POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

Stochastic Linear Bandit with Global-Local Structure

Laurea Magistrale in Computer Science and Engineering - Ingegneria Informatica

Author: Francesco Fulco Gonzales

Advisor: Prof. Francesco Trovò

Co-advisor: Marco Mussi, Gianmarco Genalti, Prof. Marcello Restelli

Academic year: 2021-2022

1. Introduction

This work addresses the problem of Multi-Agent Stochastic Linear Bandit within a novel setting called *Partitioned Linear Setting*, where there are multiple linear bandits that face different yet related tasks. The setting assumes that the parameter vector is decomposed into a global component, shared across all agents, and a local component specific to each agent. The objective of the work is to design an algorithm that balancing exploration and exploitation trade-off by leveraging both the global information shared among bandits and effectively maximize the collective reward of the agents. We propose an algorithm that aims to exploit the shared global information and effectively collaborate among the agents to achieve the goal of maximizing the collective reward over time. To solve this setting we propose an online regret minimization algorithm, namely P-LinUCB. We tested the algorithm in a synthetic setting and we show that it outperforms the state-of-the-art baseline for multi-agent bandits across multiple scenarios.

2. Background

The stochastic multi-armed bandit [2] is a fundamental problem in sequential decision-making under uncertainty. In this setting, an agent is presented with a set of choices, or arms, each associated with an unknown probability distribution of rewards. The agent selects an arm to pull at each time step, with the goal of maximizing their total reward over a fixed number of time steps, called horizon. The agent faces a trade-off between exploring different arms to learn more about their reward distributions and exploiting the arms with the highest expected rewards based on current knowledge.
Among the different classes of bandits, we study the stochastic linear bandit, i.e., a variant of the multi-armed bandit problem where each arm is a vector and the expected reward of each arm is a linear function of the arm vector. More specifically, the expected reward of an arm is given by the dot product of the arm vector and an unknown parameter vector, which the agent aims to estimate through repeated pulls of the arms. One important property of the stochastic linear bandit is that by imposing a (linear) relationship between the arms, it lets the agent learn the environment without necessarily having to pull every arm, since it gather information on the parameter from any arm pull, thus significantly simplifying the setting and making problems with large or infinite actions spaces tractable. In the following sections, we analyze a prominent algorithm for linear bandits,

which serves as foundation for our algorithm, and then we outline the existing approaches for Multi-Agent MABs.

2.1. LinUCB

LinUCB [3] extends the famous UCB algorithm to the linear setting. The core idea is to maintain a confidence set for the vector of coefficients θ of the linear function that generates the rewards, rather than for the mean reward of each arm since it is the factor shared by all arms. At each round, the algorithm selects an estimate of the coefficients from the confidence set, and chooses an action that maximizes the predicted reward. The problem, therefore, becomes constructing confidence sets for the vector of coefficients of the linear function based on observed action-reward pairs in previous time steps.

2.2. Multi-Agent Bandits

The problem of learning across multiple bandit instances has been actively researched in the field of recommendation systems, where users and their similarities are represented as a graph. The goal of these works is to design a feedback-sharing mechanism to leverage user similarities embedded in the graph. One of the most prominent works in this line of research approaches the problem from a clustering perspective [1]. The authors assign a linear bandit instance to each user and make the assumption that users within the same cluster exhibit the same behavior, and therefore have similar bandit weights. The algorithm, called CLUB, obtains the clustering of users partitioning the graph by repeatedly deleting edges between users whose bandit weights differ significantly. Another adjacent area of research is the multi-task bandits literature, which aims to transfer knowledge across several similar bandit problems while managing the classical exploration-exploitation trade-off. The multi-task linear bandit setting [4] works under the assumption that all the tasks are similar, where the similarity of two task is defined as having L2-norm of the difference of bandit parameters smaller than some threshold. Their algorithmic solution is a variant of LinUCB that constructs two confidence bounds, one task-specific and multi-task bounds, and at each round considers the tightest confidence bound, since it's the most certain, and then selects the arm with the largest retained confidence bound, in typical UCB fashion.

[5] considers the setting where the unknown parameter in each linear bandit instance can be decomposed into a global parameter plus a sparse instance-specific term. They propose an estimator that combines the trimmed mean from robust statistics to learn across similar instances and LASSO regression to debias the results by capturing the instance-specific information. This estimator, called Robust Multitask Estimator, is then used in N (number of users) linear bandits running at each problem instance.

3. Problem Formulation

The multi-agent stochastic linear bandit setting is an extension of the stochastic linear bandit problem, where N linear bandits are trying to solve different yet related tasks. Each linear bandit i is characterized by a parameter vector $\boldsymbol{\theta}^i \in \mathbb{R}^{d_i}$. At each time step t, an external context index $i \in \{1, \dots, N\}$ arrives, determining which agent will be the actor of the current turn, out of the N available agents. At each time step t, agent i chooses a vector arm $\boldsymbol{a}_{i,t} \in \mathcal{A}_i \subseteq \mathbb{R}^{d_i}$, where $\mathcal{A}_i$ is the set of available arms and d_i is the i-th vector dimension. The set of possible arms can be either discrete or continuous, and the dimensions of the arms of different bandits can be different, hence the subscript i for the dimension. The agent then receives a scalar reward $y_t \in \mathbb{R}$, which is generated by a linear function of the arm vector, and an additive stochastic noise, that characterizes the stochastic bandit setting. The function generating the reward is:

$$y_t = \boldsymbol{\theta}^{i\top} \boldsymbol{a}_{i,t} + \eta_t,$$

where $\boldsymbol{\theta}^i$ is the unknown bandit parameter, different for each agent i, and η_t is a zero-mean subgaussian random noise.

3.1. Regret

The objective of each individual bandit i is to minimize its expected regret. Since we take into account all agents together and not each individual separately, the overall objective function of the problem is to maximize the sum of the cumulative regret throughout all bandits:

$$R_T = \sum_{t=1}^{T} \max_{\boldsymbol{a}_{i_t,t} \in \mathcal{A}_{i_t}} \boldsymbol{\theta}_*^{i_t\top} \boldsymbol{a}_{i_t,t} - \mathbb{E}\left[y_t\right],$$

where i_t denotes the agent i served at time t, and $\boldsymbol{\theta}_*^{i_t}$ is the optimal parameter vector for agent i_t, and y_t the actual reward collected by agent i_t. The expectation is with respect to the randomness in the environment and policy. The first term in this expression is the maximum expected reward using any policy, since the parameter is known to be optimal and the max operator selects the action that yields the highest reward. The second term is the expected reward collected by the learner.

3.2. Partitioned Linear Bandit

The peculiarity of the partitioned setting consists in the decomposition of the linear bandit parameter, which has k of its components shared across all bandits, with $k < d$, and the remaining $d-k$ components specific to each bandit instance. We will use the slicing notation : k or k : to denote the first or last k components of a vector. For the sake of simplicity, we will characterize them as global components, with the subscript g, and local components with the subscript l. Therefore, we have a unique parameter vector $\boldsymbol{\theta}_g$ and a set of parameter vectors specific to each agent $\{\boldsymbol{\theta}_l^1, \dots, \boldsymbol{\theta}_l^N\}$. At each time step t the agent i chooses an arm $\boldsymbol{a}_{i,t}$ and observes the reward y_t generated as follows:

$$y_t = \boldsymbol{\theta}_g^\top \boldsymbol{a}_{g,t} + \boldsymbol{\theta}_l^{i\top} \boldsymbol{a}_{i,t} + \eta_t.$$

We notice that there are two independent components that contribute to the reward: the global component, shared by all bandits, that is a linear function of the first k components of the arm, and a local component specific to the agent i to be served, which is also a different linear function of the remaining part of the arm, plus the standard σ-subgaussian noise η.

4. Proposed Algorithm

The proposed algorithm solves the setting by leveraging the assumption that each bandit parameter consists in the concatenation of the shared and local parameters, i.e., $\boldsymbol{\theta}^i = [\boldsymbol{\theta}_g, \boldsymbol{\theta}_l^i]$. We use this important piece of information to learn $\boldsymbol{\theta}_g$ using the data from all bandit instances, while learning $\boldsymbol{\theta}_l^i$ separately. The algorithm starts with each independent bandit instance learning the whole θ^i on its own, until one of them achieves a low enough regression error, such that we can consider its estimate of the θ_g reliable. From that point on, we will only update the local components of the parameter of all agents, which speeds up the learning process.

Algorithm 1 PARTITIONED LINUCB

Input: $N, T, \{\mathcal{A}_i\}_{i=1}^N, k, \varepsilon, w, \lambda$
$is\_split =$ **false**
Instantiate $\text{LINUCB}_i(\mathcal{A}_i, \lambda) \quad \forall i = 1, \dots, N$
for $t = 1, 2, \dots, T$ **do**
  Receive index i
  if is_split **then**
    $\boldsymbol{a}_l = \arg\max_{\boldsymbol{a} \in \mathcal{A}_{i,l}} \text{UCB}_i(\boldsymbol{a})$
    Pull arm $\boldsymbol{a}_{i,t} = [\boldsymbol{a}_g, \boldsymbol{a}_l]$
    Receive reward y_t
    $y_{l,t} = y_t - \boldsymbol{\theta}_g^\top \boldsymbol{a}_g$
    $\text{UPDATE}(\text{LINUCB}_i, \boldsymbol{a}_i, y_{l,t})$
  else
    Pull $\boldsymbol{a}_{i,t} = \arg\max_{\boldsymbol{a} \in \mathcal{A}_i} \text{UCB}_i(\boldsymbol{a})$
    Receive reward y_t
    $\text{UPDATE}(\text{LINUCB}_i, \boldsymbol{a}_{i,t}, y_t)$
    if $\text{AGGREGCRITERION}(\boldsymbol{y}_{i,hist}, \hat{\boldsymbol{y}}_{i,hist}, w, \varepsilon)$ **then**
      $\text{LINUCB}_j = \text{SPLIT}(\text{LINUCB}_j, k) \quad \forall j \neq i$
      $\boldsymbol{a}_g = \boldsymbol{a}_{i,t,:k}$
      $is\_split =$ **true**
    end if
  end if
end for

4.1. Set-up

Algorithm 1 takes as inputs the the number of agents N, time horizon T, the set of arms of each agent $\{\mathcal{A}_i\}_{i=1}^N$, the partitioning parameter k that splits the arms dimension in global vs local, the sliding window dimension w and the regularization hyper-parameter λ. The instantiation of the N LinUCBs happens according to the standard LinUCB algorithm [2].

4.2. Update

The update of the parameters of bandit i is denoted with $\text{UPDATE}(\text{LINUCB}_i, \boldsymbol{a}, r)$, which happens again according to LinUCB. At each time step t a bandit index i is sampled from the unknown context distribution, and a different action and update is performed based on whether we are in the first few rounds, or we have already performed the aggregation. If the aggregation has not happened yet, we follow the standard

bandit protocol, and then we check the condition for the aggregation (Line 16), following a defined criterion.

4.3. Aggregation Criterion

The split is performed only the first time the aggregation condition is met. This condition is captured in the AggregationCriterion subroutine, which defines the specific metric and condition to use to evaluate the goodness of the estimate of $\hat{\boldsymbol{\theta}}$. To assess the quality of the estimation of $\hat{\boldsymbol{\theta}}$ we evaluate the regression error on the prediction of the reward. Among the plethora of available regression metrics we chose the Mean Absolute Percentage Error (MAPE), defined as:

$$\text{MAPE} = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{y_t - \hat{y}_t}{y_t}\right|,$$

where n is the total number of observations, y_t is the reward collected at time t, which is stored in the vector denoted as $\boldsymbol{y}_{i,hist}$, and $\hat{y}_t = \boldsymbol{\theta}^{i\top}\boldsymbol{a}_{i,t}$ is the predicted reward at time t, stored in the vector $\hat{\boldsymbol{y}}_{i,hist}$. We compute the MAPE over the last w samples in a moving average fashion. This allows for a robust evaluation of the regression performance, for a big enough value of w, while also allowing to forget the past incorrect prediction, characteristic of the online learning setting. Our choice for a regression error metric fell on the MAPE because it allows for a more intuitive interpretation compared to other metrics such as the Mean Average Error or the Mean Squared Error, while also providing a relative scale, i.e., a score from 0 to 1, which makes choosing a threshold value somewhat more generalizable. In summary, when the moving average over the last w samples of the residual of the regression for $\boldsymbol{\theta}^i$ is smaller than some threshold $\varepsilon > 0$, we know that we have a good enough estimate of the true parameter of bandit i, and each agent i may focus solely on learning its local parameters. This criterion and the threshold value were derived from experimental evaluations, and although it proves to be experimentally robust and works well in practice. We suppose that a theoretical analysis could provide a more sound criterion, which would only need to be plugged in the AggregationCriterion subroutine.

Algorithm 2 AggregationCriterion

Input: $\boldsymbol{y}_{hist}, \hat{\boldsymbol{y}}_{hist}, w, \varepsilon$
$e_t := \left|\frac{y_t - \hat{y}_t}{y_t}\right|$
if $\frac{e_t + \cdots + e_{t-w}}{w} < \varepsilon$ **then**
  return true
else
  return false
end if

4.4. Split

If the above aggregation requirement is satisfied, we save the estimated optimal sub-arm and run the Split subroutine, which partitions all the parameters of the given agent. Namely, it splits the local subarms, which will be the only ones to be pulled from now on, since the global optimal subarm has been found, and recomputes the local $\hat{\theta}_{l,t}$, using the closed form solution, using the last k components of the arm history matrix H_t and reward history vector y_t.
It's worth noting that the condition is met under the assumption that there is one of the N products whose reward depends almost exclusively on the global components, such that the local ones give a negligible contribution to the expected reward. From the following time step, the algorithm will execute the first branch of the if statement, where the bandit only chooses the best local subarm, i.e., some $\boldsymbol{a}_l \in \mathcal{A}_{i,l}$. Note that this set may completely differ from all other agents both in terms of content and dimensionality. The agent then pulls the arm obtained as the concatenation of the global and local components. The reward needed for the update is corrected by removing the contribution of the global components, since we only want to update the local ones.

Algorithm 3 Split

Input: LinUCB, k
for $\boldsymbol{a} \in \mathcal{A}$ **do**
  $\boldsymbol{a} = \boldsymbol{a}_{k:}$
end for
$\hat{\boldsymbol{\theta}}_{l,t} = (H_{k:,t}^{\top} H_{k:,t} + \lambda I)^{-1} H_{k:,t}^{\top} \boldsymbol{y}_{k:,t}$
return LinUCB

5. Experiments

5.1. Baseline

A straightforward approach to solve the setting is to employ N independent LinUCB instances. This approach, named IND-LinUCB (INDependent LinUCB) by the literature, requires each bandit to learn the whole $\boldsymbol{\theta}_i$ parameter vector, but by doing so it ignores the underlying structure of the problem. Learning the global components independently for each bandit hurts the overall performance, since it cannot use the feedback received by other agents. We chose IND-LinUCB as a baseline for the experiments because it fits best the problem setting, since, besides adopting the linear bandit structure, it is also able to learn a different local parameter vector per context.

5.2. Threshold

5.2.1 Goal

This experiment aims to evaluate the performance of our algorithm in a setting as plain as possible, which does not take advantage of the strengths of our algorithm. In fact, here we seek to empirically demonstrate that for a reasonable choice of the threshold ε, our algorithm's regret is upper bounded by IND-LinUCB, meaning that for a small enough value of ε, P-LinUCB has the same performance as IND-LinUCB in the worst case. This is the case in which the algorithm does not find a good enough $\boldsymbol{\theta}^i$ to aggregate the bandit instances and therefore adopts the same strategy as IND-LINUCB. Instead, in most cases, our algorithm has a gain in performance. This experiment shows that there is virtually no downside in using P-LinUCB as opposed to IND-LinUCB, with at least a moderate gain in performance in most cases and significant gain a few quite common scenarios.

5.2.2 Setting

This experiment uses the standard parameters reported at the beginning of the section and vary the values of the hyper-parameter ε, for which we test three configurations, two extremes to show the edge cases and a reasonable choice.

5.2.3 Results

As expected, we can observe that the extreme cases perform poorly, while for an appropriate choice of the threshold, P-LinUCB yields a good result when compared to the baseline algorithm. Table 1 breaks down the cumulative regrets averaged over 10 runs for the two algorithms, along with the standard deviation, and their percentage difference, for the three values of ε. We notice immediately that P-LinUCB falling back to IND-LinUCB when a good estimate of $\boldsymbol{\theta}$ is not found, where the definition of good is solely dictated by the threshold ε on the moving MAPE. By setting an aggressively low allowed error we prevent the aggregation to happen. On the opposite extreme, too large of a threshold leads to very bad results, since the global bandit parameter $\boldsymbol{\theta}_g$ is fixed before giving it enough time to be accurately learned, and we want to absolutely avoid this case, since it underperforms IND-LinUCB. The reasonable-valued scenario instead performs quite well, even though this very plain setting that does not take advantage of the strengths of the algorithm. This experiment is an empirical demonstration that for a reasonable choice of the threshold P-LinUCB will perform at least as well as IND-LinUCB.

ε	IND	P-LinUCB	Δ-regret
0.01	694 (14)	694 (14)	0%
0.1	694 (14)	**581** (84)	−16.28%
10^5	**694** (14)	1254 (2871)	80.69%

Table 1: Regret for different values of ε.

5.3. Long Tail Context Distribution

5.3.1 Goal

One of the primary benefits of our algorithm is its ability to learn the global parameter $\boldsymbol{\theta}_g$ at the same rate as the top-performing agent, which we will call *Leader*. The stronger the *Leader* compared to other agents the bigger are the gains in performance with respect to a strategy that does not leverage cross-agent structure, such as our baseline. A strong *Leader* is one that learns its parameter vector much faster than other agents, since it will satisfy the splitting criterion sooner, which will benefit all agents. One scenario that induces the creation of a strong *Leader* is the

long-tail economic model, which refers to the phenomenon where the vast majority of product sell in relatively small quantities, making up the tail-end of the distribution, in contrast to a few popular products that sell in large quantities and dominate the head of the curve. In this setting the most popular product will collect much more data, since it will be sampled much more often than the others and will be able to explore more, thus reducing its confidence interval and improving the estimate of its parameter much faster.

5.3.2 Setting

We perform four runs using a different context sampling distribution for each. In one run we use the uniform distribution to serve as comparison, and in the remaining three we simulate a long-tail distribution by sampling one context p times, while the rest of the context share the remaining probability uniformly, i.e., each has probability $\frac{p}{N-1}$ of being sampled. The three long-tail experiments are run with $p = 0.2$, $p = 0.5$, $p = 0.9$. Note that the uniform distribution is equivalent to $p = \frac{1}{N} = \frac{1}{11} = 0.09$.

5.3.3 Results

We can clearly see that the uniform distribution in Table 2 is at a great disadvantage compared to the rest, and that the more the distribution is unbalanced, the greater the gain in performance. In fact we observe that the regret gets smaller for larger values of p. This follows from the fact that the *Leader* will be sampled increasingly frequently, and therefore achieves a good estimate of his parameter sooner, leading to an overall smaller cumulative regret. IND-LinUCB instead is not able to share the learned parameter across bandits, and only the *Leader* will benefit from the additional data, thus leading to a much more modest improvement w.r.t. the uniform case.

p	**IND**	**P-LinUCB**	**Δ-regret**
Uni.	707 (15)	**581** (92)	−17.82%
0.2	704 (12)	**551** (91)	−21.73%
0.5	650 (11)	**405** (87)	−37.69%
0.9	432 (9)	**231** (51)	−46.53%

Table 2: Context distribution study.

6. Conclusions

This work introduces a new problem setting, called Partitioned Linear Setting, in which multiple agents make decisions to maximize their collective reward over time by balancing the exploration-exploitation trade-off and collaborating effectively. To solve this setting we develop an algorithm called Partitioned LinUCB, which exploits the shared components of the bandit parameter vector to learn the global component from the most accurate bandit instance and shares it across all bandits when the estimate is accurate, while it learns the local components separately for each instance. The experimental results show that Partitioned LinUCB outperforms the naive solution of running an independent linear bandit per context and contributes to the development of more efficient and effective algorithms for similar problems.

References

[1] Claudio Gentile, Shuai Li, and Giovanni Zappella. Online clustering of bandits. In *International Conference on Machine Learning*, pages 757–765. PMLR, 2014.

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

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

[4] Marta Soare, Ouais Alsharif, Alessandro Lazaric, and Joelle Pineau. Multi-task linear bandits. In *NIPS2014 workshop on transfer and multi-task learning: theory meets practice*, 2014.

[5] Kan Xu and Hamsa Bastani. Learning across bandits in high dimension via robust statistics. *arXiv preprint arXiv:2112.14233*, 2021.