Explore-Exploit in Top-N Recommender Systems via Gaussian Processes

Hastagiri P Vanchinathan
ETH Zürich
hastagiri@inf.ethz.ch

Isidor Nikolic *
Microsoft, Zürich
inikolic@microsoft.com

Fabio De Bona
Google, Zürich
fdb@google.com

Andreas Krause
ETH Zürich
krausea@ethz.ch

ABSTRACT

We address the challenge of ranking recommendation lists based on click feedback by efficiently encoding similarities among users and among items. The key challenges are threefold: (1) combinatorial number of lists; (2) sparse feedback and (3) context dependent recommendations. We propose the CGPRANK algorithm, which exploits prior information specified in terms of a Gaussian process kernel function, which allows to share feedback in three ways: Between positions in a list, between items, and between contexts. Under our model, we provide strong performance guarantees and empirically evaluate our algorithm on data from two large scale recommendation tasks: Yahoo! news article recommendation, and Google books. In our experiments, our CGPRANK approach significantly outperforms state-of-the-art multi-armed bandit and learning-to-rank methods, with an 18% increase in clicks.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Relevance feedback; H.3.5 [**Online Information Services**]: Web-based Services

Keywords

Recommender systems; CGPRANK; User feedback; Kernel methods; Exploration–exploitation tradeoffs

1. INTRODUCTION

Traditionally, recommender systems relied on supervised learning on a batch of data. Existing approaches like collaborative filtering, content based filtering or learning to rank techniques all try to learn a fixed optimal recommendation model given training data. These approaches fail to capture the dynamic nature of the user preferences and inventory. An ideal recommendation list for a given user/context has to take recent feedback into account.

Learning this optimal ordering leads to an "explore–exploit" tradeoff, where we need to gather information about the effectiveness of orderings, while at the same time maximizing conversions based on the estimated data. Standard multi-arm bandit algorithms which solve such tasks either use similarity information or cannot select lists of items. On a web scale, this is a daunting task, for mainly three reasons: (1) There is an exponential number of lists to choose from; (2) the number of items available for recommendation is often large, compared to the relatively sparse click feedback; (3) optimality of the ordering depends on the context which in itself could be from a large set (e.g., user to whom the item should be recommended).

In this work, we propose an algorithm – CGPRANK – for (re-) ranking recommendations from click feedback. In order to address above challenges, it uses a positive definite kernel function to encode prior assumptions about the similarity of items (and possible user features). This kernel is then used to share and exploit sparse feedback in multiple ways: across positions in the list (1), across items (2) and across users/contexts (3). CGPRANK navigates the exploration–exploitation tradeoff by predicting performance of as yet unexplored lists using nonparametric Gaussian process models, whose regularity is captured in the kernel function. Exploiting such regularity allows our algorithm to be statistically efficient, i.e., maximize the benefit drawn from sparse observations about the users' preferences.

We prove strong performance guarantees for our CGPRANK algorithm. In particular, we analyze its regret, i.e., the loss in conversions/clicks compared to using (hindsight) optimal orderings. We prove that under natural separability and regularity assumptions, the average regret vanishes. Further, we analyze the effect of increasing list sizes, and find a surprising theoretical result: Under some natural conditions, increasing the list size can parallelize exploration in a way to *accelerate* convergence, instead of an exponential slowdown. We extensively evaluate our approaches on two large datasets from real world recommendation tasks: Firstly, we consider news article recommendation, using data provided by Yahoo! [1]. Secondly, we evaluate CGPRANK on Google's infrastructure, using clickstream data from Google's ebooks store, demonstrating a significant improvement over existing multi-armed bandit and learning-to-rank techniques. In particular, our approach provides an 18% lift in total clicks compared to the existing approaches.

2. RELATED WORK

Our approach relates and builds on work in multiple areas.

Recommender systems, ranking and relevance: Popular techniques include collaborative filtering, matrix factorization and frequent item set mining [21], as well as learning to rank approaches [2]. These approaches usually estimate user's preferences ("ex-

*Work done while the author was a student at ETH, Zürich and working as an intern at Google, Zürich

RecSys 2014, Foster City, USA
ACM 978-1-4503-2668-1/14/10 ...$15.00.
http://dx.doi.org/10.1145/2645710.2645733.

[1] http://webscope.sandbox.yahoo.com/

ploit") from a fixed training set collected a priori, and generally do not address how to dynamically collect data ("explore") for training in order to adapt to changing inventories and user bases. We empirically compare to learning to rank (LTR) approaches that learn from user interactions and demonstrate the benefit of our exploration strategy.

Multi-arm bandits (MAB): For single item recommendation, MAB is a classical formalism for studying the explore-exploit dilemma incurred when optimizing an unknown payoff function with noisy feedback limited only to the choices made. Early approaches such as ϵ-Greedy, UCB1 [3], do not exploit the similarity information between the choices, and thus fail when feedback is sparse. Modern research has addressed this challenge, under assumptions of linear [17] and Lipschitz [12] continuous payoff functions, as well as functions with regularity explained by a kernel [24, 15]. However, these approaches do not consider the challenges arising when selecting sets and lists.

Bandits for subset selection and ranking have been studied before in both ordered and unordered subset selection settings [14]. In particular, the best subset selection under 'bandit-feedback' setting has been studied [11]. In [26, 25], the authors study a similar problem under the setting that the feedback of a set is a submodular function of the concepts covered by the set, which allows to capture diversity, but no similarity among items. [23] considers choosing diverse rankings exploiting item similarity for the problem of ranked document retrieval. Our approach, while similar in principle to some of these works, is the first to systematically exploit sharing of feedback among items, contexts *and* positions.

Top-N recommendations are an important subclass of recommendation problems [21]. Researchers have long expressed the importance of explore–exploit schemes in dynamic top-N recommendation problems [16] and also found deficiencies in using RMSE optimisation techniques for online recommendations [7]. We develop efficient techniques for managing the explore-exploit tradeoff and use appropriate regret measures to show a marked improvement over RMSE based schemes.

3. PROBLEM STATEMENT

We have a set of items $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$ that we consider for recommendation (e.g., books, articles etc.). We model the recommendation task as a sequential decision problem over T rounds, where, in each round t, we are given a subset of items $\mathcal{S}_t \subseteq \mathcal{S}$ available for recommendation. To aid in the recommendation task, we are given a context $\mathbf{z} \in \mathcal{Z}$, which, e.g., models the key (anchor) item that the user is currently considering, or possibly also containing features describing the user. While such context could be presented as a feature vector, our algorithm does not require such vectorial representation. Our task is to select an ordered list $\mathcal{L}_t = [\mathbf{s}_t^{[1]}, \ldots, \mathbf{s}_t^{[b]}]$ of b items out of $\mathcal{S}_t$ that is recommended to the user. In response, we receive a stochastic vector of rewards, $\mathbf{y} = [y_t^{[1]}, \ldots, y_t^{[b]}]$, where $\mathbb{E}[y_t^{[i]}] = f(\mathbf{s}_t^{[i]}, \mathbf{z}_t, i)$. Hereby, we assume that there is some underlying *unknown* reward function $f : \mathcal{S} \times \mathcal{Z} \times \mathcal{P} \to \mathbb{R}$, such that the expected reward of recommending item $\mathbf{s}_t^{[i]}$ in context $\mathbf{z}_t$ at position $i \in \mathcal{P} = \{1, \ldots, b\}$ is given by $f(\mathbf{s}_t^{[i]}, \mathbf{z}_t, i)$. For concreteness, f may model the click-through rate (CTR), and the rewards $y_t^{[i]} \in [0, 1]$ model whether the user clicks on the item in position i. The total reward in round t is thus

$$y_t = \sum_{i=1}^{b} y_t^{[i]}, \tag{1}$$

and our goal is to maximize the expected cumulative reward $\mathbb{E}[\sum_t y_t]$. Note that, in this model, we assume that the items in the list do not influence the rewards (clicks) received by other items, i.e., we do not model side effects of other items in the list. In the experiments, we do study the effect of relaxing this assumption. We further assume that the expected reward

$$f(\mathbf{s}_t^{[i]}, \mathbf{z}_t, i) = r(\mathbf{s}_t^{[i]}, \mathbf{z}_t)p(i) \tag{2}$$

factorizes into a *relevance* term $r(\mathbf{s}_t^{[i]}, \mathbf{z}_t)$ (relevance measures the relatedness/interestingness of the item to the context which could be a query, user features or key item) and a position-dependent effect $p(i) \in [0, 1]$. Without loss of generality, we assume that $p(1) = 1$, and for all $i \leq j$, $p(i) \geq p(j) > 0$, i.e., showing an item in a later position in the list can only decrease the expected reward.

Under these assumptions, to maximize the reward (clicks), we need to position the items in decreasing order of their true relevance r. For a fixed position i and context $\mathbf{z}_t$, the expected number of clicks received by an item $\mathbf{s}_t^{[i]}$ will be proportional to its relevance to the context, i.e., $r(\mathbf{s}_t^{[i]}, \mathbf{z}_t)$.

The position dependence p can often be estimated effectively [6]. However, the true relevance r is a priori unknown, and must be estimated through experimentation. We face an exploration–exploitation dilemma, where we have to choose between exploiting the information we have about the best ordering, and exploring alternate orderings, which may or may not lead to higher rewards.

Instead of maximizing rewards, in the following we equivalently wish to minimize the *regret*. Hereby, the instantaneous regret r_t in round t is given by $r_t = \sum_{i=1}^{b} [f(\mathbf{s}_t^{[i]^*}, \mathbf{z}_t, i) - f(\mathbf{s}_t^{[i]}, \mathbf{z}_t, i)]$, where $\mathcal{L}_t^* = [\mathbf{s}_t^{[1]^*}, \ldots, \mathbf{s}_t^{[b]^*}]$ is an optimal (in expectation) ordered list for context $\mathbf{z}_t$ observed at round t. Our goal is to minimize the cumulative regret, $R_T = \sum_{t=1}^{T} r_t$. In particular, we desire an algorithm such that the average regret vanishes, i.e., $R_T/T \to 0$ as $T \to \infty$. Note that this is quite a stringent performance requirement: Vanishing regret requires that the algorithm learns the optimal (in expectation) mapping from context to recommendations.

4. CGPRANK APPROACH AND ANALYSIS

Given the problem setup, we address the resulting challenges in this section. In Section 4.1, we show how we can share item feedback across positions. In Section 4.2, we discuss the use of statistical models for principled generalization of feedback to items/contexts that are not yet explored. Finally, the resulting exploration–exploitation dilemma is addressed in Section 4.3 leading up to the specification of the CGPRANK algorithm. Before we describe the algorithm in Section 4.4 and provide theoretical guarantees in Section 4.5, we first highlight the key technical challenges:

1. There is a combinatorial number of possible ordered lists. When n is large, these exponentially many choices are intractable even for small b. In Section 4.1, we show how we can share item feedback across positions in order to reduce this complexity.
2. In many applications, click feedback is sparse, potentially severely delaying convergence. In Section 4.2, we discuss the use of statistical models for generalizing the feedback to items that are not yet explored in a principled way.
3. Once we have settled on a statistical model for learning about reward, we face the exploration–exploitation dilemma of trading experimentation (for the purpose of parameter estimation) and exploitation (using the model's predictions to maximize reward). This dilemma is addressed in Section 4.3.

4.1 Sharing feedback across positions

Given a context (e.g., key-item), selecting an optimal ranked list of b recommendations is challenging due to the combinatorial number of choices. In general, we may need to estimate the reward associated with each of the exponentially many rankings. However, under our assumptions (1) and (2) that the reward of a list decomposes additively, and that the reward factors into a position-dependent effect independent of the item and a "relevance" effect that is position independent, the problem becomes statistically and computationally more tractable. If we know the position effects $p(i),\ \forall i \in \{1, ..., b\}$, we can normalize the feedback received by an item across all positions that it has been shown at so far. That is, given context $\mathbf{z}_t$, if we observe $y_t^{[i]}$ for some item $\mathbf{s}$ shown in position i, $y_t^{[i]}/p(i)$ provides an unbiased estimate of $r(\mathbf{s}, \mathbf{z}_t)$. Consequently, an unbiased estimate for the reward obtained when showing $\mathbf{s}$ in position j instead is given by $y_t^{[i]}p(j)/p(i)$. This insight thus allows us to share feedback across positions.

4.2 Sharing across items/contexts via kernels

In order to generalize feedback across items/contexts, we need to incorporate prior information about their respective similarities. We assume that this prior information is presented in terms of an arbitrary positive definite *kernel function* $\kappa : (\mathcal{S} \times \mathcal{Z})^2 \to \mathbb{R}$. Hereby, for two item-context pairs $(\mathbf{s}, \mathbf{z})$ and $(\mathbf{s}', \mathbf{z}')$, the kernel $\kappa\big((\mathbf{s}, \mathbf{z}), (\mathbf{s}', \mathbf{z}')\big)$ represents our assumptions about how similar we expect the rewards of presenting item $\mathbf{s}$ in context $\mathbf{z}$, as opposed to presenting item $\mathbf{s}'$ in $\mathbf{z}'$ are. A multitude of kernel functions are available for accurately capturing similarity among various types of data [22]. We detail our specific choice in Section 6. What are the consequences of choosing any particular kernel? We effectively assume the reward function r can be represented as a linear combination

$$r(\mathbf{s}, \mathbf{z}) = \sum_j \alpha_j \kappa\big((\mathbf{s}, \mathbf{z}), (\mathbf{s}_j, \mathbf{z}_j)\big),$$

i.e., as a basis function expansion around a set of context–item pairs $((\mathbf{s}_j, \mathbf{z}_j))_j$. Such functions span the *Reproducing Kernel Hilbert Space* (RKHS) associated with kernel k, and the norm of r in that space,

$$||r||_\kappa = \sum_{i,j} \alpha_i \alpha_j \kappa\big((\mathbf{s}_i, \mathbf{z}_i), (\mathbf{s}_j, \mathbf{z}_j)\big),$$

measures the "complexity" (regularity) of function r. The performance of our algorithm, as analyzed in Theorem 1, will depend on this norm. Intuitively, if the kernel matches the regularity present in real data well, the norm will be small. Capturing similarity via kernels has important consequences: In particular, it allows interpreting the relevance function r as a sample from a *Gaussian Process (GP)* prior [20], with covariance (or kernel) function κ. Consequently, one interprets the relevance as a collection of normally distributed random variables, one for each item—context pair. They are jointly distributed in a dependent manner, such that their covariances are given by the kernel:

$$\mathrm{Cov}\big(r(\mathbf{s}, \mathbf{z}), r(\mathbf{s}_j, \mathbf{z}_j)\big) = \kappa\big((\mathbf{s}, \mathbf{z}), (\mathbf{s}_j, \mathbf{z}_j)\big).$$

This joint distribution then allows us to make predictions about unobserved item–context pairs via inference in the GP model. In particular, suppose we have already observed feedback for t recommendations, i.e., obtained data
$\mathcal{D} = \{(\mathbf{s}_1, \mathbf{z}_1, y_1), \ldots, (\mathbf{s}_t, \mathbf{z}_t, y_t)\}$. Then, for a new item–context pair $(\mathbf{s}, \mathbf{z})$, its predictive distribution for $r(\mathbf{s}, \mathbf{z})$ is Gaussian, with mean and variance[2] given by

$$\mu_t(\mathbf{s}, \mathbf{z}) = \mathbf{k}_t(\mathbf{s}, \mathbf{z})^T (\mathbf{K}_t + \mathbb{I})^{-1} \bar{\mathbf{y}}_t, \quad (3)$$

$$\sigma_t^2(\mathbf{s}, \mathbf{z}) = \kappa((\mathbf{s}, \mathbf{z}), (\mathbf{s}, \mathbf{z})) - \mathbf{k}_t(\mathbf{s}, \mathbf{z})^T (\mathbf{K}_t + \mathbb{I})^{-1} \mathbf{k}_t(\mathbf{s}, \mathbf{z}), \quad (4)$$

where $\mathbf{k}_t(\mathbf{s}, \mathbf{z}) = [\kappa((\mathbf{s}_1, \mathbf{z}_1), (\mathbf{s}, \mathbf{z})), \ldots, \kappa((\mathbf{s}_t, \mathbf{z}_t), (\mathbf{s}, \mathbf{z}))]^T$ and $\mathbf{K}_t$ is the positive semi-definite kernel matrix such that $\mathbf{K}_{t,i,j} = [\kappa((\mathbf{s}_i, \mathbf{z}_i), (\mathbf{s}_j, \mathbf{z}_j))]$.

Choice of Kernels. Often, kernels over item–context pairs are naturally expressed as tensor products, where

$$\kappa\big((\mathbf{s}, \mathbf{z}), (\mathbf{s}', \mathbf{z}')\big) = \kappa_{\mathcal{S}}\big(\mathbf{s}, \mathbf{s}'\big) \cdot \kappa_{\mathcal{Z}}\big(\mathbf{z}, \mathbf{z}'\big).$$

Hereby, $\kappa_{\mathcal{S}} : \mathcal{S}^2 \to \mathbb{R}$ is a kernel (similarity) among items, and $\kappa_{\mathcal{Z}} : \mathcal{Z}^2 \to \mathbb{R}$ is a kernel (similarity) among contexts. This choice of kernel expresses our prior assumption of how smoothly the CTR changes over the item-context space.

The choice of kernel $\kappa_{\mathcal{S}}$ depends on the particular recommendation problem. Often, similarity between items is given by a usually symmetric similarity function $sim : \mathcal{S}^2 \to R$. A valid kernel function however must additionally be positive definite (i.e., all resulting covariance matrices must be positive definite). Among various available candidates, we use diffusion kernels, a family of kernels first introduced in [13]. The first step is to consider the items $\mathcal{S}$ as nodes in a weighted, undirected graph G, so that the weight $w(i, j)$ of each edge (i, j) is given by the similarity function $\mathrm{sim}(i, j)$. The diffusion kernel is then given as matrix-exponential $\mathbf{K}_{\mathcal{S}} = \exp(\alpha \mathbf{L})$ of the graph Laplacian $\mathbf{L}$ of G. In the contextual setting, if the context is given as a key item, the same diffusion kernel can be used both for items and contexts. If the context is given in terms of user features, $\kappa_{\mathcal{Z}}$ can be chosen, e.g., as linear kernel $\kappa_{\mathcal{Z}}(\mathbf{z}, \mathbf{z}') = \mathbf{z}^T \mathbf{z}'$, or Gaussian kernel $\kappa_{\mathcal{Z}}(\mathbf{z}, \mathbf{z}') = \exp(||\mathbf{z} - \mathbf{z}'||_2^2 / h^2)$. If no similarity information is known between contexts, the diagonal kernel $\kappa_{\mathcal{Z}}(\mathbf{z}, \mathbf{z}') = \mathbf{1}_{[\mathbf{z}=\mathbf{z}']}$ can be used. When features are explicitly available, we can use linear kernels, other kernels defined over Euclidean spaces or combinations thereof. In a special case of CGPRANK, we can recover the exact algorithms presented presented in [4] and [18] by choosing appropriate linear kernels. We employ CGPRANK with both diffusion and linear kernels and demonstrate their performance. However, in several real world applications, features are not easily available either for the contexts or the items and the nature of CGPRANK allows us to use any kind of kernel that can be computed from the similarity information that is available.

4.3 Explore-Exploit in List Selection

How should we use the predictive model (Equations (3) and (4)) to make recommendations? One approach could be to greedily maximize the expected reward according to our current model (i.e., rank items in order of their predictive mean (3)). However, this approach ignores the predictive uncertainty (4). If our goal were to conduct experiments to most effectively reduce uncertainty about the model, we may instead consider to pick items according to the predictive variance (4). Such an approach however would incur much regret, since it would equally explore high- and low value items. Therefore, in each step, we must trade off experimentation (showing items we have not explored yet) and exploitation (showing items with high expected reward). One way to achieve this is linearly trade off the relative importance of the predictive mean and the predictive variance to score each candidate item, i.e., select the item $\mathbf{s}$ that maximizes, for the current context $\mathbf{z}_t$ the surrogate objective $UCB_{\mathbf{s},\mathbf{z}_t}$, where

$$UCB_{\mathbf{s},\mathbf{z}_t} = \mu_{t-1}(\mathbf{s}, \mathbf{z}_t) + \beta_{t,\mathbf{s}}^{1/2} \sigma_{t-1}(\mathbf{s}, \mathbf{z}_t). \quad (5)$$

[2] using noise variance 1, according to our assumptions

For Gaussian predictive distributions, this criterion captures an upper confidence bound (UCB), i.e., an upper bound on the relevance function that holds with a certain probability that can be controlled via the tradeoff factor $\beta_{t,\mathbf{s}}$. I.e., the respective weighting of the mean and variance is handled by an item and time dependent variable, $\beta_{t,\mathbf{s}}$. We show how to pick $\beta_{t,\mathbf{s}}$ in Section 4.4 such that, with high probability, the UCB provides a valid upper bound on the true mean. At the same time, the choice is small enough so that the instantaneous regret provably decreases quickly over iterations.

In order to pick multiple items in each round, a first attempt would be to score every item $\mathbf{s}$ according to the selection rule (5), and select the b highest scoring items. However, given the regularity imposed by the kernel function, for a fixed context $\mathbf{z}$, the highest scoring items are likely very similar. Thus, the resulting list will explore sets of highly related items together, in a possibly redundant manner.

Instead, it may be desirable to encourage diversity when selecting lists to explore. One natural, and computationally efficient way is to *anticipate the reduction in uncertainty* achieved by the items already selected. Looking at the predictive mean and variance μ_t and σ_t in Equations (3) and (4), it can be observed that, while μ_t depends on the actual feedback $\bar{\mathbf{y}}_t$ observed so far, the predictive variance σ_t^2 does *not* depend on previous feedback. We can utilize this insight in the following way. Suppose, in round t, we receive context $\mathbf{z}_t$, and wish to recommend a list $\mathcal{L}_t = [\mathbf{s}_t^{[1]}, \ldots, \mathbf{s}_t^{[b]}]$. We select the first item $\mathbf{s}_t^{[1]}$ according to (5). Then, we update the predictive variance (4) *as if we had already observed* the feedback for the first item. The predictive mean is not updated (or equivalently, it is updated with its own prediction). Note that this will have the effect that the predictive variance – and hence the score (5) – for similar items is decreased. We now select the second item $\mathbf{s}_t^{[2]}$ according to the updated score, and proceed in this manner until the full list of b items has been selected.

After the ranked list has been selected, feedback $\mathbf{y}_t = [y_t^{[1]}, \ldots, y_t^{[b]}]$ is observed. According to our factorization assumption (2), each observation $y_t^{[i]}$ in position i provides a noisy observation of the underlying relevance score $\mathbb{E}[y_t^{[i]}] = p(i) \cdot r(\mathbf{s}_t^{[i]}, \mathbf{z}_t)$. Hence, we feed back $y_t^{[i]}/p(i)$ as unbiased estimate of $r(\mathbf{s}_t^{[i]}, \mathbf{z}_t)$.

4.4 Computing the Tradeoff Parameter

We now describe how to compute a value for $\beta_{t,\mathbf{s}}$ that allows us to prove rigorous bounds on the regret of CGPRANK. Note that in practice a more aggressive choice than this conservative prescription can lead to faster convergence. Our algorithm extends and generalizes the work of [24, 15, 8]. In these, the tradeoff parameter β_t ensures that, in each iteration, the true relevance function is contained within the constructed confidence bands $(\mu_t(\mathbf{s}) \pm \beta_t \sigma_t(\mathbf{s}))$ with high probability. Similarly, for our problem, we compute $\beta_{t,\mathbf{s}}$ as

$$\beta_{t,\mathbf{s}} = C_b' \Big(2M^2 + 300 \ln^3\Big(\frac{tb}{\delta}\Big) \Big(C_t + \frac{1}{2} \log(1+\sigma^{-2}\sigma_{t-1}^2(\mathbf{s}, \mathbf{z}_t)) \Big) \Big) \tag{6}$$

$$\text{where } C_b' = \frac{1}{p(b)^2} \max_{L \subseteq S \times \{\mathbf{z}_t\}; |L| = b} \frac{1}{2} \mid \mathbb{I} + \sigma^{-2} \mathbf{K}_{t-1}(L, L) \mid$$

$$C_t = \frac{1}{2} \sum_{\tau=1}^{t-1} \sum_{i=1}^{b} \log(1 + \sigma^{-2} \sigma_{\tau-1,i}^2(\mathbf{s}_\tau^{[i]}, \mathbf{z}_\tau)).$$

Hereby M is a bound on the RKHS norm of the reward function r, and $\sigma_{\tau-1,i-1}^2(\mathbf{s}_\tau^{[i]}, \mathbf{z}_\tau)$ is the predictive variance after having selected items 1 to $i-1$ in iteration τ. Note that C_t can be computed efficiently incrementally over the course of the algorithm. C_b' depends on the maximum determinant of any (posterior) kernel matrix $\mathbf{K}_{t-1}(L, L)$ that can be constructed using at most b items paired with the current context. While computing this quantity exactly requires solving a combinatorial optimization problem, it can be approximated efficiently and accurately during each iteration by running a simple greedy algorithm (uncertainty sampling) – see the longer version of the paper for details. For several commonly used kernel functions (linear, Gaussian and combinations thereof), $\beta_{t,\mathbf{s}}$ can be tightly bounded by the simple expression $\beta_t = C \log^{d'} t$ with suitable constants C, d'. It is this form that we use in the experiments.

Algorithm 1 presents pseudo-code for our CGPRANK algorithm. The procedure `GP-Inference`(κ, D) takes a kernel function κ and data set D, and returns the predictive mean and variance functions according to (3) and (4).

4.5 Regret Analysis of CGPRANK

Our analysis builds on and extends results of [15] for contextual GP bandit optimization (selecting individual items) and [8] for non-contextual GP bandit optimization with delayed feedback. We state our main result in the form of the following theorem and for reasons of space, reserve the details of the proof to a longer version of this paper.

THEOREM 1. *Let $\delta \in (0,1)$, $M > 0$, and κ be a kernel function, such that $||r||_\kappa \leq M$. In each round, choosing $\beta_{t,\mathbf{s}}$ as specified in Equation 6 and running* CGPRANK *for T rounds, it holds that*

$$Pr\{R_T \leq \frac{\sqrt{Tb\gamma_{Tb}(C_1 + C_2\gamma_{Tb}\ln^3(Tb/\delta))}}{p(b)} \forall T \geq 1\} \geq 1 - \delta$$

where $C_1 = \frac{16\exp(2\gamma_b)M^2}{\log(1+\sigma^{-2})}$, $C_2 = 300$ and

$$\gamma_n = \max_{\mathcal{D} \subseteq \mathcal{S} \times \mathcal{Z}, |\mathcal{D}| \leq n} \log |\mathbb{I} + \sigma^{-2} \mathbf{K}(\mathcal{D}, \mathcal{D})|$$

The regret bound in Theorem 1 depends on the quantity γ_n, which quantifies the effective degrees of freedom of the kernel matrix $\mathbf{K}(\mathcal{D}, \mathcal{D})$ that can be constructed from n context-item pairs. This quantity was analyzed in prior work [24], showing that for many common kernels (such as linear and Gaussian), γ_n only grows polylogarithmically in n. How can Theorem 1 be interpreted? It is instructive to consider the average regret per list slot, R_T/Tb. We can infer that, as long as γ_n grows only polylogarithmically in n (the common case),

$$\frac{R_T}{Tb} = O\Big(\frac{p(1)}{p(b)}\sqrt{\frac{\gamma_{Tb}M^2\exp(2\gamma_b)}{Tb}}\Big) = O^*\Big(\frac{M}{p(b)}\sqrt{\frac{\exp(2\gamma_b)}{Tb}}\Big),$$

where the O^* notation hides logarithmic factors in T and b. Thus, for fixed list size b, the average regret per slot decays to 0 at an essential rate of $O^*(\frac{1}{\sqrt{T}})$. It grows linearly with the complexity M of the reward function r, and inversely proportional to the decay of the position effect $p(b)$.

How does the regret scale with list size? Since $\exp(\gamma_b) = \Omega(b)$, as the list size b increases, straightforward application of the algorithm will incur average regret per slot that increases with b. However, in the non-contextual case (or the case of a finite set of contexts), it is possible to slightly modify the algorithm, such that, as long as $b = O(\log T)$, it can be ensured that γ_b remains bounded *irrespective* of b, at the cost of additional regret bounded by $O(\text{poly}\log(T))$. Thus, in this setting, one can achieve an average regret per slot of

$$\frac{R_T}{Tb} = O^*\Big(\frac{M}{p(b)}\sqrt{\frac{1}{Tb}}\Big). \tag{7}$$

Algorithm 1 The CGPRANK algorithm

Input: Kernel κ, selection batch size, b
Initialize data set of observations $D = \{\}$.
for $t = 1, 2, \ldots, T$ **do**
  Observe context $\mathbf{z}_t \in \mathcal{Z}$
  Receive set of available items $\mathcal{S}_t$
  Set $\hat{D} \leftarrow D$
  for $i = 1, 2, \ldots, b$ **do**
    $[\mu(\cdot), \sigma^2(\cdot)] \leftarrow$ GP-Inference$(\kappa, \hat{D})$
    $\mathbf{s}_t^{[i]} \leftarrow \text{argmax}_{\mathbf{s} \in \mathcal{S}_t} \mu(\mathbf{s}, \mathbf{z}_t) + \beta_{t,\mathbf{s}}^{1/2} \sigma(\mathbf{s}, \mathbf{z}_t)]$
    $\hat{D} \leftarrow \hat{D} \cup \{(\mathbf{s}_t^{[i]}, \mathbf{z}_t, \mu(\mathbf{s}_t^{[i]}, \mathbf{z}_t))\}$
  end for
  Recommend list $\mathcal{L}_t = [\mathbf{s}_t^{[1]}, \ldots, \mathbf{s}_t^{[b]}]$
  Observe feedback $\mathbf{y}_t = [y_t^{[1]}, \ldots, y_t^{[b]}]$
  $D \leftarrow D \cup \{(\mathbf{s}_t^{[1]}, \mathbf{z}_t, y_t^{[1]}/p(1)), \ldots, (\mathbf{s}_t^{[b]}, \mathbf{z}_t, y_t^{[b]}/p(b))\}$
end for

position loop

This result suggests that, perhaps surprisingly, as long as $p(b) \geq 1/\sqrt{b}$, increasing the list size can lead to *faster* convergence. This finding is further supported by our experimental results in Section 7.

5. SCALING CGPRANK TO WEB SCALE RECOMMENDATION TASKS

Naively implementing Algorithm 1 can be prohibitively slow for large data sets. For general kernels, the data set size $\mathcal{D}$ grows with the number of observations Tb, and performing exact Bayesian inference according to Equations (3) and (4) requires solving linear systems in Tb variables.

Scaling GP Inference. Fortunately, much work has been done scaling GP inference to massive data sets [20], also in online/streaming settings [10]. Since such inference is the essential subroutine in CGPRANK, it can immediately benefit from these techniques. Furthermore, in many practical applications (such as the recommendation tasks considered in our experiments), the kernel κ is of bounded rank d, in which case inference only requires solving a linear system in d dimensions. Often, approximate solution is acceptable for practical performance.

Speeding up selection. In order to speed up the selection rule (5), another computational trick can dramatically accelerate performance. Note that, in order to evaluate (5) naively, the mean $\mu_{t-1}(\mathbf{s}, \mathbf{z}_t)$ and variance $\sigma_{t-1}^2(\mathbf{s}, \mathbf{z}_t)$ has to be computed for each choice of $\mathbf{s} \in \mathcal{S}_t$. Inspecting Equations (3) and (4), it can be seen that computing (3) requires solving only one linear system, while computing (4) requires solving $|\mathcal{S}_t|$ linear systems. By exploiting the fact that, in GPs, predictive variance must monotonically decrease, i.e., $\sigma_t^2(\mathbf{s}, \mathbf{z}) \geq \sigma_{t+1}^2(\mathbf{s}, \mathbf{z})$, previous estimates can be used as upper bounds. This insight allows to use priority queues to dramatically reduce the number of linear systems that need to be solved. Similar ideas have been exploited in [8].

Delaying feedback. Instead of continuously performing updates, CGPRANK can be accelerated by reusing the same recommendation multiple times, accumulating feedback and performing delayed updates. However, delaying feedback for long periods of time can incur higher regret in the intermediate period where a fixed suboptimal ordering is chosen. Hence, careful choice of the frequency depending on the problem domain and taking into consideration speed of accumulation of feedback is an important aspect of scaling up CGPRANK.

For instance, exploiting some of these techniques, in our experiments with the Yahoo! dataset and using linear kernels, we were able to achieve an average selection time of 0.4 millisecond per slot including updating the model based on non-delayed feedback (timed on unoptimized C++ code compiled using gnu compiler and running on a single core of a Quad Core Intel Xeon E3, 3.5GHZ machine with 32GB RAM).

6. EXPERIMENTS

We extensively evaluate CGPRANK on two real-world recommendation tasks. The following questions guide our experimental study:

1. Can we exploit similarity to achieve accelerated convergence?
2. Can one parallelize exploration across lists to achieve faster convergence?
3. Can improved performance be achieved by incorporating context?

Benchmarks. In our experiments, we use the following approaches and compare the performance:

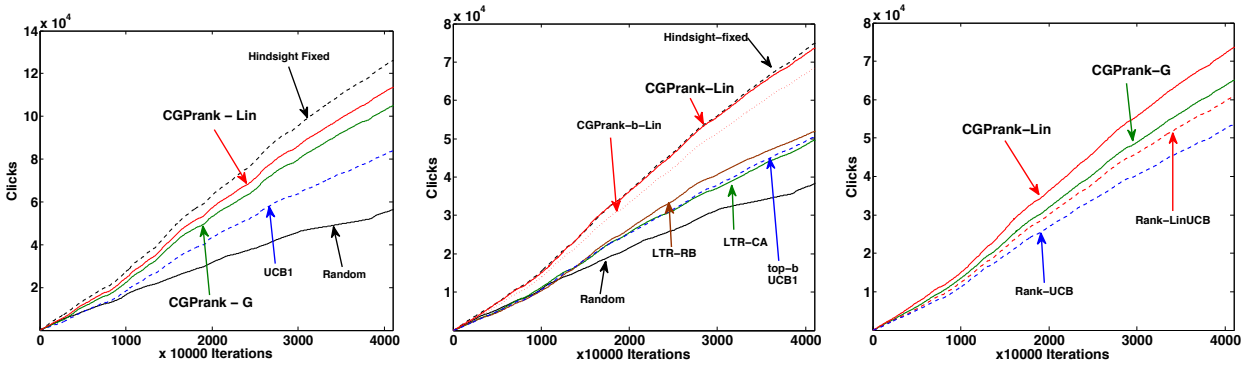
- CGPRANK-Lin, as described in Section 4. This builds on the LINUCB-HYBRID algorithm for single-item selection discussed in [17], which can be seen as a special case of CGPRANK.
- CGPRANK-G, is the version of CGPRANK with graph kernels on the items and clustering of contexts to model similarity. and can be used when we do not have access to user or item features. We use this version of CGPRANK for the experiments on Google books data.
- CGPRANK-b-Lin, which simply selects the top b items according to score (5)
- UCB1 of [3]. For list selection, we pick the top b items according to the UCB score. We also use a clustered version where we maintain an independent instance of the algorithm per cluster of contexts.
- Hierarchical versions of both UCB1 and CGPRANK (RANK-UCB and RANK-LINUCB), based on [26, 25].
- Learning to Rank Approaches: These are non-adaptive baselines that use a fraction of the data to train the model and then provide a ranking for every user request. In particular, we compare against two best performing algorithms from the RankLib module of the Lemur Toolkit software [3] (Coordinate Ascent (LTR-CA) [19] and Rank Boost (LTR-RB) [9])
- *Random* selection of lists.
- *Hindsight-Fixed* selection: Picking lists that are optimized in hindsight. This is an (unrealistic) upper bound benchmark.

6.1 Yahoo! news article recommendation

Data set. We first evaluate our algorithm on clickstream data made available by Yahoo! as part of the Yahoo! Webscope program [1]. Specifically, we use the R6A dataset containing a part of the user view/click log for articles displayed on the Today Module of Yahoo! during ten contiguous days. This data was collected in May 2009 and the displayed article was chosen uniformly at random from the list of available articles. This makes this dataset ideal for unbiased, offline evaluation of exploration–exploitation approaches. One can find detailed information on the dataset, the data collection methodology and an explanation of the unbiased offline evaluation in [1, 18, 5]. The dataset consists of more than 45 million lines of log. Each line consists of the following information:

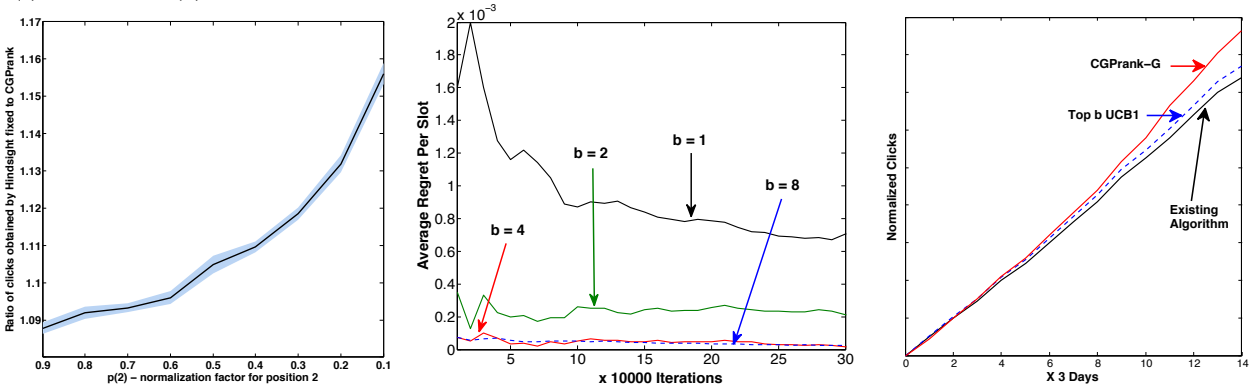
- The timestamp of the user visit.

[3] http://http://www.lemurproject.org/



(a) *Article selection on Yahoo! News* (b) *Clicks over time, b=4* (c) *Clicks over time, b=4*

Figure 1: (a): Total clicks received by the algorithms when employed to select a single article per round, using unbiased estimates from log data. (b): Contextual list selection task with $b = 4$. These results are based on click feedback simulated according to the logs. (c): Similar to 1(b). Contextual list selection task with $b = 4$.



(a) *Position effects for b=2* (b) *Average regret per slot* (c) *Performance on Google logs*

Figure 2: (a) Effect of $p(b)$ on the total regret demonstrated for a $b = 2$ item selection task. The regret increases as $p(2)$ decreases, but not dramatically so. (b) demonstrates the power of parallelizing exploration within lists to achieve accelerated convergence. In terms of the per-slot average regret, it shows how the relative performance compared to *Ideal* – perhaps surprisingly – *increases* with b (also compare Section 4.5). (c): Results of using CGPRANK-G to rerank recommendations on a clickstream log from Google's ebooks store. For confidentiality, we only present normalized numbers. Note that CGPRANK-G offers a significant ($\sim$18%) improvement over the existing non-adaptive method and also outperforms top-b UCB1.

- The article ID of the actual displayed article and whether a click was recorded or not.
- Anonymized user features denoting the context
- Article features for all the articles that were available in the pool for selection

Parameter choice. We initially chose the first 10% of the log entries as initialization data for optimizing parameters of the algorithms, training the learning-to-rank methods, and also to extract user features for clustering (see below). We used the results of this clustering for the contextual versions of CGPRANK-G and UCB1. We ignored this part of the data for all further evaluations the results reported are completely evaluated on the remaining 90% of the log entries.

The *hindsight-fixed* benchmark for these experiments used a weight vector obtained by solving to a linear regression problem for the click prediction task based on the entire log. The result is a single weight vector that maps any item-context pair $(\mathbf{s}, \mathbf{z})$ to an expected click probability.

The nature of the data set makes linear kernels the ideal choice for this task. With this choice, CGPRANK-Lin for single-item selection corresponds to the LINUCB-HYBRID algorithm as provided in [18]. In order to use CGPRANK-G on this dataset, we require a kernel function on the articles. We decided to model the articles as nodes in a graph. The weight on the edge connecting any two articles is simply the euclidean distance between their feature vectors. This choice allows us to compute a diffusion kernel on the articles. Note that computation of an appropriate diffusion kernel requires tuning of the heat parameter α. For our evaluations, we used the article features and their corresponding clickthrough rates to tune the parameter alpha α on the first 10% of the data.

For contextual versions of CGPRANK-G and UCB1, we used a simple technique of clustering the user features given in the logs, and maintaining one instance of all the evaluated algorithms per cluster (corresponding to a diagonal kernel $\kappa_{\mathcal{Z}}$). We used k-means clustering on the user contexts extracted from the initialization bucket with $k = 10$, picking the best solution from multiple random restarts. During the actual evaluation, in each round t, the user context given in the log line was mapped to its nearest cluster center, and $\mathbf{z}_t$ was set to this cluster index.

We carry out experiments both in the contextual and non-contextual setting, and vary the size of the lists selected. For list size $b = 1$, we use the actual click feedback given by the log. For $b > 1$, we use simulated click feedback as described above. In order to maintain consistency over the amount of feedback available, we randomly sample portions of the actual log during our simulated feedback since the rejection sampling technique used for $b = 1$ provides feedback once in 20 iterations on an average.

Feedback. While the goal of our work was to choose the optimal ordered list for recommendation, this dataset only contains click stream data for the choice of a single item. Hence, for the purpose of evaluating the list selection procedures, we simulated list feedback. The feedback for an article at a given position depended on the base Clickthrough rate (CTR hereafter) of the article and the bias introduced by the position. Hence, given context $\mathbf{z}$, if an article $\mathbf{s}$ with base CTR $r(\mathbf{s}, \mathbf{z})$ was shown at position j with a bias of $p(j)$, then the stochastic feedback for this placement was simulated as a Bernoulli draw with click probability $r(\mathbf{s}, \mathbf{z})p(j)$. Estimating the positional effects $p(j)$ on CTR is a well studied problem. The base CTR $r(\mathbf{s}, \mathbf{z})$ used in the simulation was computed as the CTR predicted by the *hindsight-fixed* algorithm for the given $(\mathbf{s}, \mathbf{z})$ pair.

6.2 Google e-books recommendation

Data set. We carry out our second set of experiments on clickstream logs from the Google ebooks store. Here, the recommendation task is, given a key book (context) the user is currently exploring, recommend a set of related books that the user may also be interested in. At the time of this work, Google used metadata information about the books and also inputs from other sources to compute the ordering of the related list of books to any given key book. This is a good first approximation of true "relatedness" in the absence of any real click data. But, as we receive feedback in terms of clicks on the recommendations, we can modify the original ordering in order to reflect the tastes of the users and this new ordering represents the true "relatedness" of books in the presence of a large number of clicks. In this dataset, the only context available was the current item being viewed (key item).

We evaluated our algorithm on the clicklog data of Google's book store that was collected over 42 days in the beginning of 2012. Each event in the anonymized click log data consists of two components:

- The volume id, identifying the key book (anchor item);
- The position of the related book on which the user clicked in the related list.

We estimated the unbiased position effect on the CTR using standard techniques.

Parameter choice. For each key book $\mathbf{z}$, we created a graph structure capturing the initial ordering given by the metadata-similarity in terms of the edge weights. Because of computational considerations, we only consider the similarity between the key book $\mathbf{z}_t$ and all of its candidate books $\mathcal{S}_t$, but not the similarities between the candidate books themselves. This results in a star graph with the key book in the center. The weights on the edges are the similarity scores between the books as computed using the metadata of the books. Using the obtained related graph G, the diffusion kernel K can be computed using techniques presented in [13].

Feedback. Based on the data, we simulate feedback for each item when it was displayed in a specific related list. Note that the clicks are being aggregated over users and sessions such that we group feedback on a specific related list. Position independent base CTR models how much the users prefer seeing a related book $\mathbf{s}$ in the recommendation list of key item $\mathbf{z}$. We define this CTR as the number of position-normalized clicks that item $\mathbf{s}$ got while being shown in the related list for key book $\mathbf{z}$, divided by the position-normalized number of times $\mathbf{s}$ was shown as a related book for $\mathbf{z}$.

Based on these estimates, we use offline evaluation techniques to simulate feedback for any new ordering. Since we computed the position independent feedback for each of the items in the original list and we also have the position weight terms $p(j)$, given context $\mathbf{z}$, we simulate feedback for any item $\mathbf{s}$ with base CTR $r(\mathbf{s}, \mathbf{z})$ at position j with position weight $p(j)$ by sampling from a Bernoulli distribution with bias $r(\mathbf{s}, \mathbf{z})p(j)$.

7. RESULTS AND DISCUSSION

Performance comparison. The results on the Yahoo! webscope dataset presented in Figure 1(a) and the results from the Google books evaluation presented in Figure 2(c) show that all versions of CGPRANK offers a consistent performance improvement over approaches that do not take item similarity into account. The ability of CGPRANK to generalize feedback received from few items to a larger set of related items allows it to quickly estimate their relevance. Also, in a dynamic system where new items regularly become available for selection, this feature of CGPRANK allows it to reliably estimate relevance of new items faster. Thus, CGPRANK is well suited to handle the cold start problem in recommendations. For the Yahoo! Webscope dataset, CGPRANK produced an overall final CTR of 0.0496 for the context-free setting and 0.0603 for the contextual setting, which compares favorably with the Ideal policy (0.0559 and 0.064). In the case of the Google books dataset, CGPRANK outperformed the then-existing algorithm by a margin of 18%. These findings substantiate our first hypothesis, that sharing feedback across similar items helps.

Performance without features. We decided to further test the performance of CGPRANK in settings where no explicit features are available. Instead, similarity information was provided in the form of a kernel function and the similarity between contexts was taken into account by clustering. The evaluation of this is also presented in Figures 1(a) and 1(c). While the overall performance of CGPRANK-G was outperformed by CGPRANK-Lin (CTR of 0.0574 compared to 0.0603 for single item selection), it still manages to perform well and is applicable even when explicit features are not present.

Figures 1(b) and 1(c) present the results of the list selection task with $b = 4$. From the plots, it can be inferred that adaptively learning the order is better than any fixed model learnt from training data. It can also be seen that even using context in an arguably naive manner (in terms of clustering users) provides a substantial improvement over not using context. For the single article selection case with actual click feedback, there was a 14% increase in the CTR while utilizing context information, further substantiating our hypothesis that exploiting context helps.

Relaxing the independence assumption. CGPRANK assumes that items do not influence the feedback of other items within a list. This simplifies the algorithm and its analysis, but is not necessarily true in practice. Hence, we relax this assumption by clustering the articles and model the user as diversity-preferring by ensuring that at most one article from a cluster is clicked on in a round. The recommended list might still contain multiple items from the same cluster. Although the total regret is $\sim 5\%$ more than in the independent case, CGPRANK still outperforms all other baselines and is better than the next best baseline by $\sim$10%.

Parallelizing exploration within lists. In our analysis in Section 4.5, we found that, perhaps surprisingly, increasing the list size can lead to accelerated convergence – at least under certain technical assumptions – as exploration is "parallelized" across list slots. We empirically assess this finding in Figure 2(b) which considers the per-slot average regret. In this experiment, we apply CGPRANK on the log data, using different batch sizes b. As b increases, faster convergence is obtained in relative terms compared to the *hindsight-fixed* predictions.

The experiments with multiple item selection corroborate our theoretical claims that having to select multiple items is beneficial if we consider the per slot regret as long as we gather enough feedback in the lower ranked slots. From the figures, it can be seen that

while average regret per slot decreases as we move from single item selection to 2 and then 4 items, there is diminishing returns when we select 8 items. This is due to the low position CTR at positions higher than 4 and also the sparse nature of feedback in the problem. As long as $p(i)$ is high enough to garner enough feedback, the opportunity cost incurred by making mistakes down the order is less than that at the top of the list. This is because of the decreasing expected CTR $p(i)$ as position i increases. During the actual execution, it can be noticed that CGPRANK quickly settles on the top positions while continuing to experiment with different articles down the order.

To assess the quantitative dependence of the regret on the smallest CTR $p(b)$, we conduct an experiment varying $p(2)$ (for $b = 2$), shown in Figure 2(a). We note that the ratio of ideal clicks to clicks garnered by CGPRANK increases as $p(2)$ decreases. While our theoretical results suggests a much stronger dependence on $p(b)$, the effect is not as dramatic in the experiments. This is explained by the fact that our bounds are high probability bounds and in reality, the feedback is more benign.

8. CONCLUSION

We have developed a novel algorithm – CGPRANK– for efficiently reranking lists to reflect user preferences over the items displayed, taking context into account. It exploits assumptions on similarity of items and contexts as given by a positive definite kernel function, and separability assumptions on position dependence. In this way, CGPRANK is able to share sparse feedback across positions, items and contexts. We proved strong theoretical guarantees on its regret, indicating that, perhaps surprisingly, under natural assumptions, parallelization of exploration across lists can help accelerate convergence. Support for computational and statistical parallelism makes it a suitable candidate for adoption for large-scale online recommendation engines. We extensively evaluate CGPRANK on two real world recommendation tasks. On the Google ebooks recommendation task, CGPRANK achieves 18% click lift over the previous state of the art recommender system. We also showed significant improvements over state-of-the-art bandit and learning-to-rank approaches on the Yahoo! Webscope dataset. The experiments with the Yahoo! dataset demonstrates empirically our claim about being able to exploit similarity between items and context information and also shows that parallelizing exploration across the list leads to better performance compared to the single item selection problem. We believe our results present an important step towards addressing challenging, large-scale exploration–exploitation tradeoffs in practical recommender systems.

9. ACKNOWLEDGMENTS

This research was supported in part by SNSF grant 200021_137971, ERC StG 307036 and a Microsoft Research Faculty Fellowship.

References

[1] Yahoo! webscope program. `http://webscope.sandbox.yahoo.com`. Accessed: 20/02/2013.

[2] E. Agichtein, E. Brill, and S. Dumais. Improving web search ranking by incorporating user behavior information. In *ACM SIGIR*, 2006.

[3] P. Auer, N. C. Bianchi, and P. Fischer. Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning*, 47(2-3):235–256, May 2002.

[4] W. Chu, L. Li, L. Reyzin, and R. E. Schapire. Contextual bandits with linear payoff functions. In *AISTATS*, 2011.

[5] W. Chu, S.-T. Park, T. Beaupre, N. Motgi, A. Phadke, S. Chakraborty, and J. Zachariah. A case study of behavior-driven conjoint analysis on yahoo!: front page today module. In *ACM SIGKDD*, 2009.

[6] N. Craswell, O. Zoeter, M. Taylor, and B. Ramsey. An experimental comparison of click position-bias models. In *WSDM*, 2008.

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

[8] T. Desautels, A. Krause, and J. Burdick. Parallelizing exploration-exploitation tradeoffs with gaussian process bandit optimization. In *ICML*, 2012.

[9] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. *J. Mach. Learn. Res.*, 4:933–969, Dec. 2003.

[10] R. Gomes and A. Krause. Budgeted nonparametric learning from data streams. In *ICML*, 2010.

[11] S. Kale, L. Reyzin, and R. E. Schapire. Non-stochastic bandit slate problems. In *NIPS*, 2010.

[12] R. Kleinberg, A. Slivkins, and E. Upfal. Multi-armed bandits in metric spaces. In *STOC*, 2008.

[13] R. I. Kondor and J. Lafferty. Diffusion kernels on graphs and other discrete structures. In *ICML*, 2002.

[14] W. M. Koolen, M. K. Warmuth, and J. Kivinen. Hedging structured concepts. In *COLT*, 2010.

[15] A. Krause and C. S. Ong. Contextual gaussian process bandit optimization. In *NIPS*, 2011.

[16] R. Lempel. Recommendation challenges in web media settings. In *RecSys*, 2012.

[17] L. Li, W. Chu, J. Langford, and R. E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *WWW*, 2010.

[18] L. Li, W. Chu, J. Langford, and X. Wang. Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms. In *WSDM*, 2011.

[19] D. Metzler and W. Bruce Croft. Linear feature-based models for information retrieval. *Inf. Retr.*, 10(3):257–274, June 2007.

[20] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[21] F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, editors. *Recommender Systems Handbook*. Springer, 2011.

[22] B. Scholkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA, 2001.

[23] A. Slivkins, F. Radlinski, and S. Gollapudi. Learning optimally diverse rankings over large document collections. In *ICML*, 2010.

[24] N. Srinivas, A. Krause, S. Kakade, and M. Seeger. Information-theoretic regret bounds for gaussian process optimization in the bandit setting. *IEEE Transactions on Information Theory*, 58(5):3250–3265, May 2012.

[25] M. Streeter, D. Golovin, and A. Krause. Online learning of assignments. In *NIPS*, 2009.

[26] Y. Yue and C. Guestrin. Linear submodular bandits and their application to diversified retrieval. In *NIPS*, 2011.