Mixing Bandits: A Recipe for Improved Cold-Start Recommendations in a Social Network

Stéphane Caron*
University of Tokyo
Japan
stephane.caron@ens.fr

Smriti Bhagat
Technicolor
Palo Alto, CA
smriti.bhagat@technicolor.com

ABSTRACT

Recommending items to new or "cold-start" users is a challenging problem for recommender systems. Collaborative filtering approaches fail when the preference history of users is not available. A promising direction that has been explored recently [12] is to utilize the information in the social networks of users to improve the quality of cold-start recommendations. That is, given that users are part of a social network, a new user shows up in the network with no preference history and limited social links, the recommender system tries to learn the user's tastes as fast as possible.

In this work, we model the learning of preferences of cold-start users using multi-armed bandits [5] embedded in a social network. We propose two novel strategies leveraging neighborhood estimates to improve the learning rate of bandits for cold-start users. Our first strategy, MixPair, combines estimates from pairs of neighboring bandits. It extends the well-known UCB1 algorithm [5] and inherits its asymptotically optimal guarantees. Although our second strategy, MixNeigh, is a heuristic based on consensus in the neighborhood of a user, it performed the best among the evaluated strategies. Our experiments on a dataset from Last.fm show that our strategies yield significant improvements, learning 2 to 5 times faster than our baseline, UCB1.

Categories and Subject Descriptors

H.3.5 [**Information Systems**]: On-line Information Services; I.2.6 [**Computing Methodologies**]: Learning

General Terms

Algorithms, Experimentation

Keywords

Cold-start problem, multi-armed multi-bandits, social networks, recommender systems

*The work was done when the author was at Technicolor.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SNA-KDD The 7th Workshop on Social Network Mining and Analysis, August 11, 2013, Chicago, USA.
Copyright 2013 ACM 978-1-4503-2330-7 ...$15.00.

1. INTRODUCTION

Collaborative filtering methods [13] are widely used by recommendation services to predict the items that users are likely to enjoy. These methods rely on the consumption history of users to determine the similarity between users (or items), with the premise that similar users consume similar items. Collaborative filtering approaches are highly effective when there is sufficient data about user preferences. However, they face a fundamental problem when new users who have no consumption history join the recommendation service [22]. A new user needs to enter a significant amount of data before collaborative filtering methods start providing useful recommendations. In this work, we study this specific problem of recommending items to new users, referred to as the "cold-start" recommendation problem.

Ideally, a recommender system would like to quickly learn the likes and dislikes of cold-start users (i.e., new users), while providing good initial recommendations with fewest mistakes. To minimize its mistakes, a recommender system could recommend the item predicted as the "best" from its current knowledge of the user. However, this may not be optimal as the system has very limited knowledge of a cold-start user. On the other hand, the system may try to gather more information about the user's preferences by recommending items that may not appear to be the "best", and learning from the user's response. This inherent trade-off between exploration (trying out all items) and exploitation (selecting the best item so far) is aptly captured by the Multi-Armed Bandit model [4, 5], a framework which has been extensively studied within the Machine Learning community.

In this model, a decision maker repeatedly chooses among a finite set of K actions. At each step t, the action a chosen yields a reward $X_{a,t}$ drawn from a probability distribution intrinsic to a and unknown to the decision maker. The goal for the latter is to learn, as fast as possible, which are the actions yielding maximum reward in expectation. Multiple algorithms have been proposed within this framework. In particular, a family of policies based on Upper Confidence Bounds (UCBs) [5, 4, 23, 6] has been shown to achieve optimal asymptotic performances (in a sense defined by a seminal paper [14] by Lai and Robbins) in terms of the number of steps t.

However, recent works have shown that one can leverage side observations from the social network to provide even stronger guarantees and faster learning rates [18, 7]. Building on the idea of using additional information from an underlying social network among users, we model the cold-start

problem as one of learning bandits in a graph where each node is a bandit, and neighboring bandits have "close" reward distributions. After formally defining this multi-armed multi-bandit setting, we propose novel strategies to improve the learning rate of "young" bandits (i.e., which have been played a small number of times) by leveraging the information from their neighbors in the network.

Intuitively, our solution converges faster when friends have similar tastes, a well-accepted indicator for predicting users' preferences [8]. In addition, Jamali et al. [12] have shown the usefulness of social data in improving the quality of cold-start recommendations. To the best of our knowledge, we are the first to leverage the neighborhood over multiple multi-armed bandits embedded in a graph.

The presence of a social network among users of recommender systems is fairly common. Some services such as Last.fm (`www.last.fm`) and YouTube (`www.youtube.com`) allow users to explicit declare their social connections, while others including Hulu (`www.hulu.com`), Digg (`digg.com`), and Spotify (`www.spotify.com`), are integrated with Facebook (`www.facebook.com`). Facebook integration provides users with the simplicity and convenience of logging in to many different services with a single Facebook account, and in return, the services get access to rich social data that users share on Facebook. In this work, we propose a principled approach for exploiting the information known about the friends of a new user in order to quickly learn the preferences of this new user.

In this study, we focus on recommender systems for the music domain[1]. Our setup is as follows. Users sign in to the recommender system to get music recommendations. When new users sign in for the first time, their social graph information is gathered. This may be from their account on social networking sites such as Facebook, their email address books, or an interface where users can explicitly friend other users of the recommender system. Once a user is signed in, the recommender system picks an artist and samples a song by that artist to recommend to the user. The user may choose to skip the song if she does not enjoy it and move to the next song. From such repetitive feedback, the system wants to learn as fast as possible a set of artists that the user likes, giving her an incentive to continue to use the service.

The key contributions of our work include:

- We are the first to propose mixing strategies over multiple bandits embedded in ego-centric networks.
- In contrast to prior work that optimizes accuracy of the estimates (measured using RMSE) [12], our focus is on *fast* discovery of top items for cold-start users.
- We propose novel metrics for comparing the performances of bandit strategies in the cold-start regime (whereas the bandit literature usually focuses on the asymptotic regime).
- We empirically evaluate our strategies and show that they achieve up to 5× improvement in learning preferences of cold-start users.

Outline. The rest of the paper is organized as follows. In Section 2 we describe our framework, and present the background on bandit algorithms. Next, we describe baseline strategies for recommendation using bandits, followed by two strategies–MixPair and MixNeigh–geared towards improving cold-start recommendations in Section 3. We detail our evaluation metrics in Section 4 and present our experimental results in Section 5. We discuss related work in Section 6 and conclude in Section 7.

[1]Our framework is general and can be applied to other domains such as news, movies, etc.

$N(u)$	neighborhood of user u
A_u	set of artists user u has listened to
S_u	set of artists to suggest to user u
$\mathsf{pc}_u[a]$	# times user u listened to artist a
$p_{u,a}$	$\mathbb{P}$[user u likes a song from artist a]
$\overline{\mathbf{X}} = (X_a)$	estimate vector (intrinsic)
$\overline{\mathbf{Y}} = (Y_a)$	estimate vector (from neighbors)
$\mathbf{n} = (n_a)$	sample-size vectors
r	reward profile of a strategy
b, c	confidence radii

Table 1: Notations

2. PROBLEM SETUP

Consider a social graph $G = (V, E)$ and let us reserve letters u and v to denote vertices of this graph, i.e., users of a music recommender system. The set of neighbors of a vertex u is $N(u) := \{v \in V | (u, v) \in E\}$. Our scenario is the following: user u just joined the music recommender service, indicating neighbors $v \in N(u)$ that are already known to the system. The system has already collected information on them through their listening history, and we want to leverage this information to improve the performances of a multi-armed bandit $\mathcal{B}_u$ associated with user u. Table 1 describes the notation used in the rest of the paper.

After providing background on multi-armed bandits, we detail in this section how we model user preferences through their implicit feedback (play counts) and how we define the bandits $\{\mathcal{B}_u\}$.

2.1 Preliminaries

A K-armed bandit problem is defined by K distributions $\mathcal{P}_1, \ldots, \mathcal{P}_K$, one for each "arm" of the bandit, with respective means $p_1, \ldots, p_K$. When the decision maker pulls arm a at time t, she receives a reward $X_{a,t} \sim \mathcal{P}_a$. All rewards $\{X_{a,t}, a \in [\![1, K]\!], t \geq 1\}$ are assumed to be independent. We also assume that all $\{\mathcal{P}_a\}$ have support in $[0, 1]$. The mean estimate for $\mathbf{E}[X_{a,\cdot}]$ after m steps is

$$\overline{X}_{a,m} := \frac{1}{m} \sum_{s=1}^{m} X_{a,s}$$

The standard measure of a bandit's performances is its expected (cumulative) *regret* after T steps, defined as

$$\mathbf{E}[R(T)] := \sum_{t=1}^{T} \mathbf{E}[X_{a^*,t} - X_{I(t),t}]$$

where $a^* = \arg\max\{p_a\}$ and $I(t)$ is the index of the arm played at time t. Another (equivalent) measure is the average per-step reward, up to the current step n:

$$r(T) := \frac{1}{T} \sum_{t<T} X_{I(t)} \;\Rightarrow\; \mathbf{E}[r(T)] = p^* - \frac{1}{T}\mathbf{E}[R(T)] \quad (1)$$

DEFINITION 1. *A* reward profile *of a strategy is a function* $t \mapsto r(t)$ *mapping any time step* $t \in [\![1, T]\!]$ *to the average per-step reward up to time* t *of a given run, as defined in Equation* (1).

In what follows, for a run of bandit $\mathcal{B}_u$, we denote by $n_{u,a}(t)$ the number of times arm a has been pulled up to time t and $\overline{X}_{u,a}(t)$ the corresponding reward estimate. We call $\mathbf{n}_u$ and $\overline{\mathbf{X}}_u$ the *sample-size* and *estimate* vectors (indexed by arms), respectively.

2.2 Modeling user preferences

The k-hop *ego-centric* network of user u is defined as the sub-graph of G comprising u and all vertices that are less than k hops away from u. Each user u has listened to a given set of artists A_u, and for $a \in A_u$, we denote by $\mathsf{pc}_u[a]$ (play counts) the number of times u has listened to any song from a. Thus, the probability that a song sampled uniformly from u's listening history comes from a is

$$\pi_{u,a} := \frac{\mathsf{pc}_u[a]}{\sum_{a \in A_u} \mathsf{pc}_u[a]}$$

Within our framework, each user u is associated with a multi-armed bandit $\mathcal{B}_u$ whose arms correspond to artists $a \in A_u$. During consecutive steps t, the bandit strategy picks an artist $a \in A_u$ and recommends it to u; the reward is 1 if u accepts the recommendation and 0 otherwise. We model this reward as a random variable following a Bernoulli distribution $B(p_{u,a})$, where $p_{u,a} = \mathbb{P}[u \text{ likes a song from } a]$ will be modeled from the data.

It has been established that user play counts distributions tend to follow a power law, which we also observed in our dataset. Therefore, using $\pi_{u,a}$ as ground-truth $p_{u,a}$ would result in putting all the weight on a single artist and give similar losses to the others, a trivial learning problem where one would only discover *the* top artist. Instead, we are interested in fast learning of a set of top-artist. This is a well-known issue in recommender systems with implicit feedback [11]. An effective solution is to transform π_u using a logistic function and define $\mathbf{p}_u$ as:

$$p_{u,a} := \frac{1}{1 + e^{-\gamma_u(\pi_{u,a} - \nu_u)}}, \tag{2}$$

where γ_u and ν_u are scalars defined w.r.t. the complete distribution π_u. We experimentally found the values $\nu_u := \text{median}(\pi_u)$ and $\gamma_u := 5/\nu_u$ to discriminate well between the most and least liked artists in the crawled artist sets.

2.3 Artists selection

Our solutions follow two steps: (1) compute a set S_u of artists that u may like, then (2) learn online the top artists of u in S_u. As we focus on the first recommendations made to u, we want to keep $|S_u|$ reasonably small: otherwise, by the time the learning algorithm has tried every action once, one wouldn't be in a cold start situation any more. We define

$$S_u := \bigcap_{v \in N(u)} A_v, \tag{3}$$

i.e., artists that all your neighbors have heard of. This follows the homophily property that users are more likely to listen and like artists that their friends listen to. Taking a strict intersection is a conservative option; a more general approach would be to consider artists that at least k neighbors have heard of, i.e., $S_u := \cup_{\{v_1,\ldots,v_k\} \subset N(u)} \cap_{i=1}^{k} A_{v_i}$. We leave a thorough exploration of this for future work.

To summarize, we have a multi-armed bandit $\mathcal{B}_u$ associated with cold-start user u. The arm set of $\mathcal{B}_u$ is $S_u \subset A_u$, and the expected reward of arm $a \in S_u$ is $p_{u,a}$ as defined by Equation (2). Similarly, each neighbor $v \in N(u)$ has a bandit $\mathcal{B}_v$. We assume that these neighbors are already subscribers of the recommender system and their respective bandits $\mathcal{B}_v$ have been trained when u joins the service. The goal of the strategies we propose in the next section is to learn $\mathcal{B}_u$'s optimal arms as fast as possible.

Algorithm 1 UCB1

$\overline{\mathbf{X}}, \mathbf{n} \leftarrow \mathbf{0}, \mathbf{0}$
for $t \geq 1$ **do**
  $a \leftarrow \arg\max_{a \in S_u} \left\{ \overline{X}_a + \sqrt{2\log(t)/n_a} \right\}$
  pull arm a, getting reward $X_{a,t}$
  $n_a \leftarrow n_a + 1$
  $\overline{X}_a \leftarrow 1/n_a X_{a,t} + (1 - 1/n_a)\overline{X}_i$

3. STRATEGIES

In this section we describe three strategies. The first strategy is the well-known UCB1 policy which we use as a baseline. Next, we propose two novel strategies combining information from both the current bandit $\mathcal{B}_u$ and neighboring bandits $\{\mathcal{B}_v\}_{v \in N(u)}$.

3.1 UCB1: A baseline

One of the most prominent algorithms in the stochastic bandit literature, UCB1 from Auer et al. [5] achieves a logarithmic expected regret:

$$\mathbf{E}[R(t)] = O(\log t) \;\Leftrightarrow\; \mathbf{E}[r(t)] \underset{t \to \infty}{\sim} p^* - \frac{\kappa \log t}{t}$$

for some constant κ. A previous result from Lai and Robbins [14] states that this is optimal in terms of the number of steps t, i.e., the asymptotic regret of any multi-armed bandit strategy is $\Omega(\log t)$.

UCB1 chooses which arm a to play at time t based on an Upper Confidence Bound (UCB), which is the sum of two terms: the empirical average $\overline{X}_a$ on its past rewards, and a *bias* term $b_a(t) := \sqrt{2\log t/n_a}$, which is a confidence bound on the exactness of $\overline{X}_a$. The arm chosen is the one maximizing the sum of these two terms. Intuitively, an arm is thus played because it has either high historical rewards or its outcome is uncertain. This strategy is summarized in Algorithm 1.

Later in Section 5, we validate UCB1 as a reasonable baseline by showing that it performs better than naive strategies in our cold-start setting.

3.2 MixPair strategy

Our first strategy, MixPair, considers single edges $(u, v) \in E$ of the social graph, where user u is a cold-start user and the system has already collected a listening history for v. Following the intuition that u and v have similar tastes, our MixPair strategy computes upper confidence bounds based on all samples seen so far, both from $\mathcal{B}_u$ and $\mathcal{B}_v$.

Formally, let us denote by $\overline{\mathbf{X}}(t)$ and $\mathbf{n}(t)$ (resp. $\overline{\mathbf{Y}}$ and m) the estimate and sample-size vectors of $\mathcal{B}_u$ (resp. $\mathcal{B}_v$) after

Algorithm 2 MixPair

1: $\overline{\mathbf{X}}, \mathbf{n} \leftarrow \mathbf{0}, \mathbf{0}$
2: **for** $t \geq 1$ **do**
3: **pick** $v \in N(u)$
4: $a \leftarrow \arg\max_{a \in S_u} \left\{ \overline{Z}_a(v) + b_a(v,t) \right\}$
5: **pull** arm a, getting reward $X_{a,t}$
6: $n_a \leftarrow n_a + 1$
7: $\overline{X}_a \leftarrow 1/n_a X_{a,t} + (1 - 1/n_a)\overline{X}_i$

$\mathcal{B}_u$ has been played t times. We assume that neighboring bandits are not being played while $\mathcal{B}_u$ is trained, hence $\overline{\mathbf{X}}(t)$ and $\mathbf{n}(t)$ are parametrized by t while $\overline{\mathbf{Y}}$ and $\mathbf{m}$ are step-independent. For an arm $a \in S_u$, we define

$$\overline{Z}_a(v) := \frac{n_a(t)}{n_a(t) + m_a}\overline{X}_a(t) + \frac{m_a}{n_a(t) + m_a}\overline{Y}_a,$$

$$b_a(v,t) := \sqrt{\frac{2 \log t}{n_a(t) + m_a}}.$$

MixPair is an UCB strategy, using $\overline{\mathbf{Z}}(v)$ and $\mathbf{b}(v, \cdot)$ as its exploitation and bias term, respectively. As emphasized by its name, it is designed for a pair of neighbors $(u, v) \in E$, while $|N(u)|$ is often more than 1. To aggregate data from the whole neighborhood, the neighbor v is then re-sampled from $N(u)$ at the beginning of each step t. Algorithm 2 summarizes the complete strategy.

Note that t here only accounts for the number of times bandit $\mathcal{B}_u$ has been played, which biases MixPair towards exploitation on arms a for which m_a is large (which should also correspond to high-reward arms in $\mathcal{B}_v$ if its regret is small).

There are multiple sampling processes that can be used at line 3 of Algorithm 2. We considered two natural solutions:

- Uniform sampling, based on the assumption that all neighbors are equally similar to user u;
- "Bandit" sampling: define a multi-armed bandit on neighbors $v \in N(u)$, and learn the most similar ones online.

Later, we evaluate these two approaches and observe that the Bandit sampling is more risk-averse, as detailed in Section 5.3.

3.3 MixNeigh strategy

Our previous strategy combines bandit estimates, and then aggregates them on the neighborhood through sampling. The strategy we propose thereafter, coined MixNeigh, works the other way round: we first aggregate reward estimates from all neighbors, and then choose between this aggregate and the user's empirical estimate using a heuristic based on confidence radii, as explained below.

Formally, consider a user u and an artist $a \in S_u$. Let $\mathcal{X}_a := \left\{ \overline{X}_{v,a}; v \in N(u) \right\}$ denote the set of empirical estimates for arm a in all neighboring bandits, with its average $\overline{Y}_a := \text{avg}(\mathcal{X}_a)$ and standard deviation $c_a := \sigma(\mathcal{X}_a)$. Algorithm 3 describes the MixNeigh strategy. At each step t, it decides between estimates $\overline{X}_{u,a}$ and $\overline{Y}_a$ of the reward for arm a based on a confidence criterion.

This criterion can be interpreted as follows: at step t, b_a is such that p_a lies w.h.p. in the interval $[\overline{X}_a - b_a/2; \overline{X}_a + b_a/2]$

Algorithm 3 MixNeigh

Require: neighbor estimates $\overline{Y}_a$, c_a for all $a \in S_u$
1: $\overline{\mathbf{X}}, \mathbf{n} \leftarrow \mathbf{0}, \mathbf{0}$
2: **for** $t \geq 1$ **do**
3: **for** $a \in S_u$ **do**
4: $b_a \leftarrow \sqrt{\frac{2 \log t}{n_a}}$
5: $\overline{Z}_a \leftarrow \begin{cases} \overline{Y}_a & \text{if } |\overline{X}_a - \overline{Y}_a| < \frac{b_a - c_a}{2} \\ \overline{X}_a & \text{otherwise} \end{cases}$
6: $a \leftarrow \arg\max_{a \in S_u} \left\{ \overline{Z}_a \right\}$
7: **pull** arm a, getting reward $X_{a,t}$
8: $n_a \leftarrow n_a + 1$
9: $\overline{X}_a \leftarrow 1/n_a X_{a,t} + (1 - 1/n_a)\overline{X}_i$

(see the proof of UCB1's upper bound in [5] for further details); similarly, we interpret c_a as confidence radius on $\overline{Y}_a$. When $\left[\overline{Y}_a - c_a/2; \overline{Y}_a + c_a/2\right] \subset \left[\overline{X}_a - b_a/2; \overline{X}_a + b_a/2\right]$, MixNeigh chooses it as the estimate for $p_{u,a}$, a case we illustrate in Figure 1. Otherwise, $\overline{X}_a$ is deemed as the best option. The intuition behind this heuristic is to use $\overline{Y}_a$ when it is both a more precise estimate than $\overline{X}_a$ and an admissible value for $p_{u,a}$. In our empirical evaluation, this

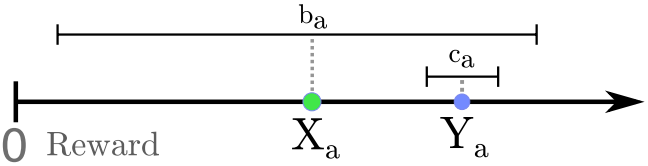


Figure 1: Case where MixNeigh will choose the estimate $\overline{Y}_a$ instead of $\overline{X}_a$

strategy interpolated nicely between using the neighborhood estimate $\overline{Y}_a$ as a prior, and exploiting the empirical estimate $\overline{X}_a$ when its precision had become good enough. Results reported in Section 5 point it out as our best strategy for the cold start problem; however, contrary to MixPair, its asymptotic behavior differs from that of UCB1 so it does not inherit the theoretical guarantees regarding convergence to the optimal solution.

4. EVALUATION METHODOLOGY

It is not straightforward to extend existing collaborative filtering-based solutions for the cold-start problem to the online framework. The evaluation metric in the former is RMSE, whereas time is a critical factor for cold-start recommendations. Hence we restrict our evaluation to baselines defined in the online setting. In this section, we propose metrics that we will use in our experiments to compare the learning rates of multi-armed bandit strategies.

4.1 Comparing two bandit runs

figwidth Consider two strategies (1) and (2) for the bandit $\mathcal{B}_u$. Running each strategy once yields two reward profiles r_1, r_2, as described in Definition 1. The standard regret metric is equivalent to comparing $r_1(t)$ and $r_2(t)$ for some time step t. However, in a cold-start setting where we want to measure an initial speedup in learning rate, the choice of an appropriate step t becomes a difficult question. We

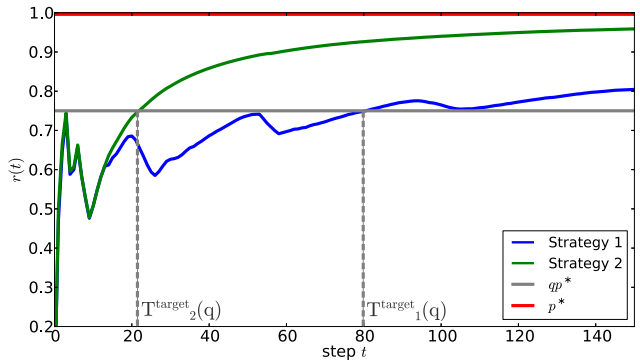


Figure 2: Example of T^{target} for two sample reward profiles

circumvent it by looking at its reciprocal: how long does it take before a strategy's average reward $r_s(t)$ gets "close enough" to the optimum value?

Let $p^* := \max_{a \in S_u}(p_{u,a})$ denote the optimal reward of bandit $\mathcal{B}_u$, and let $q \in [0, 1]$. We define a metric called *time to target*, T^{target}, as,

$$T^{\text{target}}(r, q) := \inf \{t \mid t > 10\%T,\ r(t) > qp^*\},$$

so that $T^{\text{target}}(r_s, q)$ is the first time at which strategy s achieves an average reward greater than a fraction q of the optimum. Figure 2 shows an example of two sample reward profiles in a setting where $q = 75\%$ and $p^* = 1$.

The improvement of run r_1 over run r_2 is then defined as the relative variation from $T^{\text{target}}(r_2, q)$ to $T^{\text{target}}(r_1, q)$:

$$s(r_1, r_2; q) := 1 - \frac{T^{\text{target}}(r_1, q)}{T^{\text{target}}(r_2, q)}$$

For instance, $s(r_1, r_2; q) = 50\%$ corresponds to a $2\times$ improvement of r_2 over r_1, $s(r_1, r_2; q) = 75\%$ to a $4\times$ improvement, and so on.

One can argue that this metric is optimistic as the reward profiles are not always monotonic. They tend to increase (resp. decrease) when the algorithm exploits (resp. explores), so one may observe "bumps", i.e., ordered pairs $t_1 < t_2$ such that $r(t_1) > qp^* > r(t_2)$. $T^{\text{target}}s(q)$ is thus an optimistic indicator of strategy s's learning speed. However, we note that this optimism is applied equally to both strategies $s \in 1, 2$ (we also noted in our experiments that such bumps become rare as $q \to 1$). The main idea here is to compute the time taken to first achieve a target reward. This can be extended to more relaxed definitions, where the target is said to be achieved when the reward is over a threshold for some t number of steps.

4.2 Comparing two strategies

Our metric $s(r_1, r_2; q)$ is defined over pairs of reward profiles (r_1, r_2), one per strategy $s \in \{1, 2\}$. We define the *speedup* $\phi_u(1, 2; q)$ of strategy (1) against strategy (2) in bandit $\mathcal{B}_u$ as the expected value of $s(r_1, r_2; q)$ for two random runs r_1, r_2:

$$\phi_u(1, 2; q) := \mathbf{E}_{(r_1, r_2)}\left[s(r_1, r_2; q)\right]$$

In practice, we computed $\phi_u(1, 2; q)$ through successive runs (r_{11}, r_{21}), ..., (r_{1k}, r_{2k}), until a Student's test [10] on the set $\{s(r_{1j}, r_{2j}, q); 1 \le j \le k\}$ backed the average $\widetilde{Q}_u(1, 2; q) := \frac{1}{k}\sum_{j=1}^{k} s(r_{1j}, r_{2j}; q)$ as a reliable estimate for $\phi_u(1, 2; q)$.

The final step of our comparison procedure is the analysis of the distribution $\{\phi_u(1, 2; q)\}$ over users u of the system. In particular, we focus on the following properties of this distribution for a user u sampled uniformly at random,

- $\mu := \mathbf{E}[\phi_u(1; 2)]$, the average speedup in learning for strategy (1) over strategy (2)
- $p^+ := \mathbb{P}[\phi_u(1; 2) > 0]$, the probability that strategy (1) learns faster than strategy (2)
- $\mu^+ := p^+ \cdot \mathbf{E}[\phi_u(1; 2) \mid \phi_u(1; 2) > 0]$, average speedup when (1) improves on (2), weighed by the probability of this event
- $\mu^- := (1 - p^+) \cdot \mathbf{E}[\phi_u(1; 2) \mid \phi_u(1; 2) < 0]$ average slowdown (negative speedup) when (1) is worse than (2), weighed by the probability of this event

We make use of these aggregate metrics in the following section.

5. EXPERIMENTS

In this section, we describe our dataset and how we crawled it. We show that UCB1 improves on naive bandit strategies, justifying its use as a baseline. We then compare MixPair and MixNeigh to this baseline. Finally, we evaluate the impact of the parameter q on our scoring metric. In all experiments that follow, q is set to 80%, unless specified otherwise.

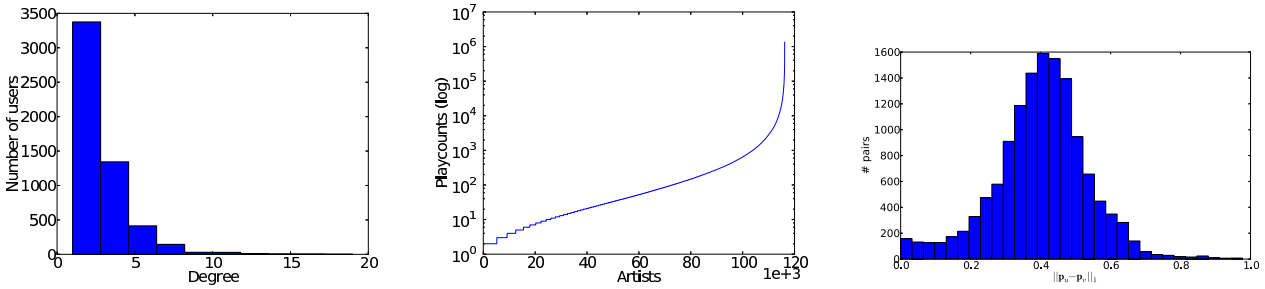
5.1 Data

We evaluate our algorithms on a dataset crawled from a music recommendation service, Last.fm, where users can listen to specific songs or radio stations compiled by the system. Users also have the option of creating their profile, where they can enter their demographic details, friend other users, and record their favorite tracks etc. We crawled a portion of the Last.fm social graph in May 2012 using the Last.fm API[2] service. We started with the user "RJ", one of the co-founders of Last.fm, and performed a breadth-first crawl from this user to obtain complete ego-centric networks around the crawled users.

Last.fm has two notions of links among users. It allows users to explicitly "friend" other users of Last.fm, and implicitly defines "neighbors" as pairs of users with similar "music tastes". We crawled the explicit friend relationships to avoid any artificial performance boost from Last.fm neighbors. For user music preferences, we crawled the 500 most frequently heard artists[3], or *top artists*, along with the number of times each artist was listened to, i.e., *artist play count*. To build a dataset for evaluating our algorithms, we included a crawled user u in our dataset if there were at least three artists in S_u heard five times each. This filtering was required to ensure that each bandit has more than two arms that we learn on, and that we have reasonable estimate on the preference of that arm (artist) by a user. The resulting dataset has ego-centric networks and artist preferences of 5,357 users of Last.fm.

[2] `http://last.fm/api`

[3] The choice of 500 artist was to ensure we have some information on the overlap among top artists of pairs of users.



(a) Degree distribution (b) Distribution of artist playcounts (c) L1 norm of the preference vectors of neighboring pairs

Figure 3: Last.fm dataset

Figure 3 illustrates the properties of the crawled graph. The filtering step resulted in small ego-centric networks, with a maximum degree of any node being 18 as shown in Figure 3(a). There are about 120K artists with play counts as shown in Figure 3(b). Finally, to get an intuition on the disagreement in the likeness of artists by neighboring nodes, we plot a histogram of the L1 norm between the preference vectors computed as $\|\mathbf{p}_u - \mathbf{p}_v\|_1$ for all $(u, v) \in E$ presented in Figure 3(c). The pairs with small values of the norm indicate higher agreement in the preferences of the two neighboring nodes. Interestingly, the mean of the distribution is 0.4, which illustrates that our dataset is not heavily biased towards agreeing pairs of nodes.

Artist selection. We defined in Equation (3) the artists S_u to be used as arms in the bandit $\mathcal{B}_u$ as those who have been listened to by all neighbors of user u. This set is easy to compute for the recommender system. However, for some users in our dataset it occurred that $A_u \cap S_u \subset S_u$, i.e., some artists heard by all neighbors have never been listened to by u. For such artists a, we did not have ground-truth to model $p_{u,a}$. In order to perform simulations, we thus used $\widetilde{S}_u := S_u \cap A_u$ instead of S_u as the set of arms for $\mathcal{B}_u$. We would like to point out that the evaluation of online learning algorithms with offline data is a difficult problem [17], hence the need to resort to such measures to build well-grounded simulation environments.

5.2 Baseline

Our first experiment compares UCB1 (which will be our baseline for the MixPair and MixNeigh strategies) to two simple multi-armed bandit strategies:

- Uniform: pick arms uniformly at random;
- ϵ_n-greedy: a variant of ϵ-greedy proposed in [5].

This last strategy has two parameters (c, d) and provable asymptotic guarantees for a optimal (instance-based) values of these parameters [5]. In this experiment, we used instance-*independent* values $c = 5$ and $d = 1$, which were the most efficient on average.

Table 2 quantifies the aggregate improvement UCB1 brings over these two baselines. As shown in the p^+ column, it is more efficient most of the time with a significant expected improvement ($\mu^+ > 30\%$). In the case of ϵ_n-greedy, however, the score distribution is heavy tailed on negative values, yielding a non-negligible μ^- as well. Figure 4 shows

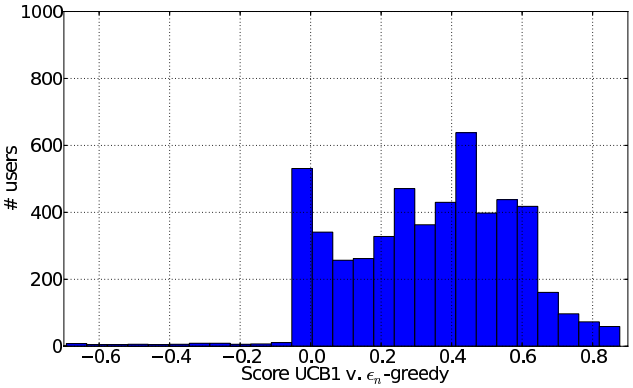


Figure 4: Performance of UCB1 against ϵ_n-greedy

	μ	p^+	μ^+	μ^-
v. Uniform	33%	**0.88**	34%	5.3%
v. ϵ_n-greedy	4.3%	**0.73**	39%	-37%

Table 2: Aggregate improvement of UCB1 over simple bandit strategies

a histogram of this distribution. The conclusion of this first experiment is two-fold. First, our statistical environment and our scoring metric are non-trivial: neither a purely exploratory nor a statically-tuned exploration/exploitation tradeoff perform well. Second, it suggests that UCB-based strategies are an interesting direction to explore in our setting.

5.3 MixPair strategy

From now on, UCB1 will be our baseline strategy. In this second experiment, we evaluate the MixPair algorithm from Section 3.2. Table 3 and Figure 5 summarize its performance in the case where neighbors are sampled uniformly at random. In particular, we note that, on the overall dataset, MixPair learns 1.54× faster than UCB1 on average.

Uniform sampling of neighbors relies on the assumption that all of them are evenly similar to the user. As this hypothesis may be questioned, we evaluated a second sampling strategy: train a multi-armed bandit on the neighbors of user u, which will learn the most similar neighbors and play them more often. In practice, for $q = 75\%$, this approach

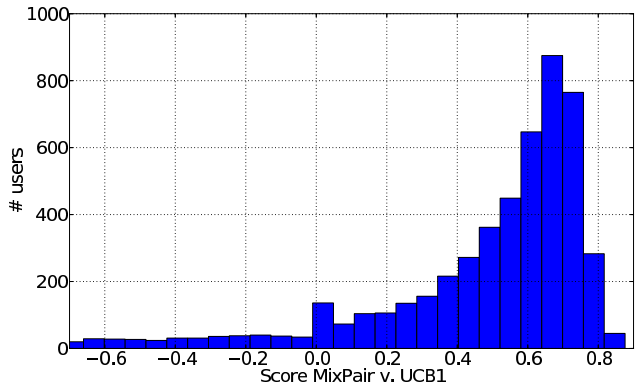


Figure 5: Performance of the MixPair strategy (neighbors sampled uniformly)

μ	p^+	μ^+	μ^-
35%	**0.84**	48%	-15%

Table 3: Aggregate improvement of MixPair over UCB1

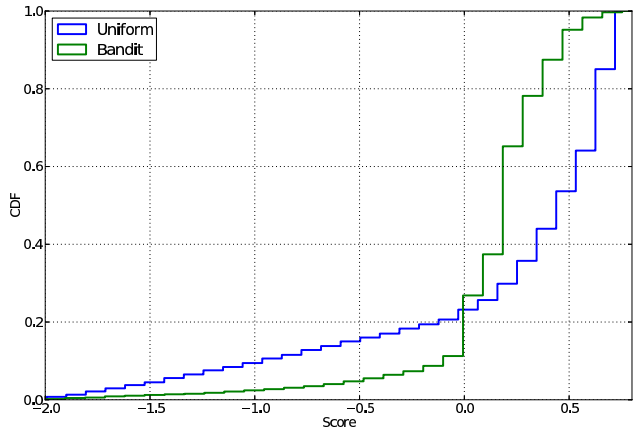


Figure 6: Score CDF for MixPair with Uniform and Bandit neighbor sampling ($q = 75\%$)

yielded a similar average score $\mu \approx 5\%$ while reducing risk: improvements are less significant (smaller μ^+) but failures are of smaller extent as well (μ^- closer to 0). Figure 6 compares the score CDFs of both neighbor sampling strategies.

5.4 MixNeigh strategy

In this section we evaluate the MixNeigh algorithm from Section 3.3. Table 4 and Figure 7 summarize the improvements it brings compared to UCB1 and MixPair. Most notably, it learns 2.94× faster on average than UCB1 ($\mu = 66\%$).

Recall that, at each step t, MixNeigh chooses between the empirical estimate $\overline{X}_a$ of $\mathcal{B}_u$ and an aggregate estimate $\overline{Y}_a$ coming from the neighborhood. In our simulations, the latter turned out to be close to 80% of the time, which shows that this information plays a crucial role in the improvement brought by MixNeigh over UCB1.

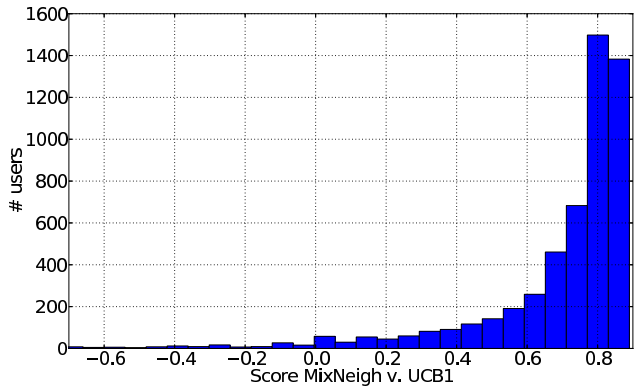


Figure 7: Performance of the MixNeigh strategy compared to UCB1

μ	p^+	μ^+	μ^-
66%	**0.96**	69%	-3%

Table 4: Aggregate improvement of MixNeigh over UCB1

5.5 Impact of the q parameter

Our performance metric is heavily dependent on the parameter $q \in [0, 1]$ quantifying the fraction of the optimum p^* the average reward needs to reach before one considers the strategy has a sufficient estimate of user preferences. The higher q, the more demanding this criterion is. We therefore study the impact of q on the performance of our various strategies. Results are reported in Tables 5 and 6. For low

q	μ	p^+	μ^+	μ^-
60%	-7%	0.21	2.5%	-39%
75%	+18%	0.79	37%	-20%
80%	+35%	0.84	48%	-14%
85%	+35%	0.80	40%	-16%
90%	+6%	0.29	9.1%	-64%

Table 5: MixPair vs. UCB1 for various values of q

q	μ	p^+	μ^+	μ^-
60%	-0.7%	0.26	3.1%	-105%
75%	+47%	0.92	52%	-7.6%
80%	+66%	0.95	68%	-3.8%
85%	+74%	0.97	75%	-2.0%
90%	+66%	0.97	67%	-2.4%

Table 6: MixNeigh vs. UCB1 for various values of q

values of q ($< 70\%$) we observe means μ close to 0, which is due to the fact that almost any strategy quickly reaches these threshold regardless of the exploration/exploitation dilemma. There is a sweet spot for $80\% \leq q \leq 85\%$, which our strategies reach much faster than the baseline. However, as $q \to 1$, all strategies converge to the same asymptotic regime. Figure 8 shows score CDFs in the "MixPair vs. UCB1" setting for various values of q; worst cases correspond to $q = 60\%$ and $q = 90\%$, and best is at $q = 80\%$.

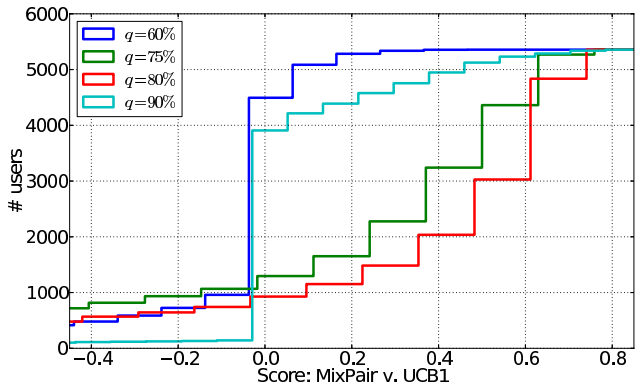


Figure 8: MixPair vs. UCB1 score CDFs for various values of q

6. RELATED WORK

Collaborative filtering. Collaborative filtering [1, 13] is the de-facto standard in recommender systems. These methods recommend an item i to a user u if the item is liked by other users whose preferences are similar to that of u. Since they rely on the historical ratings or preferences provided by users, their performance is poor for cold-start users. As a simplistic solution, hybrid approaches were proposed that combine collaborative filtering with content-based methods which learn similarities among items based on features of the items themselves. The hybrid approaches were shown to perform better than collaborative filtering for cold-start recommendations by Schein et al. [24]. Taking hybrid approaches a step forward, Park et al. [21] proposed an approach that constructs tensor profiles for user/item pairs from their respective features and performs tensor regression to minimize pairwise preference loss. They show that with meta-data about both users and items, cold-start recommendations can be made even for recommending a new item to a new user.

Cold-start problem. There have been other approaches proposed for the cold start problem, for instance, the use of *filterbots* by Park et al. [22]. Filterbots are bots that inject ratings into the recommender system to reduce the data sparsity. This simple approach improves cold-start recommendations, however, inserts "fake" ratings into the system. In a different setting, Leroy et al. [16] studied the cold-start problem for recommending friends to new users in a social network using group membership preferences of the users.

Social information. Closer to our setting of recommender systems with access to additional social information, Lam et al. [15] and later Jamali et al. [12] studied how incorporating social data about users can improve collaborative filtering results. In particular, Jamali et al. focus on cold-start recommendations and show that their method, TrustWalker, which performs a random walk on the social network among users to find items to recommend improves the F-measure on predicted ratings of cold start users by about 50% compared with standard item-based collaborative filtering. The results of Jamali et al. validate the usefulness of social data for improving the cold-start recommendations. Unlike the offline setting of [12] our interest is centered on fast online learning of cold-start user preferences. Finally, Noel et al. [19] recently proposed a social collaborative filtering framework, however, they do not focus on the cold-start problem.

Multi-armed bandits. One of the key results in the bandit literature [14] states that the asymptotic expected regret for any stochastic bandit is always $\Omega(\log t)$, where t is the number of steps. Policies based on Upper Confidence Bounds achieved this lower-bound up to a constant multiplicative factor [5, 4] while being simple to implement. They were further improved under additional assumptions, e.g., statistical [3] or leveraging an underlying *structure space* between arms [20, 6].

Bandits for recommendations. Bandits have been widely used in online advertising and content recommendation settings. Refer to [2] for an overview of bandits algorithms in recommender systems. Li et al. [17] applied Linearly Parametrized Bandits [9, 23] to personalized news recommendations. Recent works [18, 7] have shown the usefulness of side-observations to improve learning rates: in this setting, a reward is earned from the arm pulled at time t, but side observations reveal (without earning) the feedback from neighbors of that arm as well. In our present setting, we "mix" the reward estimates from playing an arm at a cold-start user with that of the neighbor(s) of the cold-start user. To the best of our knowledge, this work is the first to propose mixing strategies for combining bandits in an ego-centric network.

7. CONCLUDING REMARKS

In this work we study the problem of cold-start users in recommender systems. We formulate the problem with the multi-bandit multi-arm framework, popular in the online learning community. We proposed novel strategies for mixing bandits embedded in egocentric networks to learn them faster, which corresponds to faster learning of preferences of cold-start users. Most previous works on the cold-start problem that extend collaborative filtering methods focus on achieving high quality (often, RMSE) in recommendations made to cold-start users. In contrast, we focus on making high quality recommendations *quickly*. In addition, we propose novel evaluation metrics that can be used to effectively compare the learning rates of two bandit strategies.

A concrete next step of our work is to study different ways of selecting artists as mentioned in Section 2.3. For instance, instead of only considering artists that all neighbors like, one can consider those that a majority of neighbors have like, or those that are in the top-10 of any neighbor. As discussed in [17], evaluating bandit strategies is challenging. It would be interesting to get access to multiple snapshots of data from a recommender service and evaluate our algorithms on real cold-start users. Further, it would be interesting to find the features that discriminate users on which one strategy does better than the other. In particular, which strategy performs well when the given user's interests match that of neighbors', or the interests are aligned with only one neighbor, or if they are orthogonal to the neighbors' interests.

Our work on the cold-start problem is just one example of using multiple multi-armed bandits embedded in a social network. Several applications such as user profiling, online advertising, and advertising on social networks can benefit from this framework.

Acknowledgements. We thank Alicia Bel and Thibaut Horel for their patience and insights in the design of the MixNeigh strategy.

8. REFERENCES

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowl. and Data Eng.*, 2005.

[2] D. Agarwal and B.-C. Chen. Machine learning for large scale recommender systems. Tutorial at ICML 2011 available at `http://pages.cs.wisc.edu/~beechung/icml11-tutorial/`.

[3] J.-Y. Audibert, R. Munos, and C. Szepesvari. Tuning bandit algorithms in stochastic environments. *Algorithmic Learning Theory*, pages 150–165, 2007.

[4] P. Auer. Using confidence bounds for exploitation-exploration trade-offs. *J. Mach. Learn. Res.*, 3:397–422, Mar. 2003.

[5] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2):235–256, 2002.

[6] S. Bubeck, R. Munos, G. Stoltz, and C. Szepesvári. X-armed bandits. *Journal of Machine Learning Research*, 12:1655–1695, 2011.

[7] S. Caron, B. Kveton, M. Lelarge, and S. Bhagat. Leveraging side observations in stochastic bandits. In *UAI*, 2012.

[8] D. Crandall, D. Cosley, D. Huttenlocher, J. Kleinberg, and S. Suri. Feedback effects between similarity and social influence in online communities. In *KDD*, 2008.

[9] V. Dani, T. P. Hayes, and S. M. Kakade. Stochastic linear optimization under bandit feedback. In *Computational Learning Theory*, pages 355–366, 2008.

[10] R. V. Hogg and A. Craig. *Introduction to Mathematical Statistics*. Macmillan, 1978.

[11] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *Eighth IEEE International Conference on Data Mining*, 2008.

[12] M. Jamali and M. Ester. Trustwalker: a random walk model for combining trust-based and item-based recommendation. In *KDD*, 2009.

[13] Y. Koren and R. Bell. Advances in collaborative filtering. In *Recommender Systems Handbook*. 2011.

[14] T. L. Lai and H. Robbins. Asymptotically efficient adaptive allocation rules. *Adv. in Appl. Math.*, 6(1):4–22, 1985.

[15] C. Lam. Snack: incorporating social network information in automated collaborative filtering. In *EC*, 2004.

[16] V. Leroy, B. B. Cambazoglu, and F. Bonchi. Cold start link prediction. In *KDD*, 2010.

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

[18] S. Mannor and O. Shamir. From bandits to experts: On the value of side-observations. In *NIPS*, 2011.

[19] J. Noel, S. Sanner, K.-N. Tran, P. Christen, L. Xie, E. V. Bonilla, E. Abbasnejad, and N. Della Penna. New objective functions for social collaborative filtering. In *WWW*, 2012.

[20] S. Pandey, D. Chakrabarti, and D. Agarwal. Multi-armed bandit problems with dependent arms. In *ICML*, 2007.

[21] S.-T. Park and W. Chu. Pairwise preference regression for cold-start recommendation. In *RecSys*, 2009.

[22] S.-T. Park, D. Pennock, O. Madani, N. Good, and D. DeCoste. Naïve filterbots for robust cold-start recommendations. In *KDD*, 2006.

[23] P. Rusmevichientong and J. N. Tsitsiklis. Linearly parameterized bandits. *Math. Oper. Res.*, 35(2):395–411, May 2010.

[24] A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock. Methods and metrics for cold-start recommendations. In *SIGIR*, 2002.