On component interactions in two-stage recommender systems

Anonymous Author(s)
Affiliation
Address
email

Abstract

Thanks to their scalability, two-stage recommenders are used by many of today's largest online platforms, including YouTube, LinkedIn, and Pinterest. These systems produce recommendations in two steps: (i) multiple *nominators*, tuned for low prediction latency, preselect a small subset of candidates from the whole item pool; (ii) a slower but more accurate *ranker* further narrows down the nominated items, and serves them to the user. Despite their popularity, the literature on two-stage recommenders is scarce. Such treatment presupposes that the two-stage performance is explained by the behavior of individual components if they were deployed independently. This is not the case: using synthetic and real-world data, we find that interactions between the ranker and the nominators substantially affect the overall performance. Motivated by these findings, we derive a lower bound which shows that careful choice of each nominator's training set is sometimes the only difference between a poor and optimal two-stage recommender. Since finding a good choice manually is difficult, we learn one instead. In particular, using a Mixture-of-Experts approach, we train the nominators (experts) to specialize on different subsets of the item pool. This significantly improves performance.

1 Introduction

Recommender systems play a central role in online ecosystems, affecting what media we consume, which products we buy, or even with whom we interact. A key design challenge is ensuring these systems can sift through billions of items to deliver a personalized experience to millions of users with *low response latency*. A widely adopted solution to this problem are two-stage recommender systems [12, 20, 28, 94, 97] where (i) a set of computationally efficient *nominators* (or *candidate generators*) preselect a small number of candidates, which are then (ii) further narrowed down, reranked, and served to the user by a slower but more statistically accurate *ranker*.

Nominators are often heterogeneous, ranging from associative rules to recurrent neural networks [18]. A popular choice are matrix factorization [53, 69] and two-tower [94] architectures which model user feedback by the dot product between separate user and item embeddings. While user embeddings may change after each interaction, item embeddings can typically be precomputed before deployment, making the computational cost of a single recommendation dominated by (approximation of) the dot products. In contrast, the ranker typically takes *both* the user and item features as an input, which makes the computational cost linear in the number of items even at deployment [20, 64].

With a few exceptions [42, 50, 64], the two-stage system specific literature is small compared to that on *single-stage* systems [e.g., 17, 39, 44, 53, 69, 75, 80], which do not construct an explicit candidate set during a separate nominator stage. This is especially concerning given the considerable ethical challenges entailed by the enormous reach of two-stage systems: according to the recent systematic survey by Milano et al. [67], recommender systems have been (partially) responsible for unfair

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

treatment of disadvantaged groups, privacy leaks, political polarization, spread of misinformation, and 'filter bubble' or 'echo chamber' effects. While many of these issues are primarily within the realm of 'human-algorithm' interactions, the additional layer of 'algorithm-algorithm' interactions introduced by the two-stage systems poses a further challenge to understanding and alleviating them.

The main aim of our work is thus to narrow the knowledge gap between single- and two-stage systems. After setting up the notation in Section 2, we present an empirical comparison of single- and two-stage systems over a variety of architectural and hyperparameter settings in Section 3.1, revealing that two-stage performance is significantly affected by interactions between the ranker and the nominators. Section 3.2 investigates these interactions theoretically. While we find that a ranker trained independently of the nominators will typically perform reasonably well (Proposition 1), we show the choice of each nominator's training set may be the only difference between a two-stage recommender which performs optimally, and one no better than random guessing (Proposition 2). Since manually finding a good choice of the training sets can be difficult, we facilitate the process by gradient descent. In particular, we propose a Mixture-of-Experts [45, 47] approach in Section 4, where nominators are treated as experts. We demonstrate the advantages of our proposal empirically.

2 Two-stage recommender systems

The goal of recommender systems is to learn a policy π which maps contexts $x \in \mathcal{X}$ to distributions $\pi(x)$ over a finite set of items (or *actions*) $a \in \mathcal{A}$, such that the expected reward $\mathbb{E}_x \, \mathbb{E}_{a\sim\pi(x)}[r_a \mid x]$ is maximized. The context x typically represents information about the user (e.g., interaction history, demographic data), and r_a is the user feedback associated with item a (e.g., rating, clickthrough, watch-time). We assume that $r_a|x$ follows a fixed distribution with mean $f^\star(x,a) \coloneqq \mathbb{E}[r_a \mid x]$ for all the (x,a) pairs. To simplify, we further assume only one item a_t is to be recommended for each given context x_t, where $t \in [T]$ with $[T] \coloneqq \{1,\dots,T\}$ for $T \in \mathbb{N}$.

Two-stage systems distinguish themselves from single-stage systems by the two-step strategy of selecting a_t. First, each nominator $n \in [N]$ selects a single candidate $a_{n,t}$ from its *assigned pool* $\mathcal{A}_n \subseteq \mathcal{A}$ ($\mathcal{A}_n \neq \varnothing$, $\bigcup_n \mathcal{A}_n = \mathcal{A}$). Second, the ranker selects an item a_t from the *candidate pool* $\mathcal{C}_t \coloneqq \{a_{1,t}, \dots, a_{N,t}\}$, and observes the reward $r_t = r_{ta_t}$ associated with a_t. Since the goal is *expected* reward maximization, recommendation quality is measured by *instantaneous regret* $r_t^\star - r_t$ where $r_t^\star = r_{ta_t^\star}$ is the reward associated with an optimal arm $a_t^\star \in \operatorname{argmax}_{a\in\mathcal{A}} f^\star(x_t,a)$. This leads us to an important identity for the *(cumulative) regret* in two-stage systems used throughout

$$\mathrm{R}_T^{\mathtt{2s}} = \sum_{t=1}^{T} r_t^\star - r_t = \underbrace{\sum_{t=1}^{T}(r_t^\star - r_{t\tilde{a}_t})}_{=:\mathrm{R}_T^{\mathtt{N}}} + \underbrace{\sum_{t=1}^{T}(r_{t\tilde{a}_t} - r_t)}_{=:\mathrm{R}_T^{\mathtt{R}}}, \tag{1}$$

with $\tilde{a}_t \in \operatorname{argmax}_{a\in\mathcal{C}_t} f^\star(x_t,a)$. In words, $\mathrm{R}_T^{\mathtt{N}}$ is the *nominator regret*, quantifying the difference between the best action presented to the ranker $\tilde{a}_t$ and the best overall action $a_t^\star$, and $\mathrm{R}_T^{\mathtt{R}}$ is the *ranker regret* which measures the gap between the choice of the ranker a_t and the best action in the candidate set $\mathcal{C}_t$. The two-stage recommendation process is summarized in Figure 1.

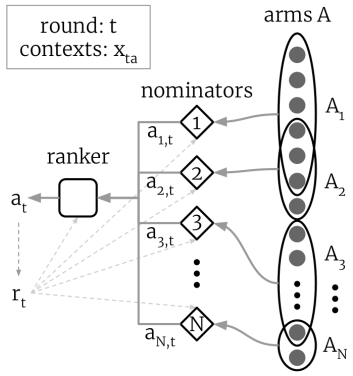

Figure 1: The two-stage recommendation setup.

While Equation (1) is typical for the *bandit* literature (i.e., data collected interactively, only r_{ta_t} revealed for each t), we also consider the *offline* learning case (i.e., a dataset with all $\{r_{ta}\}_{ta}$ revealed is given). In particular, Section 3 presents an empirical comparison of single- and two-stage systems in the bandit setting, followed by a theoretical analysis with implications for both learning setups.

3 Comparing single- and two-stage systems

Since the ranker and nominators could each be deployed independently, one may wonder whether the performance of a two-stage system is significantly affected by factors beyond the hypothetical single-stage performance of its components. This question is both theoretically (new developments needed?) and practically interesting (e.g., training components independently, as common, assumes targeting single-stage behavior is optimal). In Section 3.1, we empirically show that while factors known from the single-stage literature also affect two-stage systems, there are two-stage *specific* properties which can be even more important. Section 3.2 then investigates these properties theoretically, revealing a non-trivial interaction between the nominator training set curation and the pool allocations $\{\mathcal{A}_n\}_n$.

3.1 Empirical observations

We study the effects of item pool size, dimensionality, misspecification, nominator count, and the choice of ranker and nominator algorithms. We compare single- and two-stage systems where each (component) estimates the expected rewards using a linear model $f_t(x, a) = \langle \hat{\theta}_t, x_a \rangle$, with a different x_a given for each a. Abbreviating $x_t = x_{ta_t}$, the estimates are converted into a policy via either:

1. **UCB (U)** [6, 61] computes the ridge-regression estimate with regularizer $\lambda > 0$

$$\hat{\theta}_t \coloneqq \Sigma_t \sum_{i=1}^{t-1} x_i r_i \,, \qquad \Sigma_t \coloneqq \left(\lambda I + \sum_{i=1}^{t-1} x_i x_i^\top \right)^{-1} , \tag{2}$$

   and selects actions with exploration bonus $\alpha > 0$: $a_t \in \operatorname{argmax}_a \langle \hat{\theta}_t, x_{ta} \rangle + \alpha \sqrt{x_{ta}^\top \Sigma_t x_{ta}}$.

2. **Greedy (G)** [8, 51] can be viewed as a special case of UCB with $\alpha = 0$.

The argmax is restricted to $\mathcal{A}_n$ (resp. $\mathcal{C}_t$) in two-stage systems, with pool allocation $\{\mathcal{A}_n\}_n$ designed to minimize overlaps and approximately equalize the number of items in each pool (see Appendix B).

We use one synthetic and one real-world dataset. The **synthetic dataset** is generated using a linear model $r_{ta} = \langle \theta^\star, x_{ta} \rangle + \varepsilon_{ta}$ for each time step $t \in [T]$ and action $a \in \mathcal{A}$. The vector $\theta^\star$ is drawn uniformly from the d-dimensional unit sphere at the beginning and then kept fixed, the contexts x_{ta} are sampled independently from $\mathcal{N}(0, I)$, and $\varepsilon_{ta} \sim \mathcal{N}(0, 0.1^2)$ is independent observation noise.

The **real-world dataset** 'AmazonCat–13K' contains Amazon reviews and the associated product category labels [10, 66]. We use only the raw text features, convert them to 768-dimensional embeddings using the HuggingFace pretrained model `'bert-base-uncased'` [24, 93];[1] we further subset to the first $d = 400$ dimensions of the embeddings, which does not substantially affect the results. Since 'AmazonCat–13K' is a multi-label classification dataset, we convert it into a bandit one by taking only the label with the third highest occurrence frequency to achieve the positivity rate 10.73% when encoded as a 0-1 reward vector. An $|\mathcal{A}|$–armed linear bandit is then created by sampling $|\mathcal{A}|$ reviews uniformly from the whole dataset, and treating the associated embeddings as the contexts x_{ta}. This method of creating bandit datasets is standard in the literature [27, 35, 63, 64].

While the ranker can always access all d features, the typical lower flexibility of the nominators (*misspecification*) is modelled by restricting each to a different *random subset* of s out of the total d features, on both datasets. The same restriction is applied to the single-stage systems for comparison. Both UCB and Greedy are then run with x_t replaced by the s-dimensional $x_{n,t} = x_{n,ta_t}$ everywhere. In all rounds, each nominator is updated with $(x_{n,t}, a_t, r_t)$, regardless of whether $a_t \in \mathcal{A}_n$ (this assumption is revisited in Section 3.2). Thirty independent random seeds were used to produce the (often barely visible) two-sigma standard error regions in Figures 2 and 3. More details, including the hyperparameter tuning protocol and additional results, can be found in Appendices B and C.

Starting with the synthetic results in Figure 2, we see that the number of arms and the feature dimension d are both correlated with increased regret in single- *and* two-stage systems. Another

[1] Encoded dataset: https://twostage.s3-us-west-2.amazonaws.com/amazoncat-13k-bert.zip.

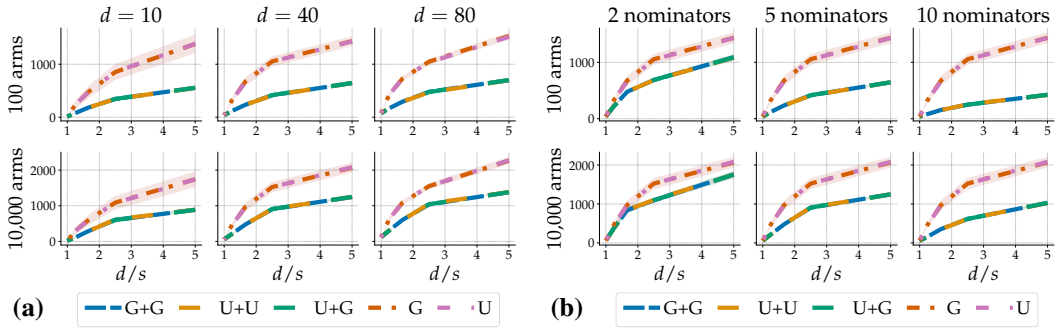

Figure 2: Synthetic data results. The x-axis is the ratio between the true feature dimension d and the size of the subset available to the nominators and the single-stage systems s. The y-axis shows the expected regret at $T = 1000$. In plot **(a)**, $N = 5$ nominators are used, and columns represent the total number of features d. In plot **(b)**, $d = 40$ features are used, and columns show the number of nominators N. The legend describes model architectures, where two-stage systems are labeled by `'[ranker]+[nominator]'` (e.g., `'U+G'` is a UCB ranker with Greedy nominators).

similarity between all the algorithms is that misspecification—as measured by d/s—also has a negative effect on the performance.[2] This is also the case for the Amazon dataset in Figure 3.

The influence of the number of arms, dimensionality, and misspecification on single-stage systems is well known [57]. Figures 2 and 3 suggest similar effects also exist for two-stage systems. On the other hand, while the directions of change in regret agree, the magnitudes do not. In particular, two-stage systems perform significantly better than their single-stage counterparts. This is possible because the ranker can exploit its access to all d features to improve upon even the best of the nominators (recall that nominators and single-stage systems share the same model architecture). In other words, *the single-stage performance of individual components does not fully explain the two-stage behavior*.

To develop further intuition about the differences between single- and two-stage systems, we zoom in on the Amazon experiments in Figure 3. The top row suggests the performance of two-stage systems improves as the number of nominators grows. Strikingly, the accompanying UCB ranker + nominator plots in the bottom row show the nominator regret $\mathrm{R}_T^{\mathtt{N}}$ dominates when there are few nominators, but gives way to the ranker regret $\mathrm{R}_T^{\mathtt{R}}$ as their number increases.

To explain this, note that the *single-stage performance of the ranker can be read off from the bottom left corner of each plot where* $d = s$ (because all the components are identical at initialization, and then updated with the same data). Since the size of the candidate set $\mathcal{C}_t$ increases with the number of nominators, the two-stage performance in the $d > s$ case eventually approaches that of the single-stage UCB ranker as well, even if the nominators are no better than random guessing. In fact, because $\approx 10\%$ of the items yield optimal reward, the probability that a set of ten uniformly random nominators with non-overlapping item pools nominates at least one optimal arm is on average $1 - (\frac{9}{10})^{10} \approx 0.65$, i.e., the instantaneous nominator regret would be zero 65% of the time.

To summarize, we have seen evidence that properties known to affect single-stage performance—number of arms, feature dimensionality, misspecification—have similar qualitative effects on two-stage systems. However, two-stage recommenders perform significantly better than any of the nominators alone, especially as the nominator count and the size of the candidate pool increase. Complementary to the evidence from offline learning [64], and the effect of ranker pretraining [42], these observations add to the case that two-stage systems should not be treated as just the sum of their parts. We add theoretical support to this argument in the next section.

[2]Misspecification error typically translates into a linear regret term ϵT [21, 25, 31, 32, 36, 55, 58]. We can thus gain *intuition* for the concavity of $d/s \mapsto \mathrm{R}_T$ from the L^2 error $\epsilon = \min_{\theta_n} (\mathbb{E}[(r_a - \langle \theta_n, x_{n,a} \rangle)^2])^{1/2}$ where $a \sim \mathrm{Unif}(\mathcal{A})$ [55]. Using $x_{ta} \sim \mathcal{N}(0, I)$, the minimum is achieved by $(\mathbb{E}[x_{n,a} x_{n,a}^\intercal])^{-1} \mathbb{E}[x_{n,a} r_a] = \theta_n^\star$, with $\theta_n^\star$ the s entries of $\theta^\star$ corresponding to the dimensions available to n. The L^2 error is thus a concave function of d/s by symmetry: $\epsilon = \sqrt{\mathbb{E}[\mathbb{E}[(r_{ta} - \langle \theta_n^\star, x_{n,ta} \rangle)^2 \mid \theta^\star]]} = \sqrt{\mathbb{E}[\|\theta^\star\|_2^2 - \|\theta_n^\star\|_2^2]} = \sqrt{1 - s/d}$.

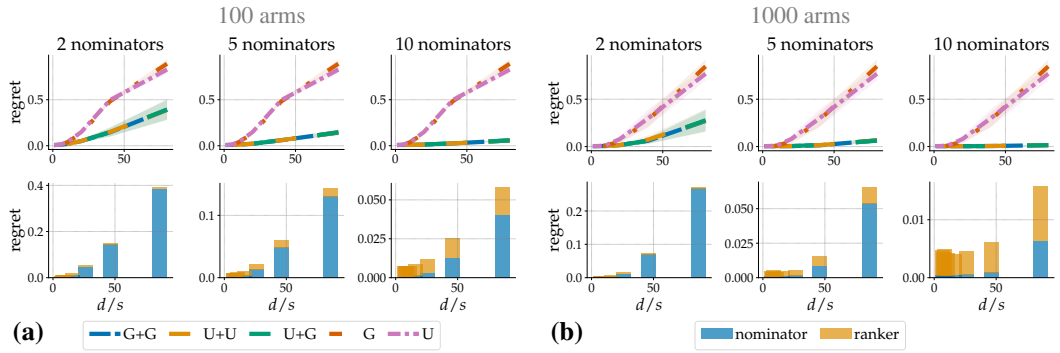

Figure 3: Amazon data results. The axes are the same as in Figure 2, except the y-axis is plotted at $T = 5000$ with the regret divided by that of a uniformly random agent. The feature dimension is fixed to $d = 400$, and the number of arms to 100 in plot **(a)**, and to 1000 in plot **(b)**. The columns represent varying number of nominators N. The legend is shared, where the one in **(a)** corresponds to the top row plots and has the same interpretation as in Figure 2, and the one in **(b)** belongs to the bottom row plots which show the proportion of ranker and nominator regret (see Equation (1)) for the `'U+U'` two-stage system as a representative example (all two-stage systems perform similarly here).

3.2 Theoretical observations

We focus on *least-squares regression oracle* (LSO) based algorithms, which estimate the expected reward by minimizing the sum of squared errors and a regularizer $\|\cdot\|_{\mathcal{F}}$, over a given model class $\mathcal{F}$

$$f_t \in \operatorname{argmin}_{f \in \mathcal{F}} \left\{ \|f\|_{\mathcal{F}} + \sum_{i=1}^{t-1} (r_i - f(x_i, a_i))^2 \right\}. \tag{3}$$

These estimates are then converted into a policy either greedily, $\pi_t(x) = \mathrm{Unif}(\operatorname{argmax}_a f_t(x, a))$, or by incorporating an exploration bonus as in LinUCB [6, 19, 22, 61, 78], or the more recent class of black-box reductions from bandit to online or offline regression [30–32, 55, 82]. The resulting algorithms are often minimax optimal, and (some) also perform well on real-world data [11].

We choose LSO-based algorithms because they (i) include the Greedy and (Lin)UCB models studied in the previous section, and (ii) allow for an easier exposition than the similarly popular cost-sensitive classification approaches [e.g., 2, 18, 20, 26, 56]. The following proposition is an application of the fact that algorithms like LinUCB or SquareCB [1, 30, 32] provide regret guarantees robust to contexts chosen by an *adaptive* adversary, and thus also to those chosen by the nominators.

Proposition 1 *Assume the ranker achieves a* single-stage *regret guarantee* $\mathrm{R}_T \leq \mathrm{B}_T^{\mathtt{R}}$ *for some constant* $\mathrm{B}_T^{\mathtt{R}} \in \mathbb{R}$ *(either in expectation or with high probability), even if the contexts* $\{x_t\}_{t=1}^T$ *are chosen by an adaptive adversary. The ranker regret then satisfies*

$$\mathrm{R}_T^{\mathtt{R}} = \sum_{t=1}^{T} r_{t\bar{a}_t} - r_t \leq \mathrm{B}_T^{\mathtt{R}},$$

in the sense of the original bound (i.e., in expectation, or with high probability).

While proving Proposition 1 may be straightforward, its consequences are not. First, if $\mathrm{B}_T^{\mathtt{R}}$ is in some sense optimal, then Equation (1) implies the two-stage regret $\mathrm{R}_T^{\mathtt{2s}}$ will be dominated by the nominator regret $\mathrm{R}_T^{\mathtt{N}}$ (unless it satisfies a similar guarantee). Second, $\mathrm{R}_T^{\mathtt{R}} \leq \mathrm{B}_T^{\mathtt{R}}$ holds exactly when the ranker is trained in the 'single-stage mode'; i.e., the tuples (x_t, a_t, r_t) are fed to the algorithm without any adjustment for the fact $\mathcal{C}_t$ is selected by a set of adaptive nominators from the whole item pool $\mathcal{A}$.

The above however does not mean that the ranker has no substantial effect on the overall behavior of the two-stage system. In particular, the feedback observed by the ranker also becomes the feedback observed by the nominators, which has the *primary* effect of influencing the nominator regret $\mathrm{R}_T^{\mathtt{N}}$, and the *secondary* effect of influencing the candidate pools $\mathcal{C}_t$, thus creating a feedback loop. Since the secondary effect is essentially handled by Proposition 1, the rest of this section focuses on the primary effect, and in particular its dependence on how the nominators are trained.

Pitfalls in designing the nominator training objective

The primary effect stems from a key property of two-stage systems: unlike the ranker, the nominators do not observe feedback for every item they nominate. While importance weighting can be used to adjust the nominator training objective [64], it does not tell us which adjustment would be optimal.

We thus study two major types of updating strategies: (i) 'training-on-all,' and (ii) 'training-on-own.' Both can be characterized in terms of the following weighted oracle objective for the nth nominator

$$f_{n,t} \in \operatorname{argmin}_{f_n \in \mathcal{F}_n} \left\{ \|f_n\|_{\mathcal{F}_n} + \sum_{i=1}^{t-1} w_{n,t}(r_i - f_n(x_{n,i}\,, a_i))^2 \right\}, \tag{4}$$

where $\mathcal{F}_n$ is the class of functions the nominator can fit, $\|\cdot\|_{\mathcal{F}_n}$ the regularizer, and $w_{n,t} = w_{n,a_t} \geq 0$ the sample weight. **'Training-on-all'**—used in Section 3.1—takes $w_{n,a} = 1$ for all (n, a), which means *all* data points are valued equally, disregarding whether a particular a_t belongs to the nominator's pool $\mathcal{A}_n$. 'Training-on-all' may potentially waste the already limited modelling capacity of the nominators if the pools $\mathcal{A}_n$ are not identical. The **'training-on-own'** alternative therefore uses $w_{n,a} = \mathbb{1}\{a \in \mathcal{A}_n\}$ so that only the data points for which $a_t \in \mathcal{A}_n$ influence the objective.[3]

While 'training-on-all' and 'training-on-own' are not the only options we could consider, they are representative of two very common strategies. In particular, 'training-on-all' is the default easy-to-implement option which can however perform surprisingly well [11, 77]. In contrast, 'training-on-own' approximates the (on-policy) 'single-stage mode' where the nominator observes feedback only for the items it selects (in particular, if the pools are non-overlapping, $a_t \in \mathcal{A}_n$ only if $a_t = a_{n,t}$).

Proposition 2 below shows that neither 'training-on-all' nor 'training-on-all' is guaranteed to perform better than random guessing in the infinite data limit ($T \to \infty$). We consider the linear setup $f^\star(x, a) = \langle \theta^\star, x_a \rangle$ for all (x, a), for fixed $\theta^\star$, where nominators use the ridge regression oracles $f_{n,t}(x_{n,}, a) = \langle \hat{\theta}_{n,t}, x_{n,a} \rangle$ defined as in Equation (2) with $\lambda \geq 0$ fixed, and $x_{n,a}$ is again a *subset* of the full feature vector x_a. We also assume the nominators take the predicted best action with non-vanishing probability (Assumption 1), which holds for all the cited LSO-based algorithms.

Assumption 1 *Let $f_{n,t}$ be as in Equation* (4)*, and denote $\mathcal{A}^{\mathtt{G}}_{n,t} := \operatorname{argmax}_{a \in \mathcal{A}_n} f_{n,t}(x_{n,t}, a)$. We assume that $\liminf_{T\to\infty} T^{-1} \sum_{1 \leq 1 \leq T} \mathbb{P}(a_{n,t} \in \mathcal{A}^{\mathtt{G}}_{n,t}) \geq \delta$, for some constant $\delta > 0$ and all n.*

Proposition 2 *There exist two* distinct *context distributions with associated pool allocations $\{\mathcal{A}_n\}_n$, and reward distributions where $r_a \in [0, 1]$ almost surely (a.s.) for all $a \in \mathcal{A}$, such that 'training-on-own' (resp. 'training-on-all') leads to a linear asymptotic two-stage regret*

$$\liminf_{T\to\infty} \frac{\mathbb{E}[\mathrm{R}^{\mathtt{2s}}_T]}{T} > 0\,.$$

Moreover, the asymptotic regret of 'training-on-all' is sublinear under the context distribution and pool allocation where 'training-on-own' suffers linear regret, and vice versa.

Proof 1 *Throughout, we use $\hat{\theta}_{n,t} \to (\mathbb{E}[x_{n,a} x_{n,a}^\top])^{-1}\, \mathbb{E}[x_{n,a} r_a] =: \theta^\star_n$ (a.s.), assuming invertibility, and that a_t is i.i.d. (Lemma 1, Appendix A); note $\theta^\star_n = \operatorname{argmin}_{\theta_n} \mathbb{E}[(r_a - \langle \theta_n, x_{n,a} \rangle)^2]$. We further take the reward noise to be zero mean and such that $r_a \in [0, 1]$ (a.s.) for all (x, a).*

Step I (full feedback): *Consider a single context setup with two nominators and three items. Let*

$$X := \begin{bmatrix} — & x_{a_{(1)}} & — \\ — & x_{a_{(2)}} & — \\ — & x_{a_{(3)}} & — \end{bmatrix} = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{5}$$

$\mathcal{A}_1 = \{a_{(1)}\}$, $\mathcal{A}_2 = \{a_{(2)}, a_{(3)}\}$, and restrict the nominators to the features in the last two columns of X. As $|\mathcal{A}_1| = 1$, the first nominator always proposes $a_{(1)}$. If feedback from all arms could be used to update $\hat{\theta}_{n,t}$ at every *t, the second nominator's oracle would in the $T \to \infty$ limit approximate*

$$\begin{aligned} \theta^\star_2 &= \operatorname{argmin}_{\theta \in \mathbb{R}^2} \mathbb{E}_{a \sim \mathit{Unif}(\mathcal{A})}[(r_a - \theta_1 x_{a,2} - \theta_2 x_{a,3})^2] \\ &= \operatorname{argmin}_{\theta \in \mathbb{R}^2} \left\{ (\bar{r}_1 + \theta_2)^2 + (\bar{r}_2 - \theta_1)^2 + (\bar{r}_3 - \theta_2)^2 \right\} = [\bar{r}_2, \tfrac{\bar{r}_3 - \bar{r}_1}{2}]^\top, \end{aligned}$$

[3] There are two possible definitions of 'training-on-own': (i) $w_{n,t} = \mathbb{1}\{a_t \in \mathcal{A}_n\}$; (ii) $w_{n,t} = \mathbb{1}\{a_t = a_{n,t}\}$. While the main text considers the former, Proposition 2 can be extended to the latter with minor modifications.

where $x_{a,i}$ is the ith entry of x_a, and $\bar{r}$ the mean reward vector for the single context ($\theta^\star$ is then $X^{-1}\bar{r}$). If $\bar{r}_1 < \bar{r}_2 < \bar{r}_3$ and $\frac{\bar{r}_3 - \bar{r}_1}{2} < \bar{r}_2$ (e.g., $\bar{r} = [\frac{1}{4}, \frac{1}{2}, 1]^\top$), then $\operatorname{argmax}_{a \in \mathcal{A}_2} \langle \theta_2^\star, [x_{a,2}, x_{a,3}]^\top \rangle = a_{(2)}$, which is not the overall *optimal $a_{(3)}$ (this is crucial in the upcoming Step II). In contrast, 'training-on-own' would yield the estimate $\theta_2^\star = [\bar{r}_2, \bar{r}_3]^\top$, and thus correctly identify $a_{(3)}$ via* argmax.

In contrast, consider the modified setup with $\mathcal{A}_1 = \{a_{(1)}\}$, $\mathcal{A}_2 = \{a_{(2)}, a_{(3)}, a_{(4)}\}$, and

$$X := \begin{bmatrix} — & x_{a_{(1)}} & — \\ — & x_{a_{(2)}} & — \\ — & x_{a_{(3)}} & — \\ — & x_{a_{(4)}} & — \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{6}$$

where the X from Equation (5) *is embedded in the bottom right block. If we restrict nominators only to the features in the last two columns of X, the second nominator will approximate $\theta_2^\star = [\bar{r}_3, \frac{\bar{r}_4 - \bar{r}_2}{2}]^\top$ when access to all labels is provided. However, the 'training-on-all' estimate would instead be $\theta_2^\star = [\bar{r}_3, \frac{\bar{r}_1 + \bar{r}_4 - \bar{r}_2}{3}]^\top$. Hence for $\bar{r} = [1 - \varepsilon, \varepsilon, \frac{1}{2}, 1]^\top$ with $\varepsilon \in (0, 1)$, 'training-on-own' leads the nominator to pick the now suboptimal $a_{(3)}$, whereas 'training-on-all' successfully identifies $a_{(4)}$.*

Step II (bandit reduction): *Take X_j, $j \in [|\mathcal{A}|]$, to be a deterministic context matrix which is the same as X from Equation* (5) *(resp. Equation* (6)*), but with all except the jth row replaced by a zeros. Take the Unif($\{X_j\}_j$) context distribution, and observe $a_t^\star = a_{(j)}$ when X_j is drawn, since the expected reward vector $X_j \theta^\star$ has only one* strictly *positive component. The all but the jth rows equal to zero also cause $\hat{\theta}_{n,t} = [\lambda I + \sum_{1 \le i \le t-1} \mathbb{1}\{a_i = a_i^\star\} w_{n,t} x_{n,i} x_{n,i}^\top]^{-1} \sum_{1 \le i \le t-1} \mathbb{1}\{a_i = a_i^\star\} w_{n,t} x_{n,i} r_i$. Crucially, $(w_{n,t}, x_{n,t}, r_t)$ are* i.i.d. conditioned on *$a_t = a_t^\star$ by construction. Hence $\hat{\theta}_{n,t} \to \theta_n^\star$ (a.s.) from Step I by Lemma 1 (Appendix A) as long as $S_t^\star := \max(1, \sum_{1 \le i \le t-1} \mathbb{1}\{a_i = a_i^\star\}) \to \infty$ (a.s.).*

Now since $\min_{a \neq a_t^\star} f^\star(x_t, a_t^\star) - f^\star(x_t, a) \ge \Delta$ (a.s.) for a fixed $\Delta > 0$ by construction, an algorithm can achieve sublinear (asymptotic) regret only if $\lim_{T \to \infty} T^{-1} \sum_{1 \le t \le T} \mathbb{P}(a_t \neq a_t^\star) = 0$. However, by Lemma 2 (Appendix A), $S_t^\star \to \infty$ (a.s.) when the regret is sublinear, implying $\hat{\theta}_{2,t} \to \theta_2^\star$ (a.s.) by the above paragraph. Because $|\mathcal{A}| < \infty$, there will be a (random) time $\tau < \infty$ such that $\operatorname{argmax}_{a \in \mathcal{A}_2} \langle \hat{\theta}_{2,t}, x_{2,ta} \rangle = \operatorname{argmax}_{a \in \mathcal{A}_2} \langle \theta_2^\star, x_{2,ta} \rangle$ for all $t \ge \tau$. We thus obtain a contradiction by combining the results of Step I, Assumption 1, and the dominated convergence theorem

$$\begin{aligned} \liminf_{T \to \infty} \mathbb{E}[T^{-1} \mathrm{R}_T^{2\mathtt{s}}] &\overset{DCT}{=} \mathbb{E}\left[\liminf_{T \to \infty} T^{-1} \sum_{t=1}^{T} \bar{r}_t^\star - \bar{r}_t\right] \ge \Delta\, \mathbb{E}\left[\liminf_{T \to \infty} \sum_{t=\tau}^{T} T^{-1} \mathbb{1}\{a_{2,t} \in \mathcal{A}_{2,t}^{\mathtt{G}}\}\right] \\ &\overset{DCT}{=} \Delta \liminf_{T \to \infty} T^{-1}\, \mathbb{E}\left[\sum_{t=\tau}^{T} \mathbb{1}\{a_{2,t} \in \mathcal{A}_{2,t}^{\mathtt{G}}\}\right] \overset{\text{Assumption 1}}{>} 0\,. \end{aligned}$$

Proposition 2 shows that training set curation can make all the difference between poor and optimal two-stage recommender.[4] Moreover, neither 'training-on-own' nor 'training-on-all' guarantees sublinear regret, and one can fail exactly when the other works. The main culprit is the difference between context distribution in and outside of each pool; combined with the misspecification, either one can result in more favorable optima from the overall two-stage performance perspective. Inspecting Steps I and II of the above proof, this is the case *both in the offline and the bandit setting*.

To summarize, beyond the actual number of nominators identified in the previous section, we have found that the combination of training objective and pool allocation can heavily influence the overall performance. We exploit these insights to improve two-stage systems in the next section.

4 Learning pool allocations with Mixture-of-Experts

Revisiting the proof of Proposition 2, we see the employed pool allocations are essentially adversarial with respect to the context distributions. However, we are typically free to design the pools ourselves, with the only constraints imposed by computational and statistical performance requirements. Proposition 2 thus hints at a positive result: a good pool allocation can help us achieve an (asymptotically) optimal performance *even in cases where this is not possible using any one of the nominators alone.*

[4] Covington et al. [20] reported empirically observing that the the training objective choice has an outsized influence on the performance of two-stage recommender systems. Proposition 2 can be seen as a theoretical complement which shows that the range of important choices goes beyond the selection of the objective.

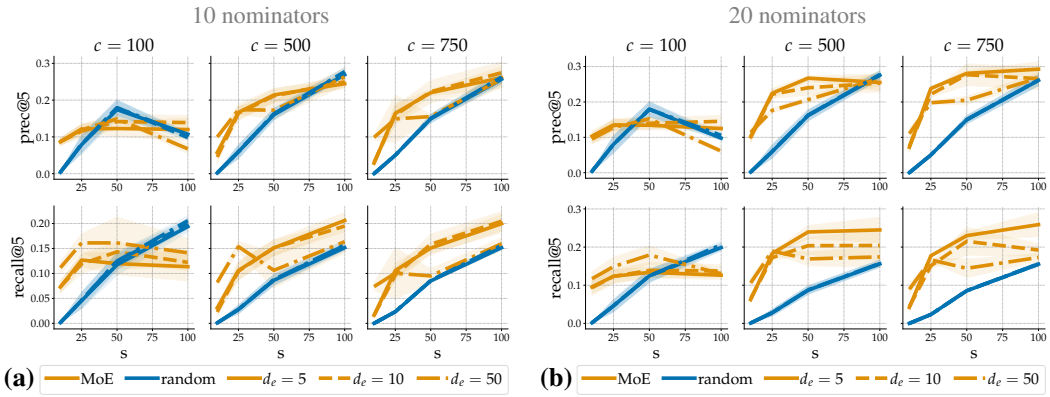

Figure 4: Mixture-of-Experts results on the 100-item Amazon dataset. The x-axis is the size of the BERT embedding dimension subset. The y-axis shows the average `precision@5` (top row) and `recall@5` (bottom row) over 50,000 entries from an independent test set (both set to zero for entries with no positive labels—about 5.5% of the test set). The columns in both plots **(a)** and **(b)** correspond to the number of examples per arm c in the training set. Ten (resp. twenty) nominators were used in **(a)** (resp. **(b)**). The legend shows whether pool allocations were learned (MoE) or randomly assigned (random), and the dimension of item embeddings d_e employed by both model types.

Crafting a good pool allocation *manually* may be difficult. We therefore propose to *learn* one instead using a **Mixtures-of-Experts** (MoE) approach [45, 47, 48, 95]. A MoE computes predictions by weighting the individual expert outputs using a trainable gating mechanism. The weighting allows each expert to specialize on a different subset of the input space, which makes the MoE more flexible than any one of the experts alone. We focus on the Gaussian MoE [45], trained by likelihood maximization (Equation (7)). Despite its occasional failure to find a good local optimum [65], we use gradient ascent, since it is easy to scale to large datasets with stochastic approximation

$$\frac{1}{T}\sum_{t=1}^{T}\log\sum_{n=1}^{N}p_{n,t}\exp\left\{-\frac{(r_t-\hat{r}_{n,t})^2}{2\sigma^2}\right\}\approx\frac{1}{B}\sum_{t=1}^{B}\log\sum_{n=1}^{N}p_{n,t}\exp\left\{-\frac{(r_t-\hat{r}_{n,t})^2}{2\sigma^2}\right\}, \quad (7)$$

with $p_{n,t}\geq 0$ ($\sum_n p_{n,t}=1$) the gating weight assigned to expert n on example t, $\hat{r}_{n,t}$ the matching expert prediction, $\sigma>0$ a hyperparameter approximating reward variance, and $B\in\mathbb{N}$ the batch size.

Exploiting the input specialization capability of MoEs, we propose to learn pool allocations by treating the individual nominators as experts. This can be done in (at least) two ways:

1. Use a *provisional* gating mechanism, and then distill the pool allocations from the learned weights, e.g., by computing the per arm average weight assigned to each nominator, and then thresholding (an alternative to the averaging is to restrict the gating network inputs only to the item features from the beginning). Once pools are divided, nominators and the ranker may be trained and deployed as usual, *using any existing infrastructure*.
2. Make the gating mechanism *permanent*, either as (i) a replacement for the ranker, or (ii) part of the nominator stage, reweighting the predictions before the candidate pool is generated.

Unusually for MoEs, we may want to use a different subset of the inputs for the gating mechanism and each of the experts, depending on which of the above options is selected. Furthermore, if some (e.g., rule based) nominators are not trained by gradient descent, they can still be plugged in during the MoE optimization, allowing the trained experts to specialize on other items.

We use the 'AmazonCat-13K' dataset [10, 66] to investigate the setup with a logistic regression gating mechanism as a part of the nominator stage. Due to the success of greedy methods in Section 3.1, and the existence of black-box reductions from bandit to offline learning [31, 82], we simplify by focusing only on offline evaluation. We employ the same preprocessing as in Section 3.1, but take the 100 highest occurrence frequency labels instead of just one (skipping the two most common labels as before) to enable nominator specialization. We compare the MoE against the same model with gating replaced by a random pool allocation fixed at the beginning. The experts in both models use a simple two-tower architecture, where d_e-dimensional dense embeddings are learned for each

item, the s-dimensional subset of the BERT embeddings is mapped to $\mathbb{R}^{d_e}$ by another trained matrix, and the final prediction is computed as the dot product on $\mathbb{R}^{d_e}$. To support low latency inference, the gating mechanism models the logits $\{\log p_n\}_n$ as a sum of learned user and item embeddings. Further details, including hyperparameter selection, are described in Appendix B.

Figure 4 shows that MoEs are able to outperform random pool allocation for most combinations of model architecture and training set size. Since the gating mechanism can learn weights which exactly recover any random pool allocation, the MoE model can perform worse only when the optimizer fails or the model overfits. This seems to be happening for the smallest training set size ($c = 100$ samples per arm), and also when the item embedding dimension d_e is high. In practice, these effects can be counteracted by regularization or alternative training approaches based on expectation–maximization or tensor decomposition [48, 65, 95], potentially yielding further performance gains.

5 Other related work

Scalable recommender systems. Interest in scalable recommenders has been driven by the growing dataset sizes [37, 60, 79, 87]. The two-stage architectures examined in this paper have seen widespread adoption in recommendation [12, 20, 28, 97, 98], and beyond [5, 96].

Off-policy learning and evaluation. Updating the recommendation policy online, without human oversight, runs the risk of compromising the service quality, and introducing unwanted behavior. Offline learning from logged data [27, 70, 85, 88] is an increasingly popular alternative [18, 46, 64, 86]. It has also found applications in search engines, advertising, robotics, and more [4, 46, 59, 84].

Ensembling and expert advice. The goal of 'learning with expert advice' [3, 7, 62, 83] is to achieve performance comparable with the best expert if deployed on its own. This is not a good alternative to our MoE approach since two-stage systems typically outperform any one of the nominators alone (Section 3). A better alternative may possibly be found in the literature on 'aggregation of weak learners' [14, 15, 33, 34, 41], or recommender ensembling (see [16] for a recent survey).

6 Discussion

We used a combination of empirical and theoretical tools to investigate the differences between single- and two-stage recommenders. Our **first major contribution** is demonstrating that beyond common factors like item pool size and model misspecification, the nominator count and division of items between the nominator pools can have even larger impact on performance in the two-stage setup. As a consequence, two-stage systems cannot be fully understood by studying their components in isolation, and the common practice of training each component independently may lead to suboptimal results. The importance of the nominator training set curation inspired our **second major contribution**: identification of a link between two-stage recommenders and Mixture-of-Experts models. Allowing each nominator to specialize on a different subset of the item pool, we were able to significantly improve the two-stage performance. Consequently, splitting items into pools within the nominator stage is not just means to lower latency, but can also improve performance.

Due to the the absence of access, a major limitation of our work is the lack of evaluation on a production system, which may be problematic due to the notorious difficulty of offline evaluation [9, 54, 76]. We further assumed that recommendation performance can be quantified in a few measurements like regret or precision/recall at K, even though design of meaningful evaluation criteria remains a challenge [23, 40, 49, 68]; we caution against deployment without careful analysis of downstream effects and broader impact assessment. Several topics were left to future work: (i) slate (multi-item) recommendation; (ii) theoretical understanding of how much can the ranker reduce the regret compared to the best of the (misspecified) nominators; (iii) alternative ways of integrating MoEs, including explicit distillation of pool allocations from the learned gating weights, learning the optimal number of nominators [74], using categorical likelihood [95], and sparse gating [29, 81].

Overall, we believe better understanding how two-stage recommenders work matters due to the enormous reach of the platforms which employ them. We hope our work inspires further inquiry into two-stage systems in particular, and the increasingly more common 'algorithm-algorithm' interactions between independently trained and deployed learning algorithms more broadly.

References

[1] Naoki Abe and Philip M Long. Associative reinforcement learning using linear probabilistic concepts. In *ICML*, 1999.

[2] Alekh Agarwal, Daniel Hsu, Satyen Kale, John Langford, Lihong Li, and Robert Schapire. Taming the monster: A fast and simple algorithm for contextual bandits. In *ICML*, 2014.

[3] Alekh Agarwal, Haipeng Luo, Behnam Neyshabur, and Robert E Schapire. Corralling a band of bandit algorithms. In *COLT*, 2017.

[4] Aman Agarwal, Ivan Zaitsev, Xuanhui Wang, Cheng Li, Marc Najork, and Thorsten Joachims. Estimating position bias without intrusive interventions. In *ACM WSDM*, 2019.

[5] Nima Asadi and Jimmy Lin. Fast candidate generation for two-phase document ranking: Postings list intersection with bloom filters. In *ACM CIKM*, 2012.

[6] Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. *JMLR*, 2002.

[7] Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, and Robert E Schapire. The nonstochastic multiarmed bandit problem. *SICOMP*, 2002.

[8] Hamsa Bastani, Mohsen Bayati, and Khashayar Khosravi. Mostly exploration-free algorithms for contextual bandits. *Management Science*, 2021.

[9] Joeran Beel and Stefan Langer. A comparison of offline evaluations, online evaluations, and user studies in the context of research-paper recommender systems. In *TPDL*, 2015.

[10] Kush Bhatia, Kunal Dahiya, Himanshu Jain, Purushottam Kar, Anshul Mittal, Yashoteja Prabhu, and Manik Varma. The extreme classification repository: Multi-label datasets and code, 2016.

[11] Alberto Bietti, Alekh Agarwal, and John Langford. A contextual bandit bake-off. *arXiv preprint*, 2018.

[12] Fedor Borisyuk, Krishnaram Kenthapadi, David Stein, and Bo Zhao. Casmos: A framework for learning candidate selection models over structured queries and documents. In *ACM SIGKDD*, 2016.

[13] James Bradbury, Roy Frostig, Peter Hawkins, Matthew J Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: Composable transformations of Python+NumPy programs, 2018.

[14] Leo Breiman. Bagging predictors. *Machine Learning*, 1996.

[15] Leo Breiman. Arcing classifiers. *Annals of Statistics*, 1998.

[16] Erion Çano and Maurizio Morisio. Hybrid recommender systems: A systematic literature review. *Intelligent Data Analysis*, 2017.

[17] Jingyuan Chen, Hanwang Zhang, Xiangnan He, Liqiang Nie, Wei Liu, and Tat-Seng Chua. Attentive collaborative filtering: Multimedia recommendation with item-and component-level attention. In *ACM SIGIR*, 2017.

[18] Minmin Chen, Alex Beutel, Paul Covington, Sagar Jain, Francois Belletti, and Ed H Chi. Top-K off-policy correction for a REINFORCE recommender system. In *ACM WSDM*, 2019.

[19] Wei Chu, Lihong Li, Lev Reyzin, and Robert Schapire. Contextual bandits with linear payoff functions. In *AISTATS*, 2011.

[20] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for YouTube recommendations. In *ACM RecSys*, 2016.

[21] Koby Crammer and Claudio Gentile. Multiclass classification with bandit feedback using adaptive regularization. *Machine Learning*, 2013.

[22] Varsha Dani, Thomas P Hayes, and Sham M Kakade. Stochastic linear optimization under bandit feedback. In *COLT*, 2008.

[23] Sarah Dean, Sarah Rich, and Benjamin Recht. Recommendations and user agency: the reachability of collaboratively-filtered information. In *ACM FAccT*, 2020.

[24] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, 2019.

[25] Simon S Du, Sham M Kakade, Ruosong Wang, and Lin F Yang. Is a good representation sufficient for sample efficient reinforcement learning? In *ICLR*, 2020.

[26] Miroslav Dudík, Daniel Hsu, Satyen Kale, Nikos Karampatziakis, John Langford, Lev Reyzin, and Tong Zhang. Efficient optimal learning for contextual bandits. In *UAI*, 2011.

[27] Miroslav Dudík, John Langford, and Lihong Li. Doubly robust policy evaluation and learning. In *ICML*, 2011.

[28] Chantat Eksombatchai, Pranav Jindal, Jerry Zitao Liu, Yuchen Liu, Rahul Sharma, Charles Sugnet, Mark Ulrich, and Jure Leskovec. Pixie: A system for recommending 3+ billion items to 200+ million users in real-time. In *ACM WWW*, 2018.

[29] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *arXiv preprint*, 2021.

[30] Dylan Foster and Alexander Rakhlin. Beyond UCB: Optimal and efficient contextual bandits with regression oracles. In *ICML*, 2020.

[31] Dylan Foster, Alekh Agarwal, Miroslav Dudik, Haipeng Luo, and Robert Schapire. Practical contextual bandits with regression oracles. In *ICML*, 2018.

[32] Dylan Foster, Claudio Gentile, Mehryar Mohri, and Julian Zimmert. Adapting to misspecification in contextual bandits. In *NeurIPS*, 2020.

[33] Jerome H Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 2001.

[34] Jerome H Friedman. Stochastic gradient boosting. *Computational Statistics and Data Analysis*, 2002.

[35] Claudio Gentile and Francesco Orabona. On multilabel classification and ranking with bandit feedback. *JMLR*, 2014.

[36] Avishek Ghosh, Sayak Ray Chowdhury, and Aditya Gopalan. Misspecified linear bandits. In *AAAI*, 2017.

[37] Prem Gopalan, Jake M Hofman, and David M Blei. Scalable recommendation with hierarchical poisson factorization. In *UAI*, 2015.

[38] Charles R Harris, K Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E Oliphant. Array programming with NumPy. *Nature*, 2020.

[39] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *ACM WWW*, 2017.

[40] Jonathan L Herlocker, Joseph A Konstan, Loren G Terveen, and John T Riedl. Evaluating collaborative filtering recommender systems. *TOIS*, 2004.

[41] Tin Kam Ho. Random decision forests. In *ICDAR*, 1995.

[42] Jiri Hron, Karl Krauth, Michael I Jordan, and Niki Kilbertus. Exploration in two-stage recommender systems. *REVEAL (ACM RecSys workshop)*, 2020.

[43] John D Hunter. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 2007.

[44] Eugene Ie, Vihan Jain, Jing Wang, Sanmit Narvekar, Ritesh Agarwal, Rui Wu, Heng-Tze Cheng, Tushar Chandra, and Craig Boutilier. SlateQ: A tractable decomposition for reinforcement learning with recommendation sets. In *IJCAI*, 2019.

[45] Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural Computation*, 1991.

[46] Thorsten Joachims, Adith Swaminathan, and Tobias Schnabel. Unbiased learning-to-rank with biased feedback. In *ACM WSDM*, 2017.

[47] Michael I Jordan and Robert A Jacobs. Hierarchies of adaptive experts. In *NeurIPS*, 1992.

[48] Michael I Jordan and Robert A Jacobs. Hierarchical mixtures of experts and the EM algorithm. In *IEEE IJCNN*, 1993.

[49] Marius Kaminskas and Derek Bridge. Diversity, serendipity, novelty, and coverage: a survey and empirical analysis of beyond-accuracy objectives in recommender systems. *TiiS*, 2016.

[50] Wang-Cheng Kang and Julian McAuley. Candidate generation with binary codes for large-scale top-N recommendation. In *ACM CIKM*, 2019.

[51] Sampath Kannan, Jamie H Morgenstern, Aaron Roth, Bo Waggoner, and Zhiwei Steven Wu. A smoothed analysis of the greedy algorithm for the linear contextual bandit problem. In *NeurIPS*, 2018.

[52] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, Carol Willing, and Jupyter development team. Jupyter notebooks - a publishing format for reproducible computational workflows. In *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, 2016.

[53] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 2009.

[54] Karl Krauth, Sarah Dean, Alex Zhao, Wenshuo Guo, Mihaela Curmei, Benjamin Recht, and Michael I Jordan. Do offline metrics predict online performance in recommender systems? *arXiv preprint*, 2020.

[55] Sanath K Krishnamurthy, Vitor Hadad, and Susan Athey. Tractable contextual bandits beyond realizability. In *AISTATS*, 2021.

[56] John Langford and Tong Zhang. The epoch-greedy algorithm for multi-armed bandits with side information. In *NeurIPS*, 2008.

[57] Tor Lattimore and Csaba Szepesvári. *Bandit algorithms*. Cambridge University Press, 2020.

[58] Tor Lattimore, Csaba Szepesvári, and Gellert Weisz. Learning with good feature representations in bandits and in RL with a generative model. In *ICML*, 2020.

[59] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint*, 2020.

[60] Chao Li, Zhiyuan Liu, Mengmeng Wu, Yuchi Xu, Huan Zhao, Pipei Huang, Guoliang Kang, Qiwei Chen, Wei Li, and Dik Lun Lee. Multi-interest network with dynamic routing for recommendation at Tmall. In *ACM CIKM*, 2019.

[61] Lihong Li, Wei Chu, John Langford, and Robert E Schapire. A contextual-bandit approach to personalized news article recommendation. In *ACM WWW*, 2010.

[62] Nick Littlestone and Manfred K Warmuth. *The weighted majority algorithm*. UC Santa Cruz, Computer Research Laboratory, 1989.

[63] Romain Lopez, Inderjit Dhillon, and Michael I Jordan. Learning from extreme bandit feedback. In *AAAI*, 2021.

[64] Jiaqi Ma, Zhe Zhao, Xinyang Yi, Ji Yang, Minmin Chen, Jiaxi Tang, Lichan Hong, and Ed H Chi. Off-policy learning in two-stage recommender systems. In *ACM WWW*, 2020.

[65] Ashok Makkuva, Pramod Viswanath, Sreeram Kannan, and Sewoong Oh. Breaking the gridlock in Mixture-of-Experts: Consistent and efficient algorithms. In *ICML*, 2019.

[66] Julian McAuley and Jure Leskovec. Hidden factors and hidden topics: Understanding rating dimensions with review text. In *ACM RecSys*, 2013.

[67] Silvia Milano, Mariarosaria Taddeo, and Luciano Floridi. Recommender systems and their ethical challenges. *AI & Society*, 2020.

[68] Smitha Milli, Luca Belli, and Moritz Hardt. From optimizing engagement to measuring value. In *ACM FAccT*, 2021.

[69] Andriy Mnih and Ruslan Salakhutdinov. Probabilistic matrix factorization. *NeurIPS*, 2007.

[70] Remi Munos, Tom Stepleton, Anna Harutyunyan, and Marc Bellemare. Safe and efficient off-policy reinforcement learning. In *NeurIPS*, 2016.

[71] The pandas development team. pandas-dev/pandas: Pandas, 2020.

[72] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019.

[73] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jak Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in Python. *JMLR*, 2011.

[74] Carl Edward Rasmussen and Zoubin Ghahramani. Infinite mixtures of Gaussian process experts. In *NeurIPS*, 2001.

[75] Steffen Rendle. Factorization machines. In *IEEE ICDM*, 2010.

[76] Marco Rossetti, Fabio Stella, and Markus Zanker. Contrasting offline and online results when evaluating recommendation algorithms. In *ACM RecSys*, 2016.

[77] Mark Rowland, Will Dabney, and Remi Munos. Adaptive trade-offs in off-policy learning. In *AISTATS*, 2020.

[78] Paat Rusmevichientong and John N Tsitsiklis. Linearly parameterized bandits. *Mathematics of Operations Research*, 2010.

[79] Badrul M Sarwar, George Karypis, Joseph Konstan, and John Riedl. Recommender systems for large-scale e-commerce: Scalable neighborhood formation using clustering. In *IEEE ICCIT*, 2002.

[80] Tobias Schnabel, Adith Swaminathan, Ashudeep Singh, Navin Chandak, and Thorsten Joachims. Recommendations as treatments: Debiasing learning and evaluation. In *ICML*, 2016.

[81] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated Mixture-of-Experts layer. In *ICLR*, 2017.

[82] David Simchi-Levi and Yunzong Xu. Bypassing the monster: A faster and simpler optimal algorithm for contextual bandits under realizability. *SSRN*, 2020.

[83] Adish Singla, Hamed Hassani, and Andreas Krause. Learning to interact with learning agents. In *AAAI*, 2018.

[84] Alexander L Strehl, John Langford, Lihong Li, and Sham M Kakade. Learning from logged implicit exploration data. In *NeurIPS*, 2010.

[85] Adith Swaminathan and Thorsten Joachims. Batch learning from logged bandit feedback through counterfactual risk minimization. *JMLR*, 2015.

[86] Adith Swaminathan, Akshay Krishnamurthy, Alekh Agarwal, Miroslav Dudík, John Langford, Damien Jose, and Imed Zitouni. Off-policy evaluation for slate recommendation. In *NeurIPS*, 2017.

[87] Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. Scalable collaborative filtering approaches for large recommender systems. *JMLR*, 2009.

[88] Philip Thomas and Emma Brunskill. Data-efficient off-policy policy evaluation for reinforcement learning. In *ICML*, 2016.

[89] Guido van Rossum and Fred L Drake. *Python 3 Reference Manual*. CreateSpace, 2009.

[90] Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J van der Walt, Matthew Brett, Joshua Wilson, K Jarrod Millman, Nikolay Mayorov, Andrew R J Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E A Quintero, Charles R Harris, Anne M Archibald, Antônio H Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 2020.

[91] Michael L Waskom. seaborn: statistical data visualization. *Journal of Open Source Software*, 2021.

[92] Wes McKinney. Data structures for statistical computing in Python. In *Python in Science*, 2010.

[93] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *EMNLP*, 2020.

[94] Xinyang Yi, Ji Yang, Lichan Hong, Derek Zhiyuan Cheng, Lukasz Heldt, Aditee Kumthekar, Zhe Zhao, Li Wei, and Ed Chi. Sampling-bias-corrected neural modeling for large corpus item recommendations. In *ACM RecSys*, 2019.

[95] Seniha Esen Yuksel, Joseph N. Wilson, and Paul D. Gader. Twenty years of mixture of experts. *IEEE Transactions on Neural Networks and Learning Systems*, 2012.

[96] Longkai Zhang, Houfeng Wang, and Xu Sun. Coarse-grained candidate generation and fine-grained re-ranking for Chinese abbreviation prediction. In *EMNLP*, 2014.

[97] Zhe Zhao, Lichan Hong, Li Wei, Jilin Chen, Aniruddh Nath, Shawn Andrews, Aditee Kumthekar, Maheswaran Sathiamoorthy, Xinyang Yi, and Ed Chi. Recommending what video to watch next: a multitask ranking system. In *ACM RecSys*, 2019.

[98] Han Zhu, Xiang Li, Pengye Zhang, Guozheng Li, Jie He, Han Li, and Kun Gai. Learning tree-based deep model for recommender systems. In *ACM SIGKDD*, 2018.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] Sections 1 and 6.
   (b) Did you describe the limitations of your work? [Yes] Section 6.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] Sections 1 and 6.

(d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] All assumptions are included in the original statements in the main paper.
   (b) Did you include complete proofs of all theoretical results? [Yes] Complete proofs either follow the original statements directly or are provided in full detail in Appendix A
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] The code required to run our experiments is included in the supplemental material. We include code to generate the synthetic data. The Amazon dataset is too large, so we provide a URL in Section 3.1 instead.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] We describe the key parameters in the main body, and all remaining details in Appendix B.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We use 30 different random seeds in Section 3.1, and 3 in Section 4.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Appendix B. We remark that we cannot reconstruct the overall CPU hours used that led up to this submission.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [No] The cited sources do not include any licensing information.
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [No] The Amazon reviews dataset was created by scraping publicly posted reviews, but the original creators do not discuss if and how was consent obtained.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No] The Amazon product reviews may contain offensive language, and contain personally identifiable information.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]