^k$. The user u is assumed to scan this list top-down, click on relevant items and then stop. The agent is only provided with *partial feedback* based on the position of items clicked. The objective of the agent at time t is to find the parameters $\boldsymbol{\theta}_u$, so that sorting the items by their scores in descending order maximizes the Normalized Discounted Cumulative Gain (NDCG) [24]. We assume no historical information is available at the beginning. The agent can update its parameters $\boldsymbol{\theta}$ based on the feedback from each user interaction. It needs to balance exploration and exploitation in as few interactions as possible to minimize poor user experience due to irrelevant recommendations.

3.1 Baseline Algorithms

Linear Cascading Bandits: LinUCB [15] and LinTS [1] assume that each user u is associated with an unknown preference parameter $\boldsymbol{\theta}_u \in \mathbb{R}^d$. This preference parameter, together with the given item's context vector $\boldsymbol{d}_j \in \mathbb{R}^d$, determines the score of item j by $r_{j,u} = \boldsymbol{d}_j^T \boldsymbol{\theta}_u + \eta$, where the random noise η is drawn from a zero-mean Gaussian distribution $N(0, \sigma^2)$. **CascadeLinUCB** and **CascadeLinTS** [28] extend the former to support top-K recommendation by assuming a cascading click model. The ranked list is produced by scoring each candidate item and sorting the scores in descending order. For updating the agent weights, each click $\in (0, 1)$ is used as a feedback for the corresponding item. We use CascadeLinTS and CascadeLinUCB as baselines in our experiments.
Collaborative Cascading Bandits: Contextual bandit algorithms assume that the learner has access to the features of users and items ahead of time. This precludes use of collaborative features. hLinUCB [23] improves upon LinUCB, and learns latent features for users and items. The learning agent can be represented as:

$$r_{j,u} = (\boldsymbol{d}_j^{\boldsymbol{o}}; \boldsymbol{d}_j^{\boldsymbol{h}})^T (\boldsymbol{\theta}_u^{\boldsymbol{o}}; \boldsymbol{\theta}_u^{\boldsymbol{h}}) + \eta, \tag{1}$$

where $r_{j,u}$ is the payoff of item j for user u, η is random noise drawn from a zero-mean Gaussian distribution, and $\boldsymbol{d}_j^{\boldsymbol{o}}$ and $\boldsymbol{d}_j^{\boldsymbol{h}}$ are the observable and latent item features respectively. $\boldsymbol{\theta}_u^{\boldsymbol{o}}$ is the weights of the model learned for the observable item features, and $\boldsymbol{\theta}_u^{\boldsymbol{h}}$ is the latent user features. The observable item features $\boldsymbol{d}_j^{\boldsymbol{o}}$ is an optional input to the model, and in its absence, the model will learn only with the latent features of users and items, i.e. $\boldsymbol{\theta}_u^{\boldsymbol{h}}$ and $\boldsymbol{d}_j^{\boldsymbol{h}}$ respectively.

hLinUCB was designed to produce the top-1 recommendation only. Following Zong et al. [28], we extend hLinUCB to output top-k recommendations, assuming a cascade model. We refer to this extension as k-hLinUCB (Algorithm 1). We further create a variant of k-hLinUCB to produce k-hLinUCB-Greedy, by turning off the UCB exploration in k-hLinUCB. Analyzing the performance of this algorithm against k-hLinUCB will help us see the benefits of exploration. Without exploration, k-hLinUCB-Greedy can be seen as a form of online matrix factorization [16]. We use **k-hLinUCB** and **k-hLinUCB-Greedy** as baselines for comparison with our proposed CPR algorithm.
Neural Collaborative Filtering (NCF) [10] uses neural networks for collaborative filtering. We use NCF as a baseline to highlight the trade-offs between expressivity of neural networks with high representational power and the complexity of frequent re-training

in dynamic environments. We use this model in an online setting where we store the interactions in an experience buffer and train the model for 10 iterations after every 50 interactions with a batch size of 512.

Algorithm 1: k-hLinUCB

Input: *agent*: a hLinUCB instance
for $t = 1$ **to** T **do**
  User $u \sim$ Uniform(1,U)
  for item $j = 1$ **to** D **do**
    $R_t(j) \leftarrow$ agent.get_score(u, j)
  end for
  for $i = 1$ **to** K **do**
    $a_i^t \leftarrow arg\max_{j \in [D] - \{a_1^t, \ldots, a_{i-1}^t\}} R_t(j)$
  end for
  // Recommend K ranked items and get feedback
  $\boldsymbol{A}_t \leftarrow (a_1^t, \ldots, a_K^t)$
  Observe clicks $\boldsymbol{C}_t$
  for $i = 1$ **to** K **do**
    // Ingest feedback and update weights
    agent.observe_feedback(u, j, c_t^i)
  end for
end for

Algorithm 2: Collaborative Pairwise Ranking (CPR)

1: **Input:** Randomly initialized weights θ_u, γ_d for all users/items; predictor f; learning rate η
2: **for** $t = 1$ **to** T **do**
3: User $u \sim$ Uniform(1,U)
4: $D_t \leftarrow generate\_candidate\_items(u)$
5: $\boldsymbol{R}_t \leftarrow sample\_ranked\_list(f_{\theta_{u,t}\gamma_d}(.), D_t)$ // (Eq. 2)
6: $c_t \leftarrow receive\_clicks(\boldsymbol{R}_t)$
7: $\nabla f_{\theta_{u,t}} \leftarrow \boldsymbol{0}$
8: $\nabla f_{\gamma_{d,t}} \leftarrow \boldsymbol{0}$ for all $d \in \boldsymbol{R}_t$
9: **for** $d_i \succ_c d_j \in c_t$ **do**
10: $w \leftarrow \rho(d_i, d_j, R, D)$
11: $w \leftarrow w \times P(d_i \succ_c d_j \mid \theta_{u,t}, \gamma_{i,t}, \gamma_{j,t}) P(d_j \succ_c d_i \mid \theta_{u,t}, \gamma_{i,t}, \gamma_{j,t})$
12: $\nabla f_{\theta_{u,t}} \leftarrow \nabla f_{\theta_{u,t}} + w(\gamma_{i,t} - \gamma_{j,t})$
13: $\nabla f_{\gamma_{i,t}} \leftarrow \nabla f_{\gamma_{i,t}} + w(\theta_{u,t})$
14: $\nabla f_{\gamma_{j,t}} \leftarrow \nabla f_{\gamma_{j,t}} - w(\theta_{u,t})$
15: **end for**
16: $\theta_{u,t+1} \leftarrow \theta_{u,t} + \eta \nabla f_{\theta_{u,t}}$
17: $\gamma_{i,t+1} \leftarrow \gamma_{i,t} + \eta \nabla f_{\gamma_{i,t}}$
18: $\gamma_{j,t+1} \leftarrow \gamma_{j,t} + \eta \nabla f_{\gamma_{j,t}}$
19: **end for**

4 COLLABORATIVE PAIRWISE RANKING

The original PDGD formulation is designed for document retrieval where the agent takes a feature representation of a document $\boldsymbol{d}$ as input and outputs a score using $f_{\boldsymbol{\theta}}(\boldsymbol{d})$. Using PDGD requires item features from the environment. Moreover, a separate agent $f_{\boldsymbol{\theta}_u}(\boldsymbol{d})$ is learned for each user u, which is inefficient because there is no collaborative knowledge shared across users. Collaborative Pairwise Ranking (CPR) extends PDGD to learn collaborative user and item features online. State-of-the-art collaborative filtering solutions are based on latent factor models, which outperform traditional content-based methods [3, 13] . Such agents learn effectively by propagating the user feedback across users and items. Although CPR can leverage observable user/item features, we leave its empirical analysis for future work.

CPR optimizes a learning agent $f_{\boldsymbol{\theta}_u \boldsymbol{\gamma}_d}(u, d)$, where $\boldsymbol{\theta}_u$ and $\boldsymbol{\gamma}_d$ are the latent features of user u and item d respectively. The aim of the algorithm is to find the parameters $\boldsymbol{\theta}_u$ and $\boldsymbol{\gamma}_d$ for all the users and items so that sorting the items by their scores in descending order provides optimal rankings. In our experiments, we assume the scoring function f to be a dot product between $\boldsymbol{\theta}_u$ and $\boldsymbol{\gamma}_d$, but the approach can be extended to non-linear models like neural networks.

Algorithm 2 captures the details of CPR. Given a user u and a set of candidate items D, a Plackett-Luce model is applied to the ranking function $f_{\boldsymbol{\theta}_u \boldsymbol{\gamma}_d}(.)$ resulting in a distribution over the item set D:

$$P(d \mid D) = \frac{e^{f_{\theta_u \gamma_d}(u,d)}}{\sum_{d' \in \boldsymbol{D}} e^{f_{\theta_u \gamma_{d'}}(u,d')}} \tag{2}$$

A ranking R to display to the user u is then created by sampling from the distribution k times, where after each placement the distribution is re-normalized to prevent duplicate placements. The user then interacts with the list and may choose to click on some or none of the items. The algorithm assumes that the clicked items are preferred over unclicked ones. Since the algorithm does not know which items were observed, it assumes that every item preceding a clicked item and the first subsequent unclicked item was observed. We will denote preferences between items inferred from clicks as: $d_i \succ_c d_j$ where d_i is preferred over d_j .

The weights of the model are updated by optimizing pairwise probabilities over the preference pairs; for each inferred item preference $d_i \succ_c d_j$, the probability that the preferred item d_i is sampled before d_j is sampled is increased:

$$\begin{aligned} P(d_i \succ_c d_j) &= \frac{P(d_i \mid D)}{P(d_i \mid D) + P(d_j \mid D)} \\ &= \frac{e^{f_{\theta_u \gamma_i}(u,i)}}{e^{f_{\theta_u \gamma_i}(u,i)} + e^{f_{\theta_u \gamma_j}(u,j)}}. \end{aligned} \tag{3}$$

The gradient is estimated as the following weighted sum:

$$\nabla f(\theta, \gamma) \approx \sum_{d_i \succ_c d_j} \rho(d_i, d_j, R, D)[\nabla P(d_i \succ_c d_j)]. \tag{4}$$

Following Oosterhuis and de Rijke [18], we use a reweighing function $\rho(d_i, d_j, R, D)$ to make the gradient unbiased w.r.t the item pair preferences. ρ uses the reverse pair ranking: $R^*(d_i, d_j, R)$, which is the same ranking R but with items d_i and d_j swapped. The reweighing function ρ (equation 5) is shown to be stable and produces an unbiased gradient [18]. Lines 12-18 of Algorithm 2 show the gradient update equations.

$$\rho(d_i, d_j, R, D) = \frac{P(R^*(d_i, d_j, R) \mid D)}{P(R \mid D) + P(R^*(d_i, d_j, R) \mid D)} \tag{5}$$

CPR has some notable benefits over collaborative cascading bandit algorithms like k-hLinUCB. Firstly, CPR uses a pairwise ranking loss to order the items. It directly optimizes for the ordering of a particular pair according to user feedback. Cascading bandits, on the other hand, use a regression target - click or no click to optimize the models and hence rank each item independently according to its score. Secondly, CPR can use any differentiable model as a predictor, including non-linear models like neural networks. And lastly, the explore-exploit trade-off in CPR is taken care of implicitly by sampling from the Plackett-Luce distribution. CPR does not maintain a separate covariance matrix for each user/item to model the uncertainty in weights, as done by most bandit algorithms. Hence, CPR has a lot less parameters than cascading bandits.

5 EXPERIMENTS

5.1 Dataset and Interactive Setup

We evaluate the algorithms on a real world recommendation dataset - MovieLens 100K [9]. This movie rating dataset has been widely used to evaluate collaborative filtering algorithms [6, 12, 22, 25]. It has a total of 100K ratings given by 943 users for 1682 items. The ratings were given on a scale of 1-5 stars. We adopt the setup used by Kawale et al. [12] to evaluate an interactive learning agent on a static dataset of the ratings matrix. Specifically, at the 0^{th} interaction round, no entry of the reward matrix has been revealed. At every round t, we randomly sample one of the users present in the data to interact with the system; this can be a user the system has already interacted with in previous rounds (warm-start), or a new one (cold-start). We generate a candidate list of items by randomly sampling L movies from the movies that this user has rated. If a user has rated less than L movies, then we use all of the rated movies as candidates. We do not show all the rated movies as candidates on purpose to simulate a highly dynamic environment in the beginning, with cold users and items being exposed over time. The learning agent decides which items to show to the user. It ranks the L items and presents the top K items to the user in every round.

We simulate users following the OLTR setup of Oosterhuis and de Rijke [18]. At each item, the user decides whether to click it or not, modeled as a probability conditioned on the true relevance label $R : P(click = 1 \mid R)$. After a click has occurred, the userâĂŹs information need may be satisfied and they may stop viewing items. The probability of a user stopping after a click is modeled as $P(stop = 1 \mid click = 1, R)$. As shown in Table 1, we use three different click model probability configurations to represent three different types of users – 1) a *perfect user*, who clicks on all relevant items and does not stop browsing until they have visited all of the items. This type of users contribute the least noise, as they make no mistakes and the feedback is entirely accurate. 2) a *navigational user*, who is very likely to click on the first highly relevant item that they see and stop there. 3) an *informational user*, who sometimes clicks on irrelevant items and contributes a significant amount of noise in click feedback.

We use Cumulative (online) Normalized Discounted Cumulative Gain (NDCG) to evaluate the learning agents. Cumulative NDCG is calculated by summing NDCG scores from successive iterations with a discount factor γ set to 0.99995. We apply the discount factor to reduce the weightage of interactions that happen later in the process as compared to the cold start interactions. In other words, we want to weigh the cold start interactions more. This will capture the user experience more accurately. We use a high discount factor so that the contributions of interactions later in the process do not fade away. Another measure of interest can be the cumulative regret over T rounds, $R_T = \mathbb{E}[\sum_1^T r_{A*,\theta^*} - r_{A_t,\theta_t}]$, where A^* is the optimal ranking and A_t is the ranking that the agent showed at time t. We refrain from evaluating agents using this metric as it is rank unaware. If we include position weights into account for calculating the reward, then cumulative regret becomes similar to cumulative NDCG and hence we just report the latter.

Table 1: Instantiations of Cascading Click Models as used for simulating user behavior in experiments

	$P(click = 1 \mid R)$					$P(stop = 1 \mid click = 1, R)$				
Rating	1	2	3	4	5	1	2	3	4	5
perf	0	.2	.4	.8	1	0	0	0	0	0
nav	.05	.3	.5	.7	.95	.2	.3	.5	.7	.9
inf	.4	.6	.7	.8	.9	.1	.2	.3	.4	.5

Table 2: Cumulative NDCG@10 for different agents and clicks models after 30,000 impressions. The numbers are scaled down by a factor of 10,000 for easier readability.

	Perf	Nav	Inf
CPR	1.137	.908	.894
k-hLinUCB	1.090	.881	.859
k-hLinUCB-Greedy	1.104	.826	.819
CascadeLinUCB	1.076	.849	.845
CascadeLinTS	1.032	.841	.829
NCF	.926	.802	.773
Random	.790	.769	.751

5.2 Experimental runs

We simulate runs consisting of 30,000 impressions; each run was repeated 10 times with different random seeds under each of the three click models. We set $\eta = 0.1$ and use uniform random initialization of weights. We fix the size of recommended items list to $k = 10$. For collaborative models, we keep the embedding dimension fixed to 16. To generate the candidate list, we randomly sample $L = 50$ movies from the list of movies that the user has rated in the dataset.

5.3 Results

As seen in Figure 1 and Table 2, CPR consistently outperforms other agents in terms of the user experience which is measured by Online NDCG. In the perfect click model assumption, we see that the second best performer is k-hLinUCB-Greedy. Through this observation, we conclude that in case of a perfect scenario with no noise, the agent can be confident about what it has learned and need not continuously explore. Initial exploration in k-hLinUCB-Greedy is implicit due to random initialization of the parameters.

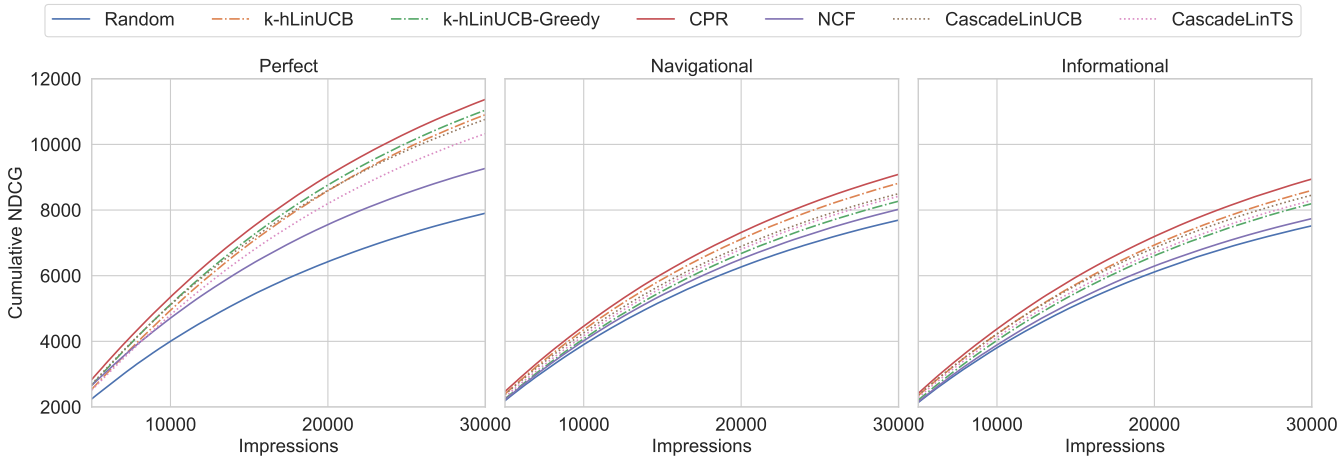

Figure 1: Performance (online NDCG@10) of different agents on MovieLens-100K under three different click model assumptions - Perfect, Navigational and Informational. The *Random* agent recommends items at random from the candidate list.

We also observe that k-hLinUCB, which learns latent features of users and items, performs consistently better than CascadeLinUCB and CascadeLinTS. The latter rely on using movie genres as features. As we move towards navigational and informational models, which are noisier, we can see the benefits of exploration. Performance of k-hLinUCB gets better, while k-hLinUCB-Greedy starts getting worse.

The Neural Collaborative Filtering performs worse than all other algorithms, which can be attributed to three reasons: 1) It does not react to user feedback instantly, as we update the model after every 50 interactions. 2) The model ranks the items greedily, so the performance gets worse in the noisier feedback environments, i.e., with navigational and informational simulated users. 3) The NCF model is the most complex of all the models and might require more data or better hyperparameter tuning.

6 CONCLUSION AND FUTURE WORK

We study the problem of interactive recommender systems, that can react quickly to changes in dynamic environments by capturing user feedback online. We introduced a novel OLTR learning algorithm called Collaborative Pairwise Ranking (CPR), which learns latent features of users and items to perform effective collaborative learning across users. We provide an extensive study of CPR with other state-of-the-art approaches in contextual and collaborative bandits, and their extensions to ranking. We showcase the superior performance of CPR on a real-world dataset, by simulating a noisy interactive environment.

For future work, we want to explore more complex model architectures, like neural networks and how to train them efficiently in an online manner. We also plan to incorporate different click modeling solutions for more accurate and realistic user behavior modeling.

REFERENCES

[1] Shipra Agrawal and Navin Goyal. 2012. Analysis of thompson sampling for the multi-armed bandit problem. In *Conference on learning theory*. 39–1.

[2] Marko Balabanović and Yoav Shoham. 1997. Fab: content-based, collaborative recommendation. *Commun. ACM* 40, 3 (1997), 66–72.

[3] Robert M Bell and Yehuda Koren. 2007. Lessons from the Netflix prize challenge. *Acm Sigkdd Explorations Newsletter* 9, 2 (2007), 75–79.

[4] Nick Craswell, Onno Zoeter, Michael Taylor, and Bill Ramsey. 2008. An experimental comparison of click position-bias models. In *Proceedings of the 2008 international conference on web search and data mining*. 87–94.

[5] James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, and Dasarathi Sampath. 2010. The YouTube Video Recommendation System. In *Proceedings of the Fourth ACM Conference on Recommender Systems* (Barcelona, Spain) *(RecSys âĂŹ10)*. Association for Computing Machinery, New York, NY, USA, 293âĂŞ296. https://doi.org/10.1145/1864708.1864770

[6] Luis M De Campos, Juan M Fernández-Luna, Juan F Huete, and Miguel A Rueda-Morales. 2010. Combining content-based and collaborative recommendations: A hybrid approach based on Bayesian networks. *International journal of approximate reasoning* 51, 7 (2010), 785–799.

[7] Souvik Debnath, Niloy Ganguly, and Pabitra Mitra. 2008. Feature weighting in content based recommendation system using social network analysis. In *Proceedings of the 17th international conference on World Wide Web*. 1041–1042.

[8] Carlos A. Gomez-Uribe and Neil Hunt. 2016. The Netflix Recommender System: Algorithms, Business Value, and Innovation. *ACM Trans. Manage. Inf. Syst.* 6, 4, Article 13 (Dec. 2016), 19 pages. https://doi.org/10.1145/2843948

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

[10] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*. 173–182.

[11] Katja Hofmann, Anne Schuth, Shimon Whiteson, and Maarten De Rijke. 2013. Reusing historical interaction data for faster online learning to rank for IR. In *Proceedings of the sixth ACM international conference on Web search and data mining*. 183–192.

[12] Jaya Kawale, Hung H Bui, Branislav Kveton, Long Tran-Thanh, and Sanjay Chawla. 2015. Efficient Thompson Sampling for Online Matrix-Factorization Recommendation. In *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (Eds.). Curran Associates, Inc., 1297–1305.

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

[14] Andreas Krause and Cheng S Ong. 2011. Contextual gaussian process bandit optimization. In *Advances in neural information processing systems*. 2447–2455.

[15] Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. 2010. A Contextual-Bandit Approach to Personalized News Article Recommendation. In *Proceedings of the 19th International Conference on World Wide Web* (Raleigh, North Carolina, USA) *(WWW âĂŹ10)*. Association for Computing Machinery, New York, NY, USA, 661âĂŞ670. https://doi.org/10.1145/1772690.1772758

[16] Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. 2010. Online Learning for Matrix Factorization and Sparse Coding. *Journal of Machine Learning*

Research 11, 2 (2010), 19–60. http://jmlr.org/papers/v11/mairal10a.html
[17] Atsuyoshi Nakamura. 2015. A ucb-like strategy of collaborative filtering. In *Asian Conference on Machine Learning*. 315–329.
[18] Harrie Oosterhuis and Maarten de Rijke. 2018. Differentiable unbiased online learning to rank. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. 1293–1302.
[19] Steffen Rendle. 2010. Factorization machines. In *2010 IEEE International Conference on Data Mining*. IEEE, 995–1000.
[20] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. 1994. GroupLens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*. 175–186.
[21] Anne Schuth, Harrie Oosterhuis, Shimon Whiteson, and Maarten de Rijke. 2016. Multileave gradient descent for fast online learning to rank. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*. 457–466.
[22] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. 2015. Autorec: Autoencoders meet collaborative filtering. In *Proceedings of the 24th international conference on World Wide Web*. 111–112.
[23] Huazheng Wang, Qingyun Wu, and Hongning Wang. 2016. Learning hidden features for contextual bandits. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. 1633–1642.
[24] Yining Wang, Liwei Wang, Yuanzhi Li, Di He, Wei Chen, and Tie-Yan Liu. 2013. A theoretical analysis of NDCG ranking measures. In *Proceedings of the 26th annual conference on learning theory (COLT 2013)*, Vol. 8. 6.
[25] Bin Xu, Jiajun Bu, Chun Chen, and Deng Cai. 2012. An exploration of improving collaborative recommender systems via user-item subgroups. In *Proceedings of the 21st international conference on World Wide Web*. 21–30.
[26] Yisong Yue and Thorsten Joachims. 2009. Interactively optimizing information retrieval systems as a dueling bandits problem. In *Proceedings of the 26th Annual International Conference on Machine Learning*. 1201–1208.
[27] Xiaoxue Zhao, Weinan Zhang, and Jun Wang. 2013. Interactive collaborative filtering. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*. 1411–1420.
[28] Shi Zong, Hao Ni, Kenny Sung, Nan Rosemary Ke, Zheng Wen, and Branislav Kveton. 2016. Cascading bandits for large-scale recommendation problems. *arXiv preprint arXiv:1603.05359* (2016).