Cascading Bandits for Large-Scale Recommendation Problems

Shi Zong
Dept of Electrical and Computer Engineering
Carnegie Mellon University
szong@andrew.cmu.edu

Hao Ni
Dept of Electrical and Computer Engineering
Carnegie Mellon University
haon@cmu.edu

Kenny Sung
Dept of Electrical and Computer Engineering
Carnegie Mellon University
tsung@andrew.cmu.edu

Nan Rosemary Ke
Dépt d'informatique et de recherche opérationnelle
Université de Montréal
nke001@gmail.com

Zheng Wen
Adobe Research
San Jose, CA
zwen@adobe.com

Branislav Kveton
Adobe Research
San Jose, CA
kveton@adobe.com

Abstract

Most recommender systems recommend a list of items. The user examines the list, from the first item to the last, and often chooses the first attractive item and does not examine the rest. This type of user behavior can be modeled by the *cascade model*. In this work, we study *cascading bandits*, an online learning variant of the cascade model where the goal is to recommend K most attractive items from a large set of L candidate items. We propose two algorithms for solving this problem, which are based on the idea of linear generalization. The key idea in our solutions is that we learn a predictor of the attraction probabilities of items from their features, as opposing to learning the attraction probability of each item independently as in the existing work. This results in practical learning algorithms whose regret does not depend on the number of items L. We bound the regret of one algorithm and comprehensively evaluate the other on a range of recommendation problems. The algorithm performs well and outperforms all baselines.

1 INTRODUCTION

Most recommender systems recommended a list of K items, such as restaurants, songs, or movies. The user *examines* the recommended list from the first item to the last, and typically clicks on the first item that *attracts* the user. The *cascade model* [10] is a popular model to formulate this kind of user behavior. The items before the first clicked item are *not attractive*, because the user examines these items but does not click on them. The items after the first attractive item are *unobserved*, because the user never examines these items. The key assumption in the cascade model is that each item attracts the user independently of the other items. Under this assumption, the optimal solution in the cascade model, the list of K items that maximizes the probability that the user finds an attractive item, are K most attractive items. The cascade model is simple, intuitive, and surprisingly effective in explaining user behavior [7].

In this paper, we study on an online learning variant of the cascade model, which is known as *cascading bandits* [15]. In this model, the learning agent does not know the preferences of the user over recommended items and the goal is to learn them by interacting with the user. At time t, the agent recommends to the user a list of K items out of L candidate items and observes the click of the user. If the user clicks on an item, the agent receives a reward of one. If the user does not click on any item, the agent receives a reward of zero. The performance of the learning agent is evaluated by its cumulative reward in n steps, which is the total number of clicks in n steps. The goal of the agent is to maximize it.

Kveton *et al.* [15] proposed two computationally and sample efficient algorithms for cascading bandits. They also proved a $\Omega(L - K)$ lower bound on the regret in cascading bandits, which shows that the regret grows linearly with the

number of candidate items L. Therefore, cascading bandits are impractical for learning when L is large. Unfortunately, this setting is common practice. For instance, consider the problem of learning a personalized recommender system for $K = 10$ movies from the ground set of $L = 100$k movies. In this setting, each movie would have to be shown to the user at least once, which means at least 10k interactions with the recommender system, before the system starts behaving intelligently. Such a system would clearly be impractical. The main contribution of our work is that we propose *linear cascading bandits*, an online learning framework that makes learning in cascading bandits practical at scale. The key step in our approach is that we assume that the attraction probabilities of items can be predicted from the features of items. Features are often available in practice or can be easily derived.

To the best of our knowledge, this is the first work that studies a top-K recommender problem in the bandit setting with cascading feedback and context. Specifically, we make four contributions. First, we propose linear cascading bandits, a variant of cascading bandits where we make an additional assumption that the attraction probabilities of items are a linear function of the features of items. This assumption is the key step in designing a sample efficient learning algorithm for our problem. Second, we propose two computationally efficient learning algorithms, `CascadeLinTS` and `CascadeLinUCB`, which are motivated by *Thompson sampling (TS)* [23, 3] and *linear UCB* [1, 24], We believe this is the first application of linear generalization in the cascade model under partial monitoring feedback. Third, we derive an upper bound on the regret of `CascadeLinUCB` and discuss why a similar upper bound should hold for `CascadeLinTS`. Finally, we evaluate `CascadeLinTS` on a range of recommendation problems; in the domains of restaurant, music, and movie recommendations; and demonstrate that it performs well even when our modeling assumptions are violated.

Our paper is organized as follows. In Section 2, we review the cascade model and cascading bandits. In Section 3, we present linear cascading bandits; propose `CascadeLinTS` and `CascadeLinUCB`; and bound the regret of `CascadeLinUCB`. In Section 4, we evaluate `CascadeLinTS` on several recommendation problems. We review related work in Section 5 and conclude in Section 6.

To simplify exposition, we denote random variables by boldface letter. We define $[n] = \{1, \ldots, n\}$ and denote the cardinality of set A by $|A|$.

2 BACKGROUND

In this section, we review the cascade model [10] and cascading bandits [15].

2.1 Cascade Model

The *cascade model* [10] is a popular model of user behavior. In this model, the user is recommended a list of K items $A = (a_1, \ldots, a_K) \in \Pi_K(E)$, where $\Pi_K(E)$ is the set of all K-*permutations* of some *ground set* $E = [L]$, which is the set of all possibly recommended items. The model is parameterized by L *attraction probabilities* $\bar{w} \in [0, 1]^E$ and the user scans the list A sequentially from the first item a_1 to the last a_K. After the user examines item a_k, the item attracts the user with probability $\bar{w}(a_k)$, *independently* of the other items. If the user is attracted by item a_k, the user clicks on it and stop examining the remaining items. If the user is not attracted by item a_k, the user examines the next recommended item a_{k+1}. It is easy to see that the probability that item a_k is examined is $\prod_{i=1}^{k-1}(1 - \bar{w}(a_i))$, and that the probability that at least one item in A is attractive is $1 - \prod_{i=1}^{K}(1 - \bar{w}(a_i))$. This objective is maximized by K most attractive items.

The cascade model is surprising effective in explaining how users scan lists of items [7]. The reason is that lower ranked items typically do not get clicked because the user is attracted by higher ranked items, and never examines the rest of the recommended list.

2.2 Cascading Bandits

Kveton *et al.* [15] proposed a learning variant of the cascading model, which is known as a cascading bandit. Formally, a *cascading bandit* is a tuple $B = (E, P, K)$, where $E = [L]$ is a *ground set* of L items, P is a probability distribution over a binary hypercube $\{0, 1\}^E$, and $K \leq L$ is the number of recommended items.

The learning agent interacts with our problem as follows. Let $(\mathbf{w}_t)_{t=1}^n$ be an i.i.d. sequence of n *weights* drawn from P, where $\mathbf{w}_t \in \{0, 1\}^E$ and $\mathbf{w}_t(e)$ is the preference of the user for item e at time t. More precisely, $\mathbf{w}_t(e) = 1$ if and only if item e attracts the user at time t. At time t, the agent recommends a list of K items $\mathbf{A}_t = (\mathbf{a}_1^t, \ldots, \mathbf{a}_K^t) \in \Pi_K(E)$. The list is a function of the observations of the agent up to time t. The user examines the list, from the first item $\mathbf{a}_1^t$ to the last $\mathbf{a}_K^t$, and clicks on the first attractive item. If the user is not attracted by any item, the user does not click on any item. Then time increases to $t + 1$.

The reward of the agent at time t is one if and only if the user is attracted by at least one item in $\mathbf{A}_t$. Formally, the reward at time t can be expressed as $\mathbf{r}_t = f(\mathbf{A}_t, \mathbf{w}_t)$, where $f : \Pi_K(E) \times [0, 1]^E \to [0, 1]$ is a *reward function* and we define it as:

$$f(A, w) = 1 - \prod_{k=1}^{K} (1 - w(a_k))$$

for any $A = (a_1, \ldots, a_K) \in \Pi_K(E)$ and $w \in [0, 1]^E$. The

agent at time t receives feedback:

$$\mathbf{C}_t = \min\left\{k \in [K] : \mathbf{w}_t(\mathbf{a}_k^t) = 1\right\} ,$$

where we assume that $\min \emptyset = \infty$. The feedback $\mathbf{C}_t$ is the click of the user. If $\mathbf{C}_t \leq K$, the user clicks on item $\mathbf{C}_t$. If $\mathbf{C}_t = \infty$, the user does not click on any item. Since the user clicks on the first attractive item in the list, the observed weights of all recommended items at time t can be expressed as a function of $\mathbf{C}_t$:

$$\mathbf{w}_t(\mathbf{a}_k^t) = \mathbb{1}\{\mathbf{C}_t = k\} \quad k = 1, \ldots, \min\{\mathbf{C}_t, K\} . \quad (1)$$

Accordingly, we say that item e is *observed* at time t if $e = \mathbf{a}_k^t$ for some $k \in [\min\{\mathbf{C}_t, K\}]$.

Let the attraction weights of items in the ground set E be distributed independently as:

$$P(w) = \prod_{e \in E} \mathrm{Ber}(w(e); \bar{w}(e)) ,$$

where $\mathrm{Ber}(\cdot; \theta)$ is a Bernoulli distribution with mean θ. Then the expected reward for list $A \in \Pi_K(E)$, the probability that at least one item in A is satisfactory, can be expressed as $\mathbb{E}[f(A, \mathbf{w})] = f(A, \bar{w})$, and depends only on the attraction probabilities of individual items in A. Therefore, it is sufficient to learn a good approximation to $\bar{w}$ to act optimally.

The agent's policy is evaluated by its *expected cumulative regret*:

$$R(n) = \mathbb{E}\left[\sum_{t=1}^{n} R(\mathbf{A}_t, \mathbf{w}_t)\right] , \quad (2)$$

where $R(\mathbf{A}_t, \mathbf{w}_t) = f(A^*, \mathbf{w}_t) - f(\mathbf{A}_t, \mathbf{w}_t)$ is the *instantaneous stochastic regret* of the agent at time t and:

$$A^* = \operatorname*{arg\,max}_{A \in \Pi_K(E)} f(A, \bar{w})$$

is the *optimal list* of items, the list that maximizes the reward at any time t. For simplicity of exposition, we assume that the optimal solution, as a set, is unique.

2.3 Algorithm `CascadeUCB1`

Kveton *et al.* [15] proposed and analyzed two learning algorithms for cascading bandits, `CascadeUCB1` and `CascadeKL-UCB`. In this section, we review `CascadeUCB1`.

`CascadeUCB1` belongs to the family of UCB algorithms. The algorithm operates in three stages. First, it computes the *upper confidence bounds (UCBs)* $\mathbf{U}_t \in [0, 1]^E$ on the attraction probabilities of all items in E. The UCB of item e at time t is:

$$\mathbf{U}_t(e) = \hat{\mathbf{w}}_{\mathbf{T}_{t-1}(e)}(e) + c_{t-1, \mathbf{T}_{t-1}(e)} , \quad (3)$$

where $\hat{\mathbf{w}}_s(e)$ is the average of s observed attraction weights of item e, $\mathbf{T}_t(e)$ is the number of times that item e is observed in t steps, and:

$$c_{t,s} = \sqrt{(1.5 \log t)/s}$$

is the radius of a confidence interval around $\hat{\mathbf{w}}_s(e)$ after t steps such that $\bar{w}(e) \in [\hat{\mathbf{w}}_s(e) - c_{t,s}, \hat{\mathbf{w}}_s(e) + c_{t,s}]$ holds with high probability. Second, `CascadeUCB1` recommends a list of K items with largest UCBs:

$$\mathbf{A}_t = \operatorname*{arg\,max}_{A \in \Pi_K(E)} f(A, \mathbf{U}_t) .$$

Finally, after the user provides feedback $\mathbf{C}_t$, the algorithm updates its estimates of the attraction probabilities $\bar{w}(e)$ based on the observed weights of items, which are defined in (1) for all $e = \mathbf{a}_k^t$ such that $k \leq \mathbf{C}_t$.

3 LINEAR CASCADING BANDITS

Kveton *et al.* [15] showed that the n-step regret of `CascadeUCB1` is $O((L - K)(1/\Delta) \log n)$, where L is the number of items in ground set E; K is the number of recommended items; and Δ is the gap, which measures the sample complexity. This means that the regret increases linearly with the number of items L. As a result, `CascadeUCB1` is not practical when L is large. Unfortunately, this setting is common practice. For instance, consider the problem of learning a personalized recommender for 10 movies from the ground set of 100k movies. To learn, `CascadeUCB1` would need to show each movie to the user at least once, which means that the algorithm would require at least 10k interactions with the user to start behaving intelligently. This is clearly impractical.

In this work, we propose *practical algorithms* for large-scale cascading bandits, in the setting where L is large. The key assumption, which allows us to learn efficiently, is that we assume that the attraction probability of each item e, $\bar{w}(e)$, can be approximated by a linear combination of some known d-dimensional feature vector $x_e \in \mathbb{R}^{d \times 1}$ and an unknown d-dimensional parameter vector of $\theta^* \in \mathbb{R}^{d \times 1}$, which is shared among all items. More precisely, we assume that there exists $\theta^* \in \Theta$ such that:

$$\bar{w}(e) \approx x_e^\top \theta^* \quad (4)$$

for any $e \in E$. The features are problem specific and we discuss how to construct them in Section 4.3. We propose two learning algorithms, which we call *cascading linear Thompson sampling* (`CascadeLinTS`) and *cascading linear UCB* (`CascadeLinUCB`). We prove that when the above linear generalization is perfect, the regret of `CascadeLinUCB` is independent of L and sublinear in n. Therefore, `CascadeLinUCB` is suitable for learning to recommend from large ground sets E. We also discuss why a similar regret bound should hold for `CascadeLinTS`, though we do not prove this bound formally.

Algorithm 1 `CascadeLinTS`

Inputs: Variance σ^2

// Initialization
$\mathbf{M}_0 \leftarrow I_d$ and $\mathbf{B}_0 \leftarrow \mathbf{0}$

for all $t = 1, \ldots, n$ **do**
  $\bar{\theta}_{t-1} \leftarrow \sigma^{-2}\mathbf{M}_{t-1}^{-1}\mathbf{B}_{t-1}$
  $\theta_t \sim \mathcal{N}(\bar{\theta}_{t-1}, \mathbf{M}_{t-1}^{-1})$

  // Recommend a list of K items and get feedback
  for all $k = 1, \ldots, K$ **do**
    $\mathbf{a}_k^t \leftarrow \arg\max_{e \in [L] - \{\mathbf{a}_1^t, \ldots, \mathbf{a}_{k-1}^t\}} x_e^\intercal \theta_t$
  $\mathbf{A}_t \leftarrow (\mathbf{a}_1^t, \ldots, \mathbf{a}_K^t)$
  Observe click $\mathbf{C}_t \in \{1, \ldots, K, \infty\}$
  Update statistics using Algorithm 3

Algorithm 2 `CascadeLinUCB`

Inputs: Variance σ^2, constant c (Section 3.2)

// Initialization
$\mathbf{M}_0 \leftarrow I_d$ and $\mathbf{B}_0 \leftarrow \mathbf{0}$

for all $t = 1, \ldots, n$ **do**
  $\bar{\theta}_{t-1} \leftarrow \sigma^{-2}\mathbf{M}_{t-1}^{-1}\mathbf{B}_{t-1}$
  for all $e \in E$ **do**
    $\mathbf{U}_t(e) \leftarrow \min\left\{x_e^\intercal \bar{\theta}_{t-1} + c\sqrt{x_e^\intercal \mathbf{M}_{t-1}^{-1} x_e}, 1\right\}$

  // Recommend a list of K items and get feedback
  for all $k = 1, \ldots, K$ **do**
    $\mathbf{a}_k^t \leftarrow \arg\max_{e \in [L] - \{\mathbf{a}_1^t, \ldots, \mathbf{a}_{k-1}^t\}} \mathbf{U}_t(e)$
  $\mathbf{A}_t \leftarrow (\mathbf{a}_1^t, \ldots, \mathbf{a}_K^t)$
  Observe click $\mathbf{C}_t \in \{1, \ldots, K, \infty\}$
  Update statistics using Algorithm 3

Algorithm 3 Update of statistics in Algorithms 1 and 2

$\mathbf{M}_t \leftarrow \mathbf{M}_{t-1}$
$\mathbf{B}_t \leftarrow \mathbf{B}_{t-1}$
for all $k = 1, \ldots, \min\{\mathbf{C}_t, K\}$ **do**
  $e \leftarrow \mathbf{a}_k^t$
  $\mathbf{M}_t \leftarrow \mathbf{M}_t + \sigma^{-2} x_e x_e^\intercal$
  $\mathbf{B}_t \leftarrow \mathbf{B}_t + x_e \mathbb{1}\{\mathbf{C}_t = k\}$

3.1 Algorithms

Our learning algorithms are based on the ideas of Thompson sampling [23, 3] and linear UCB [1], and motivated by the recent work of Wen *et al.* [24], which proposes computationally and sample efficient algorithms for large-scale stochastic combinatorial semi-bandits. The pseudocode of both algorithms is in Algorithms 1 and 2, and we outline them below.

Both `CascadeLinTS` and `CascadeLinUCB` represent their past observations as a positive-definite matrix $\mathbf{M}_t \in \mathbb{R}^{d \times d}$ and a vector $\mathbf{B}_t \in \mathbb{R}^{d \times 1}$. Specifically, let $\mathbf{X}_t$ be a matrix whose rows are the feature vectors of all observed items in t steps and $\mathbf{Y}_t$ be a column vector of all observed attraction weights in t steps. Then:

$$\mathbf{M}_t = \sigma^{-2}\mathbf{X}_t^\intercal \mathbf{X}_t + I_d$$

is the *gram matrix* in t steps and:

$$\mathbf{B}_t = \mathbf{X}_t^\intercal \mathbf{Y}_t\,,$$

where I_d is a $d \times d$ identity matrix and $\sigma > 0$ is parameter that controls the learning rate.[1]

Both `CascadeLinTS` and `CascadeLinUCB` operate in three stages. First, they estimated the expected weight of each item e based on their model of the world. `CascadeLinTS` randomly samples parameter vector θ_t from a normal distribution, which approximates its posterior on θ^*, and then estimates the expected weight as $x_e^\intercal \theta_t$. `CascadeLinUCB` computes an upper confidence bound $\mathbf{U}_t(e)$ for each item e. Second, both algorithms choose the optimal list $\mathbf{A}_t$ with respect to their estimates. Finally, they receive feedback, and update $\mathbf{M}_t$ and $\mathbf{B}_t$ using Algorithm 3.

[1] Ideally, σ^2 should be the variance of the observation noises. However, based on recent literature [24], we believe that both algorithms will perform well for a wide range of σ^2.

We would like to emphasize that both `CascadeLinTS` and `CascadeLinUCB` are computationally efficient. In practice, we would update $\mathbf{M}_t^{-1}$ instead of $\mathbf{M}_t$. In particular, note that:

$$\mathbf{M}_t \leftarrow \mathbf{M}_t + \sigma^{-2} x_e x_e^\intercal$$

can be equivalently updated as:

$$\mathbf{M}_t^{-1} \leftarrow \mathbf{M}_t^{-1} - \frac{\mathbf{M}_t^{-1} x_e x_e^\intercal \mathbf{M}_t^{-1}}{x_e^\intercal \mathbf{M}_t^{-1} x_e + \sigma^2}\,,$$

and hence $\mathbf{M}_t^{-1}$ can be updated incrementally and computationally efficiently in $O(d^2)$ time. It is easy to to see that the per-step time complexities of both `CascadeLinTS` and `CascadeLinUCB` are $O(L(d^2 + K))$.

3.2 Analysis and Discussion

We first derive a regret bound on `CascadeLinUCB`, under the assumptions that (1) $\bar{w}(e) = x_e^\intercal \theta^*$ for all $e \in E$ and (2) $\|x_e\|_2 \leq 1$ for all $e \in E$. Note that condition (2) can be always ensured by rescaling feature vectors. The regret bound is detailed below.

Theorem 1. *Under the above assumptions, for any $\sigma > 0$ and any*

$$c \geq \frac{1}{\sigma}\sqrt{d \log\left(1 + \frac{nK}{d\sigma^2}\right) + 2\log(nK)} + \|\theta^*\|_2,$$

if we run `CascadeLinUCB` *with parameters* σ *and* c*, then*

$$R(n) \leq 2cK\sqrt{\frac{dn\log\left[1+\frac{nK}{d\sigma^2}\right]}{\log\left(1+\frac{1}{\sigma^2}\right)}}+1.$$

Note that if we choose $\sigma = 1$ and

$$c = \sqrt{d\log\left(1+\frac{nK}{d}\right)+2\log\left(nK\right)}+\eta,$$

for some constant $\eta \geq \|\theta^*\|_2$, then $R(n) \leq \tilde{O}\left(Kd\sqrt{n}\right)$ where the $\tilde{O}$ notation hides logarithmic factors.

The proof is in Appendix and we outline it below. First, we define event $\mathcal{G}_{t,k} = \{\text{item } \mathbf{a}_k^t \text{ is examined in step } t\}$ for any time t and $k \in [K]$, and bound the n-step regret as

$$R(n) \leq \mathbb{E}\left[\sum_{t=1}^{n}\sum_{k=1}^{K}\mathbb{1}\{\mathcal{G}_{t,k}\}\left[\bar{w}(\mathbf{a}_k^{*,t})-\bar{w}(\mathbf{a}_k^t)\right]\right],$$

where $\mathbf{a}_k^{*,t}$ is an optimal item in A^* matched to item $\mathbf{a}_k^t$ in step t. Second, we define an event

$$\mathcal{E} = \left\{\left|x_e^T(\bar{\theta}_{t-1}-\theta^*)\right| \leq c\|x_e\|_{\mathbf{M}_{t-1}^{-1}}\ \forall t \leq n,\ \forall e \in E\right\},$$

where $\|x_e\|_{\mathbf{M}_{t-1}^{-1}} = \sqrt{x_e^\intercal \mathbf{M}_{t-1}^{-1} x_e}$. Then we prove a high-probability bound $P(\mathcal{E}) \geq 1-1/nK$ for any c that satisfies the condition of Theorem 1. Finally, we show that by conditioning on $\mathcal{E}$, we have

$$\begin{aligned}
&\sum_{t=1}^{n}\sum_{k=1}^{K}\mathbb{1}\{\mathcal{G}_{t,k}\}\left[\bar{w}(\mathbf{a}_k^{*,t})-\bar{w}(\mathbf{a}_k^t)\right] \\
&\quad\leq 2c\sum_{t=1}^{n}\sum_{k=1}^{K}\mathbb{1}\{\mathcal{G}_{t,k}\}\left\|x_{\mathbf{a}_k^t}\right\|_{\mathbf{M}_{t-1}^{-1}} \\
&\quad\leq 2cK\sqrt{\frac{dn\log\left[1+\frac{nK}{d\sigma^2}\right]}{\log\left(1+\frac{1}{\sigma^2}\right)}},
\end{aligned}$$

where the first inequality follows from the definition of $\mathcal{E}$ and the second inequality follows from a worst-case bound. The bound in Theorem 1 follows from putting the above results together.

Recent work [21, 24] demonstrated close relationships between UCB-like algorithms and Thompson sampling algorithms in related bandit problems. Therefore, we believe that a similar regret bound to that in Theorem 1 also holds for `CascadeLinTS`. However, it is highly non-trivial to derive a regret bound for `CascadeLinTS`. Unlike in [24], `CascadeLinTS` cannot be analyzed from the Bayesian perspective because the Gaussian posterior is inconsistent with the fact that $\bar{w}(e)$ is bounded in $[0, 1]$. Moreover, a subtle statistical dependence between partial monitoring and Thompson sampling prevents a frequentist analysis similar to that in [4]. Therefore, we leave the formal analysis of `CascadeLinTS` for future work. It is well known that Thompson sampling tends to outperform UCB-like algorithms in practice [3]. Therefore, we only empirically evaluate `CascadeLinTS`.

4 EXPERIMENTS

We validate `CascadeLinTS` on several problems of various sizes and from various domains. In each problem, we conduct several experiments that demonstrate that our approach is scalable and stable with respect to its tunable parameters, the number of recommended items K and the number of features d.

Our experimental section is organized as follows. In Section 4.1, we outline the experiments that are conducted on each dataset. In Section 4.2, we introduce our metrics and baselines. In Section 4.3, we describe how we construct the features of items E. We present our empirical results in the rest of the section.

4.1 Experimental Setting

All of our learning problems can be viewed as follows. The feedback of users is a matrix $W \in \{0, 1\}^{m\times L}$, where row i corresponds to user $i \in [m]$ and column j corresponds to item $j \in E$. Entry (i, j) of W, $W_{i,j} \in \{0, 1\}$, indicates that user i is attracted by item j. The user at time t, the row of W, is chosen at random from the pool of all users. Our goal is to learn the list of items A^*, the columns of W, that maximizes the probability that the user at time t is attracted by at least one recommended item.

In each of our problems, we conduct a set of experiments. In the first experiment, we compare `CascadeLinTS` to baselines (Section 4.2) and also evaluate its scalability. We experiment with three variants of our problems: $L = 16$ items, $L = 256$ items, and the maximum possible value of L in a given experiment. The number of recommended items is $K = 4$ and the number of features is $d = 20$.

In the second experiment, we show that the performance of `CascadeLinTS` is robust with respect to the number of features d, in the sense that d affects the performance but `CascadeLinTS` performs reasonably well for all settings of d. We experiment with three settings for the number of features: $d = 10$, $d = 20$, and $d = 40$. The ground set contains $L = 256$ items and the number of recommended items is $K = 4$.

In the third experiment, we evaluate `CascadeLinTS` on an interesting subset of each dataset, such as *Rock Songs*. The setting of this experiment is identical to the second experiment. This experiment validates that `CascadeLinTS` can also learn to recommend items in the context, of a subset of the dataset.

In the last experiment, we evaluate how the performance of

CascadeLinTS varies with the number of recommended items K. We experiment with three settings for the number of recommended items: $K = 4$, $K = 8$, and $K = 12$. The ground set contains $L = 256$ items and the number of features is $d = 20$.

All experiments are conducted for $n = 100$k steps and averaged over 10 randomly initialized runs. The tunable parameter σ in CascadeLinTS is set to 1.

4.2 Metrics and Baselines

The performance of CascadeLinTS is evaluated by its expected cumulative regret, which is defined in (2). In most of our experiments, our modeling assumptions are violated. In particular, the items are not guaranteed to attract users independently because the attraction indicators $\mathbf{w}_t(e)$ are correlated across items e. The result is that:

$$A^* = \arg\max_{A \in \Pi_K(E)} \mathbb{E}\left[f(A, \mathbf{w})\right] > \arg\max_{A \in \Pi_K(E)} f(A, \bar{w}) \,.$$

It is NP-hard to find A^*, because $\mathbb{E}\left[f(A, \mathbf{w})\right]$ does not decompose into the product of expectations as we assume in our model (Section 2.2). However, since $\mathbb{E}\left[f(A, \mathbf{w})\right]$ is submodular and monotone in A, a $(1 - 1/e)$ approximation to A^* can be computed greedily, by iteratively adding items that attract most users that are not attracted by any previously added item. We denote this approximation by A^* and use it instead of the optimal solution.

We compare CascadeLinTS to two baselines. The first baseline is CascadeUCB1 (Section 2.3). This baseline does not leverage the structure of our problem and learns the attraction probability of each item e independently. The second baseline is RankedLinTS (Algorithm 4). This baseline is a variant of ranked bandits (Section 5), where the base bandit algorithm is LinTS. This base algorithm is the same as in CascadeLinTS. Therefore, any observed difference in the performance of cascading and ranked bandits must be due to the efficiency of using the base algorithm, and not the algorithm itself. In this sense, our comparison of CascadeLinTS and RankedLinTS is fair. The tunable parameter σ in RankedLinTS is also set to 1.

4.3 Features

In most recommender problems, good features of items are rarely available. Thus, they are typically learned from data [14]. As an example, in movie recommendations, all state of the art approaches are based on collaborative filtering rather than on the features of movies, such as movie genres.

Motivated by the successes of collaborative filtering in recommender systems, we derive the features of our items using low-rank matrix factorization. In particular, let $W \in \{0,1\}^{m \times L}$ be our feedback matrix for m users and L items. We randomly divide the rows of W into two matrices, training matrix $W_{\text{train}} \in \{0,1\}^{(m/2) \times L}$ and test matrix

Algorithm 4 Ranked bandits with linear TS.

Inputs: Variance σ^2

// Initialization
$\forall k \in [K] : \mathbf{M}_0^k \leftarrow I_d$ and $\mathbf{B}_0^k \leftarrow \mathbf{0}$

for all $t = 1, \ldots, n$ **do**
  for all $k = 1, \ldots, K$ **do**
    $\bar{\theta}_{t-1}^k \leftarrow \sigma^{-2} (\mathbf{M}_{t-1}^k)^{-1} \mathbf{B}_{t-1}^k$
    $\theta_t^k \sim \mathcal{N}(\bar{\theta}_{t-1}^k, (\mathbf{M}_{t-1}^k)^{-1})$
    $\mathbf{a}_k^t \leftarrow \arg\max_{e \in [L] - \{\mathbf{a}_1^t, \ldots, \mathbf{a}_{k-1}^t\}} x_e^\intercal \theta_t^k$

  // Recommend a list of K items and get feedback
  $\mathbf{A}_t \leftarrow (\mathbf{a}_1^t, \ldots, \mathbf{a}_K^t)$
  Observe click $\mathbf{C}_t \in \{1, \ldots, K, \infty\}$

  // Update statistics
  $\forall k \in [K] : \mathbf{M}_t^k \leftarrow \mathbf{M}_{t-1}^k$
  $\forall k \in [K] : \mathbf{B}_t^k \leftarrow \mathbf{B}_{t-1}^k$
  for all $k = 1, \ldots, \min\{\mathbf{C}_t, K\}$ **do**
    $e \leftarrow \mathbf{a}_k^t$
    $\mathbf{M}_t^k \leftarrow \mathbf{M}_t^k + \sigma^{-2} x_e x_e^\intercal$
    $\mathbf{B}_t^k \leftarrow \mathbf{B}_t^k + x_e \mathbb{1}\{\mathbf{C}_t = k\}$

$W_{\text{test}} \in \{0,1\}^{(m/2) \times L}$. We use W_{train} to learn the features of items and W_{test} in place of W to evaluate our learning algorithms. Most existing real-world recommender systems already have some data about their users. Such data can be used to construct W_{train}.

Let $W_{\text{train}} \approx U \Sigma V^\intercal$ be rank-d truncated SVD of W_{train}, where $U \in \mathbb{R}^{(m/2) \times d}$, $\Sigma \in \mathbb{R}^{d \times d}$, and $V \in \mathbb{R}^{L \times d}$. Then the features of items are the rows of $V\Sigma$. Specifically, for each item $e \in E$ and feature $i \in [d]$, $x_e(i) = V_{e,i} \Sigma_{i,i}$.

4.4 Restaurant Recommendations

Our dataset is from Yelp Dataset Challenge[2]. This dataset has five parts, including business information, checkin information, review information, tip information, and user information. We only consider the business and review information. The dataset contains 78k businesses, out of which 11k are restaurants; and 2.2M reviews written by 550k users. We extract $L = 3$k most reviewed restaurants and $m = 20$k most reviewing users.

Our objective is to maximize the probability that the user is attracted by at least one recommended restaurant. We build the model of users from past review data and assume that the user is attracted by the restaurant if the user reviewed this restaurant before. This indicates that the user visited the restaurant at some point in time, likely because the restaurant attracted the user at that time.

[2] https://www.yelp.com/dataset_challenge

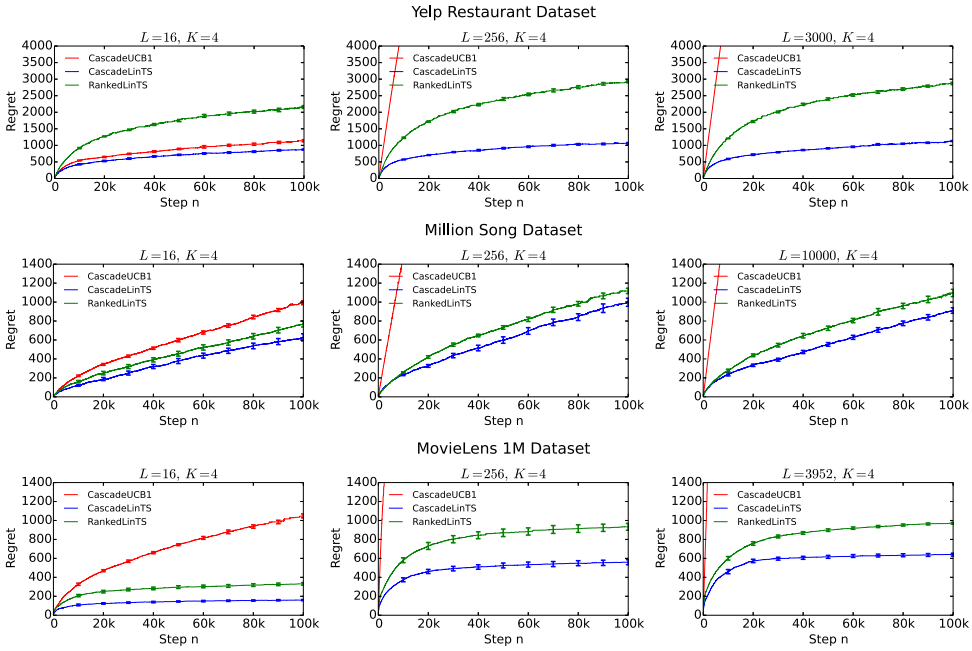

Figure 1: The n-step regret of `CascadeUCB1`, `CascadeLinTS` and `RankedLinTS` on three problems. We vary the number of items in the ground set E, from $L = 16$ to the maximum value in each problem.

4.4.1 Results

The results of our first experiment are reported in Fig. 1. When the ground set is small, $L = 16$, all compared methods perform similarly. In particular, the regret of `CascadeLinTS` is similar to that of `RankedLinTS`. The regret of `CascadeUCB1` is about two times larger than that of `CascadeLinTS`. As the size of the ground set increases, the gap between `CascadeLinTS` and the other methods increases. In particular, when $L = 3$k, the regret of `CascadeUCB1` is orders of magnitude larger than that of `CascadeLinTS`, and the regret of `RankedLinTS` is almost three times larger.

In the second experiment (Fig. 2a), we observe that `CascadeLinTS` performs well for all settings of d. When the number of features doubles to $d = 40$, the regret roughly doubles. When the number of features is halved to $d = 10$, the regret improves and is roughly halved.

In the third experiment (Fig. 2b), `CascadeLinTS` is evaluated on the subset of *American Restaurants*. This is the largest restaurant category in our dataset. We observe that `CascadeLinTS` can learn for any number of features d, similarly to Fig. 2a.

In the last experiment (Fig. 2c), we observe that the regret of `CascadeLinTS` increases with the number of recommended items, from $K = 4$ to $K = 8$. This result is surprising and seems to contradict to Kveton *et al.* [15], who find both theoretically and empirically that the regret in cascading bandits decreases with the number of recommended items K. We investigate this further and plot the cumulative reward of `CascadeLinTS` in Fig. 2d. The reward increases with K, which is expected and validates that `CascadeLinTS` learns better policies for larger K. Therefore, the increase in the regret in Fig. 2c must be due to the fact that the expected reward of the optimal solution, $f(A^*, \bar{w})$, increases faster with K than that of the learned policies. We believe that the optimal solutions for larger K are harder to learn because our modeling assumptions are violated. In particular, the linear generalization in (4) is imperfect and the items in E are not guaranteed to attract users independently.

4.5 Million Song Recommendation

Million Song Dataset[3] is a collection of audio features and metadata for a million contemporary pop songs. Instead of storing any audio, the dataset consists of features derived from the audio, user-song profile data, and genres of songs. We extract $L = 10$k most popular songs from this dataset, as measured by the number of song-listening events; and $m = 400$k most active users, as measured by the number of song-listening events.

Our objective is to maximize the probability that the user is attracted with at least one recommended song and plays it. We build the model of users from their past listening patterns and assume that the user is attracted by the song if the user listened to this song before. This indicates that the

[3] http://labrosa.ee.columbia.edu/millionsong/

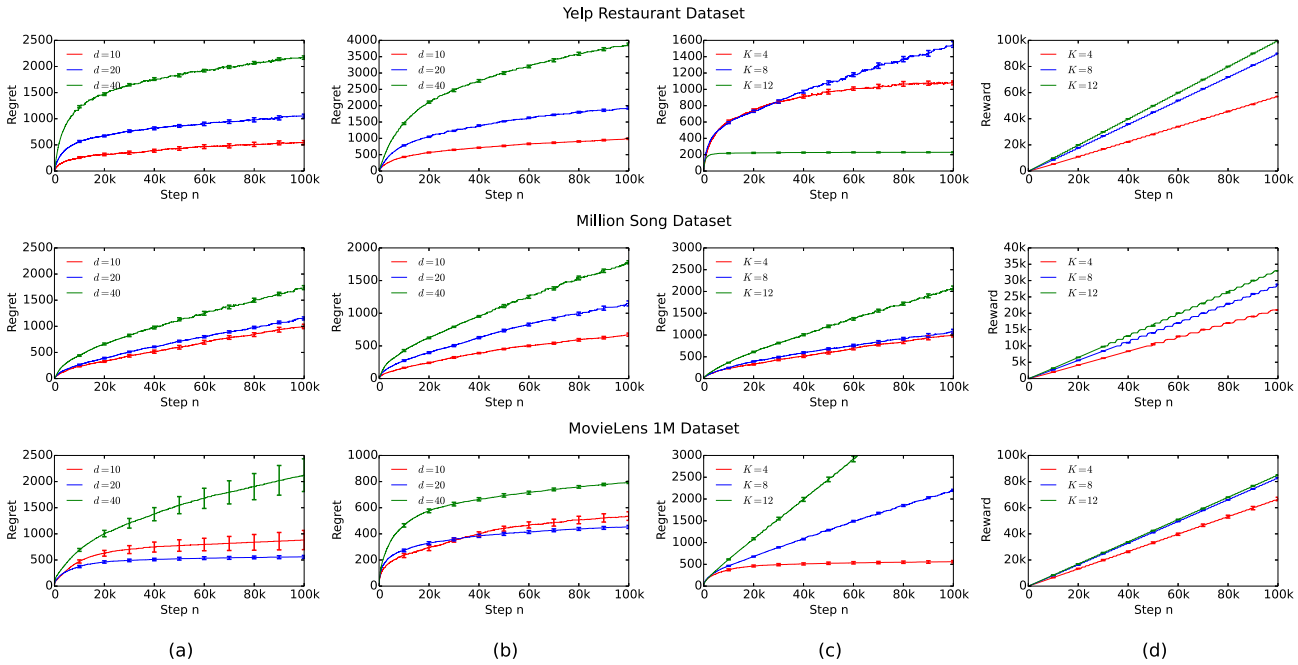

Figure 2: **a**. The n-step regret of `CascadeLinTS` for varying number of features d. **b**. The n-step regret of `CascadeLinTS` in a subset of each dataset for varying number of features d. **c**. The n-step regret of `CascadeLinTS` for varying number of recommended items K. **d**. The n-step reward of `CascadeLinTS` for varying number of recommended items K.

user was attracted by the song at some point in time.

4.5.1 Results

The results of our first experiment are reported in Fig. 1. Similarly to Section 4.4, we observe that when the ground set is small, $L = 16$, the regret of all compared methods is similar. As the size of the ground set increases, the gap between `CascadeUCB1` and the rest of the methods increases, and the regret of `CascadeUCB1` is orders of magnitude larger than that of `CascadeLinTS`. The regret of `CascadeLinTS` is similar to that of `RankedLinTS` for all settings of L.

We report the regret of `CascadeLinTS` for various numbers of features d, on the whole dataset and its subset of *Rock Songs*, in Fig. 2a and 2b, respectively. Similarly to Section 4.4, we observe that `CascadeLinTS` performs well for all settings of d. The lowest regret in both experiments is achieved at $d = 10$.

In the last experiment (Fig. 2c), we observe that the regret of `CascadeLinTS` increases with the number of recommended items K. As in Section 4.4, we observe that the cumulative reward of our learned policies increases with K. Therefore, the increase in the regret must be due to the fact that the expected reward of the optimal solution, $f(A^*, \bar{w})$, increases faster with K than that of the learned policies. This is due to the mismatch between our model and real-world data.

4.6 Movie Recommendation

MovieLens datasets[4] contain the ratings of users for movies from the MovieLens website. The datasets come in different sizes and we choose MovieLens 1M for our experiments. This dataset contains 1M anonymous ratings of 4k movies by 6k users who joined MovieLens in 2000.

We build the model of users from their historical ratings. The ratings are on a 5-star scale and we assume the user is attracted by a movie if the user rates it with more than 3 stars. Thus, the feedback matrix is defined as $W_{i,j} = \mathbb{1}\{\text{user } i \text{ rates movie } j \text{ with more than 3 stars}\}$. Our goal is to maximize the probability of recommending at least one attractive movie.

4.6.1 Results

The results of our first experiment are reported in Fig. 1. Similarly to Section 4.4, we observe that the regret of all compared methods is similar when the ground set is small, L = 16. The gap between `CascadeUCB1` and the rest of the methods increases when the size of the ground set increases. In particular, the regret of `CascadeUCB1` is orders of magnitude larger than that of `CascadeLinTS`. The regret of `CascadeLinTS` is always lower than that of `RankedLinTS` for all settings of L.

We report the regret of `CascadeLinTS` for various num-

[4] http://grouplens.org/datasets/movielens/

bers of features d, on the whole dataset and its subset of *Adventures*, in Fig. 2a and 2b, respectively. Similarly to Sections 4.4 and 4.5, we observe that `CascadeLinTS` performs well for all settings of d. The lowest regret in both experiments is achieved at $d = 20$.

In the last experiment (Fig. 2c), we observe that the regret of `CascadeLinTS` increases with the number of recommended items K. As in Sections 4.4 and 4.5, the cumulative reward of our learned policies increases with K. Therefore, the increase in the regret must be due to the fact that the expected reward of the optimal solution, $f(A^*, \bar{w})$, increases faster with K than that of the learned policies.

5 RELATED WORK

Our work is closely related to *cascading bandits* [15, 8], which are learning variants of the cascade model of user behavior [10]. The key difference is that we assume that the attraction weights of items are a linear function of known feature vectors, which are associated with each item; and an unknown parameter vector, which is learned. This leads to very efficient learning algorithms whose regret is sublinear in the number of items L. We compare `CascadeLinTS` to `CascadeUCB1`, one of the proposed algorithms by Kveton *et al.* [15], in Section 4.

Ranked bandits [20] are a popular approach in learning to rank. The key idea in ranked bandits is to model each position in the recommended list as an independent bandit problem, which is then solved by a *base bandit algorithm*. The solutions in ranked bandits are $(1 - 1/e)$ approximate and their regret grows linearly with the number of recommended items K. On the other hand, ranked bandits do not assume that items attract the user independently. Slivkins *et al.* [22] proposed contextual ranked bandits. We compare `CascadeLinTS` to contextual ranked bandits with linear generalization in Section 4.

Our learning problem is a partial monitoring problem where we do not observe the attraction weights of all recommended items. Bartok *et al.* [5] studied general partial monitoring problems. The algorithm of Bartok *et al.* [5] scales at least linearly with the number of actions, which is $\binom{L}{K}$ in our setting. Therefore, the algorithm is impractical for large L and moderate K. Agrawal *et al.* [2] studied a variant of partial monitoring where the reward is observed. The algorithm of Agrawal *et al.* [2] cannot be applied to our problem because the algorithm assumes a finite parameter set. Lin *et al.* [19] and Kveton *et al.* [17] studied combinatorial partial monitoring. Our feedback model is similar to that of Kveton *et al.* [17]. Therefore, we believe that our algorithm and analysis can be relatively easily generalized to combinatorial action sets.

Our learning problem is combinatorial as we learn K most attractive items out of L candidate items. In this sense, our work is related to stochastic combinatorial bandits, which are frequently studied with a linear reward function and semi-bandit feedback [11, 6, 16, 18, 24, 9]. Our work differs from these approaches in both the reward function and feedback. Our reward function is a non-linear function of unknown parameters. Our feedback model is less than semi-bandit, because the learning agent does not observe the attraction weights of all recommended items.

6 CONCLUSIONS

In this work, we propose linear cascading bandits, a framework for learning to recommend in the cascade model at scale. The key assumption in linear cascading bandits is that the attraction probabilities of items are a linear function of the features of items, which are known; and an unknown parameter vector, which is unknown and we learn it. We design two algorithms for solving our problem, `CascadeLinTS` and `CascadeLinUCB`. We bound the regret of `CascadeLinUCB` and suggest that a similar regret bound can be proved for `CascadeLinTS`. We comprehensively evaluate `CascadeLinTS` on a range of recommendation problems and compare it to several baselines. We report orders of magnitude improvements over learning algorithms that do not leverage the structure of our problem, the features of items. We observe empirically that `CascadeLinTS` performs very well.

We leave open several questions of interest. For instance, we only bound the regret of `CascadeLinUCB`. Based on the existing work [24], we believe that a similar regret bound can be proved for `CascadeLinTS`. Moreover, note that our analysis of `CascadeLinUCB` is under the assumption that items attract the user independently and that the linear generalization is perfect. Both of these assumptions tend to be violated in practice. Our current analysis cannot explain this behavior and we leave it for future work.

The main limitation of the cascade model [10] is that the user clicks on at most one item. This assumption is often violated in practice. Recently, Katariya *et al.* [13] proposed a generalization of cascading bandits to multiple clicks, by proposing a learning variant of the dependent click model [12]. We strongly believe that our results can be generalized to this setting and leave this for future work.

References

[1] Yasin Abbasi-Yadkori, David Pal, and Csaba Szepesvari. Improved algorithms for linear stochastic bandits. In *Advances in Neural Information Processing Systems 24*, pages 2312–2320, 2011.

[2] Rajeev Agrawal, Demosthenis Teneketzis, and Venkatachalam Anantharam. Asymptotically efficient adaptive allocation schemes for controlled i.i.d. processes: Finite parameter space. *IEEE Transactions on Automatic Control*, 34(3):258–267, 1989.

[3] Shipra Agrawal and Navin Goyal. Analysis of Thompson sampling for the multi-armed bandit problem. In *Proceeding of the 25th Annual Conference on Learning Theory*, pages 39.1–39.26, 2012.

[4] Shipra Agrawal and Navin Goyal. Thompson sampling for contextual bandits with linear payoffs. In *Proceedings of the 30th International Conference on Machine Learning*, pages 127–135, 2013.

[5] Gabor Bartok, Navid Zolghadr, and Csaba Szepesvari. An adaptive algorithm for finite stochastic partial monitoring. In *Proceedings of the 29th International Conference on Machine Learning*, 2012.

[6] Wei Chen, Yajun Wang, and Yang Yuan. Combinatorial multi-armed bandit: General framework, results and applications. In *Proceedings of the 30th International Conference on Machine Learning*, pages 151–159, 2013.

[7] Aleksandr Chuklin, Ilya Markov, and Maarten de Rijke. *Click Models for Web Search*. Morgan & Claypool Publishers, 2015.

[8] Richard Combes, Stefan Magureanu, Alexandre Proutiere, and Cyrille Laroche. Learning to rank: Regret lower bounds and efficient algorithms. In *Proceedings of the 2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, 2015.

[9] Richard Combes, Mohammad Sadegh Talebi, Alexandre Proutiere, and Marc Lelarge. Combinatorial bandits revisited. In *Advances in Neural Information Processing Systems 28*, pages 2107–2115, 2015.

[10] Nick Craswell, Onno Zoeter, Michael Taylor, and Bill Ramsey. An experimental comparison of click position-bias models. In *Proceedings of the 1st ACM International Conference on Web Search and Data Mining*, pages 87–94, 2008.

[11] Yi Gai, Bhaskar Krishnamachari, and Rahul Jain. Combinatorial network optimization with unknown variables: Multi-armed bandits with linear rewards and individual observations. *IEEE/ACM Transactions on Networking*, 20(5):1466–1478, 2012.

[12] Fan Guo, Chao Liu, and Yi Min Wang. Efficient multiple-click models in web search. In *Proceedings of the 2nd ACM International Conference on Web Search and Data Mining*, pages 124–131, 2009.

[13] Sumeet Katariya, Branislav Kveton, Csaba Szepesvari, and Zheng Wen. DCM bandits: Learning to rank with multiple clicks. In *Proceedings of the 33rd International Conference on Machine Learning*, 2016.

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

[15] Branislav Kveton, Csaba Szepesvari, Zheng Wen, and Azin Ashkan. Cascading bandits: Learning to rank in the cascade model. In *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

[16] Branislav Kveton, Zheng Wen, Azin Ashkan, Hoda Eydgahi, and Brian Eriksson. Matroid bandits: Fast combinatorial optimization with learning. In *Proceedings of the 30th Conference on Uncertainty in Artificial Intelligence*, pages 420–429, 2014.

[17] Branislav Kveton, Zheng Wen, Azin Ashkan, and Csaba Szepesvari. Combinatorial cascading bandits. In *Advances in Neural Information Processing Systems 28*, pages 1450–1458, 2015.

[18] Branislav Kveton, Zheng Wen, Azin Ashkan, and Csaba Szepesvari. Tight regret bounds for stochastic combinatorial semi-bandits. In *Proceedings of the 18th International Conference on Artificial Intelligence and Statistics*, 2015.

[19] Tian Lin, Bruno Abrahao, Robert Kleinberg, John Lui, and Wei Chen. Combinatorial partial monitoring game with linear feedback and its applications. In *Proceedings of the 31st International Conference on Machine Learning*, pages 901–909, 2014.

[20] Filip Radlinski, Robert Kleinberg, and Thorsten Joachims. Learning diverse rankings with multi-armed bandits. In *Proceedings of the 25th International Conference on Machine Learning*, pages 784–791, 2008.

[21] Daniel Russo and Benjamin Van Roy. Learning to optimize via posterior sampling. *Mathematics of Operations Research*, 39(4):1221–1243, 2014.

[22] Aleksandrs Slivkins, Filip Radlinski, and Sreenivas Gollapudi. Ranked bandits in metric spaces: Learning diverse rankings over large document collections. *Journal of Machine Learning Research*, 14(1):399–436, 2013.

[23] William. R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3-4):285–294, 1933.

[24] Zheng Wen, Branislav Kveton, and Azin Ashkan. Efficient learning in large-scale combinatorial semi-bandits. In *Proceedings of the 32nd International Conference on Machine Learning*, 2015.