Non-stochastic Low Rank Bandit

Author names withheld

Abstract

We study the problem of learning the maximum entry of a low-rank non-negative matrix, from sequential observations. In this setting, the learner chooses tuples of rows and columns at every round and observes the corresponding submatrix. The main challenge in this setting is that the learner does not observe the individual latent values of rows and columns as its feedback. Diverging from previous works, we assume that the preference matrix is non-stochastic and hence our setting is more general in nature. Existing methods for solving similar problems rely on UCB-type algorithms based on constructing conservative confidence intervals with the strong assumption that underlying distributions are stochastic. We depart from this standard approach and consider the case when the best row and column pair can be learned jointly with the help of two separate bandit algorithms working individually on rows and columns. We propose a simple and computationally efficient algorithm that implements this procedure, which we call Low Rank Bandit, and prove a sub-linear bound on its n-step regret in the rank-1 special case. We evaluate the algorithm empirically on several synthetic and real-world datasets. In all experiments, we outperform existing state-of-the-art algorithms.

A: Should we say 'existing state-of-the-art algorithms adapted to this setting'?

1 Introduction

In this work, we study the problem of learning the maximum entry of a low-rank matrix from sequential observations. The low-rank structure is observed in many real-world applications and is a standard assumption in recommender systems [Koren *et al.*, 2009; Ricci, 2011]. Our learning model is motivated by a real-world scenario, where a marketer wants to advertise a product and has K users and L marketing channels. The marketer can choose pairs of users and marketing channels to promote its product. Let the users and marketing channels be the rows and columns of a low-rank matrix. Then the goal of the learner is to maximize its click-through rate (CTR) by finding the maximum entry of this matrix.

A: The application mentioned here asks for assigning each user to a marketing channel. After reading the above para, the reader might wonder 'Why just the maximum entry?'

We formalize our learning problem as the following online learning problem. At round t, the learning agent chooses d-tuples of rows and columns where d is the rank of a non-negative and low-rank matrix M. We use the terminology row/user and column/item interchangeably, keeping in sync with our proposed application area.

A: Proposed application area doesn't say item but marketing channel.

This matrix M is formed by the outer product of row and column latent vectors over d topics. Hence, M encodes the reward of choosing the row-column pairs and is termed as the reward matrix. Note that the learner does not observe the row or column latent vectors but just their product. The reward matrix M is low-rank at each round t, can vary substantially over time, and does not have to be stochastic.

A: What about the condition that the maximum entry of M stays within the same dxd block?

The goal of our learning agent is to minimize the cumulative regret with respect to the best solution in hindsight by finding the maximum entry in M as quickly as possible.

We make four major contributions. First, we formulate our online learning problem as a non-stochastic bandit problem on a class of non-negative low-rank matrices. We identify a family of non-negative low-rank matrices where our problem can be solved statistically efficiently, without actually observing the latent values of individual rows and columns. Second, we propose a computationally efficient algorithm that implements this idea, which we call a low rank bandit algorithm (LRB). The algorithm has two components, column learning and row learning, which learn the pair of optimal columns and rows, respectively. Since we are in the non-stochastic setting, we use a variation of Exp3 [Auer *et al.*, 2002b] as our row and column learner. Note that we do not construct any confidence interval or eliminate rows and columns like the existing works. In fact, we use the well-known fact that exponentially-weighted algorithms, like Exp3, are robust and fast learners to design our algorithm. Third, we analyze LRB and up to problem-specific factors, we prove a $O\left(\frac{(\sqrt{L}+\sqrt{K})\sqrt{n}}{\alpha}\right)$ upper bound on its n-step regret in the special case when rank is one and for some $\alpha > 0$. The regret of a naive solution is $O(\sqrt{KLn})$ and is much worse than that of LRB when $K \approx L$. Finally, we evaluate LRB empirically on several synthetic and real-world problems. Perhaps surprisingly, LRB performs well even when our modeling assumptions are violated.

The paper is organized as follows. In Section 2, we introduce the rank-1 setting. In Sections 3 and 4, we propose our rank-1 algorithm and derive a sublinear upper bound on its regret. In Sections 5 and 6, we introduce the rank-d setting and propose an algorithm for it. In Section 7, we evaluate our algorithms empirically. We discuss related works in Section 8 and conclude in Section 9.

2 Rank-1 Background and Setting

Notation: We denote $[n] = \{1, \dots, n\}$ as the set of the first n positive integers. Let M denote any arbitrary matrix of size $m \times n$ matrix. Let the rank of the matrix be denoted by d. We denote by A^B the set of all vectors whose entries take values from set A and are indexed by set B, where A and B can be any arbitrary sets. For any d and $I \in [m]^d$, $M(I, :)$ denotes a $d \times n$ submatrix of M whose i-th row is $M(I(i), :)$. Similarly, for any d and $J \in [n]^d$, $M(:, J)$ denotes a $m \times d$ submatrix of M whose j-th column is $M(:, J(j))$. Finally, we denote Π_d as the set of all d-permutations. For an element $\pi \in \Pi_d$ and d-dimensional vector v, we denote by $\pi(v)$ the permutation of the entries of v according to π.

Rank-1 Setting: We study the online learning problem of finding the maximum entry of a family of non-stochastic, low-rank and non-negative matrices which we call as the *non-stochastic low-rank bandit problem*. We first analyze the simple rank-1 setup and propose our solution for this setting. Many of the key aspects of our design principle are captured in this rank-1 setting. Let $U_t \in [0,1]^{K \times 1}$ be the latent row vector over a single topic and $V_t \in [0,1]^{L \times 1}$ be the latent column vector over a single topic. The *reward matrix* $M_t = U_t V_t^\mathsf{T}$ is non-negative, non-stochastic, and rank one. We assume that

$$i_t^* = \arg\max_{i \in [K]} U_t(i), \qquad j_t^* = \arg\max_{j \in [L]} V_t(j). \tag{1}$$

do not change with $t \in [n]$, i.e., $i_t^* = i^*, j_t^* = j^*$ for some fixed $i^* \in [K]$ and $j^* \in [L]$. This assumption makes sure that the row and column latent vectors can change over time t, but the indices of best row i^* and column j^* do not change. At every round t, the learner chooses one pair of row and column indices i_t and j_t, respectively, and observes their product $U_t(i_t)V_t(i_t)$ as its feedback.

Regret Definition (Rank-1): The goal of the learner is to minimize the expected n-step regret with respect to the optimal solution in the hindsight as follows,

$$R(n) = \sum_{t=1}^{n} \mathbb{E}\left[M_t(i^*, j^*) - M_t(i_t, j_t)\right] , \tag{2}$$

where the expectation is over any random choice of rows and columns.

3 Rank-1 Algorithm

We present the LRB for the rank 1 setting in Algorithm 1. The LRB consist of two key components, a row learning algorithm and a column learning algorithm. At every round t, the row algorithm suggests the row $i_t \in [K]$ and the column algorithm suggests the column $j_t \in [L]$. Note that in this non-stochastic scenario we use instances of Exp3 as the row and column learning algorithm. The row Exp3 has K arms and the column Exp3 has L arms. The main idea is to use the row Exp3 to learn the best row on average while the column Exp3 learns the best column on average. The learner then observes the reward $M_t(i_t, j_t)$ and updates the row and column Exp3 simultaneously. A key insight to this design is that when both the row and column learners are run simultaneously, they learn the most rewarding row and column on average and converge on the maximum entry of the matrix M. From the definition of U_t, for any sequence of n columns, the maximum value is in row i^* (see eq (1)). This means that the row algorithm learns irrespective of what the column algorithm does. From the definition of V_t, for any sequence of n rows, the maximum value is in column j^* (see eq (1)). This means that the column algorithm learns irrespective of what the row algorithm does.

Algorithm 1 Low Rank Bandit (LRB) (Rank-1)

```
1: Input: Time horizon $n$                          ▷ Initialization
2: Initialize RowAlg
3: Initialize ColAlg
4: for $t = 1, \dots, n$ do                          ▷ Generate response
5:     Row $i_t$ suggested by RowAlg
6:     Column $j_t$ suggested by ColAlg
7:     Observe $M_t(i_t, j_t)$                       ▷ Update statistics
8:     Update arm $i_t$ of RowAlg with reward $M_t(i_t, j_t)$.
9:     Update arm $j_t$ of ColAlg with reward $M_t(i_t, j_t)$.
```

4 Analysis of Rank-1 Setting

In this section we analyze the rank-1 LRB and show its regret for a horizon n.

Theorem 1. *Let* ColAlg *and* RowAlg *in* LRB *be* Exp3 *algorithm, respectively. Then the expected n-step regret of* LRB *is bounded as*

$$R(n) = O\left(\frac{\left(\sqrt{L} + \sqrt{K}\right)\sqrt{n}}{\alpha}\right)$$

for any $\alpha > 0$ such that $U_t(i, 1) \geq \alpha$ and $V_t(j, 1) \geq \alpha$ for all $i \in [K]$, $j \in [L]$, and $t \in [n]$.

Proof. Let, $(U_t V_t^\mathsf{T})_{t=1}^n$ be a sequence of n non-negative rank-1 matrices such that $U_t \in [0,1]^{K \times 1}, V_t \in [0,1]^{L \times 1}$, and the highest entry, without loss of generality, is $U_t(1)V_t(1)$. Let, $((i_t, j_t))_{t=1}^n$ be a sequence of n row-column pairs chosen by a learning agent. Then the expected n-step regret of the agent is,

$$R(n) = \sum_{t=1}^{n} \mathbb{E}\left[U_t(1)V_t(1) - U_t(i_t)V_t(j_t)\right]$$

where the expectation is over the randomness of the agent. For any two vectors U, V and indices i, j we have,

$$
\begin{aligned}
&2(U(1)V(1) - U(i)V(j)) \\
&= 2U(1)V(1) - U(i)V(1) - U(1)V(j) + \\
&U(i)V(1) + U(1)V(j) - 2U(i)V(j) \\
&= U(1)(V(1) - V(j)) + V(1)(U(1) - U(i)) + \\
&U(i)(V(1) - V(j)) + V(j)(U(1) - U(i)) \\
&= (U(1)+U(i))(V(1)-V(j)) + (V(1)+V(j))(U(1)-U(i))
\end{aligned}
$$

Therefore, the expected n-step regret can be decomposed as,

$$
\begin{aligned}
R(n) = &\sum_{t=1}^{n} \mathbb{E}[(V_t(1) + V_t(j_t))(U_t(1) - U_t(i_t))] \\
&+ \sum_{t=1}^{n} \mathbb{E}[(U_t(1)+U_t(i_t))(V_t(1) - V_t(j_t))]
\end{aligned}
$$

Now suppose that all entries of U_t and V_t for all $t = 1, 2, \ldots, n$ are bounded from below by some $\alpha > 0$. Then we get that,

$$
\begin{aligned}
R(n) = &\sum_{t=1}^{n} \mathbb{E}[(1 + V_t(1)/V_t(j_t))V_t(j_t)(U_t(1) - U_t(i_t))] + \\
&\sum_{t=1}^{n} \mathbb{E}[(1 + U_t(1)/U_t(i_t))U_t(i_t)(V_t(1) - V_t(j_t))] \\
&\leq (1 + \frac{1}{\alpha})\Bigg[\sum_{t=1}^{n} \mathbb{E}[U_t(1)V_t(j_t) - U_t(i_t)V_t(j_t)] + \\
&\sum_{t=1}^{n} \mathbb{E}[U_t(i_t)V_t(1) - U_t(i_t)V_t(j_t)]\Bigg]
\end{aligned}
$$

The ColAlg chooses the column j_t at round t and observes reward $U_t(i_t)V_t(j_t)$. Note that from eq (1) we know that the best row i^* and best column j^* is fixed for all $t \in [n]$. Hence, we can show that the ColAlg using Exp3 will converge on j^* for any sequence of rows. Therefore, the first sum above is bounded by $\sqrt{Ln}$ for any sequence of i_t, and thus also in expectation over the randomness in i_t. Similarly RowAlg using Exp3 chooses the row i_t at round t, and observe reward is $U_t(i_t)V_t(j_t)$. Following eq (1) we can again show that RowAlg will converge on i^* for any sequence of columns. Therefore, the second sum above is bounded by $\sqrt{Kn}$ for any sequence of j_t, and thus also in expectation over the randomness in j_t. Therefore we get the final regret as,

$$
R(n) = O\left(\frac{(\sqrt{L} + \sqrt{K})\sqrt{n}}{\alpha}\right)
$$

□

Discussion: The main idea in Theorem 1 is to decompose the regret of LRB into two parts, where ColAlg does not suggest j^* and the RowAlg does not suggest i^*. The first part is analyzed as follows. ColAlg has a sub-linear regret, as we use Exp3 as the ColAlg. Thus, the expected regret that ColAlg suggests sub-optimal columns for *any sequence* of rows is bounded. This regret scales as $O(\sqrt{Ln})$ based on the analysis of Auer *et al.* [2002b]. Similarly, we analyze the regret for the RowAlg as it also uses the exponentially weighted algorithm Exp3. The RowAlg suffers a regret of $O(\sqrt{Kn})$ for suggesting sub-optimal rows for *any sequence* of columns. Hence, combining both the parts we get the regret of order $O\left(\frac{(\sqrt{L}+\sqrt{K})\sqrt{n}}{\alpha}\right)$ in Theorem 1. We use non-stochastic algorithm Exp3 for ColAlg and RowAlg because our environment is non-stationary. Finally, we can consider any matrix M of dimension $K \times L$ as a K-bandit problem with each bandit having L arms and the optimal columns are learned separately for each bandit. Such a trivial setting gives rise to the regret bound of order $O(\sqrt{KLn})$. The regret bound for our algorithm is much better than this for certain parameter ranges of the problem.

5 Rank-d Background and Setting

In this section, we study the online learning problem of finding the maximum entry of a family of non-stochastic, low-rank and non-negative matrices for the general rank-d setting.

Hott-topics Assumption: We focus on a family of low-rank matrices, which are known as hott topics [Recht *et al.*, 2012]. We define a *hott-topics matrix* of rank d as $M = UV^\mathsf{T}$, where U is a $K \times d$ non-negative matrix and V is a $L \times d$ non-negative matrix that both have the hott-topics structure. We say a matrix U has hott-topics structure if there exists d rows U whose indices are‘ I^* such that each row in U can be represented as a convex combination of these rows and the zero vector. Hence, for $A = \{a \in [0,1]^{d\times 1} : \|a\|_1 \leq 1\}$ each row of U can be expressed as,

$$
\forall i \in [K]\ \exists \alpha \in A : U(I^*, :)\alpha = U(i, :)\,, \tag{3}
$$

Similarly, we assume that there exist d rows in V whose indices are J^* such that each row of V can be expressed as a convex combination of these rows and the zero vector. $M(i, j)$ is the reward of row i and column j. The rank d of M is the number of latent topics. The matrix U are latent vectors of K rows over d topics, where $U(i, :)$ are the coordinates of row $i \in [K]$. The matrix V are latent vectors of L columns in the space of d topics, where $V(j, :)$ are the coordinates of column $j \in [L]$. Without loss of generality, we assume that $U \in [0,1]^{K\times d}$ and $V \in [0,1]^{L\times d}$. We assume that the coordinates are points in a simplex, that is $\|U(i, :)\|_1 \leq 1$ for all $i \in [K]$ and $\|V(j, :)\|_1 \leq 1$ for all $j \in [L]$. Note that our assumptions imply that $M(i, j) \geq 0$ for any $i \in [K]$ and $j \in [L]$.

Rank-d Setting: Let for all rounds $t \in [n]$, $U_t \in [0,1]^{K\times d}$, $V_t \in [0,1]^{L\times d}$ and $M_t = U_t V_t^\mathsf{T}$. Similar to (1) we assume that,

$$
I^* = \operatorname*{arg\,max}_{I\in[K]^d, t\in[n]} U_t(I, :), \quad J^* = \operatorname*{arg\,max}_{J\in[L]^d, t\in[n]} V_t(J, :) \tag{4}
$$

A: The above makes no sense. $U_t(I, :)$ is a matrix and not scalar. What do you mean by arg-max??

We also assume that for all rounds $t \in [n]$, the rows of U_t can be written as a convex combination of the rows of $U_t(I^*, :$

) and the rows of V_t can be written as a convex combination of the rows of $V_t(J^*, :)$ as defined in (3) and (**??**).

A: You have already defined what hott-topics is. Why are you repeating it here?

Hence, assumption (4) on reward matrices M_t ensures that for any row $i \in [K]$, any column $j \in [L]$ and for all round t,

$$\underset{(i,j)\in[K]\times[L],t\in[n]}{\arg\max} \quad M_t(i,j) \in (I^*, J^*), \tag{5}$$

where I^* and J^* is defined in (3) and (**??**). The hott-topics assumption makes it possible to learn the maximum entry of M_t statistically efficiently at any round $t \in [n]$ because the maximum entry $M_t(i^*, j^*)$ will be in $M_t(I^*, J^*)$. Note that even though different entries of U_t and V_t can attain high rewards at different times but the I^* and J^* remain fixed for all round $t \in [n]$ (eq 4). At every round t, the learner chooses d-tuples of rows and columns from M_t denoted by $(I_t, J_t) \in \Pi_d([K]) \times \Pi_d([L])$. It then observes all the values from the matrix $M_t(I_t, J_t)$ for all $i_t \in I_t$ and $j_t \in J_t$. The *reward* for the agent for choosing arms (I_t, J_t) at round t is denoted by $M_t(i^*(I_t, J_t), j^*(I_t, J_t))$ such that,

$$(i^*(I,J), j^*(I,J)) = \underset{(i,j)\in(I\times J)}{\arg\max} \; M_t(i,j) \tag{6}$$

Regret Definition (Rank-d): Now we are ready to define our notion of optimality and regret for the general rank-d scenario. Our goal is to minimize the expected n-step regret,

$$R(n) = \sum_{t=1}^{n} \mathbb{E}\left[M_t(i_t^*, j_t^*) - M_t(i^*(I,J), j^*(I,J))\right] \,, \tag{7}$$

where the expectation is with respect to both randomly choosing rows (I_t) and columns (J_t) by the learning algorithm and potential randomness in the environment.

6 Rank-d Algorithm

Now, we propose the general *Low Rank Bandit (*`LRB`*)* algorithm for solving the family of non-stochastic, non-negative and low-rank matrices of rank d. Again, the goal is to identify the maximum entry of the matrix by quickly identifying the d-best rows or columns. The pseudocode of rank-d `LRB` is in Algorithm 2. `LRB` has two main components, column learning and row learning algorithm.

At every round t, the row learning algorithm recommends a list of d rows. But we exploit an additional structure (eq (5)) in our problem to learn the optimal rows I^*. The row learning algorithm is d instances of multi-armed bandit algorithms, which we denote by `RowAlg`(k) for algorithm $k \in [d]$. `RowAlg`(1) learns the most rewarding row on average, `RowAlg`(2) learns the second most rewarding row on average conditioned on the first learned row, and so on.

A: These are hand-wavy statements at best, and doesn't make much sense mathematically. Don't say RowAlg(2) learns something without knowing that it does. Same applies below.

Similarly, the column learning algorithm recommends a list of d columns by exploiting the same structure in our rewards. Again the goal of the column learning algorithm `ColAlg` is to learn the optimal set of columns J^*. Hence, `ColAlg`$(1), \dots,$ `ColAlg`(d) learns the most rewarding columns in J^* on average. Note that this sequence of learning the rows or columns first does not matter because the *hott-topics* structure is defined on both U and V matrix (eq (3), (**??**)) generating $M_t = U_t V_t^\mathsf{T}$, and so we will be learning the d-best rows or columns in average. Another way of looking at this is to first realize that if we fix the column selection strategy, which is simply some distribution over d-tuples of chosen columns then for any such distribution, the d hott-topic rows are the optimal solution to the row selection problem. By symmetry, the same is true for the column selection problem. If we run both in parallel, and the distributions in the other dimensions do not change too fast (this is true by our design in eq (5)), then $i_1, \dots, i_d$ and $j_1, \dots, j_d$ would slowly converge to the d hott-topic rows and columns.

A: I find these statements to be ambiguously between rigorous mathematical statements and informal intuition. You have to make it clear that it is the latter and choose words appropriately.

Finally, `LRB` observes the individual rewards of $M_t(i,j)$ for all $(i,j) \in (I_t, J_t)$. Then we update both column and row learning algorithms. The reward of the arm in `RowAlg`(k), which selects the k-th row in I_t, is updated as follows. If the k-th arm was not one of the previously suggested rows its reward is updated d times such that, $\max_{k\le k_1} M_t(i_k, j_{k_2}) - \max_{k<k_1} M_t(i_k, j_{k_2})$ for all $k_1, k_2 \in [d]$. By the choice of our design and previous argument a similar update is performed on the k-th column learning algorithm `ColAlg` such that its reward is also updated d times with rewards, $\max_{k\le k_2} M_t(i_{k_1}, j_k) - \max_{k<k_2} M_t(i_{k_1}, j_k)$ for all $k_1, k_2 \in [d]$.

A: The above has lot of mistakes. I think you are mixing up k, k_1 and k_2. Please rewrite the above 3 sentences. Fix the issues pointed out in the algorithm first.

Otherwise, if any of the row or column has been previously selected by the corresponding `RowAlg` or `ColAlg` algorithm then we update it with reward 0. This type of update makes sure that the `RowAlg`(k) or `ColAlg`(k) learns the k-th best row or column conditioned on the selection of previous $1, \dots, k-1$ `RowAlg` or `ColAlg` respectively. If there is conflict of suggestion for either rows or columns, only one `RowAlg` or `ColAlg` is updated so that each can focus on a distinct row or column respectively. A similar type of update is also done in a much simpler setting of Ranked Bandit Algorithm (RBA) in Radlinski *et al.* [2008].

6.1 Practical Considerations

Motivated by rank-1, we use a non-stochastic algorithm `Exp3` as `ColAlg` and `RowAlg`, respectively. For experimental purposes, stochastic algorithms like `UCB1` or Thompson Sampling can also be used to improve the performance of `LRB`. This has also been explored in Radlinski *et al.* [2008] where `RBA` uses `UCB1` for ranking items. The proposed `LRB` algorithm only updates $(K+L)d$ entries for d instances of `ColAlg` and `RowAlg`, respectively, at every round t. This is in stark contrast to some of the existing matrix completion algorithms, which reconstruct a $K \times L$ matrix [Sen *et al.*, 2016] or decompose tensors [Gopalan *et al.*, 2016].

7 Experiments

In this section, we compare `LRB` to several bandit algorithms in three experiments. The first two experiments are on synthetic problems where all modeling assumptions hold. The third experiment is on a real-world dataset where we evaluate LRB when our modeling assumptions fail. All results are averaged over 10 independent random runs. We test in both rank 1 and rank 2 settings to clearly illustrate the failures of the current rank 1 algorithms and show the efficiency of our proposed method.

7.1 Experiments on Rank-1 Setting

Evaluated Algorithms for Rank-1: We compare against several state-of-the-art rank 1 algorithms. Note all the rank 1 algorithms suggest a single row and column at every round. The UCB1 algorithm from Auer *et al.* [2002a] builds a confidence set at every round t over all the entries of M_t as $c_{i,j}(t) = \sqrt{\frac{2\log t}{N_{i,j}(t)}}$ where $N_{i,j}(t)$ denotes the number of times the (i,j)-th entry has been observed. It suggests the best row-column pair based on the term $\hat{M}_t(i,j) + c_{i,j}(t)$ where $\hat{M}_t(i,j)$ is the empirical mean of all observed rewards of entry (i,j). The UCB1-Elim [Auer and Ortner, 2010] is similar to UCB1 but it eliminates sub-optimal rows and columns based on a similar confidence set $c_{i,j}(t)$ till it finally converges on the best pair of row and column. The algorithm LinUCB was first proposed in Li *et al.* [2010] for the contextual bandit setting. Note that rank-1 bandit generalizes to the stochastic linear bandit setting and can be solved by LinUCB. Similarly, GLM-UCB from Filippi *et al.* [2010] can also be used to solve the rank 1 bandit problem. Finally, we compare against the algorithm $\texttt{Rank1}-\texttt{Elim}$ from Katariya *et al.* [2016] which is an improved version of UCB1-Elim and employs row and column elimination and aggressive exploration to converge on the best row and column pair. For LRB we use the Algorithm 1 from Section 3. We use eq (2) definition to calculate regret in rank-1 setting.

Synthetic Experiment 1 **for Rank-**1**:** This experiment is conducted to test the performance of LRB over a small number of rows and columns and to show how LRB scales with increasing number of rows and columns. Note that in this experiment all our modeling assumptions hold. This simulated testbed consist of two scenarios: (1) 8 rows and 8 columns and (2) 16 rows and 16 columns. In this setting, $U = \{0.7, 0.9\}^{K\times 1}$ and similarly $V = \{0.7, 0.9\}^{L\times 1}$ with only the entry $U(K/2, 1) = V(L/2, 1) = 0.9$. Hence, the matrix $M = UV^\intercal$ is rank 1 and the hott-topics structure is maintained. At every round t, we generate the matrix $M_t = UD_tV^\intercal$ where D_t is a randomly generated value from $[0, 1]$. Note for all round $t \in [n]$, M_t is rank-1 matrix with its maximum value always at $M_t(K/2, L/2)$, and other entries changing arbitrarily but always less than $M_t(K/2, L/2)$. The learner observes the entry $M_t(i,j)$ when it selects the i-th row and j-th column. A similar environment has been discussed as B_{spike} in Katariya *et al.* [2016]. From Figure 1(a) and 1(b) we can clearly see that LRB outperforms all the other algorithms. The regret curve of LRB flattens, indicating that it has learned the best row-column pair. As we scale the number of rows and columns we see that LRB performs even better than other algorithms.

7.2 Experiments on Rank-2 Setting

Evaluated Algorithms for Rank-2**:** We design the rank-2 algorithms by modifying the rank-1 algorithms. Again note that all the rank-2 algorithms suggest two pairs of rows and columns at every round t. For all of the algorithms UCB1, UCB1-Elim, LinUCB, $\texttt{GLM}-\texttt{UCB}$, and $\texttt{Rank1}-\texttt{Elim}$ we modify these algorithms so that they suggest 2 pairs of rows and columns based on their respective confidence interval set $c_{i,j}(t)$. The row and column pair with the highest and the second highest $\hat{M}_t(i,j) + c_{i,j}(t)$ are suggested for each round t and consequently after observing all the entries of $M_t(i,j)$ all of the algorithms update their estimates of $\hat{M}_t(i,j)$ for each $i,j \in [d]$. For LRB we use the Algorithm 2 from Section 6. Note that there are two RowAlg and ColAlg, each running a Exp3 algorithm with the exploration parameters as discussed before. For LRB-rank2 we use the Algorithm 2 from Section 6. We also modify the rank-1 LRB (Algorithm 1) so that the algorithm works in rank-2 setting. After LRB-rank1 has sampled one pair of rows and columns from $[K]$ and $[L]$, it then samples again another choice that does not clash with the first pair and then updates all the pairs with the feedback observed. We use eq (7) definition to calculate regret in rank-2 setting.

Synthetic Experiment 2**:** This experiment is conducted to test the performance of LRB over a large number of rows and columns. This simulated testbed consist of 64 rows, 64 columns, and rank$(M) = 2$. The vectors spanning U and V, generating the row-column preference matrix M, are shown Figure 1(c). The rows and columns are evenly distributed into a $50 : 50$ split such that 50% of rows prefer column 1 and 50% rows prefer column 2. The column hott-topics are $V(1,:) = (1,0)$ and $V(2,:) = (0, 0.6)$ while 50% remaining columns has feature $V(j',:) = (0.45, 0.5)$ and the rest have $V(j,:) = (0.5, 0.45)$. Similarly, we create the row feature matrix U having a $50 : 50$ split such that $U(1,:) = (1,0)$, $U(2,:) = (0, 0.6)$ and the remaining 50% rows having $U(i,:) = (0.5, 0.4)$ and the rest having $U(i',:) = (0.4, 0.5)$. At every timestep t the resulting matrix $M_t = UD_tV^\intercal$ is generated where D_t is a randomly-generated diagonal matrix. From Figure 1(b) we can clearly see that LRB-rank2 outperforms all the other algorithms. It's regret curve flattens, indicating that it has learned the best row-column pair. The key realization is that LRB takes advantage of the hott-topics structure and quickly identifies them. Note that for any rank d setting the best row-column pair must be one of the hott-topics in (I^*, J^*). Also note the failure of LRB-rank1 in this setting which clearly shows why a general rank-d algorithm with our specific type of update is required.

Real-world Experiment 3**:** We conduct the third experiment to test the performance of LRB when our modeling assumptions are violated. We use the Jester dataset [Goldberg *et al.*, 2001] which consist of over 4.1 million continuous ratings of 100 jokes from 73,421 users collected over 5 years. In this dataset there are many users who rated all jokes and we work with these users. We sample ran-

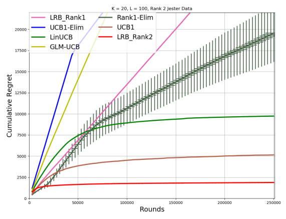
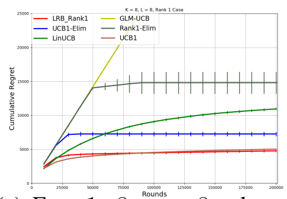
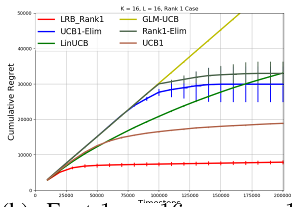


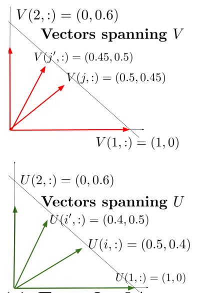
Figure 2: Expt-3: 20 rows, 100 columns, Jester Dataset



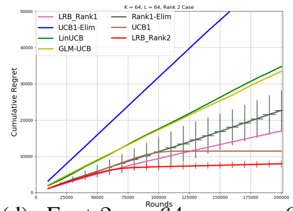
(a) Expt-1: 8 rows, 8 columns, Rank 1 Setting



(b) Expt-1: 16 rows, 16 columns, Rank 1 Setting



(c) Expt-2: 64 rows, 64 columns, row and column vectors



(d) Expt-2: 64 rows, 64 columns, Rank 2 Setting

Figure 1: A comparison of LRB vs state-of-the-art algorithms.

domly 20 users (who have rated all jokes) from this dataset and use singular value decomposition (SVD) to obtain a rank 2 approximation of this user-joke rating matrix M. In the resultant matrix M, most of the rows belong to the two classes preferring jokes 98, and 28, while a very small percentage of users prefer some other jokes. Note that this condition results from the fact that this real-life dataset does not have the hott-topics structure. Furthermore, in this experiment, we assume that the noise is independent Bernoulli over the entries of M and hence this experiment deviates from our modeling assumptions. In Figure 2 we see that LRB-rank2 outperform other algorithms. Finally, LRB-rank1 again fails to perform well in this setting.

8 Related Works

Previous works that have studied this setting have focused either on the rank-1 setting or proposed solution where the underlying distributions are stochastic with additional structures and assumptions.

Rank-1 Setting: The work of Katariya *et al.* [2016] was proposed for a rank-1 bandit model with the assumption that the underlying distributions are stochastic. Similarly, Katariya *et al.* [2017] was proposed for the special case when the underlying distributions are Bernoulli. A simpler setting has also been studied in Maillard and Mannor [2014]. All of these works used different variations of the Upper Confidence Bound (UCB) algorithm Auer *et al.* [2002a], [Auer and Ortner, 2010] algorithm to construct a confidence interval set over row-column pairs to identify and eliminate sub-optimal rows and columns. These naturally results in algorithms that explore conservatively (for the sake of row and column elimination) and cannot work beyond the stochastic distribution assumption. As opposed to these earlier works our method is focused on the more general and natural non-stochastic setting.

Rank-d Setting: The work of Kveton *et al.* [2017] can be viewed as a generalization of rank 1 bandits of Katariya *et al.* [2016] to a higher rank of d. However, this work proposes a phase-based algorithm that calculates the square of the determinant of a $d \times d$ sub-matrix to eliminate sub-optimal rows and columns at the end of phases which is impractical for very large non-negative low-rank matrices. The theoretical guarantees hold for only stochastic distributions. Some other approaches involving non-negative matrix factorization Sen *et al.* [2016] or tensor-based methods [Gopalan *et al.*, 2016] to reconstruct the matrix have also been proposed. These works require strong assumptions on the structure of the matrix such as all the matrices satisfy a weak statistical Restricted Isometric Property (RIP) or calculate third order tensors as in Anandkumar *et al.* [2014]. On the contrary, our simple and statistically efficient algorithm is easily generalizable to rank-d and do not require any sort of costly matrix inversion or reconstruction operations or even row or column eliminations and hence are much easier to implement.

9 Conclusions

In this paper, we studied the problem of finding the highest entry of a non-stochastic, non-negative low-rank matrix. We formulated the above problem as an online-learning problem and proposed the LRB algorithm for this setting. We proved that an instance of algorithm has a regret bound in the special case of rank 1 setting that scales as $O\left(\frac{(\sqrt{L}+\sqrt{K})\sqrt{n}}{\alpha}\right)$ and has the correct order with respect to rows, columns and rank of the row-column preference matrix M. We also evaluated our proposed algorithm on several simulated and real-life datasets and show that it outperforms the existing state-of-the-art algorithms. There are several directions where this work can be extended. Note that we only proved our theoretical results for the rank 1 setting. Proving theoretical guarantees for LRB algorithm will require additional assumptions on the structure of rewards and the matrix M.

References

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.

Peter Auer and Ronald Ortner. Ucb revisited: Improved regret bounds for the stochastic multi-armed bandit problem. *Periodica Mathematica Hungarica*, 61(1-2):55–65, 2010.

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

Sarah Filippi, Olivier Cappe, Aurélien Garivier, and Csaba Szepesvári. Parametric bandits: The generalized linear case. In *Advances in Neural Information Processing Systems*, pages 586–594, 2010.

Ken Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. Eigentaste: A constant time collaborative filtering algorithm. *information retrieval*, 4(2):133–151, 2001.

Aditya Gopalan, Odalric-Ambrym Maillard, and Mohammadi Zaki. Low-rank bandits with latent mixtures. *arXiv preprint arXiv:1609.01508*, 2016.

Sumeet Katariya, Branislav Kveton, Csaba Szepesvari, Claire Vernade, and Zheng Wen. Stochastic rank-1 bandits. *arXiv preprint arXiv:1608.03023*, 2016.

Sumeet Katariya, Branislav Kveton, Csaba Szepesvári, Claire Vernade, and Zheng Wen. Bernoulli rank-1 bandits for click feedback. *arXiv preprint arXiv:1703.06513*, 2017.

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

Branislav Kveton, Csaba Szepesvari, Anup Rao, Zheng Wen, Yasin Abbasi-Yadkori, and S Muthukrishnan. Stochastic low-rank bandits. *arXiv preprint arXiv:1712.04644*, 2017.

Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. A contextual-bandit approach to personalized news article recommendation. *Proceedings of the 19th International Conference on World Wide Web, WWW 2010, Raleigh, North Carolina, USA, April 26-30, 2010*, pages 661–670, 2010.

Odalric-Ambrym Maillard and Shie Mannor. Latent bandits. In *International Conference on Machine Learning*, pages 136–144, 2014.

Filip Radlinski, Robert Kleinberg, and Thorsten Joachims. Learning diverse rankings with multi-armed bandits. In *Proceedings of the 25th international conference on Machine learning*, pages 784–791. ACM, 2008.

Ben Recht, Christopher Re, Joel Tropp, and Victor Bittorf. Factoring nonnegative matrices with linear programs. In *Advances in Neural Information Processing Systems*, pages 1214–1222, 2012.

Francesco Ricci. Liorrokach, and brachashapira." introduction to recommender systems handbook, 2011.

Rajat Sen, Karthikeyan Shanmugam, Murat Kocaoglu, Alexandros G Dimakis, and Sanjay Shakkottai. Contextual bandits with latent confounders: An nmf approach. *arXiv preprint arXiv:1606.00119*, 2016.

Algorithm 2 Low Rank Bandit (LRB) (Rank-d)

```
1: Input: Time horizon $n$, Rank $d$
2: for $k = 1, \dots, d$ do                                   ▷ Initialization
3:     Initialize RowAlg($k$)
4:     Initialize ColAlg($k$)
5: for $t = 1, \dots, n$ do
6:     for $k = 1, \dots, d$ do                             ▷ Generate response
7:         $\hat{i}_k \leftarrow$ Suggested row $i_t$ by RowAlg($k$)
8:         if $\hat{i}_k \in \{i_1, \dots, i_{k-1}\}$ then
9:             $i_k \leftarrow$ Random row not in $\{i_1, \dots, i_{k-1}\}$
10:        else
11:            $i_k \leftarrow \hat{i}_k$
12:        $\hat{j}_k \leftarrow$ Suggested column $j_t$ by ColAlg($k$)
13:        if $\hat{j}_k \in \{j_1, \dots, j_{k-1}\}$ then
14:            $j_k \leftarrow$ Random column not in $\{j_1, \dots, j_{k-1}\}$
15:        else
16:            $j_k \leftarrow \hat{j}_k$
17:    $I_t \leftarrow (i_1, \dots, i_d)$
18:    $J_t \leftarrow (j_1, \dots, j_d)$
19:    Observe $M_t(I_t(k), J_t(k))$ for all $k \in [d]$
20:    for $k_1 = 1, \dots, d$ do                              ▷ Update statistics
21:        for $k_2 = 1, \dots, d$ do
22:            if $i_k = \hat{i}_k$ then
               A: What is $k$? You are looping over $k_1$ and $k_2$.
23:                Update arm $i_k$ of RowAlg($k$) with reward
                   A: This is all messed up. You are using $k$ as an argument in max and also in the sentence above.
                   $\max_{k \le k_1} M_t(i_k, j_{k_2}) - \max_{k < k_1} M_t(i_k, j_{k_2})$
24:            else
25:                Update $\hat{i}_k$ of RowAlg($k$) with reward 0
26:            if $j_k = \hat{j}_k$ then
27:                Update arm $j_k$ of ColAlg($k$) with reward
                   $\max_{k \le k_2} M_t(i_{k_1}, j_k) - \max_{k < k_2} M_t(i_{k_1}, j_k)$
28:            else
29:                Update $\hat{j}_k$ of ColAlg($k$) with reward 0
```