JMLR: Workshop and Conference Proceedings 29:1–19, 2014 ACML 2014

Pseudo-reward Algorithms for Contextual Bandits with Linear Payoff Functions

Ku-Chun Chou R99922095@CSIE.NTU.EDU.TW
Department of Computer Science and Information Engineering, National Taiwan University

Chao-Kai Chiang CHAO-KAI.CHIANG@UNILEOBEN.AC.AT
Department of Mathematics and Information Technology, University of Leoben

Hsuan-Tien Lin HTLIN@CSIE.NTU.EDU.TW
Department of Computer Science and Information Engineering, National Taiwan University

Chi-Jen Lu CJLU@IIS.SINICA.EDU.TW
Institute of Information Science, Academia Sinica

Abstract

We study the contextual bandit problem with linear payoff functions, which is a generalization of the traditional multi-armed bandit problem. In the contextual bandit problem, the learner needs to iteratively select an action based on an observed context, and receives a linear score on only the selected action as the reward feedback. Motivated by the observation that better performance is achievable if the other rewards on the non-selected actions can also be revealed to the learner, we propose a new framework that feeds the learner with pseudo-rewards, which are estimates of the rewards on the non-selected actions. We argue that the pseudo-rewards should better contain over-estimates of the true rewards, and propose a forgetting mechanism to decrease the negative influence of the over-estimation in the long run. Then, we couple the two key ideas above with the linear upper confidence bound (LinUCB) algorithm to design a novel algorithm called linear pseudo-reward upper confidence bound (LinPRUCB). We prove that LinPRUCB shares the same order of regret bound to LinUCB, while enjoying the practical observation of faster reward-gathering in the earlier iterations. Experiments on artificial and real-world data sets justify that LinPRUCB is competitive to and sometimes even better than LinUCB. Furthermore, we couple LinPRUCB with a special parameter to formalize a new algorithm that yields faster computation in updating the internal models while keeping the promising practical performance. The two properties match the real-world needs of the contextual bandit problem and make the new algorithm a favorable choice in practice.

Keywords: contextual bandit, pseudo-reward, upper confidence bound

1. Introduction

We study the *contextual bandit problem* (Wang et al., 2005), or known as the k-armed bandit problem with context (Auer et al., 2002a), in which a learner needs to interact iteratively with the external environment. In the contextual bandit problem, the learner has to make decisions in a number of iterations. During each iteration, the learner can first access certain information, called *context*, about the environment. After seeing the context, the learner is asked to strategically *select* an action from a set of candidate actions. The external environment then reveals the reward of the selected action to the learner as the feedback, while hiding the rewards of all other actions. This type of feedback scheme is known as the bandit setting (Auer et al., 2002b). The learner then *updates* its internal models with the feedback to reach the goal of gathering the most cumulative reward over all the iterations. The difference of cumulative reward between the best strategy and the learner's strategy is called the regret, which is often used to measure the performance of the learner.

The contextual bandit problem can be viewed as a more realistic extension of the traditional bandit problem (Lai and Robbins, 1985), which does not provide any context information to the learner. The use of context allows expressing many interesting applications. For instance, consider an online advertising system operated by a contextual bandit learner. For each user visit (iteration), the advertising algorithm (learner) is provided with the known properties about the user (context), and an advertisement (action) from a pool of relevant advertisements (set of candidate actions) is selected and displayed to that user. The company's earning (reward) can be calculated based on whether the user clicked the selected advertisement and the value of the advertisement itself. The reward can then be revealed to the algorithm as the feedback, and maximizing the cumulative reward directly connects to the company's total earning. Many other web applications for advertisement and recommendation can be similarly modeled as a contextual bandit problem (Li et al., 2010, 2011; Pandey et al., 2007), and hence the problem has been attracting much research attention (Auer et al., 2002a,b; Auer, 2002; Kakade et al., 2008; Chu et al., 2011; Chen et al., 2014).

The bandit setting contrasts the full-information setting in traditional online learning (Kakade et al., 2008; Chen et al., 2014), in which the reward of every action is revealed to the learner regardless of whether the action is selected. Intuitively, it is easier to design learners under the full-information setting, and such learners (shorthanded full-information learners) usually perform better than their siblings that run under bandit-setting (to be shown in Section 2). The performance difference motivates us to study whether we can mimic the full-information setting to design better contextual bandit learners.

In particular, we propose the concept of *pseudo-rewards*, which embed estimates to the hidden rewards of non-selected actions under the bandit setting. Then, the revealed reward and the pseudo-rewards can be jointly fed to full-information learners to mimic the full-information setting. We study the possibility of feeding pseudo-rewards to linear full-information learners, which compute the estimates by linear ridge regression. In particular, we propose to use pseudo-rewards that *over-estimate* the hidden rewards. Such pseudo-rewards embed the intention of exploration, which means selecting the action that the learner is less certain about. The intention has been shown useful in many of the existing works on the contextual bandit problem (Auer et al., 2002b; Li et al., 2010). To neutralize the effect of over-estimating the hidden rewards routinely, we further propose a forgetting mechanism that decreases the influence of the pseudo-rewards received in the earlier iterations.

Combining the over-estimated pseudo-rewards, the forgetting mechanism, and the state-of-the-art Linear Upper Confidence Bound (LinUCB) algorithm (Li et al., 2010; Chu et al., 2011), we design a novel algorithm, Linear Pseudo-reward Upper Confidence Bound (LinPRUCB). We prove that LinPRUCB results in a matching regret bound to LinUCB. In addition, we demonstrate empirically with artificial and real-world data sets that LinPRUCB enjoys two additional properties. First, with the pseudo-rewards on hand, LinPRUCB is able to mimic the full information setting more closely, which leads to faster reward-gathering in earlier iterations when compared with LinUCB. Second, with the over-estimated pseudo-rewards on hand, LinPRUCB is able to express the intention of exploration in the *model updating* step rather than the *action selection* step of the learner. Based on the second property, we derive a variant of LinPRUCB, called LinPRGreedy, to perform action selection faster than LinPRUCB and LinUCB while maintaining competitive performance to those two algorithms. The two properties above make our proposed LinPRGreedy a favorable choice in real-world applications that usually demand fast reward-gathering and fast action-selection.

This paper is organized as follows. In Section 2, we give a formal setup of the problem and introduce related works. In Section 3, we describe the framework of using pseudo-rewards and derive LinPRUCB with its regret bound analysis. We present experimental simulations on artificial data sets and large-scale benchmark data sets in Section 4. Finally, we introduce the practical variant LinPRGreedy in Section 5, and conclude in Section 6.

2. Preliminaries

We use bold lower-case symbol like $\mathbf{w}$ to denote a column vector and bold upper-case symbol like $\mathbf{Q}$ to denote a matrix. Let $[N]$ represents the set $\{1, 2, \cdots, N\}$. The contextual bandit problem consists of T iterations of decision making. In each iteration t, a contextual bandit algorithm (learner) first observes a context $\mathbf{x}_t \in \mathbb{R}^d$ from the environment, and is asked to select an action a_t from the action set $[K]$. We shall assume that the contexts are bounded by $\|\mathbf{x}_t\|_2 \leq 1$. After selecting an action a_t, the algorithm receives the corresponding reward $r_{t,a_t} \in [0,1]$ from the environment, while other rewards $r_{t,a}$ for $a \neq a_t$ are hidden from the algorithm. The algorithm should strategically select the actions in order to maximize the cumulative reward $\sum_{t=1}^{T} r_{t,a_t}$ after the T iterations.

In this paper, we study the contextual bandit problem with linear payoff functions (Chu et al., 2011), where the reward is generated by the following process. For any context $\mathbf{x}_t$ and any action $a \in [K]$, assume that the reward $r_{t,a}$ is a random variable with conditional expectation $\mathbb{E}[r_{t,a}|\mathbf{x}_t] = \mathbf{u}_a^\top \mathbf{x}_t$, where $\mathbf{u}_a \in \mathbb{R}^d$ is an unknown vector with $\|\mathbf{u}_a\|_2 \leq 1$. The linear relationship allows us to define $a_t^* = \operatorname{argmax}_{a\in[K]} \mathbf{u}_a^\top \mathbf{x}_t$, which is the optimal strategy of action selection in the hindsight. Then, the goal of maximizing the cumulative reward can be translated to minimizing the regret of the algorithm, which is defined as $\text{regret}(T) = \sum_{t=1}^{T} r_{t,a_t^*} - \sum_{t=1}^{T} r_{t,a_t}$.

Next, we introduce a family of existing linear algorithms for the contextual bandit problem with linear payoff functions. The algorithms all maintain $\mathbf{w}_{t,a}$ as the current estimate of $\mathbf{u}_a$. Then, $\mathbf{w}_{t,a}^\top \mathbf{x}_t$ represents the estimated reward of selecting action a upon seeing $\mathbf{x}_t$. The most naïve algorithm of the family is called Linear Greedy (LinGreedy). During iteration t, with the estimates $\mathbf{w}_{t,a}$ on hand, LinGreedy simply selects an action a_t that maximizes the estimated reward. That is, $a_t = \operatorname{argmax}_{a\in[K]} \mathbf{w}_{t,a}^\top \mathbf{x}_t$. Then, LinGreedy computes the weight vectors $\mathbf{w}_{t+1,a}$ for the next iteration by ridge regression

$$\mathbf{w}_{t+1,a} = (\lambda \mathbf{I}_d + \mathbf{X}_{t,a}^\top \mathbf{X}_{t,a})^{-1}(\mathbf{X}_{t,a}^\top \mathbf{r}_{t,a}), \tag{1}$$

where $\lambda > 0$ is a given parameter for ridge regression; $\mathbf{I}_d$ is a $d \times d$ identity matrix. We use $(\mathbf{X}_{t,a}, \mathbf{r}_{t,a})$ to store the observed contexts and rewards *only* when action a gets selected by the algorithm: $\mathbf{X}_{t,a}$ is a matrix that contains $\mathbf{x}_\tau^\top$ as rows, where $1 \le \tau \le t$ and $a_\tau = a$; $\mathbf{r}_{t,a}$ is a vector that contains the corresponding $r_{\tau,a}$ as components. That is, each $\mathbf{x}_\tau$ is only stored in $\mathbf{X}_{t,a_\tau}$. Then, it is easy to see that $\mathbf{w}_{t+1,a} = \mathbf{w}_{t,a}$ for $a \neq a_t$. That is, only the weight vector $\mathbf{w}_{t+1,a}$ with $a = a_t$ will be updated.

Because only the reward r_{t,a_t} of the selected action a_t is revealed to the algorithm, it is known that LinGreedy suffers from its myopic decisions on action selection. In particular, LinGreedy only focuses on exploitation (selecting the actions that are seemly more rewarding) but does not conduct exploration (*trying* the actions that the algorithm is less certain about). Thus, it is possible that LinGreedy would miss the truly rewarding actions in the long run. One major challenge in designing better contextual bandit algorithms is to strike a balance between exploitation and exploration (Auer, 2002).

For LinGreedy, one simple remedy for the challenge is to replace the a_t of LinGreedy by a randomly selected action with a non-zero probability ϵ in each iteration. This remedy is known as ϵ-greedy, which is developed from reinforcement learning (Sutton and Barto, 1998), and has been extended to some more sophisticated algorithms like epoch-greedy (Langford and Zhang, 2007) that controls ϵ dynamically. Both ϵ-greedy and epoch-greedy explores other potential actions with a non-zero probability, and thus generally performs better than LinGreedy in theory and in practice.

Another possible approach of balancing exploitation and exploration is through the use of the upper confidence bound (Auer, 2002). Upper-confidence-bound algorithms cleverly select an action based on some mixture of the estimated reward and an uncertainty term. The uncertainty term represents the amount of information that has been received for the candidate action and decreases as more information is gathered during the iterations. The mixture allows the algorithms to select either an action with high estimated reward (exploitation) or with high uncertainty (exploration).

Linear Upper Confidence Bound (LinUCB) (Chu et al., 2011; Li et al., 2010) is a state-of-the-art representative within the family of upper-confidence-bound algorithms. In addition to calculating the estimated reward $\mathbf{w}_{t,a}^\top \mathbf{x}_t$ like LinGreedy, LinUCB computes the uncertainty term by $c_{t,a} = \sqrt{\mathbf{x}_t \mathbf{Q}_{t-1,a}^{-1} \mathbf{x}_t}$, where $\mathbf{Q}_{t-1,a} = (\lambda \mathbf{I}_d + \mathbf{X}_{t-1,a}^\top \mathbf{X}_{t-1,a})$ is the matrix that gets inverted when computing $\mathbf{w}_{t,a}$ by (1). For any given $\alpha > 0$, the upper confidence bound of action a can then be formed by $\mathbf{w}_{t,a}^\top \mathbf{x}_t + \alpha c_{t,a}$. LinUCB takes this bound for selecting the action to balance exploitation and exploration. That is,

$$a_t = \operatorname*{argmax}_{a \in [K]} \left(\mathbf{w}_{t,a}^\top \mathbf{x}_t + \alpha c_{t,a} \right). \tag{2}$$

Then, after receiving the reward r_{t,a_t}, similar to LinGreedy, LinUCB takes ridge regression (1) to update only the weight vector $\mathbf{w}_{t+1,a_t}$.

Note that LinGreedy can then be viewed as an extreme case of LinUCB with $\alpha = 0$. The extreme case of LinGreedy enjoys the computational benefit of only taking $\mathcal{O}(d)$ of time for selecting an action, which is useful in practical applications that need fast action-selection. In contrast, general LinUCB with $\alpha > 0$ requires $\mathcal{O}(d^2)$ in computing the uncertainty term. Nevertheless, LinGreedy enjoys a better theoretical guarantee and better practical performance. In particular, for proper choices of a non-zero α, LinUCB enjoys a rather low regret bound (Chu et al., 2011) and reaches state-of-the-art performance in practice (Li et al., 2010).

3. Linear Pseudo-reward Upper Confidence Bound

In LinUCB (as well as LinGreedy), the estimated weight vectors $\mathbf{w}_{t+1,a}$ are computed by $(\mathbf{X}_{t,a}, \mathbf{r}_{t,a})$: all the observed contexts and rewards *only* when action a gets selected by the algorithm. Because of the bandit setting, if an action a_t is selected in iteration t, the rewards $r_{t,a}$ for $a \neq a_t$ are unknown to the learning algorithm. Since $r_{t,a}$ are unknown, the context $\mathbf{x}_t$ is not used to update $\mathbf{w}_{t+1,a}$ for any $a \neq a_t$, which appears to be a waste of the information in $\mathbf{x}_t$. On the other hand, if we optimistically assume that all the rewards $r_{t,a}$ are revealed to the learning algorithm, which corresponds to the full-information setting in online learning, ideally $\mathbf{w}_{t+1,a}$ for every action a can be updated with $(\mathbf{x}_t, r_{t,a})$. Then, $\mathbf{w}_{t+1,a}$ shall converge to a decent estimate faster, and the resulting "full-information variant" of LinUCB shall achieve better performance.

We demonstrate the motivation above with the results on a simple artificial data set in Figure 1. More detailed comparisons with artificial data sets will be shown in Section 4. Each curve represents the average cumulative reward with different types of feedback information. The pink circles represent LinUCB that receives full information about the rewards; the red diamonds represent the original LinUCB that receives the bandit feedback. Unsurprisingly, Figure 1 suggests that the full-information LinUCB outperforms the bandit-information LinUCB by a big gap.

We include another curve colored by black in Figure 1. The black curve represents the performance of the full-information LinUCB when the received rewards are perturbed by some random white noise within $[-0.05, 0.05]$. As a consequence, the black curve is worse than the (pink-circle) curve of the full-information LinUCB with noise-less rewards. Nevertheless, the black curve is still better than the (red-diamond) curve of the bandit-information LinUCB. That is, by updating $\mathbf{w}_{t+1,a}$ on *all* actions a (full-information), even with inaccurate rewards, it is possible to improve the original LinUCB that only updates $\mathbf{w}_{t+1,a_t}$ of the *selected* action a_t (bandit-information). The results motivate us to study how contextual bandit algorithms can be designed by mimicking the full-information setting.

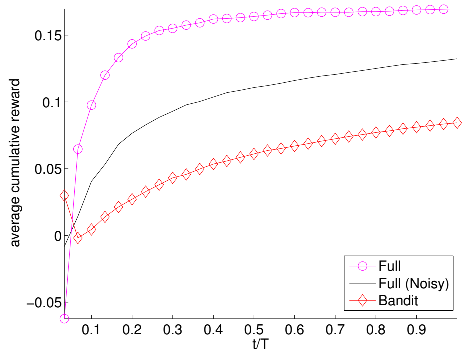


Figure 1: a motivating example

Based on the motivation, we propose a novel framework of Linear Pseudo-reward (LinPR) algorithms. In addition to using $(\mathbf{x}_t, r_{t,a_t})$ to update the weight vector $\mathbf{w}_{t+1,a_t}$, LinPR algorithms compute some *pseudo-rewards* $p_{t,a}$ for all $a \neq a_t$, and take $(\mathbf{x}_t, p_{t,a})$ to update

$\mathbf{w}_{t+1,a}$ for $a \neq a_t$ as well. Three questions remain in designing an algorithm under the LinPR framework. How are $p_{t,a}$ computed? How are $(\mathbf{x}_t, p_{t,a})$ used to update $\mathbf{w}_{t+1,a}$? How are $\mathbf{w}_{t+1,a}$ used in selecting the next action? We answer those questions below to design a concrete and promising algorithm.

Choosing $p_{t,a}$. Ideally, for the purpose of mimicking the full-information setting, we hope $p_{t,a}$ to be an accurate estimate of the true reward $r_{t,a}$. If so, $p_{t,a}$ serves as a free feedback without costing the algorithm any action on exploration. Of course, the hope cannot be achieved in general. We propose to use $p_{t,a}$ to *over-estimate* the true reward instead. The over-estimates tilt $\mathbf{w}_{t,a}$ towards those actions a that the algorithm is less certain about when observing $\mathbf{x}_t$. That is, the over-estimates guide the algorithm towards exploring the less certain actions during the model updating step, which fits the intent to strike a balance between exploitation and exploration (Auer, 2002).

Formally, we propose to borrow the upper confidence bound in LinUCB, which is an over-estimate of the true reward, for computing the pseudo-reward $p_{t,a}$. That is,

$$p_{t,a} = \mathbf{w}_{t,a}^\top \mathbf{x}_t + \beta \hat{c}_{t,a}, \text{ with } \hat{c}_{t,a} = \sqrt{\mathbf{x}_t^\top \hat{\mathbf{Q}}_{t-1,a}^{-1} \mathbf{x}_t}, \tag{3}$$

where $\hat{\mathbf{Q}}_{t-1,a}$ is a matrix extended from $\mathbf{Q}_{t-1,a}$ and will be specified later in (5); $\beta > 0$ is a given parameter like α. Similar to the upper confidence bound in LinUCB, the first term $\mathbf{w}_{t,a}^\top \mathbf{x}_t$ computes the estimated reward and the second term $\beta \hat{c}_{t,a}$ corresponds to some measure of uncertainty.

With the definition of $p_{t,a}$ in (3), we use the vector $\mathbf{p}_{t,a}$ to gather all the pseudo-rewards calculated for action a, and the matrix $\bar{\mathbf{X}}_{t,a}$ for the corresponding contexts. That is, $\bar{\mathbf{X}}_{t,a}$ is a matrix that contains $\mathbf{x}_\tau^\top$ as rows, where $1 \leq \tau \leq t$ and $a_\tau \neq a$; $\mathbf{p}_{t,a}$ is a vector that contains the corresponding $p_{\tau,a}$ as components. Next, we discuss how the information within $(\bar{\mathbf{X}}_{t,a}, \mathbf{p}_{t,a})$ can be used to properly update $\mathbf{w}_{t+1,a}$.

Updating $\mathbf{w}_{t+1,a}$ with $(\mathbf{x}_t, p_{t,a})$. Recall that $(\mathbf{X}_{t,a}, \mathbf{r}_{t,a})$ contains all the observed contexts and rewards *only* when action a gets selected by the algorithm, and $(\bar{\mathbf{X}}_{t,a}, \mathbf{p}_{t,a})$ contains the contexts and pseudo-rewards only when action a is *not* selected by the algorithm. Then, a natural attempt is to directly include the pseudo-rewards *as if* they are the true rewards when updating the model. That is, $\mathbf{w}_{t+1,a}$ can be updated by solving the ridge regression problem of

$$\mathbf{w}_{t+1,a} = \operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^d} \left(\lambda \|\mathbf{w}\|^2 + \|\mathbf{X}_{t,a}\mathbf{w} - \mathbf{r}_{t,a}\|^2 + \|\bar{\mathbf{X}}_{t,a}\mathbf{w} - \mathbf{p}_{t,a}\|^2 \right). \tag{4}$$

The caveat with the update formula in (4) is that the pseudo-rewards affect the calculation of $\mathbf{w}_{t+1,a}$ permanently, even though some of them, especially the earlier ones, may be rather inaccurate. The inaccurate ones may become irrelevant or even misleading in the latter iterations. Thus, intuitively we may not want those inaccurate pseudo-rewards to have the same influence on $\mathbf{w}_{t+1,a}$ as the more recent, and perhaps more accurate, pseudo-rewards.

Following the intuition, we propose the following forgetting mechanism, which puts emphases on more recent pseudo-rewards and their contexts than earlier ones. Formally, we take a forgetting parameter $\eta \in [0, 1)$ to control how fast the algorithm forgets previous pseudo-rewards, where smaller η corresponds to faster forgetting. Note that every pair $(\mathbf{x}_t, p_{t,a})$ corresponds to an error term of $(\mathbf{w}^\top \mathbf{x}_t - p_{t,a})^2$ within the objective function of (4). When updating $\mathbf{w}_{t+1,a}$, we treat the most recent pseudo-reward as if it were the true reward, and do not modify the associated term. For the second most recent pseudo-reward, however, we decrease its influence on $\mathbf{w}_{t+1,a}$ by multiplying its associated term with η^2. Similarly, if $\ell_{t,a}$ pseudo-rewards have been gathered when updating $\mathbf{w}_{t+1,a}$, the term that corresponds to the earliest pseudo-reward would be multiplied by $\eta^{2(\ell_{t,a}-1)}$, which decreases along $\ell_{t,a}$ rapidly when $\eta < 1$.

Equivalently, the forgetting mechanism can be performed as follows. Note that there are $\ell_{t,a}$ rows in $\bar{\mathbf{X}}_{t,a}$ and $\mathbf{p}_{t,a}$. Let $\hat{\mathbf{X}}_{t,a}$ be an $\ell_{t,a} \times d$ matrix with its i-th row being that of $\bar{\mathbf{X}}_{t,a}$ multiplied by the factor $\eta^{\ell_{t,a}-i}$; similarly, let $\hat{\mathbf{p}}_{t,a}$ be an $\ell_{t,a}$-dimensional vector with its i-th component being that of $\mathbf{p}_{t,a}$ multiplied by $\eta^{\ell_{t,a}-i}$. We then update $\mathbf{w}_{t+1,a}$ by

$$\mathbf{w}_{t+1,a} = \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^d} \left(\lambda\|\mathbf{w}\|^2 + \|\mathbf{X}_{t,a}\mathbf{w} - \mathbf{r}_{t,a}\|^2 + \|\hat{\mathbf{X}}_{t,a}\mathbf{w} - \hat{\mathbf{p}}_{t,a}\|^2\right),$$

which yields the closed-form solution

$$\mathbf{w}_{t+1,a} = \hat{\mathbf{Q}}_{t,a}^{-1}\left(\mathbf{X}_{t,a}^\top \mathbf{r}_{t,a} + \hat{\mathbf{X}}_{t,a}^\top \hat{\mathbf{p}}_{t,a}\right), \text{ where } \hat{\mathbf{Q}}_{t,a} = \lambda \mathbf{I}_d + \mathbf{X}_{t,a}^\top \mathbf{X}_{t,a} + \hat{\mathbf{X}}_{t,a}^\top \hat{\mathbf{X}}_{t,a}. \quad (5)$$

One remark for the update formula in (5) is that the use of $\beta > 0$ to over-estimate the true reward in $p_{t,a}$ is crucial. If only $p'_{t,a} = \mathbf{w}_{t,a}^\top \mathbf{x}_t$ is used as the pseudo-reward, we would be feeding the model with its own output. Then, the weight vector $\mathbf{w}_{t,a}$ can readily make zero error on $(\mathbf{x}_t, p'_{t,a})$, and thus provably $\mathbf{w}_{t+1,a} = \mathbf{w}_{t,a}$ is not updated. When $\beta > 0$, however, the weight vectors $\mathbf{w}_{t,a}$ for $a \neq a_t$ effectively embed the intention for exploration when the algorithm is less certain about selecting action a for $\mathbf{x}_t$.

Using $\mathbf{w}_{t,a}$ for Action-selection. Two natural choices for using $\mathbf{w}_{t,a}$ for action selection is either to decide greedily based on the estimated rewards $\mathbf{w}_{t,a}^\top \mathbf{x}_t$, or to decide in an upper-confidence-bound manner by

$$a_t = \operatorname*{argmax}_{a\in[K]} \left(\mathbf{w}_{t,a}^\top \mathbf{x}_t + \alpha \hat{c}_{t,a}\right), \quad (6)$$

where $\hat{c}_{t,a} = \sqrt{\mathbf{x}_t \hat{\mathbf{Q}}_{t-1,a}^{-1} \mathbf{x}_t}$ is upgraded from the $c_{t,a}$ used by LinUCB. We introduce the latter choice here, and will discuss about the greedy choice in Section 5.

Combining the pseudo-rewards in (3), the model updating with the forgetting mechanism in (5), and the upper-confidence-bound decision in (6), we derive our proposed Linear Pseudo-reward Upper Confidence Bound (LinPRUCB) algorithm, as shown in Algorithm 1. Similar to LinUCB, our LinPRUCB selects an action based on an upper confidence bound controlled by the parameter $\alpha > 0$. The main difference from LinUCB is the loop from Step 7 to Step 16 that updates the weight $\mathbf{w}_{t+1,a}$ for all actions a. For the selected action a_t, its corresponding context vector and actual reward is updated in Step 9 as LinUCB does. For actions whose rewards are hidden, LinPRUCB computes the pseudo-rewards in Step 11, and applies the forgetting mechanism in Step 12. Finally Step 15 computes the new weight vectors $\mathbf{w}_{t+1,a}$ shown in (5) for the next iteration.

Algorithm 1 Linear Pseudo-Reward Upper Confidence Bound (LinPRUCB)

```
1: Parameters: $\alpha, \beta, \eta, \lambda > 0$
2: Initialize $\mathbf{w}_{1,a} := \mathbf{0}_d$, $\hat{\mathbf{Q}}_{0,a} := \lambda \mathbf{I}_d$, $\hat{\mathbf{V}}_{0,a} := \mathbf{V}_{0,a} := \mathbf{0}_{d\times d}$, $\hat{\mathbf{z}}_{0,a} := \mathbf{z}_{0,a} := \mathbf{0}_d$, for every $a \in [K]$
3: for $t := 1$ to $T$ do
4:   Observe $\mathbf{x}_t$
5:   Select $a_t := \operatorname{argmax}_{a\in[K]} \mathbf{w}_{t,a}^\top \mathbf{x}_t + \alpha\sqrt{\mathbf{x}_t^\top \hat{\mathbf{Q}}_{t-1,a}^{-1}\mathbf{x}_t}$
6:   Receive reward $r_{t,a_t}$
7:   for $a \in [K]$ do
8:     if $a = a_t$ then
9:       $\mathbf{V}_{t,a} := \mathbf{V}_{t-1,a} + \mathbf{x}_t\mathbf{x}_t^\top$,
10:    else
11:      $p_{t,a} := \min\left(\mathbf{w}_{t,a}^\top \mathbf{x}_t + \beta\sqrt{\mathbf{x}_t^\top \hat{\mathbf{Q}}_{t,a}^{-1}\mathbf{x}_t}, 1\right)$
12:      $\hat{\mathbf{V}}_{t,a} := \eta\hat{\mathbf{V}}_{t-1,a} + \mathbf{x}_t\mathbf{x}_t^\top$,
13:    end if
14:    $\hat{\mathbf{Q}}_{t,a} := \lambda\mathbf{I}_d + \mathbf{V}_{t,a} + \hat{\mathbf{V}}_{t,a}$
15:    $\mathbf{w}_{t+1,a} := \hat{\mathbf{Q}}_{t,a}^{-1}(\mathbf{z}_{t,a} + \hat{\mathbf{z}}_{t,a})$
16:  end for
17: end for
```

We will compare LinPRUCB to LinUCB empirically in Section 4. Before so, we establish the theoretical guarantee of LinPRUCB and compare it to that of LinUCB first. The key results are summarized and discussed as follows. Under the assumption that the received rewards are independent from each other, Lemma 2 below shows that LinPRUCB can achieve a regret of $(1+\alpha+\rho)\left(\sum_{t\in[T]} \hat{c}_{t,a_t}\right) + \tilde{\mathcal{O}}\left(\sqrt{T}\right)$ with probability $1-\delta$, where $\rho = \mathcal{O}(\beta)$. LinUCB under the same condition (Chu et al., 2011), on the other hand, achieves a regret of $(1+\alpha)\left(\sum_{t\in[T]} c_{t,a_t}\right) + \tilde{\mathcal{O}}(\sqrt{T})$ with probability $1-\delta$.

One difference between the two bounds is that the bound of LinPRUCB contains an additional term of ρ, which is of $\mathcal{O}(\beta)$ and comes from the pseudo-rewards. By choosing $\beta = \mathcal{O}(\alpha)$, however, the term can always be made comparable to α.

The other difference is that the bound of LinPRUCB contains $\hat{c}_{t,a_t}$ terms instead of c_{t,a_t}. Recall that $\hat{c}_{t,a} = \sqrt{\mathbf{x}_t^\top \hat{\mathbf{Q}}_{t-1,a}^{-1}\mathbf{x}_t}$, with $\hat{\mathbf{Q}}_{t-1,a} = \lambda\mathbf{I}_d + \mathbf{X}_{t-1,a}^\top\mathbf{X}_{t-1,a} + \hat{\mathbf{X}}_{t-1,a}^\top\hat{\mathbf{X}}_{t-1,a}$, while $c_{t,a} = \sqrt{\mathbf{x}_t^\top \mathbf{Q}_{t-1,a}^{-1}\mathbf{x}_t}$, with $\mathbf{Q}_{t-1,a} = \lambda\mathbf{I}_d + \mathbf{X}_{t-1,a}^\top\mathbf{X}_{t-1,a}$. Since $\hat{\mathbf{Q}}_{t-1,a}$ contains the additional term $\hat{\mathbf{X}}_{t-1,a}^\top\hat{\mathbf{X}}_{t-1,a}$ when compared with $\mathbf{Q}_{t-1,a}$, the uncertainty terms $\hat{c}_{t,a}$ are likely smaller than $c_{t,a}$, especially in the early iterations. Figure 2 shows one typical case that compares c_{t,a_t} of LinUCB and $\hat{c}_{t,a_t}$ of LinPRUCB with one artificial data set used in Section 4. We see that $\hat{c}_{t,a_t}$ are indeed generally smaller than c_{t,a_t}. The empirical observation in Figure 2 and the theoretical bound in Lemma 2 explain how LinPRUCB could perform better than LinUCB through playing with the bias-variance trade-off: decreasing the variance (expressed by $\hat{c}_{t,a_t}$) while slightly increasing the bias (expressed by ρ).

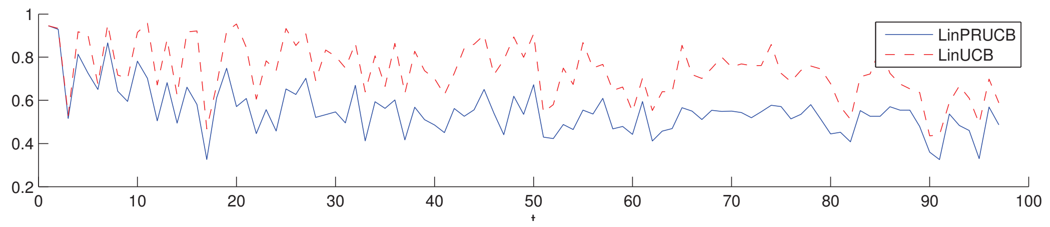


Figure 2: Comparison of the values on the uncertainty terms

The regret bounds discussed above rely on the assumption of independent rewards, which is unlikely to hold in general. To deal with this issue, we follow the approach of Chu et al. (2011) and discuss below on how LinPRUCB can be modified to a variant algorithm for which a small regret bound can actually be guaranteed without any assumptions. The key technique is to partition the iterations into several parts such that the received rewards in the same part are in fact independent from each other (Auer, 2002). More precisely, Chu et al. (2011) constructs a master algorithm named SupLinUCB, which performs the partitioning and calls a subroutine BaseLinUCB (modified from LinUCB) on each part separately. Similarly, we construct another master algorithm SupLinPRUCB, which is similar to SupLinUCB but calls BaseLinPRUCB (modified from LinPRUCB) instead. The detailed descriptions are listed in Appendix A.[1]

Then, the big picture of the analysis follows closely from that of LinUCB (Auer, 2002; Chu et al., 2011). First, as shown in Appendix A, SupLinPRUCB in iteration t partitions the first $t-1$ iterations into S parts: $\Psi_t^1, \dots, \Psi_t^S$. Using the same proof as that for Lemma 14 by Auer (2002), we can first establish the following lemma.

Lemma 1 *For any $s \in [S]$, any $t \in [T]$, and any fixed sequence of contexts $\mathbf{x}_\tau$, with $\tau \in \Psi_t^s$, the corresponding rewards r_{τ,a_τ} are independent random variables with $\mathbb{E}[r_{\tau,a_\tau}] = \mathbf{x}_\tau^\top \mathbf{u}_a$.*

Then, we can prove that BaseLinPRUCB when given such an independence guarantee can provide a good estimate of the true reward.

Lemma 2 *Suppose the input index set $\Psi_t \subseteq [t-1]$ given to* BaseLinPRUCB *has the property that for fixed contexts $\mathbf{x}_\tau$, for $\tau \in \Psi_t$, the corresponding rewards r_{τ,a_τ} are independent random variables with means $\mathbf{x}_\tau^\top \mathbf{u}_{a_\tau}$. Then, for some $\alpha = \mathcal{O}(\sqrt{\ln(TK/\delta)})$ and $\rho = (2+\beta)/\sqrt{1-\eta}$, we have with probability $1-\delta/T$ that $|\mathbf{x}_t^\top \mathbf{w}_{t,a} - \mathbf{x}_t^\top \mathbf{u}_a| \le (1+\alpha+\rho)\hat{c}_{t,a}$ for every $a \in [K]$.*

Proof For notation convenience, we drop all the subscripts involving t and a below. By definition,

$$\begin{aligned}
\left|\mathbf{x}^\top \mathbf{w} - \mathbf{x}^\top \mathbf{u}\right| &= \left|\mathbf{x}^\top \hat{\mathbf{Q}}^{-1}(\mathbf{X}^\top \mathbf{r} + \hat{\mathbf{X}}^\top \hat{\mathbf{p}}) - \mathbf{x}^\top \hat{\mathbf{Q}}^{-1}(\lambda \mathbf{I}_d + \mathbf{X}^\top \mathbf{X} + \hat{\mathbf{X}}^\top \hat{\mathbf{X}})\mathbf{u}\right| \\
&= \left|\mathbf{x}^\top \hat{\mathbf{Q}}^{-1}\mathbf{X}^\top(\mathbf{r} - \mathbf{X}\mathbf{u}) - \lambda \mathbf{x}^\top \hat{\mathbf{Q}}^{-1}\mathbf{u} + \mathbf{x}^\top \hat{\mathbf{Q}}^{-1}\hat{\mathbf{X}}^\top(\hat{\mathbf{p}} - \hat{\mathbf{X}}\mathbf{u})\right| \\
&\le \left|\mathbf{x}^\top \hat{\mathbf{Q}}^{-1}\mathbf{X}^\top(\mathbf{r} - \mathbf{X}\mathbf{u})\right| + \lambda\left|\mathbf{x}^\top \hat{\mathbf{Q}}^{-1}\mathbf{u}\right| + \left|\mathbf{x}^\top \hat{\mathbf{Q}}^{-1}\hat{\mathbf{X}}^\top(\hat{\mathbf{p}} - \hat{\mathbf{X}}\mathbf{u})\right|. \qquad (7)
\end{aligned}$$

1. The full version that includes the appendix can be found at http://www.csie.ntu.edu.tw/~htlin/paper/doc/acml14pseudo.pdf.

The first two terms in (7) together can be bounded by $(1+\alpha)\hat{c}$ with probability $1-\delta/T$ using arguments similar to those by Chu et al. (2011). The third term arises from our use of pseudo-rewards, which makes our analysis different from that of BaseLinUCB. This is where our forgetting mechanism comes to help. By the Cauchy-Schwarz inequality, this term is at most $\|\mathbf{x}^\top \hat{\mathbf{Q}}^{-1}\hat{\mathbf{X}}^\top\|\|\hat{\mathbf{p}} - \hat{\mathbf{X}}\mathbf{u}\|$, where one can show that $\|\mathbf{x}^\top \hat{\mathbf{Q}}^{-1}\hat{\mathbf{X}}^\top\| \le \hat{c}$ using a similar argument as Lemma 1 of Chu et al. (2011). Since the i-th entry of the vector $\hat{\mathbf{p}} - \hat{\mathbf{X}}^\top\mathbf{u}$ by definition is at most $(2+\beta)\eta^{\ell-i}$, we have

$$\|\hat{\mathbf{p}} - \hat{\mathbf{X}}^\top\mathbf{u}\| \le (2+\beta)\sqrt{\sum_{i\le\ell}\eta^{2(\ell-i)}} \le \frac{2+\beta}{\sqrt{1-\eta}} = \rho.$$

Combining these bounds together, we have the lemma. ■

Lemma 2 is the key distinction in our analysis, which justifies our use of pseudo-rewards and the forgetting mechanism. With Lemma 1 and Lemma 2, the rest of the regret analysis is almost identical to that of LinUCB (Auer, 2002; Chu et al., 2011), and the analysis leads to the following main theoretical result. The details of the proof are listed in Appendix A.

Theorem 3 *For some $\alpha = \mathcal{O}(\sqrt{\ln(TK/\delta)})$, for any $\beta = \mathcal{O}(\alpha)$ and any constant $\eta \in [0,1)$, SupLinPRUCB with probability $1-\delta$ achieves a regret of $\mathcal{O}\left(\sqrt{dKT\ln^3(KT/\delta)}\right)$.*

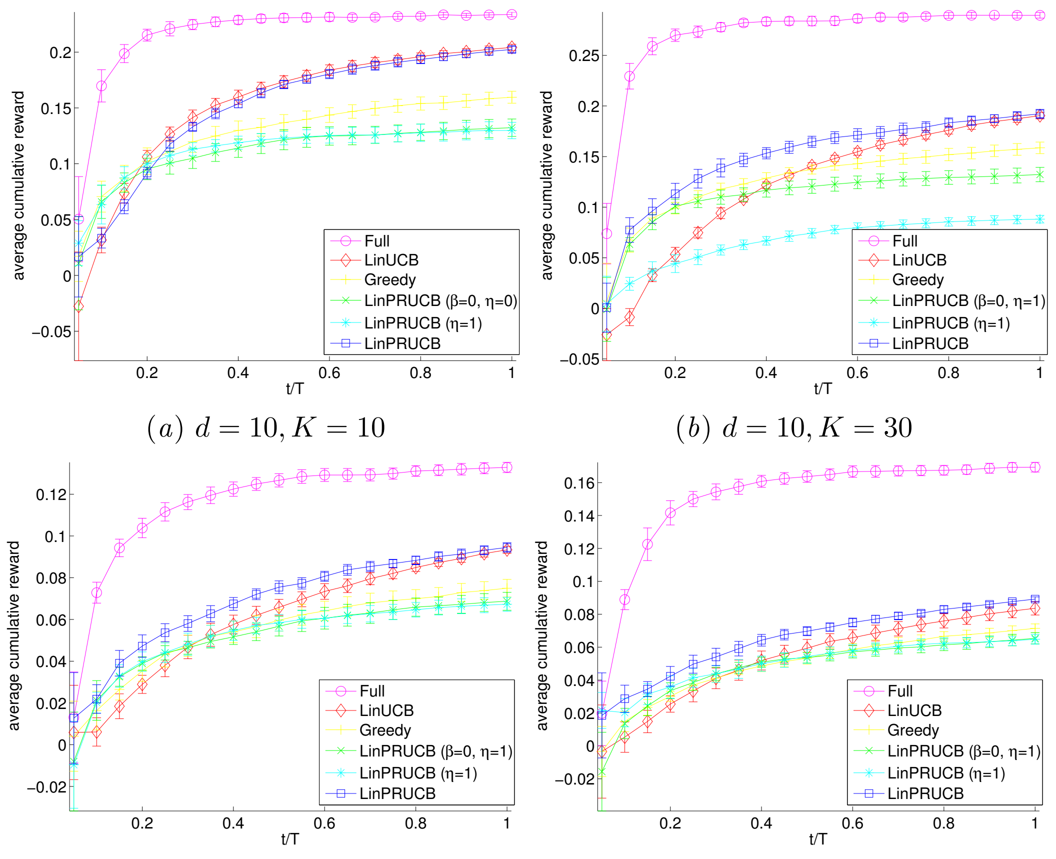
4. Experiments

Next, we conduct empirical studies on both artificial and real-world data sets to justify the idea of using pseudo-rewards and to compare the performance between LinPRUCB and the state-of-the-art LinUCB algorithm.

Artificial Data Sets. First, we simulate with artificial data sets as follows. Unit vectors $\mathbf{u}_1, \dots, \mathbf{u}_K$ for K actions are drawn uniformly from $\mathbb{R}^d$. In iteration t out of the T iterations, a context $\mathbf{x}_t$ is first sampled from an uniform distribution within $\|\mathbf{x}\| \le 1$. Then, the reward $r_{t,a_t} = \mathbf{u}_{a_t}^\top \mathbf{x}_t + v_t$ is generated, where $v_t \in [-0.05, 0.05]$ is a random white noise. For the simulation, we consider parameters $T = 1000$, $K \in \{10, 30\}$, $d \in \{10, 30\}$.

We plot the average cumulative reward (vertical axis) versus the normalized number of iterations (horizontal axis) to illustrate the key ideas of the proposed LinPRUCB algorithm and to compare it with LinUCB. Each curve is the average of 10 runs of simulations. The results are presented in Figure 3. For each (K, d) combination, we use pink-circle marks to represent the curve of LinUCB with full-information feedback, and red-diamond marks to represent the curve of LinUCB with bandit-information. Similar to the observations in the motivating Figure 1, the full-information LinUCB performs better than the bandit-information one. Another curve we include is colored yellow, which represents LinGreedy. The three curves serve as references to evaluate the performance of LinPRUCB.

To justify the idea of using over-estimated rewards as pseudo-rewards, we first set β in (3) to be 0, fix the forgetting parameter $\eta = 1$, and use grid search for the optimal $\alpha \in \{0, 0.2, \dots, 1.2\}$ based on the final cumulative reward after T iterations. This setting of (α, β, η) represents a LinPRUCB variant that estimates the pseudo-reward without the



$(a)\ d = 10, K = 10$ $(b)\ d = 10, K = 30$

$(c)\ d = 30, K = 10$ $(d)\ d = 30, K = 30$

Figure 3: Comparison of average cumulative reward for $T = 1000$.

uncertainty term $\hat{c}_{t,a}$ and never forgets the previous estimates. We use green-cross marks to represent the curve and call it LinPRUCB$(\alpha, 0, 1)$. Note that we allow LinUCB the same flexibility in choosing α with grid search.

Then, we fix $\eta = 1$ and use grid search for the optimal (α, β) within $\alpha \in \{0, 0.2, \ldots, 1.2\}$ and $\beta \in \{0, 0.2, \ldots, 1.2\}$. This setting of (α, β, η) represents a LinPRUCB variant that estimates the pseudo-reward with the uncertainty term in (3) but never forgets the previous estimates. We use light-blue-star marks to represent the curve and call it LinPRUCB$(\alpha, \beta, 1)$.

Finally, we study the full power of LinPRUCB that not only estimates the pseudo-reward according to (3) but also utilizes the forgetting mechanism. This is done by a grid search on α, β, and η, where $\eta \in [0.8, 0.95]$ with a grid step of 0.05. We use blue-square marks to represent the curve.

From the results that are plotted in Figure 3, for each given (K, d), LinPRUCB$(\alpha, 0, 1)$ and LinPRUCB$(\alpha, \beta, 1)$ cannot perform well. They are often worse than LinUCB in the end, and are even worse than LinGreedy. The results justify the importance of both the over-estimate with the uncertainty terms $\hat{c}_{t,a}$ and the forgetting mechanism that we propose.

Next, we compare the actual LinPRUCB with LinUCB in Figure 3. When focusing on the early iterations, LinPRUCB often gathers more rewards than LinUCB for each

(K, d). Interestingly, similar phenomenons also occur in the curves of LinPRUCB$(\alpha, 0, 1)$ and LinPRUCB$(\alpha, \beta, 1)$. Those results echo our discussions in Section 3 that it can be beneficial to trade some bias for a smaller variance to improve LinUCB. After T iterations, the performance of LinPRUCB is better than or comparable to that of LinUCB, which echoes our theoretical results in Section 3 of the matching regret bound.

The observations above support the key ideas in designing LinPRUCB. By allowing the algorithm to update every $\mathbf{w}_{t+1,a}$ instead of only $\mathbf{w}_{t+1,a_t}$, LinPRUCB achieves faster reward-gathering in the earlier iterations, and the *over-estimates* in the pseudo-rewards and the *forgetting* mechanism are both essential in maintaining the promising performance of LinPRUCB.

Real-world Benchmark Data Sets. Next, we extend our empirical study to two real-world benchmark data sets, R6A and R6B released by Yahoo!. To the best of our knowledge, the two data sets are the only public data sets for the contextual bandit problem. They first appeared in ICML 2012 Workshop on New Challenges for Exploration and Exploitation 3 (`https://explochallenge.inria.fr/`). The challenge is a practical problem that needs the algorithm to display news articles on a website strategically. The algorithm has to display a news article (action a_t) during the visit (iteration t) of a user (context $\mathbf{x}_t$), and the user can select to click (reward r_{t,a_t}) on the news to read more information or simply ignore it. The LinUCB algorithm and its variants are among the leading algorithms in the challenge, and there is a 36-hour run time limitation to all algorithms. This roughly translates to less then 5 ms for selecting an action, and 50 ms for updating the weight vectors $\mathbf{w}_{t+1,a}$.

In R6A, each context $\mathbf{x}_t$ is 6 dimensional and each dimension consists a value within $[0, 1]$. In R6B, each $\mathbf{x}_t$ is 136 dimensional with values within $\{0, 1\}$ and the first dimension being a constant 1. Rather than selecting an action from a fixed set $[K]$, the actions in the data sets are dynamic since new news articles may emerge over time and old article may be dropped. The dynamic nature does not affect the LinPRUCB and LinUCB algorithms, which maintain one $\mathbf{w}_{t,a}$ per action regardless of the size of the action set. For each data set, the reward r_{t,a_t} equals 1 if the user clicked on the article, and equals 0 otherwise. Then, the *click through rate* (CTR) of the algorithm, which is defined as the ratio of total clicks in the total evaluated iterations, corresponds to the average cumulative reward. In order to achieve an unbiased evaluation when using an offline data set such as R6A or R6B, we use the technique described by Li et al. (2011).

We split each data set into two sets, one for parameter tuning and the other for testing. The tuning set consists of 50,000 visits, and the testing set consists of the remaining 4,000,000 visits. We run grid search on the tuning set for every algorithm that comes with tunable parameters. Then, the parameter combination that achieves the best tuning CTR is coupled with the corresponding algorithm to evaluate the CTR on the testing set.

The experimental results for R6A and R6B are summarized in Table 1 and Table 2. The tables demonstrate that LinPRUCB is pretty competitive to LinUCB with 10% of test CTR improvement in R6A and comparable CTR performance in R6B. The results justify LinPRUCB to be a promising alternative to LinUCB in practical applications. Interestingly and somewhat surprisingly, for R6B, the best performing algorithm during

Algorithm	CTR (tuning)	CTR (testing)	selection time	updating time
LinPRUCB	0.038	**0.070**	**9.1**	3.6
LinUCB	**0.039**	0.063	9.4	**0.7**
LinGreedy	0.028	0.037	3.8	0.4
Random	0.024	0.030	0.7	0.0

Table 1: Experimental results on data set R6A. ($d = 6, K = 30$)

Algorithm	CTR (tuning)	CTR (testing)	selection time	updating time
LinPRUCB	**0.046**	**0.054**	**1082**	5003
LinUCB	**0.046**	0.053	1166	**248**
LinGreedy	0.039	0.059	25	201
Random	0.032	0.034	1	0

Table 2: Experimental results on data set R6B. ($d = 136, K = 30$)

testing is neither LinUCB nor LinPRUCB, but LinGreedy. It is possibly because the data set is of a larger d and thus exploration is a rather challenging task.

We also list the selection time and the updating time for each algorithm on data sets R6A and R6B in Table 1 and Table 2, respectively. The updating time is the time that an algorithm needed to update its model $\mathbf{w}_{t+1,a}$ after the reward is revealed. The selection time is the time that an algorithm needed to return an action a_t, which links directly to the loading speed of a web page in this competition. On both data sets we calculate the selection time and the updating time aggregated in 50,000 iterations and report them in seconds.

The results in Table 1 show that the selection time of LinPRUCB and LinUCB are similar, but LinPRUCB needs more time in updating. The situation becomes more prominent in Table 2 for larger dimension d. This is because in one iteration, LinUCB only has to update one model $\mathbf{w}_{t,a}$, but LinPRUCB needs to update every model for each action. We also note that the selection time for LinPRUCB and LinUCB are larger than the updating time. This is somewhat not practical since in real-world applications, a system usually needs to react to users within a short time, while updating on the server can take longer. To meet the real-world requirements, we present a practical variant of the proposed LinPRUCB algorithm in the next section. The variant can conduct faster action selection while enjoying similar performance to the original LinPRUCB in some cases.

5. A Practical Variant of LinPRUCB

The practical variant is derived by replacing the action selection step (6) of LinPRUCB with a greedy one $a_t = \operatorname{argmax}_{a\in[K]} \mathbf{w}_{t,a}^\top \mathbf{x}_t$. That is, we can simply set $\alpha = 0$ for LinPRUCB to drop the calculation of the uncertainty terms $\hat{c}_{t,a}$. Then, similar to how LinUCB becomes LinGreedy when $\alpha = 0$, LinPRUCB becomes the Linear Pseudo-reward Greedy (LinPRGreedy) algorithm.

Recall our discussion in Section 2 that LinGreedy is faster in action selection than LinUCB. The main reason is in calculating the uncertainty terms $c_{t,a}$, which takes $\mathcal{O}(d^2)$

Simulation	LinPRUCB	LinUCB	LinPRGreedy	LinGreedy
$d = 10, K = 10$	**0.460 ± 0.010**	**0.461 ± 0.017**	0.454 ± 0.033	0.150 ± 0.016
$d = 10, K = 30$	0.558 ± 0.005	**0.563 ± 0.007**	0.504 ± 0.011	0.155 ± 0.006
$d = 30, K = 10$	**0.270 ± 0.008**	0.268 ± 0.008	**0.271 ± 0.016**	0.074 ± 0.010
$d = 30, K = 30$	**0.297 ± 0.003**	**0.297 ± 0.005**	0.255 ± 0.014	0.091 ± 0.009

Table 3: Comparisons of average cumulative reward on artificial data sets.

Algorithm	CTR (tuning)	CTR (testing)	selection time (seconds)	updating time (seconds)
LinPRUCB	0.038	**0.070**	9.1	3.6
LinUCB	**0.039**	0.063	9.4	**0.7**
LinPRGreedy	0.036	0.068	**4.8**	3.4
Greedy	0.028	0.037	3.8	0.4

Table 4: Experimental results on data set R6A.

time even when the inverse of all $\mathbf{Q}_{t-1,a}$ have been cached during model updating. Nevertheless, LinUCB heavily relies on the uncertainty terms in selecting proper actions, and cannot afford dropping the terms.

Our proposed LinPRUCB, however, may not suffer that much from dropping the terms. In particular, because of our reuse of the upper confidence bound in (3), the term $\beta\sqrt{\mathbf{x}_t^T \hat{\mathbf{Q}}_{t,a}^{-1} \mathbf{x}_t}$ in (3) can readily carry out the job of exploration, and thus $\alpha \hat{c}_{t,a}$ in (6) may not be heavily needed. Thus, LinPRUCB provides a flexibility to move the computation loads in the action selection step to the model updating step, which matches the need of practical applications.

The experimental results on artificial data are summarized in Table 3, where the artificial data sets are generated in the same way as Figure 3. We can see that although LinPRUCB and LinUCB reach higher cumulative reward, the performance of LinPRGreedy is close to that of LinPRUCB and LinUCB when $K = 10$. The results justify our argument that LinPRUCB may not heavily rely on $\alpha \hat{c}_{t,a}$ in (6) in some cases, and thus dropping the terms does not affect the performance much. There is a larger gap between LinPRGreedy and LinUCB when $K = 30$, though.

The experimental results for for R6A and R6B are summarized in Table 4 and Table 5. The tables are the extensions of Table 1 and Table 2, respectively. Once again, we see that the performance of LinPRGreedy is competitive with LinPRUCB and LinUCB. Furthermore, note that the major advantage of LinPRGreedy is its selection time. In each table, the selection time of LinPRGreedy is much smaller than that of LinPRUCB and LinUCB, and the gap grows larger for larger dimension d. The small selection time and competitive performance makes LinPRGreedy a favorable choice for practical applications.

Algorithm	CTR (tuning)	CTR (testing)	selection time	updating time
LinPRUCB	**0.046**	0.054	1082	5003
LinUCB	**0.046**	0.053	1166	**248**
LinPRGreedy	0.045	**0.056**	**24**	4899
LinGreedy	0.039	0.059	25	201

Table 5: Experimental results on R6B.

6. Conclusion

We propose a novel contextual bandit algorithm LinPRUCB, which combines the idea of using pseudo-rewards that over-estimate the true rewards, applying a forgetting mechanism of earlier pseudo-rewards, and taking the upper confidence bound in action selection. We prove a matching regret bound of LinPRUCB to the state-of-the-art algorithm LinUCB, and discuss how the bound echoes the promising performance of LinPRUCB in gathering rewards faster in the earlier iterations. Empirical results on artificial and real-world data sets demonstrate that LinPRUCB compares favorably to LinUCB in the early iterations, and is competitive to LinUCB in the long run. Furthermore, we design a variant of LinPRUCB called LinPRGreedy, which shares similar performance benefits to LinPRUCB while enjoying much faster action selection.

Acknowledgments

The paper originates from part of the first author's M.S. thesis (Chou, 2012) and the second author's Ph.D. thesis (Chiang, 2014). The authors thank the anonymous reviewers and the oral committee members of the two leading authors for their valuable comments. The completion of the paper was partially supported by the government of Taiwan via NSC 101-2628-E-002-029-MY2 and 100-2221-E-001-008-MY3.

References

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

Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256, 2002a.

Peter Auer, Nicolo Cesa-bianchi, Yoav Freund, and Robert Schapire. The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing*, 32:2002, 2002b.

Shang-Tse Chen, Hsuan-Tien Lin, and Chi-Jen Lu. Boosting with online binary learners for the multiclass bandit problem. In *ICML*, 2014.

Chao-Kai Chiang. *Toward realistic online learning*. PhD thesis, National Taiwan University, 2014.

Ku-Chun Chou. Pseudo-reward algorithms for linear contextual bandit problems. Master's thesis, National Taiwan University, 2012.

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

Sham M. Kakade, Shai Shalev-Shwartz, and Ambuj Tewari. Efficient bandit algorithms for online multiclass prediction. In *ICML*, 2008.

T. L. Lai and Herbert Robbins. Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6(1):4–22, 1985.

John Langford and Tong Zhang. The epoch-greedy algorithm for multi-armed bandits with side information. In *NIPS*, 2007.

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

Lihong Li, Wei Chu, John Langford, and Xuanhui Wang. Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms. In *WSDM*, 2011.

Sandeep Pandey, Deepak Agarwal, Deepayan Chakrabarti, and Vanja Josifovski. Bandits for taxonomies: A model-based approach. In *SDM*, 2007.

Richard Sutton and Andrew Barto. *Reinforcement Learning: An Introduction.* MIT Press, 1998. ISBN 0262193981.

Chih-Chun Wang, Sanjeev R. Kulkarni, and H. Vincent Poor. Bandit problems with side observations. *IEEE Transactions Automatic Control*, 50(3):338–355, 2005.

Appendix A. Proof of Theorem 3

The descriptions of BaseLinPRUCB and SupLinPRUCB are given in Algorithm 2 and Algorithm 3 respectively. To bound the regret of our SupLinPRUCB, we follow the analysis in (Auer, 2002; Chu et al., 2011) and replace several critical steps that are related to our algorithm.

Algorithm 2 BaseLinPRUCB

```
1: Inputs: $\Psi_t \subseteq [t-1]$; $\alpha, \beta, \eta, \lambda > 0$
2: Initialize $\mathbf{w}_a := \mathbf{0}_d$, $\hat{\mathbf{Q}}_a := \lambda \mathbf{I}_d$, $\mathbf{V}_a := \hat{\mathbf{V}}_a := \mathbf{0}_{d\times d}$, $\hat{\mathbf{z}}_a := \mathbf{z}_a := \mathbf{0}_d$, for every $a \in [K]$
3: Observe $\mathbf{x}_t$
4: for $a \in [K]$ do
5:   for $\tau \in \Psi_t$ do
6:     if $a_\tau = a$ then
7:       $\mathbf{V}_a := \mathbf{V}_a + \mathbf{x}_\tau \mathbf{x}_\tau^\top$
8:       $\mathbf{z}_a := \mathbf{z}_a + \mathbf{x}_\tau r_{\tau,a}$
9:     else
10:      $p_{\tau,a} := \min\left(\mathbf{w}_a^\top \mathbf{x}_\tau + \beta\sqrt{\mathbf{x}_\tau^\top \hat{\mathbf{Q}}_a^{-1} \mathbf{x}_\tau}, 1\right)$
11:      $\hat{\mathbf{V}}_a := \eta \hat{\mathbf{V}}_a + \mathbf{x}_\tau \mathbf{x}_\tau^\top$
12:      $\hat{\mathbf{z}}_a := \eta \hat{\mathbf{z}}_a + \mathbf{x}_\tau p_{\tau,a}$
13:    end if
14:    $\hat{\mathbf{Q}}_a := \lambda \mathbf{I}_d + \mathbf{V}_a + \hat{\mathbf{V}}_a$
15:    $\mathbf{w}_a := \hat{\mathbf{Q}}_a^{-1}(\mathbf{z}_a + \hat{\mathbf{z}}_a)$
16:  end for
17:  $\text{width}_{t,a} := (1+\alpha+\rho)\sqrt{\mathbf{x}_t^\top \hat{\mathbf{Q}}_a^{-1} \mathbf{x}_t}$
18:  $\text{ucb}_{t,a} := \mathbf{w}_a^\top \mathbf{x}_t + \text{width}_{t,a}$
19: end for
```

Let $\Psi^0 = [T] \setminus \bigcup_{s\in[S]} \Psi_{T+1}^{(s)}$, which contains those indices not in any of those $\Psi_{T+1}^{(s)}$. Let $\Psi_{t,a}^{(s)}$ denote the subset of $\Psi_t^{(s)}$ that collects all $\tau \in \Psi_{t,a}^{(s)}$ such that $a_\tau = a$; thus, $\Psi_t^{(s)} = \bigcup_{a\in[K]} \Psi_{t,a}^{(s)}$. By definition, the expected regret is $\sum_{t\in[T]}(\mathbb{E}[r_{t,a_t^*}] - \mathbb{E}[r_{t,a_t}])$, which can be regrouped as

$$\sum_{t\in\Psi^0} \left(\mathbb{E}[r_{t,a_t^*}] - \mathbb{E}[r_{t,a_t}]\right) + \sum_{s\in[S]} \sum_{a\in[K]} \sum_{t\in\Psi_{T+1,a}^{(s)}} \left(\mathbb{E}[r_{t,a_t^*}] - \mathbb{E}[r_{t,a}]\right). \tag{8}$$

Using Lemma 1 and Lemma 2 stated in Section 3, we can establish a lemma similar to Lemma 15 by Auer (2002), which implies that with probability $1-\delta S$, the bound in (8) is at most

$$2\sqrt{T} + \sum_{s\in[S]} \sum_{a\in[K]} 2^{3-s} \left|\Psi_{T+1,a}^{(s)}\right|. \tag{9}$$

To bound the second term in (9), we need the following.

Algorithm 3 SupLinPRUCB

```
1: Initialize $S := \ln T$ and $\Psi_1^{(s)} := \emptyset$ for every $s \in [S]$
2: for $t = 1$ to $T$ do
3:     $s := 1$ and $\hat{A}^{(1)} := [K]$
4:     repeat
5:         Call BaseLinPRUCB with input $\Psi_t^{(s)}$ to compute $\text{width}_{t,a}^{(s)}$ and $\text{ucb}_{t,a}^{(s)}$ for every
           $a \in \hat{A}^{(s)}$.
6:         if $\text{width}_{t,a}^{(s)} > 2^{-s}$ for some $a \in \hat{A}^{(s)}$ then
7:             Select action $a_t := a$ and update:
               $\Psi_{t+1}^{(s)} := \Psi_t^{(s)} \cup \{t\}$, and
               $\Psi_{t+1}^{(s')} := \Psi_t^{(s')}$ for every $s' \neq s$.
8:         else if $\text{width}_{t,a}^{(s)} \leq 1/\sqrt{T}$ for every $a \in \hat{A}^{(s)}$ then
9:             Select action $a_t := \arg\max_{a \in \hat{A}^{(s)}} \text{ucb}_{t,a}^{(s)}$, and let $\Psi_{t+1}^{(s)} := \Psi_t^{(s)}$ for every $s \in [S]$.
10:        else
11:            Let $\hat{A}^{(s+1)} := \{a \in \hat{A}^{(s)} \mid \text{ucb}_{t,a}^{(s)} \geq u^{(s)} - 2^{1-s}\}$, where $u^{(s)} = \max_{a' \in \hat{A}^{(s)}} \text{ucb}_{t,a'}^{(s)}$.
               Let $s := s + 1$.
12:        end if
13:    until an action $a_t$ is selected
14: end for
```

Lemma 4 *For any $s \in [S]$ and $a \in [K]$,*

$$2^{-s} \left|\Psi_{T+1,a}^{(s)}\right| \leq 5(1+\alpha+\rho)\sqrt{d \left|\Psi_{T+1,a}^{(s)}\right| \ln \left|\Psi_{T+1,a}^{(s)}\right|}.$$

Proof From Steps 6 and 7 of SupLinPRUCB, we know that for any s and a, $\text{width}_{t,a}^{(s)} > 2^{-s}$ for any $t \in \Psi_{T+1,a}^{(s)}$, and therefore

$$2^{-s} \left|\Psi_{T+1,a}^{(s)}\right| \leq \sum_{t \in \Psi_{T+1,a}^{(s)}} \text{width}_{t,a}^{(s)}.$$

From Step 17 of BaseLinPRUCB, we know that

$$\text{width}_{t,a}^{(s)} = (1+\alpha+\rho)\sqrt{\mathbf{x}_t^\top \hat{\mathbf{Q}}_a^{-1} \mathbf{x}_t}$$

for some matrix $\hat{\mathbf{Q}}_a = \mathbf{I}_d + \mathbf{V}_a + \hat{\mathbf{V}}_a$. The matrices $\hat{\mathbf{Q}}_a$, $\mathbf{V}_a$, and $\hat{\mathbf{V}}_a$ actually depend on $\Psi_{t,a}^{(s)}$, and thus let us denote them more appropriately by $\hat{\mathbf{Q}}_{t,a}^{(s)}$, $\mathbf{V}_{t,a}^{(s)}$, and $\hat{\mathbf{V}}_{t,a}^{(s)}$, respectively, where $\mathbf{V}_{t,a}^{(s)}$ is the sum of $\mathbf{x}_\tau \mathbf{x}_\tau^\top$, over $\tau \in \Psi_{t,a}^{(s)}$, and $\hat{\mathbf{V}}_{t,a}^{(s)}$ is the sum of $\mathbf{x}_\tau \mathbf{x}_\tau^\top$ scaled by some power of η, over $\tau \in \Psi_t^{(s)} \setminus \Psi_{t,a}^{(s)}$. Using the same proof for Lemma 3 in (Chu et al., 2011), one can show that for the matrices $\mathbf{Q}_{t,a}^{(s)} = \mathbf{I}_d + \mathbf{V}_{t,a}^{(s)}$,

$$\sum_{t \in \Psi_{T+1,a}^{(s)}} \sqrt{\mathbf{x}_t^\top \left(\mathbf{Q}_{t,a}^{(s)}\right)^{-1} \mathbf{x}_t} \leq 5\sqrt{d \left|\Psi_{T+1,a}^{(s)}\right| \ln \left|\Psi_{T+1,a}^{(s)}\right|}.$$

Note that $\hat{\mathbf{Q}}_{t,a}^{(s)} \succeq \mathbf{Q}_{t,a}^{(s)}$ and $(\hat{\mathbf{Q}}_{t,a}^{(s)})^{-1} \preceq (\mathbf{Q}_{t,a}^{(s)})^{-1}$, which implies that $\mathbf{x}^\top(\hat{\mathbf{Q}}_{t,a}^{(s)})^{-1}\mathbf{x} \leq \mathbf{x}^\top(\mathbf{Q}_{t,a}^{(s)})^{-1}\mathbf{x}$ for any $\mathbf{x} \in \mathbb{R}^d$. By combining all the bounds together, we have the lemma. ■

Using Lemma 4, the second term in (9) can be upper bounded by

$$\begin{aligned}
&40 \sum_{s\in[S]} \sum_{a\in[K]} (1+\alpha+\rho)\sqrt{d\left|\Psi_{T+1,a}^{(s)}\right| \ln \left|\Psi_{T+1,a}^{(s)}\right|} \\
&\leq \quad 40(1+\alpha+\rho)\sqrt{d\ln T} \sum_{s\in[S]} \sum_{a\in[K]} \sqrt{\left|\Psi_{T+1,a}^{(s)}\right|} \\
&\leq \quad 40(1+\alpha+\rho)\sqrt{d\ln T}\sqrt{SKT},
\end{aligned}$$

by Jensen's inequality. The rest of the proof is almost identical to that by Auer (2002). That is, by replacing δ with $\delta/(S+1)$, substituting $\alpha = \mathcal{O}(\sqrt{\ln(TK/\delta)})$ and $\rho = \mathcal{O}(\beta) = \mathcal{O}(\alpha)$, and then applying Azuma's inequality, we obtain our Theorem 3.