Latent Contextual Bandits and Their Application to Personalized Recommendations for New Users

Li Zhou and **Emma Brunskill**
Computer Science Department
Carnegie Mellon University
{lizhou, ebrun}@cs.cmu.edu

Abstract

Personalized recommendations for new users, also known as the cold-start problem, can be formulated as a contextual bandit problem. Existing contextual bandit algorithms generally rely on features alone to capture user variability. Such methods are inefficient in learning new users' interests. In this paper we propose Latent Contextual Bandits. We consider both the benefit of leveraging a set of learned latent user classes for new users, and how we can learn such latent classes from prior users. We show that our approach achieves a better regret bound than existing algorithms. We also demonstrate the benefit of our approach using a large real world dataset and a preliminary user study.

1 Introduction

In general we desire recommender systems that can quickly start providing good recommendations for *new users*, which is particularly challenging as no prior information for new users is available. This is often known as the *cold-start* problem. Despite the lack of prior information for new users, such systems typically have interacted with millions of previous users. Therefore, this problem can be cast as an instance of lifelong learning across sequential decision making tasks: how should information from prior users be leveraged to help improve the recommendations for a new user? Standard techniques like collaborative filtering [Koren *et al.*, 2009] provide good answers to this challenge, but such approaches are typically limited to myopically providing a single recommendation, rather than reasoning about the multi-step interactions the system may have with the user. This is important, because across a sequence of interactions it may be useful for the system to actively gather information (potentially sacrifice immediate performance outcomes) in order to maximize its benefit over the longer run with the individual in question.

One approach to this is to model users by a contextual bandit model [Bubeck, 2012; Zhou, 2015] with a single shared set of model parameters, and all prior users' data can be leveraged to fit those model parameters for use in interacting with a new user. Example algorithms are LinUCB [Li *et al.*, 2010], Thompson sampling with linear payoffs [Agrawal and Goyal, 2013], and CofineUCB [Yue *et al.*, 2012]. However, these approaches work well only when there are many available features that describe users and capture user variability. If those features are not available, then we may need to fall back on a population average that may make poor recommendations for the current new individual.

At the other extreme is to use learning algorithms such as LinUCB and Thompson sampling with linear payoffs to learn from scratch for each new user separately. Such systems can provide full personalization to an individual (using model parameters learned only for that user), but may take an enormous amount of interactions to achieve this, yielding very little value for a long period (and potentially causing the user to get frustrated or cease using the system).

We instead propose an approach that provides *partial personalization*. We assume that users can be described as each belonging to one of a finite set of latent classes. Each class may be associated with a different set of model parameters, but within a class all individuals share the same parameters. Compared with the two extreme approaches mentioned above, partial personalization does not fully rely on user features to capture user variability, instead it leverages users' latent class structure to more quickly start providing good recommendations for new users.

Latent class structure has been explored in the non-contextual Multi-armed bandits setting [Lazaric *et al.*, 2013; Maillard and Mannor, 2014]. In the contextual setting, the most closely related work is CLUB [Gentile *et al.*, 2014]. CLUB learns an underlying graph structure of users based on user similarities and serves a group of users by taking advantage of the learned graph structure. However, as we will show later, our algorithm is theoretically and empirically better than CLUB.

In this paper, we focus on the latent contextual bandit setting. We consider both the benefit of using a set of learned latent classes, and how we can obtain such latent classes from a prior set of data or online. We also provide a formal analysis of the regret in this setting, by building on recent progress on latent variable learning of regression model mixtures using tensor methods [Chaganty and Liang, 2013], to bound the performance obtained by learning and leveraging a set of latent models learned from data. We then demonstrate the benefit of our approach in simulation and an unbiased offline evaluation using a large real world dataset, as well as a preliminary user study. Our results suggest a substantial benefit of our latent contextual bandit approach.

2 Our Approach

2.1 Problem Formulation

We assume there is a sequence of contextual bandit tasks, and each contextual bandit task involves multi-step interactions with a particular user.[1] Let there be $\tilde{U}$ users, where $\tilde{U}$ may be infinite. Each user u belongs to one of a finite set of N latent classes. Denote by c_u the (unknown) latent class of user u. Users within the same latent class share similar interests and behaviors. Each task (series of interactions with a single user) is assumed to last for T_u steps.

During a task, in each time step $t_u \in \{1, \ldots, T_u\}$, the algorithm observes both the current user u and a set $\mathcal{A}_{t_u}$ of arms (items) together with their d-dimensional feature vectors $x_{t_u,a} \in \mathbb{R}^d$ for all $a \in \mathcal{A}_{t_u}$. $||x_{t_u,a}||_2 \leq 1$. The feature vector $x_{t_u,a}$ captures the information of both user u and arm a at time t_u. For example, $x_{t_u,a}$ could be the linear concatenation of user and arm feature vectors. We assume the size of $\mathcal{A}_{t_u}$ is fixed: $|\mathcal{A}_{t_u}| = K$. These K feature vectors are together referred to as the *context* C_{t_u} at t_u: $C_{t_u} = \{x_{t_u,a} | a \in \mathcal{A}_{t_u}\}$.

The algorithm then recommends an arm a_{t_u} to the current user u, and receives reward $r_{t_u} \in [0, 1]$ from the user. We assume that the reward is a noisy linear function of the current user's latent class. More precisely, each latent class $h \in \{1, ..., N\}$ is associated with an (unknown) weight vector $\beta_h \in \mathbb{R}^d$. The reward of an arm $a \in \mathcal{A}_{t_u}$ is given by

$$r_{t_u,a} = \beta_{c_u}^\top x_{t_u,a} + \epsilon_{c_u}$$

where ϵ_{c_u} follows a Gaussian distribution with zero mean and bounded variance. Let $a_{t_u}^* = \arg\max_{a \in \mathcal{A}_{t_u}} \beta_{c_u}^\top x_{t_u,a}$ be the arm with highest expected reward at time t_u, and a_{t_u} be the arm selected at t_u. Then the algorithm's goal is to minimize the *regret*, which is defined as

$$\begin{aligned}\mathrm{Reg}(\tilde{U}, T_{u:u\in\{1,\ldots,\tilde{U}\}}) \\ = \sum_{u=1}^{\tilde{U}} \sum_{t_u=1}^{T_u} \beta_{c_u}^\top x_{t_u,a_{t_u}^*} - \sum_{u=1}^{\tilde{U}} \sum_{t_u=1}^{T_u} \beta_{c_u}^\top x_{t_u,a_{t_u}}\end{aligned}$$

2.2 Latent Contextual Bandits

Our algorithm, Latent Contextual Bandits (LCB), is described in Algorithm 1. LCB learns the set of latent models from prior users and leverages the learned models to make recommendations for new users. The algorithm consists of two phases. In phase 1, LCB simply runs LinUCB algorithm on the first J users and collects the pulled arms and rewards. (line $2 - 11$, Algorithm 1). The reason to do phase 1 is that initially when LCB starts from scratch, there are no prior users or training data for LCB to learn the latent models. Therefore, phase 1 is the bootstrap phase of the algorithm. A short phase 1 will cause a high model estimation error at the early stage of phase 2, while a long phase 1 will cause large regret in phase 1. In Section 3 we will discuss how to pick the length of phase 1 to get low overall regret bound. In real world systems, usually we already have a huge set of interactions made by prior users, then phase 1 is not needed.

In phase 2, LCB train/re-train latent models using data collected in both phases 1 and 2 so far (line 13, Algorithm 1). We will show how to learn the latent models in Section 2.3. In practice, we may want to re-train the latent models after a batch set of users instead of each user.

Meanwhile, in phase 2 LCB should leverage the learned latent models to improve performance for new tasks (users). Though there are many ways to do this, we propose an approach that first constructs a policy for each learned latent model, and then uses a contextual bandit algorithm that can adaptively select across the policies for a new task. A policy is a function that takes a context as input and returns an arm or a distribution over arms. For example, one policy could be a function that always return the arm with the highest expected reward estimated by a learned latent model. There already exist numerous contextual bandit algorithms that take as input a finite set of policies and compete with the best policy inside the policy set [Beygelzimer *et al.*, 2011; Agarwal *et al.*, 2014], so LCB can build upon these existing works. However, LCB in phase 2 offers multiple advantages relative to these prior works: the policy set N is often smaller than the set of policies considered by generic contextual bandit approaches; LCB automatically constructs the set of policies (instead of requiring an oracle or expert to provide a good set); and assuming the problem setting holds, the set of N policies is sufficient to enable optimal performance for any new task, in contrast to standard contextual bandit approaches which can only achieve performance as good as the input policies (which may not achieve optimal performance).

More precisely, LCB constructs one policy for each learned latent model (line 14, Algorithm 1), and then runs a pre-selected contextual bandit algorithm $\mathcal{B}$ that takes in the set of N learned policies for the N latent contextual bandit tasks (line $15 - 22$, Algorithm 1). We will discuss specific choices of $\mathcal{B}$ and ways to construct policies in Section 2.4, and we will shortly provide a theoretical analysis of our approach in Section 3.

2.3 Learn Latent Models from Past Users

We model latent user classes using a mixture of linear regressions [Viele and Tong, 2002]. A mixture of linear regressions consists of N mixture components, each is a linear regression model. Let $\Theta = \{\pi_h, \beta_h, \sigma_h^2 | h \in \{1, ..., N\}\}$ be the model parameters, where π_h is the mixture proportion, β_h is the coefficient vector, and σ_h^2 is the variance of the response. Then the likelihood of mixture of linear regressions is defined as

$$\mathrm{L}(\Theta; \mathcal{D}) = \prod_{(r,x)\in\mathcal{D}} \left(\sum_{h=1}^{N} \pi_h \mathcal{N}\left(r | \beta_h^\top x, \sigma_h^2\right) \right)$$

where $\mathcal{N}(r|\mu, \sigma^2)$ is the probability density function of a Gaussian distribution with mean μ and variance σ^2. One

[1]While in the paper we assume users come sequentially and interact with our algorithm, our approach can also handle interleaved users. As we will describe later, in phase 1 of our algorithm, it doesn't matter if users interleave as LinUCB is used. In phase 2, we can fix the current set of clusters and parameters for a new user when he/she first arrive, and use that for the entire time we interact with that user.

Algorithm 1: Latent Contextual Bandits (LCB)

```
Input: $J \in \mathbb{R}^+$: number of users in phase 1
       $N \in \mathbb{R}^+$: number of latent classes
       $\mathcal{P}$: policies construction strategy
       $\mathcal{B}$: contextual bandit algorithm
 1: Samples $\mathcal{D} = \varnothing$
    // Phase 1
 2: Create a LinUCB instance
 3: for user $u \in \{1, 2, ..., J\}$ do
 4:     for $t_u \in \{1, 2, ..., T_u\}$ do
 5:         Observe context $C_{t_u} = \{x_{t_u,a} \in \mathbb{R}^d | a \in \mathcal{A}_{t_u}\}$
 6:         Pull $a_{t_u} = \text{LinUCB}(C_{t_u})$
 7:         Observe reward $r_{t_u}$
 8:         Update LinUCB based on reward
 9:         Add $(x_{t_u,a_{t_u}}, r_{t_u})$ to $\mathcal{D}$
10:     end
11: end
    // Phase 2
12: for user $u \in \{J+1, J+2, ...\}$ do
13:     Learn $N$ latent models $\{\hat\beta_1, ..., \hat\beta_N\}$ using data $\mathcal{D}$
14:     Construct $N$ policies $\{\mathcal{P}(\hat\beta_1, \cdot), ..., \mathcal{P}(\hat\beta_N, \cdot)\}$
15:     Create a $\mathcal{B}$ instance for $u$
16:     for $t_u \in \{1, 2, ...T_u\}$ do
17:         Observe context $C_{t_u} = \{x_{t_u,a} \in \mathbb{R}^d | a \in \mathcal{A}_{t_u}\}$
18:         Pull $a_{t_u} = \mathcal{B}_u(C_{t_u}, \{\mathcal{P}(\hat\beta_1, \cdot), ..., \mathcal{P}(\hat\beta_N, \cdot)\})$
19:         Observe reward $r_{t_u}$
20:         Update $\mathcal{B}_u$ based on reward
21:         Add $(x_{t_u,a_{t_u}}, r_{t_u})$ to $\mathcal{D}$
22:     end
23: end
```

classic algorithm to learn a mixture model is the Expectation-Maximization (EM) algorithm. However, EM does not guarantee convergence to the globally optimal parameters, and it does not provide finite sample guarantees on the quality of the resulting parameter estimates. On the other hand, tensor decomposition based methods, as we will describe shortly, give us finite sample guarantees which can be further used to derived our regret bound.

Learn Latent Models using Spectral Experts

Anandkumar et al. [2014] showed that tensor decomposition can efficiently recover parameters for a wide class of latent variable models. They exploited a special tensor structure derived from second and third-order moments of the observations, and apply the robust tensor power method to recover model parameters. Spectral Experts [Chaganty and Liang, 2013], built on top of Anandkumar et al.'s work, provide provably consistent estimator for mixture of linear regressions. Our algorithm uses Spectral Experts to estimate parameters of mixture of linear regressions. Later in Section 3 we also use the parameter error bound provided by Spectral Experts to bound the regret of our algorithm.

Though Spectral Experts algorithm has appealing theoretical properties, it is not particularly sample efficient and it is computationally expensive. Therefore, in the following section, we also derive and implement a computationally efficient Gibbs sampling based procedure to estimate parameters of mixtures of linear regressions.

Learn Latent Models using Gibbs Sampling

Gibbs sampling is an efficient inference technique to learn latent models for large scale dataset. We derive a sampling procedure for Dirichlet Process [Neal, 2000] mixtures of linear regressions. By using a Dirichlet Process prior, we do not need to specify the number of latent models. Specifically, we assume the prior of β_h and σ_h^2 follow the Normal-inverse-Gamma distribution and the prior of π follows GEM distribution [Murphy, 2012a], which is used by the stick-breaking construction of the Dirichlet process. The generative process is as follows:

1. $\alpha \sim \text{Gamma}(u_0, v_0)$
2. $\pi \sim \text{GEM}(\alpha)$
3. For each latent model $h \in \{1, ..., N\}$
   (a) $(\beta_h, \sigma_h^2) \sim \text{NIG}(w_0, V_0, a_0, b_0)$
4. For each user $u \in \{1, ..., U\}$
   (a) $c_u \sim \text{Categorical}(\pi)$
   (b) For each interaction $t_u \in \{1, ..., T_u\}$
       i. $r_{t_u} \sim \mathcal{N}(\beta_{c_u}^\top x_{t_u,a_{t_u}}, \sigma_{c_u}^2)$

We use collapsed Gibbs sampling to sample c_u and α. Denote all the rewards of a user u by $r_u = \{r_{t_u} : t_u \in \{1, ..., T_u\}\}$. To sample c_u,

$$\begin{aligned} &P(c_u = h | c_{-u}, r_u, \alpha, w_0, V_0, a_0, b_0) \\ &\propto P(c_u = h | c_{-u}, \alpha) P(r_u | r_{-u}^h, w_0, V_0, a_0, b_0) \end{aligned} \tag{1}$$

where $c_{-u} = \{c_{u'} : u' \neq u\}$ and $r_{-u}^h = \{r_{u'} : c_{u'} = h, u' \neq u\}$. The first term in Equation (1) is given by the Chinese Restaurant Process (CRP) [Neal, 2000], the second term in Equation (1) is the posterior predictive distribution of r_u given r_{-u}^h, and it follows Multivariate t-distribution [Murphy, 2012b]. To sample α, we adopt the auxiliary variable method [Escobar and West, 1995].

2.4 Leverage Learned Models for New Users

Let $\{\hat\beta_1, ..., \hat\beta_N\}$ be the N learned latent models. We define N policies based on these models. There are two types of policies we can define, one is deterministic, and the other one is probabilistic. The deterministic one maps a context C_{t_u} to an arm $a \in \mathcal{A}_{t_u}$:

$$\mathcal{P}(\hat\beta_h, C_{t_u}) = \arg\max_{a \in \mathcal{A}_{t_u}} \hat\beta_h^\top x_{t_u,a}$$

The probabilistic one maps a context C_{t_u} to a categorical distribution over arms:

$$\mathcal{P}(\hat\beta_h, C_{t_u}) = [p_1, p_2, ..., p_{|\mathcal{A}_{t_u}|}]$$

where

$$p_a = \frac{\exp(\hat\beta_h^\top x_{t_u,a})}{\sum_{a \in \mathcal{A}_u} \exp(\hat\beta_h^\top x_{t_u,a})}$$

The constructed polices can be used by many contextual bandit algorithms to serve new users. If the policies are deterministic, possible contextual bandit algorithms include Epoch-Greedy [Langford and Zhang, 2008], ILOVETOCONBANDITS [Agarwal *et al.*, 2014], and Generalized Thompson Sampling [Li, 2013]. If the policies are probabilistic, possible contextual bandit algorithms include EXP4 [Auer *et al.*, 2002] and EXP4.P [Beygelzimer *et al.*, 2011]. The algorithm choice depends on the desired outcome, and we will shortly consider specific choices for both our theoretical analysis and empirical results.

3 Theoretical Analysis

In this section, we analyze LCB's expected regret. We assume the latent models are learned using the Spectral Experts algorithm. Let J be the number of users in phase 1, U be the number of users in phase 2, and $\tilde{U} = J + U$ be the total number of users. Let $S = \sum_{u=1}^{J} T_u$, $T = \sum_{u=J+1}^{\tilde{U}} T_u$, and $\tilde{T} = S + T$ be the total number of interactions. We denote the first n positive integers by $[n]$. For convenience, we define the *true policy* of a user $u \in [\tilde{U}]$ as the deterministic policy constructed by β_{c_u} (the true latent model the user belongs to).

For theoretical analysis, we make two minor changes to the Algorithm 1. First, instead of running a single LinUCB instance for all users in phase 1, we run a separate LinUCB instance for each user. The reason is that under our realizability assumption (each user belongs to one of the latent models) single LinUCB instance which runs for all users has linear regret $O(S)$. Second, we collect τ_u i.i.d. samples from each user $u \in [\tilde{U}]$, that is, we select arms uniformly at random for the first τ_u interactions for each user $u \in [\tilde{U}]$. When training latent models using Spectral Experts, we only use these i.i.d. samples. We do this because Spectral Experts requires i.i.d. training examples to get theoretical guarantee on the parameter error bound.

Assume $T_u \le L$ for all $u \in [\tilde{U}]$ for some constant L. Denote the minimum Euclidean distance of any two latent models by $\triangle$, that is, $||\beta_h - \beta_{h'}|| \ge \triangle$ for any $h \neq h'$. The following two theorems show a problem-independent expected regret bound which is independent of $\triangle$ and a problem-dependent expected regret bound which depends on $\triangle$.

Theorem 1. *Set $\tau_u = \sqrt{T_u}$ for all $u \in [\tilde{U}]$ and $J = \sqrt{L}$. Assume $\mathcal{P}$ constructs deterministic policies. If $\mathcal{B}$ is a contextual bandits algorithm with optimal regret bound (e.g. EXP4.P), then the problem-independent expected regret bound of LCB with respect to the true policy is*

$$\begin{aligned}\mathbb{E}\left[\mathrm{Reg}_{\mathrm{LCB}}\left(\tilde{U}, T_{u:u\in[\tilde{U}]}\right)\right] &= O\left(\sqrt{JS} + d\sqrt{JS\ln(1+S)}\right) \\ &+ O\left(3\sqrt{UT} + \sqrt{UTK\ln N}\right) \\ &= O\left(\sqrt{UTK\ln N}\right)\end{aligned}$$

as $UT \gg \max\left\{1, \frac{d^2}{K}\right\} JS$, that is, as T and U grows large. Similarly, if $\mathcal{B}$ is EXP3 [Auer et al.*, 2002] which treats each learned policy as an arm, then the problem-independent expected regret bound of LCB with respect to the true policy is*

$$\mathbb{E}\left[\mathrm{Reg}_{\mathrm{LCB}}\left(\tilde{U}, T_{u:u\in[\tilde{U}]}\right)\right] = O\left(\sqrt{UTN\ln N}\right)$$

as $UT \gg \max\left\{1, \frac{d^2}{N}\right\} JS$, that is, as T and U grows large.

Theorem 2. *Set $\tau_u = 3$ for all $u \in [\tilde{U}]$ and $J = L^2$. Assume $\mathcal{P}$ constructs deterministic policies and $\mathcal{B}$ is Epoch-Greedy, then the problem-dependent expected regret bound of LCB with respect to the true policy is*

$$\begin{aligned}&\mathbb{E}\left[\mathrm{Reg}_{\mathrm{LCB}}\left(\tilde{U}, T_{u:u\in[\tilde{U}]}\right)\right] = \\ &O\left(3L^2 + dL\sqrt{S\ln(L+1)} + \frac{UK}{\triangle^2}\left(\ln N + \ln(T+1)\right)\right)\end{aligned}$$

Proof *(Theorem 1).* The expected regret of LinUCB [Zhou, 2015] is

$$\mathbb{E}\left[\mathrm{Reg}_{\mathrm{LinUCB}}(T_u)\right] = O(d\sqrt{T_u\ln(1+T_u)})$$

so in phase 1 the expected regret of LCB is

$$\begin{aligned}&\mathbb{E}\left[\mathrm{Reg}_{\mathrm{LCB}}^{phase\_1}\left(J, T_{u:u\in[J]}\right)\right] \\ &= O\left(\sum_{u=1}^{J}\left(\sqrt{T_u} + d\sqrt{(T_u - \sqrt{T_u})\ln(1 + T_u - \sqrt{T_u})}\right)\right) \\ &= O(\sqrt{JS} + d\sqrt{JS\ln(1+S)}) \qquad (2)\end{aligned}$$

where Equation (2) follows from the Cauchy-Schwarz inequality.

We next need to bound the regret in phase 2. For each user $u \in \{J+1, ..., J+U\}$ in phase 2, define

$$\hat{\beta}_u^* = \underset{\hat{\beta}\in\{\hat{\beta}_1,\dots,\hat{\beta}_N\}}{\arg\max} \sum_{t_u=1}^{T_u} \mathrm{E}[r_{t_u,\mathcal{P}(\hat{\beta},C_{t_u})}] \qquad (3)$$

as the best model of that user within all estimated models. Also recall that β_{c_u} is the true model of the user u and $\hat{\beta}_{c_u}$ is the estimate of β_{c_u} returned by Spectral Experts. Let $a_{t_u}^*$, $\tilde{a}_{t_u}^*$ and $\hat{a}_{t_u}^*$ be the arm proposed by β_{c_u}, $\hat{\beta}_{c_u}$ and $\hat{\beta}_u^*$. $\hat{\beta}_u^*$ achieves the highest expected cumulative reward based on its definition, so it achieves higher expected cumulative reward than $\hat{\beta}_{c_u}$, so

$$\sum_{t_u=1}^{T_u} x_{t_u,\hat{a}_{t_u}^*}^\top \beta_{c_u} \ge \sum_{t_u=1}^{T_u} x_{t_u,\tilde{a}_{t_u}^*}^\top \beta_{c_u} \qquad (4)$$

Meanwhile, we can bound the gap between the expected cumulative reward achieved by β_{c_u} and by $\hat{\beta}_{c_u}$ as follows:

$$\begin{aligned}&x_{t_u,a_{t_u}^*}^\top \beta_{c_u} - x_{t_u,\tilde{a}_{t_u}^*}^\top \beta_{c_u} \\ &\le x_{t_u,a_{t_u}^*}^\top \beta_{c_u} - x_{t_u,\tilde{a}_{t_u}^*}^\top \beta_{c_u} + x_{t_u,\tilde{a}_{t_u}^*}^\top \hat{\beta}_{c_u} - x_{t_u,a_{t_u}^*}^\top \hat{\beta}_{c_u} \\ &= (x_{t_u,a_{t_u}^*}^\top \beta_{c_u} - x_{t_u,a_{t_u}^*}^\top \hat{\beta}_{c_u}) + (x_{t_u,\tilde{a}_{t_u}^*}^\top \hat{\beta}_{c_u} - x_{t_u,\tilde{a}_{t_u}^*}^\top \beta_{c_u}) \\ &\le 2||\beta_{c_u} - \hat{\beta}_{c_u}|| \qquad (5)\end{aligned}$$

The last step uses the fact that $||x_{t_u,a}||_2 \leq 1$. Using Equation (4) and (5) together we can bound the gap between the expected cumulative reward achieved by β_{c_u} and by $\hat{\beta}^*_u$:

$$\sum_{t_u=1}^{T_u} x^\top_{t_u,a^*_{t_u}} \beta_{c_u} - \sum_{t_u=1}^{T_u} x^\top_{t_u,\hat{a}^*_{t_u}} \beta_{c_u} \leq 2T_u ||\beta_{c_u} - \hat{\beta}_{c_u}||$$

Chaganty and Liang [2013] showed that $||\beta_{c_u} - \hat{\beta}_{c_u}|| = O\left(\frac{1}{\sqrt{n}}\right)$ where n is the number of training examples. Now if $\mathcal{B}$ is a bandits algorithm with optimal expected regret bound $O(\sqrt{T_u K \ln N})$, then in phase 2 the expected regret of Latent Contextual Bandits is

$$\mathbb{E}\left[\text{Reg}^{phase\_2}_{\text{LCB}}\left(U, T_{u:u\in\{J+1,\dots,J+U\}}\right)\right]$$
$$= O\left(\sum_{u=J+1}^{J+U}\left(\sqrt{T_u} + \sqrt{T_u K \ln N} + 2T_u ||\beta_{c_u} - \hat{\beta}_{c_u}||\right)\right)$$
$$= O\left(\sum_{u=J+1}^{J+U}\left(\sqrt{T_u} + \sqrt{T_u K \ln N} + \frac{2T_u}{\sqrt{\sqrt{T_u}(u-1)}}\right)\right)$$
$$= O\left(\sum_{u=J+1}^{J+U}\left(\sqrt{T_u K \ln N} + \frac{3\sqrt{T_u}}{\sqrt{(1+(u-J-1)/\sqrt{T_u})}}\right)\right) \tag{6}$$
$$= O(3\sqrt{UT} + \sqrt{UTK \ln N}) \tag{7}$$

Equation (6) follows from $J \geq \sqrt{T_u}$ for all u. Equation (7) follows from bounding the last term in (6) by $3\sqrt{T_u}$, and then applying Cauchy-Schwarz inequality.[2] Similarly, if $\mathcal{B}$ is EXP3 which achieves a regret of $O(\sqrt{TN \ln N})$, then the expected regret of LCB in phase 2 is $O(3\sqrt{UT} + \sqrt{UTN \ln N})$. Finally, by adding the regret bound of phase 1 and phase 2, we prove the theorem. □

Proof *(Theorem 2).* The proof of Theorem 1 shows that the regret in phase 1 is

$$\mathbb{E}\left[\text{Reg}^{phase\_1}_{\text{LCB}}\left(J, T_{u:u\in[J]}\right)\right]$$
$$= O\left(\sum_{u=1}^{J}\left(\tau_u + d\sqrt{T_u \ln(1+T_u)}\right)\right)$$

If $\mathcal{B}$ is Epoch-Greedy, then based on Epoch-Greedy's problem-dependent bound we have

$$\mathbb{E}\left[\text{Reg}^{phase\_2}_{\text{LCB}}\left(U, T_{u:u\in\{J+1,\dots,J+U\}}\right)\right]$$
$$= O\left(\sum_{u=J+1}^{J+U}\left(\tau_u + \frac{K\ln(N(T_u+1))}{\triangle^2} + 2T_u||\beta_{c_u} - \hat{\beta}_{c_u}||\right)\right)$$
$$= O\left(\sum_{u=J+1}^{J+U}\left(\tau_u + \frac{K\ln(N(T_u+1))}{\triangle^2} + \frac{2T_u}{\sqrt{u-1}}\right)\right)$$

[2]The last term actually decreases at the rate of $O(1/\sqrt{u})$ with respect to u, so our regret bound gets tighter as u increases in phase 2, but only by a constant factor.

Set $J = L^2$, then the last term is less than or equal to 1. Set $\tau = 3$, then the regret of LCB, by adding the regret in phase 1 and phase 2, is

$$\mathbb{E}\left[\text{Reg}_{\text{LCB}}\left(\tilde{U}, T_{u:u\in[\tilde{U}]}\right)\right] =$$
$$O\left(3L^2 + dL\sqrt{S\ln(L+1)} + \frac{UK}{\triangle^2}(\ln N + \ln(T+1))\right)$$

which proves the theorem. □

To put these results in context, we now compare our regret results to several other approaches. We compare the following algorithms: 1) **Population EXP4.P**: runs a single EXP4.P model for all users, 2) **Individual EXP4.P**: runs a separate EXP4.P model for each user, 3) **Population LinUCB**: runs a single LinUCB model for all users, 4), **Individual LinUCB**: runs a separate LinUCB model for each user, 5) **EXP4.P enum-policies**: enumerates all policies by mapping all possible contexts to all possible arms and then runs EXP4.P on each user (assuming contexts are enumerable), 6) **CLUB**, 7) **LCB**. Table 1 shows the expected regret of each algorithm; it also shows the policy each algorithm is competing with when deriving the regret bound. Keep in mind that all comparisons are under our realizability assumption that each user belongs to one of the latent models.

Within the 7 algorithms in Table 1, 3 of them does not compete with the true policy of each user: Population LinUCB doesn't distinguish between users from different classes, so it is competing with the best average policy of all users; Population/Individual EXP4.P requires a set of pre-defined policies as input, and compete with the best one inside the policy set instead of the true policy of each user.

The remaining 4 algorithms all compete with the true policy of each user; however, LCB achieves the best expected regret bound. The problem-independent regret bound of Individual LinUCB is linear with respect to the contexts' feature dimension which is often very large. Define C as the total number of contexts. EXP4.P enum-policies has a C term in its problem-independent regret bound which is often large or even infinite. LCB's problem-independent regret bound, on the other hand, only has square root dependence on U, T and $\min\{K, N\}$.

Table 1 also shows the problem-dependent regret bound of CLUB analyzed by Gentile et al. [2014]. Both LCB and CLUB's problem-dependent regret bounds depends on $1/\triangle^2$. However, CLUB has a square root dependence on $\tilde{T}$, while LCB only has a square root dependence on S (number of interactions in phase 1) which is a constant. Moreover, the analysis of CLUB (see Appendix of Gentile et al. [2014]) shows that CLUB's expected regret on a new user u is linear with respect to T_u when $T_u < B$ for some constant B that is in the order of $O(\frac{d}{\triangle^2})$. B may be enormous if $\triangle^2$ is small. Meanwhile, LCB's problem-independent regret bound guarantees that LCB's expected regret on a new user u is always sublinear (square root) with respect to T_u even when T_u is small.

4 Experiments

In this section, we evaluate our algorithm both on simulation and on a large real world dataset from Yahoo!. We compare

Algorithm	Expected Regret	Regret with respect to	Regret Type
Population EXP4.P	$O(\sqrt{\tilde{T}K\ln N})$	best policy in pre-defined policy set	problem-independent
Individual EXP4.P	$O(\sqrt{\tilde{U}\tilde{T}K\ln N})$	best policy in pre-defined policy set	problem-independent
Population LinUCB [3]	$O(d\sqrt{\tilde{T}\ln(1+\tilde{T})})$	best average policy of all users	problem-independent
Individual LinUCB	$O(d\sqrt{\tilde{U}\tilde{T}\ln(1+\tilde{T})})$	true policy of each user	problem-independent
EXP4.P enum-policies	$O(\sqrt{\tilde{U}\tilde{T}KC\ln K})$	true policy of each user	problem-independent
LCB (Theorem 1)	$O(\sqrt{UT\min\{K,N\}\ln N})$	true policy of each user	problem-independent
CLUB	$O(N+\tilde{U}\sqrt{Nd}+\frac{\tilde{U}d}{\triangle^2}\ln(\tilde{T}+1)+dN\sqrt{\tilde{T}})$	true policy of each user	problem-dependent
LCB (Theorem 2)	$O(3L^2+dL\sqrt{S}+\frac{UK}{\triangle^2}(\ln N+\ln(T+1)))$	true policy of each user	problem-dependent

Table 1: Expected regret bounds of LCB and baseline algorithms. We use N to denote both the number of policies in the three EXP4.P variants and the number of latent models in LCB and CLUB. We use C to denote the total number of contexts.

the following algorithms: 1) **LCB**: our approach. We choose Generalized Thompson Sampling as $\mathcal{B}$ in Algorithm 1. For simulation we use Spectral Experts to learn latent models, but due to time/memory constrains, for large real world dataset we use Gibbs sampling to learn latent models; 2) **LCB_GT**: this is similar to LCB, except that instead of learning the latent models, we provide true latent models to the algorithm. 3) **CLUB**; 4) **Population LinUCB**: runs single LinUCB instance on all users; 5) **Individual LinUCB**: runs a separate LinUCB instance on each user; 6) **Random** : selects each arm uniformly at random.

4.1 Simulation

We artificially created 5 latent models as shown in Figure 1. Each model had 10 parameters, and 4 of them were assigned higher weights. Users were sampled uniformly at random from these latent models. For each user interaction we generated 20 arms, that is, $|\mathcal{A}_{t_u}| = 20$. Each arm a was associated with a feature vector $x_{t_u,a}$ sampled uniformly from $[-1,1]^{10}$, and was normalized so that $||x_{t_u,a}||_2 = 1$. We sampled the reward of each arm a from $\mathcal{N}(\beta_{c_u}^\top x_{t_u,a}, \sigma^2)$ with $\sigma = 0.1$. For LCB, we set $J = 50$, and in phase 2 we re-trained the latent models after every 50 users.

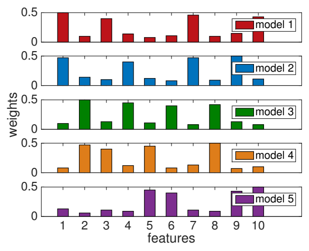
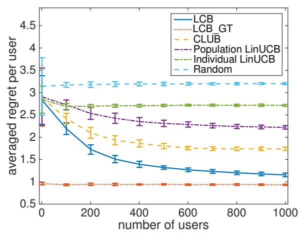


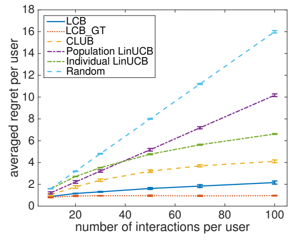
Figure 1: Models in simulation

In the first experiment, we fixed $T_u = 20$ for all users, and reported the averaged per-user regret vs. number of users. Results are shown in Figure 2a. We can see that the regret of LCB was about 35% lower than CLUB and 50% lower than Population LinUCB. In the second experiment, we fixed the number of users to 1000, and varied T_u from 10 to 100. T_u were set to the same for all users. Figure 2b shows the averaged per-user regret vs. T_u. We can see that LCB outperformed CLUB and LinUCB, and as T_u increased, the gap between their regret also increased. Also, when T_u was more than 40, Individual LinUCB started to learn a good model for each user, and outperformed Population LinUCB, however, it still had much higher regret than LCB.

[3]Under our realizability assumption, Population LinUCB has $O(\tilde{T})$ linear regret, so here we show the regret under its own realizability assumption



(a) Averaged per-user regret vs. number of users.



(b) Averaged per-user regret vs. number of interactions per users.

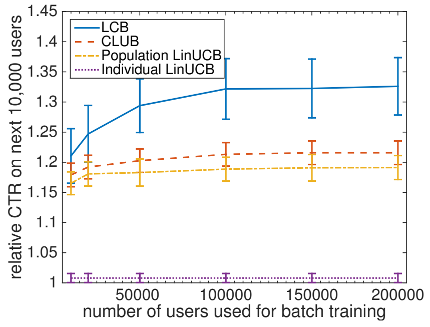
Figure 2: Experiment results on simulation

4.2 Experiments on Real World Dataset

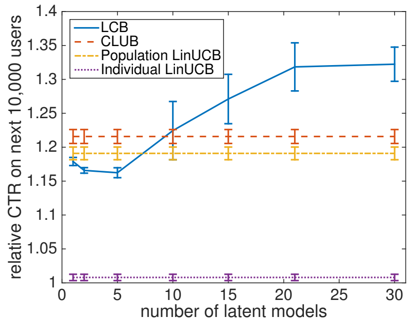
We evaluated our algorithm on a news feed dataset provided by Yahoo!. The dataset contained 500,000 users and all their visits in a one month period. In each visit, a user was shown 25 news articles from the top down. User clicks (binary feedback) were logged. There were 21 news categories, and each news article belonged to $1 \sim 3$ categories. Therefore, articles were represented as a 21-dimensional binary feature vector. User features were not available because of privacy issues.

To the best of our knowledge, in our case there is no perfect solution for *unbiased* offline evaluation. For stationary algorithm one can use propensity scoring [Strehl *et al.*, 2010], however our algorithm is nonstationary, and propensity score is not available in this dataset. One state-of-the-art solution is rejection sampling based replay method [Li *et al.*, 2011]. However, rejection sampling is quite sample inefficient on our dataset because the policy which generated our dataset was biased towards exploitation. Therefore, we adopted the queue method [Mandel *et al.*, 2015], a sample efficient offline evaluation method for *non-contextual* bandits, and extended it to our *contextual* case.

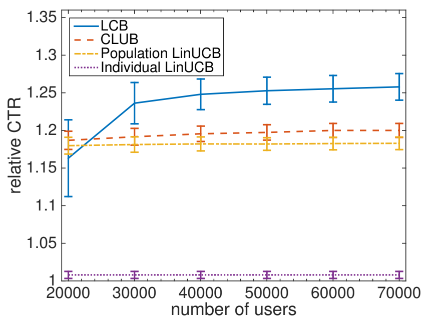
To use the queue method, we defined arms as news categories instead of news articles. As there were 21 categories,



(a) Relative CTR of 10,000 new users vs. number of users used for batch training.



(b) Relative CTR of 10,000 new users vs. number of latent models specified in batch training.



(c) Relative CTR vs. number of users that have been interacted with the algorithm.

Figure 3: Experiment results on a large real world news recommendation dataset.

we defined 21 queues for each user. The queues of each user were initialized with click labels (0 or 1) of articles shown to that user. For example, if an article belonged to two categories, then its click label was added to the two corresponding queues. Finally, we ran PCA to project the 21-dimensional article feature space to a 6-dimensional lower space so that each category (arm) can be represented as a dense vector.

We fixed the number of interactions per user (T_u) to 20 for all users to ensure all users were new users. We reported relative CTR, the algorithm's CTR divided by the CTR in the data, due to confidentiality. In the first experiment, we ran batch training: each algorithm was pre-trained with $20{,}000 - 200{,}000$ users, and then tested on the 10,000 new users. Relative CTR of the test users was reported. For LCB, we used the training users to learn 30 latent models, then we directly ran phase 2 for the test users without re-training latent models. Figure 3a shows the experiment results. We can see that LCB achieved the highest CTR, and outperformed CLUB by about 10%. Moreover, CLUB only outperformed Population LinUCB by about 2%. One reason is that the rewards in our real dataset were binary and noisy, so 20 samples per user ($T_u = 20$) were not enough for CLUB to learn a good regression model for each user and hence to learn a good latent graph structure. In the second experiment, we varied the number of latent models of LCB from 1 to 30. Similar to the batch training, each algorithm was pre-trained with 150,000 users and then tested on 10,000 new users. The result is shown in Figure 3b. We can see that with 10 or more models, LCB started to take the benefit of latent class structure and outperformed CLUB and LinUCB. With 15 and 30 latent models, our approach improved the CTR by about 5% and 10% respectively compared with CLUB and Population LinUCB. The third experiment simulated the real world environment in which users came sequentially and interacted with the algorithm. For LCB, 20,000 users were used in phase 1. In phase 2, 30 latent models were trained and re-trained after every 10,000 users. To collect i.i.d data points to better learn the latent models, we picked arms uniformly at random for the first 5 interactions of each user, and only used these data points to train/re-train latent models. For all algorithms, we reported relative CTR after every 10,000 users. Results in Figure 3c shows that LCB achieved about 5% higher CTR than CLUB and Population LinUCB.

Population LinUCB	LCB
0.196 ± 0.096	0.380 ± 0.076

Table 2: CTR mean and standard deviation in user study.

4.3 Pilot Results on User Study

In this section, we show the pilot results of our user study with 10 users, 5 for each algorithm. We compared two algorithms: Population LinUCB and LCB. Since the experiments in Section 4.2 used the Yahoo! real world dataset, so the learned models can be directly used for the user study. For LCB, we used the learned latent models from Section 4.2 and directly ran phase 2 of LCB. For Population LinUCB, we used the learned Population LinUCB model from Section 4.2 to initialize the LinUCB model used in the user study. Users interacted with the algorithms through an app developed on the Android platform. $T_u = 20$ for all users. During each user interaction, the app requested 170 latest news articles from the Yahoo! news service in real time. Similar to Section 4.2, each news article was represented by a 21-dimensional vector. The algorithm then selected one of the articles for the user and received user feedback (click). Table 2 shows the CTR mean and standard deviation achieved by these two algorithms. We can see from the pilot results that LCB outperformed Population LinUCB. User study with more users and algorithms is in progress.

5 Conclusion

In this paper, we propose Latent Contextual Bandits, a contextual bandits algorithm that learns the latent structure of users and leverages the learned latent structure to make personalized recommendations for new users. We prove both a problem-independent and a problem-dependent regret bound with respect to the true policies of users. The regret bounds significantly improved over baseline algorithms. We then demonstrate the benefit of our approach using both simulation and an unbiased offline evaluation with a large real world dataset, as well as a preliminary user study.

Acknowledgments

This work was supported by the CMU-Yahoo! InMind project. We also gratefully acknowledge the assistance and/or helpful feedback of Liangjie Hong, Suju Rajan, Michal Valko, Saloni Potdar, Zhengyang Ruan, Linxi Zou, Mingzhi Zeng and the pilot study participants.

References

[Agarwal *et al.*, 2014] 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 *Proceedings of the 31st International Conference on Machine Learning*, pages 1638–1646, 2014.

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

[Anandkumar *et al.*, 2014] Animashree Anandkumar, Rong Ge, Daniel Hsu, Sham M Kakade, and Matus Telgarsky. Tensor decompositions for learning latent variable models. *The Journal of Machine Learning Research*, 15(1):2773–2832, 2014.

[Auer *et al.*, 2002] Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, and Robert E Schapire. The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing*, 32(1):48–77, 2002.

[Beygelzimer *et al.*, 2011] Alina Beygelzimer, John Langford, Lihong Li, Lev Reyzin, and Robert E Schapire. Contextual bandit algorithms with supervised learning guarantees. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, pages 19–26, 2011.

[Bubeck, 2012] Sébastien Bubeck. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends in Machine Learning*, 5(1):1–122, 2012.

[Chaganty and Liang, 2013] Arun Tejasvi Chaganty and Percy Liang. Spectral experts for estimating mixtures of linear regressions. In *Proceedings of the 30th International Conference on Machine Learning*, pages 1040–1048, 2013.

[Escobar and West, 1995] Michael D Escobar and Mike West. Bayesian density estimation and inference using mixtures. *Journal of the american statistical association*, 90(430):577–588, 1995.

[Gentile *et al.*, 2014] Claudio Gentile, Shuai Li, and Giovanni Zappella. Online clustering of bandits. In *Proceedings of the 31st International Conference on Machine Learning*, pages 757–765, 2014.

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

[Langford and Zhang, 2008] John Langford and Tong Zhang. The epoch-greedy algorithm for multi-armed bandits with side information. In *Advances in Neural Information Processing Systems 20*, pages 817–824. Curran Associates, Inc., 2008.

[Lazaric *et al.*, 2013] Alessandro Lazaric, Emma Brunskill, et al. Sequential transfer in multi-armed bandit with finite set of models. In *Advances in Neural Information Processing Systems*, pages 2220–2228, 2013.

[Li *et al.*, 2010] Lihong Li, Wei Chu, John Langford, and Robert E Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 661–670. ACM, 2010.

[Li *et al.*, 2011] Lihong Li, Wei Chu, John Langford, and Xuanhui Wang. Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 297–306. ACM, 2011.

[Li, 2013] Lihong Li. Generalized thompson sampling for contextual bandits. *CoRR*, abs/1310.7163, 2013.

[Maillard and Mannor, 2014] Odalric-ambrym Maillard and Shie Mannor. Latent bandits. In *Proceedings of the 31st International Conference on Machine Learning*, pages 136–144, 2014.

[Mandel *et al.*, 2015] Travis Mandel, Yun-En Liu, Emma Brunskill, and Zoran Popovic. The queue method: Handling delay, heuristics, prior data, and evaluation in bandits. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[Murphy, 2012a] Kevin P Murphy. *Machine Learning: a Probabilistic Perspective*, chapter 25.2.2, pages 882–885. MIT press, 2012.

[Murphy, 2012b] Kevin P Murphy. *Machine Learning: a Probabilistic Perspective*, chapter 7.6.3, pages 234–238. MIT press, 2012.

[Neal, 2000] Radford M Neal. Markov chain sampling methods for dirichlet process mixture models. *Journal of computational and graphical statistics*, 9(2):249–265, 2000.

[Strehl *et al.*, 2010] Alex Strehl, John Langford, Lihong Li, and Sham M Kakade. Learning from logged implicit exploration data. In *Advances in Neural Information Processing Systems*, pages 2217–2225, 2010.

[Viele and Tong, 2002] Kert Viele and Barbara Tong. Modeling with mixtures of linear regressions. *Statistics and Computing*, 12(4):315–330, 2002.

[Yue *et al.*, 2012] Yisong Yue, Sue A Hong, and Carlos Guestrin. Hierarchical exploration for accelerating contextual bandits. In *Proceedings of the 29th International Conference on Machine Learning*, pages 1895–1902, 2012.

[Zhou, 2015] Li Zhou. A survey on contextual multi-armed bandits. *CoRR*, abs/1508.03326, 2015.