Exploration in two-stage recommender systems

Jiri Hron[1,*]**, Karl Krauth**[2,*]**, Michael I. Jordan**[2]**, Niki Kilbertus**[1,3]
[1]University of Cambridge, [2]UC Berkeley, [3]Max Planck Institute for Intelligent Systems

Abstract

Two-stage recommender systems are widely adopted in industry due to their scalability and maintainability. These systems produce recommendations in two steps: (i) multiple *nominators* preselect a small number of items from a large pool ($\gg$100K) using cheap-to-compute item embeddings; (ii) with a richer set of features, a *ranker* rearranges the nominated items and serves them to the user. A key challenge of this setup is that optimal performance of each stage in isolation does not imply optimal global performance. In response to this issue, Ma et al. (2020) proposed a nominator *training objective* importance weighted by the ranker's probability of recommending each item. In this work, we focus on the complementary issue of *exploration*. Modeled as a contextual bandit problem, we find LinUCB—a near optimal exploration strategy for single-stage systems—may lead to linear regret when deployed in two-stage recommenders. We therefore propose a method of *synchronising the exploration strategies* between the ranker and the nominators. Our algorithm only relies on quantities already computed by standard LinUCB at each stage and can be implemented in three lines of additional code. We end by demonstrating the effectiveness of our algorithm experimentally.

1 Introduction

Contemporary recommender systems are tasked with finding a small number of relevant items among millions or billions of candidates, personalized for each of hundreds of thousands or millions of users and their always changing needs, all of which has to happen in order of milliseconds so as not to negatively impact webpage loading speeds. One of the most widely used solutions to the problem are *two-stage recommender systems* (Borisyuk et al., 2016; Covington et al., 2016; Eksombatchai et al., 2018) in which (i) a set of *computationally efficient* nominators narrows down the search from millions to only hundreds of items, and (ii) the *slower but more accurate* ranker selects and reorders a few items which are eventually served to the user.

For example, a nominator can use a two-tower architecture (Yi et al., 2019) and focus on a narrower set of features, whereas the ranker would rely on a more powerful model and consider additional features extracted from, e.g., ratings, specialized user and item attributes, or the number and type of past interactions with given user (Covington et al., 2016; Ma et al., 2020). Importantly, the nominators are often heterogeneous both in terms of the size and type of the items from which they select the candidate items, and the algorithm used to select candidates ranging from simple associative rules to recurrent neural networks (Chen et al., 2019).We will focus on nominators which utilize statistical learning methods in the two-stage setup as in the paper most relevant to our work (Ma et al., 2020).

Ma et al. (2020) study off-policy learning for two-stage recommender systems where the goal is learning a good recommendation policy from the typically abundant logged data. The main proposal of Ma et al. is to modify the nominator training objective by adding importance weights based on the ranker's probability of recommending each item. With adjustments facilitating gradient descent optimization, the authors show significant empirical improvements not only compared to a system trained without importance weighting, but also relative to nominators importance weighted only based on the past *nominator* policy (ignoring the presence of the ranker). These results thus demonstrate that local optima of individual components do not translate to optimality of the system as a whole.

Naturally, we can ask whether there are other aspects of the recommendation problem where optimal solutions for a single-stage system result in suboptimal
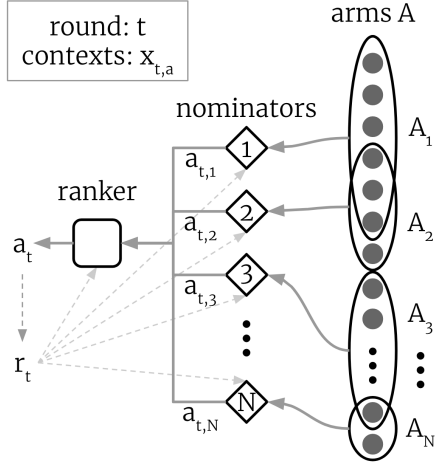
Figure 1: The two-stage recommendation setup.

performance when deployed in a two-stage system. We answer this question affirmatively in the case of *exploration* by which we mean the task of learning an optimal recommendation policy under uncertainty in a sample efficient way. An effective exploration strategy then needs to balance greedy actions based on past interactions with exploratory recommendations targetting items about which there is little or no information.

While many strategies have been proposed in the literature, ranging from more conservative ones like Boltzmann exploration (Daw et al., 2006; Chen et al., 2019) to more optimistic contextual bandit algorithms (Lattimore & Szepesvári, 2020), we will restrict our discussion to the popular LinUCB algorithm (Auer, 2002; Dani et al., 2008; Li et al., 2010) and the associated contextual bandit recommendation setup. A similar setup has been explored by (Ma et al., 2020) as it is rich enough to exhibit many of the salient properties encountered in the real world, while abstracting away some of the complexities involved in deployment of large scale recommendation systems.

Contributions: We (i) show that a mismatch of feature mappings (architecture) and the amount of data seen (regularization) between LinUCB ranker and nominators can result in large, even linear regret; (ii) demonstrate said effects and their dependence on the level of mismatch empirically on a toy dataset; and (iii) propose a simple algorithm based on synchronization of inferred statistics between the ranker and the nominators which addresses the issue. We demonstrate the efficacy of our algorithm on simulated data.

2 Setup

We consider a scenario in which a single item is to be recommended in each of the T rounds. The recommen-

dation problem is modeled in a contextual bandit setup where the individual items correspond to arms $a \in A$ (we thus also refer to items as arms or actions). For the first stage, we assume a fixed number of $N \in \mathbb{N}$ *nominators*, each of which has access only to a fixed non-empty subset of arms $A_n \subseteq A$. At every round $t \in [T]$, each nominator observes contexts $x_{t,a}$ for all $a \in A_n$, and selects a *single* action $a_{t,n}$. The ranker then chooses a *single* final recommendation among the ones nominated in the first stage $a_t \in \{a_{t,1}, \ldots a_{t,N}\}$ *based on the corresponding contexts*. Both the ranker and the nominators can ultimately be updated using the reward r_t obtained by pulling arm a_t as well as all the revealed contexts $x_{t,a}$. This two-stage recommendation process is illustrated in Figure 1.

We restrict our attention to the stochastic linear bandit setting (Abe & Long, 1999; Lattimore & Szepesvári, 2020), and the LinUCB algorithm *with ellipsoidal confidence sets* for both the ranker and the nominators (Auer, 2002; Dani et al., 2008). The linear contextual bandit setting assumes existence of a fixed embedding for each context $\phi(x_{t,a})$ such that

$$\mathbb{E}[r_t \mid x_{t,a_t}, a_t] = \langle \phi(x_{t,a_t}), \theta_\star \rangle \qquad (1)$$

for all $x_{t,a}$, a_t, and a fixed $\theta_\star \in \mathbb{R}^d$. Since $\theta_\star$ is assumed unknown, a *single-stage* LinUCB estimates it by ridge regression with regularization parameter $\lambda > 0$

$$\Sigma_t := \left[\lambda I_d + \sum_{i=1}^{t} \phi(x_{i,a_i}) \phi(x_{i,a_i})^\top \right]^{-1},$$
$$\hat{\theta}_t := \Sigma_t \sum_{i=1}^{t} r_i \, \phi(x_{i,a_i}). \qquad (2)$$

The actions are then selected according to

$$a_{t+1} \in \operatorname*{argmax}_{a \in A} \mathrm{UCB}_{t+1}(a), \qquad (3)$$

where as in (Lattimore & Szepesvári, 2020, p. 239–241)

$$\mathrm{UCB}_{t+1}(a) := \langle \phi(x_{t,a}), \hat{\theta}_t \rangle + \sqrt{\beta_t} \, \|\phi(x_{t,a})\|_{\Sigma_t},$$
$$\sqrt{\beta_t} := \sqrt{\lambda} + \sqrt{2 \log t + d \log \left(\frac{d\lambda + t}{d\lambda} \right)}.$$

LinUCB with such β_t achieves near optimal regret

$$R_T := \mathbb{E} \left[\sum_{t=1}^{T} r_{t,\star} - r_t \right]$$
$$= \sum_{t=1}^{T} \langle \theta_\star, \phi(x_{t,a_{t,\star}}) - \mathbb{E}[\phi(x_{t,a_t})] \rangle, \qquad (4)$$

when the reward noise is sub-Gaussian (Dani et al., 2008; Lattimore & Szepesvári, 2020), where

$$a_{t,\star} := \operatorname*{argmax}_{a \in A} \langle \theta_\star, \phi(x_{t,a}) \rangle, \qquad (5)$$

$r_{t,\star}$ is the reward obtained by choosing $a_{t,\star}$, and the expectation is taken with respect to the randomness of the rewards and the policy (uniform tie breaking).

Importantly for our later development, LinUCB can be interpreted in Bayesian terms in the following sense: the current mean and covariance estimates $\theta \sim \mathcal{N}(\hat{\theta}_t, \Sigma_t)$ correspond to the posterior distribution obtained by combining the prior $\mathcal{N}(0, \lambda^{-1}I_d)$ with the likelihood $r_{t,a} \sim \mathcal{N}(\langle \phi(x_{t,a}), \theta \rangle, 1)$. Since

$$\langle \phi(x_{t,a}), \theta \rangle \sim \mathcal{N}\left(\langle \phi(x_{t,a}), \hat{\theta}_{t-1} \rangle, \|\phi(x_{t,a})\|_{\Sigma_{t-1}}\right), \quad (6)$$

the selection rule employed by LinUCB can be viewed as selecting the action with the highest posterior $\Phi(\sqrt{\beta_t})$-th quantile, where Φ is the cumulative distribution function (CDF) of the standard normal distribution. In other words, the maintained estimates define a *confidence set* for the true parameter θ, and LinUCB chooses the best action compatible with this set.

As mentioned, we will assume that each of the nominators and the ranker use the LinUCB algorithm also in the two-stage setup.[1] Since the nominators often need to rapidly sift through millions of items, we assume *only* the ranker has access to the true but expensive to compute embeddings ϕ, while each nominator $n \in [N]$ uses computationally cheaper embeddings $\phi_n(x) \in \mathbb{R}^{d_n}$. In the rest of this document, we use the term *naive two-stage LinUCB* to refer to the algorithm where each nominator *independently* maintains its own estimates $\hat{\theta}_{n,t}$, $\Sigma_{n,t}$ defined as in Equation (2) with ϕ replaced by ϕ_n, nominate actions analogously to Equation (3), and update their posterior *only* with r_t and $\phi_n(x_{t,a_t})$, where we recall a_t need not equal $a_{t,n}$. Moreover, the ranker independently maintains its own estimates $\hat{\theta}$, Σ_t used to select the item ultimately served to the user.

In the next section, we will first show on a simple example that such independently maintained uncertainty estimates lead to suboptimal global performance. We then propose a solution based on synchronization of the upper bound estimates $\text{UCB}_t(a)$ used in Equation (3) between the ranker and the nominators.

3 Coordinated exploration

To understand when the *naive* two-stage LinUCB implementation does not work, it is useful to know when it does. In particular, consider the case when all the nominators are allowed to use the *same features* as the ranker $\phi_n = \phi$, and employ the *same*

---

[1]While not in line with the usual heterogeneity of the nominator algorithms, we believe this setting captures much that is salient to the interaction between general simultaneously learning ranker and nominators. It also goes one step beyond the setup in (Ma et al., 2020) by considering the existence of more than one nominator.

Inputs: $\hat{\theta}_0, \Sigma_0, (\beta_t)_t$; $\forall n: \hat{\theta}_{n,0}, \Sigma_{n,0}, (\beta_{n,t})_t$
for $t = 1, 2, \ldots, T$ **do**
    $\forall n: a_{n,t} \leftarrow \text{argmax}_{a \in A_n} \text{UCB}_{n,t}(a)$
    $a_t \leftarrow \text{argmax}_{a \in \{a_{1,t},\ldots,a_{N,t}\}} \text{UCB}_t(a)$
    $\Sigma_t^{-1} \leftarrow \Sigma_{t-1}^{-1} + \phi_t\phi_t^{\top}$
    $\hat{\theta}_t \leftarrow \Sigma_t\left(\Sigma_{t-1}^{-1}\hat{\theta}_{t-1} + r_t\phi_t\right)$
    for $n = 1, 2, \ldots, N$ **do**
        $\Sigma_{n,t}^{-1} \leftarrow \Sigma_{n,t-1}^{-1} + \phi_{n,t}\phi_{n,t}^{\top}$
        $\hat{\theta}_{n,t} \leftarrow \Sigma_{n,t}\left(\Sigma_{n,t-1}^{-1}\hat{\theta}_{n,t-1} + r_t\phi_{n,t}\right)$
        if $\|\phi_n(a_{n,t})\|_{\Sigma_{n,t}} > \|\phi(a_{n,t})\|_{\Sigma_t}$ **then**
            $\hat{\theta}_{n,t} \leftarrow \hat{\theta}_{n,t} + \frac{\langle \hat{\theta}_t, \phi_t \rangle - \langle \hat{\theta}_{n,t}, \phi_{n,t} \rangle}{\|\phi_{n,t}\|_{\Sigma_{n,t}}^2}\Sigma_{n,t}\phi_{n,t}$
            $\Sigma_{n,t}^{-1} \leftarrow \Sigma_{n,t}^{-1} + \left(\frac{1}{\|\phi_t\|_{\Sigma_t}^2} - \frac{1}{\|\phi_{n,t}\|_{\Sigma_{n,t}}^2}\right)\phi_{n,t}\phi_{n,t}^{\top}$

Algorithm 1: Two-stage synchronized LinUCB. Here $\phi_t \coloneqq \phi(a_t)$ and $\phi_{n,t} \coloneqq \phi(a_{n,t})$ for all n, t.

prior $\mathcal{N}(0, \lambda_n^{-1}I_d)$ with $\lambda_n = \lambda$. It is not hard to see that in this case $\hat{\theta}_{n,t} = \hat{\theta}_t$ and $\Sigma_{n,t} = \Sigma_t$ for all $t \in [T]$, and thus each nominator selects the same action as would be selected by the ranker constrained to the same action pool A_n. Since $\max\{c_1, \ldots, c_k\} = \max\{\max\{c_1, \ldots, c_{k_1}\}, \ldots, \max\{c_{k_{N-1}+1}, \ldots, c_k\}\}$ for any partition of $c_1, \ldots, c_k \in \mathbb{R}$, this then implies that the naive two-stage system behaves *exactly* as a single-stage LinUCB with access to all actions would.

Because we know single-stage LinUCB is close to optimal, the above implies that any potential increase in regret must come from either the already discussed mismatch of the embeddings inherent to two-stage systems, or mismatch of the prior. The latter is then most often caused by the ranker being deployed for much longer time and thus better trained. Such a scenario is common in many contemporary industrial practices (Covington et al., 2016; Ma et al., 2020), and can be modelled in our setup by using a ranker prior with lower initial uncertainty then the nominators. As demonstrated next, naive two-stage LinUCB is poorly equipped to handle such discrepancies.

3.1 Motivating example

Consider a setting with *only one* context, two nominators, three actions split between them as $A_1 = \{a_1\}$, $A_2 = \{a_2, a_3\}$, ϕ returning *one-hot encodings* of the actions, and $\phi_n = \phi$ for all nominators. The expected rewards from a_1 to a_3 are $[1/2, 1/4, 3/4] = \theta_*$ (one-hot action encodings), and observed rewards are generated by adding i.i.d. Gaussian noise $\mathcal{N}(\theta_\star, 10^{-2}I_3)$. The ranker's parameters are initialized to $\hat{\theta}_0 \sim \mathcal{N}(\theta_\star, \sigma_{\theta_\star}^2 I_3)$, and $\Sigma_0 = (\lambda + \gamma)^{-1}I_3$ where γ represents how many more samples per action the ranker has seen at $t = 1$ compared to the nominators. For both nominators $n \in [2]$, we take $\hat{\theta} = 0$, and $\Sigma_{n,0} = \lambda_n^{-1}I_3$ with regular-

Noise level $\sigma_{\theta_\star} = 0.1$: $\hat{\theta}_0 \sim \mathcal{N}(\theta_\star, 10^{-2})$

Noise level $\sigma_{\theta_\star} = 0.2$: $\hat{\theta}_0 \sim \mathcal{N}(\theta_\star, 5^{-2})$
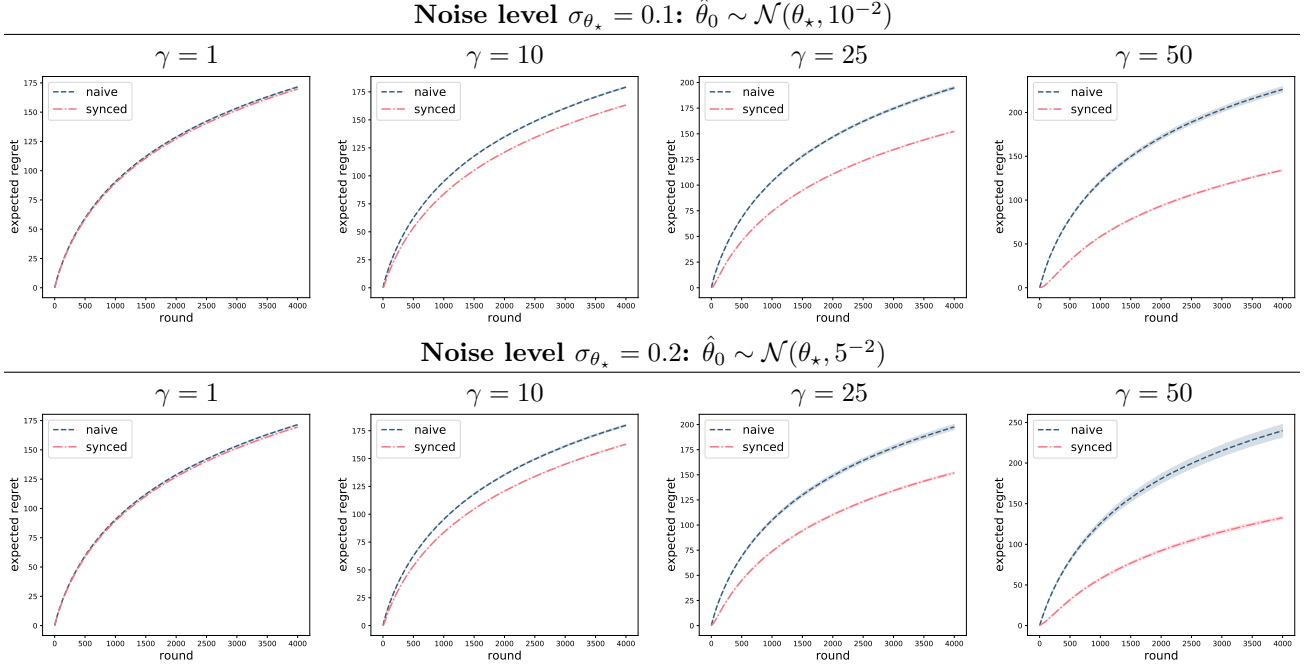

Figure 2: Naive vs. synchronized 2-stage recommendation. Setup described in Section 3.1. Expected regret and its 2-sigma confidence intervals were estimated over 400 runs. The level of pretraining of the ranker γ has outsized effect on the naive but not the synchronized two-stage LinUCB, overcoming the 'deadlock' effect.

ization parameter (prior precision) $\lambda_n = \lambda = 10^{-3}$.

To see what can go wrong in this scenario, consider the extreme case $\sigma^2_{\theta_\star} = 0$ and $\gamma \gg 0$, i.e., the ranker has seen enough data to essentially recover the true parameter. Since $A_1 = \{a_1\}$, such a ranker always picks a_3 (the best action) when $a_{2,t} = a_3$, and a_1 otherwise. In the naive implementation, this results in a *'deadlock'* where the second nominator's uncertainty about a_2 never decreases, leading it to mostly nominate a_2 over a_3, entailing *linear* regret. While the extreme case may be rare in practice, Figure 2 shows the effect remains significant even when the ranker is not fully trained, i.e., $\sigma_{\theta_\star} > 0$, and $\gamma > 1$ but not overly large.

3.2 Synchronized two-stage LinUCB

The deadlock observed in the previous section is due to a lack of communication of uncertainty between the ranker and the nominators. Since the computational constraints inherent to the two-stage setup entail distinct embedding functions, the issue cannot be addressed by simply setting nominator parameters to those of the ranker. However, because action selection only depends on the estimated marginal quantiles of the *rewards*, we can update the nominator's estimates based on the reward statistics computed by the ranker.

We propose to *synchronize* each nominator n in the rounds where $\|\phi_n(a_{n,t})\|_{\Sigma_{n,t}} > \|\phi(a_{n,t})\|_{\Sigma_t}$, i.e., when

the nominator is more uncertain about its selected action than the ranker (the cause of the 'deadlock' in Section 3.1). In particular, we want to minimally adjust the nominator posterior so that it matches the ranker's mean and variance which fully determine $\mathrm{UCB}_{n,t}(a_{n,t})$ (see Section 2). Defining the minimality in terms of Kullback-Leibler (KL) divergence, this can be achieved by solving the constrained optimization problem:

$$\min_{m,S} \quad \mathrm{KL}\left(\mathcal{N}(m, S) \,\middle\|\, \mathcal{N}\left(\hat{\theta}_{n,t}, \Sigma_{n,t}\right)\right) \tag{7}$$

$$\text{subject to} \quad \langle m, \phi_n(a_{n,t})\rangle = \langle \hat{\theta}_t, \phi(a_{n,t})\rangle, \quad \text{(mean)}$$
$$\|\phi_n(a_{n,t})\|_S = \|\phi(a_{n,t})\|_{\Sigma_t}, \quad \text{(covariance)}$$

When synchronization is performed after the usual update, the solution to Equation (7) gives us Algorithm 1.

Note that the selected KL divergence penalizes overdispersion compared to the previous distribution, meaning the resulting replacement for $\mathcal{N}(\hat{\theta}_{n,t}, \Sigma_{n,t})$ should not have more uncertainty. Furthermore, if we modify Algorithm 1 to perform the synchronization before the usual update—i.e., swap the black and blue lines within the inner-most for-loop and replace the if-condition by $\|\phi_n(a_{n,t})\|_{\Sigma_{n,t-1}} > \|\phi(a_{n,t})\|_{\Sigma_{t-1}}$—we arrive at an algorithm that only uses quantities already computed during selection of $a_{n,t}$ and a_t, minimizing the additional computation required; in experiments, both versions of the algorithm performed essentially the same, which we show in an example setting in Figure 3.
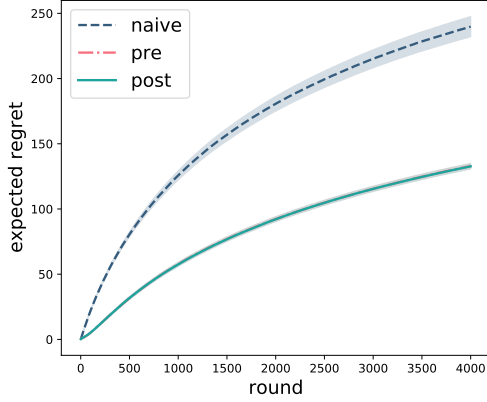
Figure 3: Comparison of post- vs. pre-update synchronization for a noise level of 0.2 and $\gamma = 50$. In this setting, the line for pre-update synchronization is barely visible, because it is covered by the line for post-update synchronization.

Finally, let us consider our synchronized two-stage LinUCB algorithm in the context of the motivating example from Section 3.1. Because we assumed $\phi = \phi_n$ are one-hot encodings of the actions, we see that the ranker variance for an action $j \in [3]$ at time t is $\|\phi(a_j)\|_{\Sigma_t}^2 = (\lambda + \gamma + n_{tj})^{-1}$ where n_{tj} is the number of times the ranker selected a_j. The *first* time a nominator select a_j, its variance before the round update will be $\|\phi_n(a_j)\|_{\Sigma_{n,t}}^2 = (\lambda + n_{tj})^{-1}$.[2] Inspecting the synchronization update for $\Sigma_{n,t}^{-1}$ in blue (Algorithm 1), the new value for the $(jj)^{\text{th}}$ entry of $\Sigma_{n,t}^{-1}$ amounts to

$$\underbrace{\lambda + n_{tj}}_{\Sigma_{n,t}^{-1}} + \underbrace{\lambda + \gamma + n_{tj}}_{\|\phi(a_j)\|_{\Sigma_t}^{-2}} - \underbrace{(\lambda + n_{tj})}_{\|\phi_n(a_j)\|_{\Sigma_{n,t}}^{-2}} = \|\phi(a_j)\|_{\Sigma_t}^2,$$

while the other variances will remain as in $\Sigma_{n,t}$ at time t. Since an analogous claim holds for the mean $\hat{\theta}_{n,t}$, we conclude the synchronization update ensures the posteriors of the ranker and the nominators match after each nominator *selected* (but not necessarily seen *recommended*) each $a \in A_n$ exactly once. Because the posteriors never diverge after they are fully matched, our algorithm starts behaving like single-stage LinUCB from thereon, which we know is near optimal for the task. This is confirmed in Figure 2 where the more pretrained the ranker is (higher γ), the better the synchronized and the worse the naive LinUCB do.

---

[2]Since we allow the pools A_n to be overlapping, n_{tj} could generally be greater than zero here. This is not the case for the example from Section 3.1 though.

4 Conclusion

We have shown that naive deployment of LinUCB in two-stage recommender systems may result in suboptimal performance, and close to linear regret in extreme cases. The suboptimality is due to the mismatch between both the embeddings used by the ranker and the nominators, and the gap between the amount of training data the ranker has seen compared to the nominators. Both of these issues are inherent to the setting in which two-stage recommenders are typically deployed in industry, and thus pose important barriers to achieving a good exploration-exploitation trade-off.

We have proposed a simple modification of the LinUCB algorithm based on communication of inferred statistics between the ranker and the nominators. Our algorithm can be implemented with minimal computational overhead, and achieves superior empirical results compared to the naive two-stage LinUCB implementation. While focusing solely on LinUCB, we suspect the 'deadlock' problem identified in Section 3.1 is pertinent to any exploration algorithm which in part selects its actions based on the level of uncertainty about them (e.g., all 'optimism in the face of uncertainty' type algorithms, Lattimore & Szepesvári, 2020). Since the principles motivating Equation (7) do not hinge on LinUCB in particular, we hope these issues could also be addressed by communication of statistics from the ranker.

This preprint is based on a workshop version of our work. We plan to publish an extended version with additional experiments on real-world data and extended discussion in the near future.

References

Abe, N. and Long, P. M. Associative reinforcement learning using linear probabilistic concepts. In *ICML*, pp. 3–11. Citeseer, 1999.

Auer, P. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3(Nov):397–422, 2002.

Borisyuk, F., Kenthapadi, K., Stein, D., and Zhao, B. Casmos: A framework for learning candidate selection models over structured queries and documents. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 441–450, 2016.

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

Covington, P., Adams, J., and Sargin, E. Deep neural networks for youtube recommendations. In *Proceed-*

ings of the 10th ACM conference on recommender systems, pp. 191–198, 2016.

Dani, V., Hayes, T. P., and Kakade, S. M. Stochastic linear optimization under bandit feedback. 2008.

Daw, N. D., O'doherty, J. P., Dayan, P., Seymour, B., and Dolan, R. J. Cortical substrates for exploratory decisions in humans. *Nature*, 441(7095):876–879, 2006.

Eksombatchai, C., Jindal, P., Liu, J. Z., Liu, Y., Sharma, R., Sugnet, C., Ulrich, M., and Leskovec, J. Pixie: A system for recommending 3+ billion items to 200+ million users in real-time. In *Proceedings of the 2018 World Wide Web Conference*, pp. 1775–1784. International World Wide Web Conferences Steering Committee, 2018.

Lattimore, T. and Szepesvári, C. *Bandit algorithms*. Cambridge University Press, 2020.

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

Ma, J., Zhao, Z., Yi, X., Yang, J., Chen, M., Tang, J., Hong, L., and Chi, E. H. Off-policy learning in two-stage recommender systems. In *Proceedings of The Web Conference 2020*, pp. 463–473, 2020.

Yi, X., Yang, J., Hong, L., Cheng, D. Z., Heldt, L., Kumthekar, A., Zhao, Z., Wei, L., and Chi, E. Sampling-bias-corrected neural modeling for large corpus item recommendations. In *Proceedings of the 13th ACM Conference on Recommender Systems*, pp. 269–277, 2019.