CMSC 858G: Bandits, Experts and Games 11/14/16

Lecture 11: Bandits with Knapsacks

Instructor: Alex Slivkins Scribed by: Mahsa Derakhshan

1 Motivating Example: Dynamic Pricing

The basic version of the dynamic pricing problem is as follows. A seller has B items for sale: copies of the same product. There are T rounds. In each round t, a new customer shows up, and one item is offered for sale. Specifically, the algorithm chooses a price $p_t \in [0, 1]$. The customer shows up, having in mind some value v_t for this item, buys the item if $v_t \geq p_t$, and does not buy otherwise. The customers' values are chosen independently from some fixed but unknown distribution. The algorithm earns p_t if there is a sale, and 0 otherwise. The algorithm stops after T rounds or after there are no more items to sell, whichever comes first; in the former case, there is no premium or rebate for the left-over items. The algorithm's goal is to maximize revenue.

The simplest version $B = T$ (*i.e.,* unlimited supply of items) is a special case of bandits with IID rewards, where arms corresponds to prices. However, with $B < T$ we have a "global" constraint: a constraint that binds across all rounds and all actions.

In more general versions of dynamic pricing, we may have multiple products for sale, with a limited supply of each product. For example, in each round the algorithm may offer one copy of each product for sale, and assign a price for each product, and the customer chooses a subset of products to buy. What makes this generalization interesting is that customers may have valuations over *subsets* of products that are not necessarily additive: *e.g.,* a pair of shoes is usually more valuable than two copies of the left shoe. Here "actions" correspond to price vectors, and we have a separate "global constraint" on each product.

2 General Framework: Bandits with Knapsacks (BwK)

We introduce a general framework for bandit problems with "global constraints" such as supply constraints in dynamic pricing. We call this framework "Bandits with Knapsacks" because of an analogy with the well-known *knapsack problem* in algorithms. In that problem, one has a knapsack of limited size, and multiple items each of which has a value and takes a space in the knapsack. The goal is to assemble the knapsack: choose a subset of items that fits in the knapsacks so as to maximize the total value of these items. Similarly, in dynamic pricing each action p_t has "value" (the revenue from this action) and "size in the knapsack" (namely, number of items sold). However, in BwK the "value" and "size" of a given action are not known in advance.

The framework of BwK is as follows. There are k arms and d resources, where each "resource" represents a global constraint such as limited supply of a given product. There are B_i units of each resource i. There are T rounds, where T is a known time horizon. In each round, algorithm chooses an arm, receives a reward, and also consumes some amount of each resource. Thus, the outcome of choosing an arm is now a $(d + 1)$-dimensional vector rather than a scalar: the first component of this vector is the reward, and each of the remaining d components describe the consumption of the corresponding resource. We have the "IID assumption", which now states that for each arm a

the outcome vector is sampled independently from a fixed distribution over outcome vectors. The algorithm stops as soon as we are out of time or any of the resources; the algorithm's goal is to maximize the total reward in all preceding rounds.

We think of time (*i.e.,* the number of rounds) as one of the d resources: this resource has supply T and is consumed at the unit rate by each action. As a technical assumption, we assume that the reward and consumption of each resource in each round lies in $[0, 1]$. Formally, an instance of `BwK` is specified by parameters $(d, k; B_1, \ldots, B_d)$ and a mapping from arms to distributions over outcome vectors.

2.1 Examples

We illustrate the generality of `BwK` with several basic examples.

Dynamic pricing. Dynamic pricing with a single product is a special case of `BwK` with two resources: time (*i.e.,* the number of customers) and supply of the product. Actions correspond to chosen prices p. If the price is accepted, reward is p and resource consumption is 1. Thus, the outcome vector is

$$\begin{cases} (p, 1) & \text{if price } p \text{ is accepted} \\ (0, 0) & \text{otherwise.} \end{cases}$$

Dynamic pricing for hiring. A contractor on a crowdsourcing market has a large number of similar tasks, and a fixed amount of money, and wants to hire some workers to perform these tasks. In each round t, a worker shows up, the algorithm chooses a price p_t, and offers a contract for one task at this price. The worker has a value v_t in mind, and accepts the offer (and performs the task) if and only if $p_t \geq v_t$. The goal is to maximize the number of completed tasks.

This problem as a special case of `BwK` with two resources: time (*i.e.,* the number of workers) and contractor's budget. Actions correspond to prices p; if the offer is accepted, the reward is 1 and the resource consumption is p. So, the outcome vector is

$$\begin{cases} (1, p) & \text{if price } p \text{ is accepted} \\ (0, 0) & \text{otherwise.} \end{cases}$$

PPC ads with budgects. There is an advertising platform with pay-per-click ads (advertisers pay only when their ad is clicked). For any ad a there is a known per-click reward r_a: the amount an advertiser would pay to the platform for each click on this ad. If shown, each ad a is clicked with some fixed but unknown probability q_a. Each advertiser has a limited budget of money that he is allowed to spend on her ads. In each round, a user shows up, and the algorithm chooses an ad. The algorithm's goal is to maximize the total reward.

This problem is a special case of `BwK` with one resource for each advertiser (her budget) and the "time" resource (*i.e.,* the number of users). Actions correspond to ads. Each ad a generates reward r_a if clicked, in which case the corresponding advertiser spends r_a from her budget. In particular, for the special case of one advertiser the outcome vector is:

$$\begin{cases} (r_a, r_a) & \text{if ad } a \text{ is clicked} \\ (0, 0) & \text{otherwise.} \end{cases}$$

Repeated auction. An auction platform such as eBay runs many instances of the same auction to sell B copies of the same product. At each round, a new set of bidders arrives, and the platform runs a new auction to sell an item. The auction s parameterized by some parameter θ: *e.g.,* the second price auction with the reserve price θ. In each round t, the algorithm chooses a value $\theta = \theta_t$ for this parameter, and announces it to the bidders. Each bidder is characterized by the value for the item being sold; in each round, the tuple of bidders' values is drawn from some fixed but unknown distribution over such tuples. Algorithm's goal is to maximize the total profit from sales; there is no reward for the remaining items.

This is a special case of `BwK` with two resources: time (*i.e.,* the number of auctions) and the limited supply of the product. Arms correspond to feasible values of parameter θ. The outcome vector in round t is:

$$\begin{cases} (p_t, 1) & \text{if an item is sold at price } p_t \\ (0,0) & \text{if an item is not sold.} \end{cases}$$

The price p_t is determined by the parameter θ and the bids in this round.

Repeated bidding on a budget. Let's look at a repeated auction from a bidder's perspective. It may be a complicated auction that the bidder does not fully understand. In particular, the bidder often not know the best bidding strategy, but may hope to learn it over time. Accordingly, we consider the following setting. In each round t, one item is offered for sale. An algorithm chooses a bid b_t and observes whether it receives an item and at which price. The outcome (whether we win an item and at which price) is drawn from a fixed but unknown distribution. The algorithm has a limited budget and aims to maximize the number of items bought.

This is a special case of `BwK` with two resources: time (*i.e.,* the number of auctions) and the bidder's budget. The outcome vector in round t is:

$$\begin{cases} (1, p_t) & \text{if the bidder wins the item and pays } p_t \\ (0,0) & \text{otherwise.} \end{cases}$$

The payment p_t is determined by the chosen bid b_t, other bids, and the rules of the auction.

2.2 `BwK` compared to the "usual" bandits

`BwK` is complicated in three different ways:

1. In bandits with IID rewards, one thing that we can almost always do is the explore-first algorithm. However, Explore-First does not work for `BwK`. Indeed, suppose we have an exploration phase of a fixed length. After this phase we learn something, but what if we are now out of supply? Explore-First provably does not work in the worst case if the budgets are small enough: less than a constant fraction of the time horizon.

2. In bandits with IID rewards, we usually care about per-round expected rewards: essentially, we want to find an arm with the best per-round expected reward. But in `BwK`, this is not the right thing to look at, because an arm with high per-round expected reward may consume too much resource(s). Instead, we need to think about the *total* expected reward over the entire time horizon.

3. Regardless of the distinction between per-round rewards and total rewards, we usually want to learn the best arm. But for BwK, the best arm is not the right thing to learn! Instead, the right thing to learn is the best *distribution* of arms. More precisely, a "fixed-distribution policy" — an algorithm that samples an arm independently from a fixed distribution in each round — may be much better than any "fixed-arm policy".

   The following example demonstrates this point. Assume we have two resources: a "horizontal" resource and a "vertical" resource, both with budget B. We have two actions: the "horizontal" action which spends one unit of the "horizontal" resource and no "vertical" resource, and the vice versa for the "vertical" action. Each action brings reward of 1. Then best fixed action gives us the total reward of B, but alternating the two actions gives the total reward of $2 \times B$. And the uniform distribution over the two actions gives essentially the same expected total reward as alternating them, up to a low-order error term. Thus, the best fixed distribution performs better by a factor of 2 in this example.

3 Regret bounds

Algorithms for BwK compete with a very strong benchmark: the best algorithm for a given problem instance $\mathcal{I}$. Formally, the benchmark is defined as

$$\texttt{OPT} \triangleq \texttt{OPT}(\mathcal{I}) \triangleq \sup_{\text{algorithms ALG}} \texttt{REW}(\texttt{ALG}|\mathcal{I}),$$

where $\texttt{REW}(\texttt{ALG}|\mathcal{I})$ is the expected total reward of algorithm ALG on problem instance $\mathcal{I}$.

It can be proved that competing with OPT is essentially the same as competing with the best fixed distribution over actions. This simplifies the problem considerably, but still, there are infinitely many distributions even for two actions.

There algorithms have been proposed, all with essentially the same regret bound:

$$\texttt{OPT} - \texttt{REW}(\texttt{ALG}) \leq \tilde{O}\left(\sqrt{k \cdot \texttt{OPT}} + \texttt{OPT}\sqrt{\frac{k}{B}}\right), \tag{1}$$

where $B = \min_i B_i$ is the smallest budget. The first summand is essentially regret from bandits with IID rewards, and the second summand is really due to the presence of budgets.

This regret bound is worst-case optimal in a very strong sense: for any algorithm and any given triple (k, B, T) there is a problem instance of BwK with k arms, smallest budget B, and time horizon T such that this algorithm suffers regret

$$\texttt{OPT} - \texttt{REW}(\texttt{ALG}) \geq \Omega\left(\min\left(\texttt{OPT},\ \sqrt{k \cdot \texttt{OPT}} + \texttt{OPT}\sqrt{\frac{k}{B}}\right)\right). \tag{2}$$

However, the lower bound is proved for a particular family of problem instances, designed specifically for the purpose of proving this lower bound. So it does not rule out better regret bounds for some interesting special cases.

4 Fractional relaxation

While in BwK time is discrete and outcomes are randomized, it is very useful to consider a "fractional relaxation": a version of the problem in which time is fractional and everything happens exactly according to the expectation. We use the fractional relaxation to approximate the expected total reward from a fixed distribution over arms, and upper-bound OPT in terms of the best distribution.

To make this more formal, let $r(\mathcal{D})$ and $c_i(\mathcal{D})$ be, resp., the expected per-round reward and expected per-round consumption of resource i if an arm is sampled from a given distribution $\mathcal{D}$ over arms. The fractional relaxation is a version of BwK where:

1. each round has a (possibly fractional) "duration"
2. in each round t, the algorithm chooses duration $\tau = \tau_t$ and distribution $\mathcal{D} = \mathcal{D}_t$ over arms,
3. the reward and consumption of each resource i are, resp., $\tau\, r(\mathcal{D})$ and $\tau\, c_i(\mathcal{D})$,
4. there can be arbitrarily many rounds, but the total duration cannot exceed T.

As as shorthand, we will say that the expected total reward of $\mathcal{D}$ is the expected total reward of the fixed-distribution policy which samples an arm from $\mathcal{D}$ in each round. We approximate the expected total reward of $\mathcal{D}$ in the original problem instance of BwK with that in the fractional relaxation. Indeed, in the fractional relaxation one can continue using $\mathcal{D}$ precisely until some resource is completely exhausted, for the total duration of $\min_i B_i/c_i(D)$; here the minimum is over all resources i. Thus, the expected total reward of $\mathcal{D}$ in the fractional relaxation is

$$\texttt{FR}(\mathcal{D}) = r(\mathcal{D}) \min_{\text{resources } i} \frac{B_i}{c_i(\mathcal{D})}.$$

Note that $\texttt{FR}(\mathcal{D})$ is not known to the algorithm, but can be approximately learned over time.

Further, one can prove that

$$\texttt{OPT} \leq \texttt{OPT}_{\texttt{FR}} \triangleq \sup_{\text{distributions } \mathcal{D} \text{ over arms}} \texttt{FR}(\mathcal{D}).$$

In fact, the algorithms for BwK compete with the "relaxed benchmark" $\texttt{OPT}_{\texttt{FR}}$ rather than with the original benchmark OPT.

Informally, we are interested in distributions $\mathcal{D}$ such that $\texttt{FR}(\mathcal{D}) = \texttt{OPT}_{\texttt{FR}}(\mathcal{D})$; we call them "fractionally optimal". Interestingly, one can prove (using some linear algebra) that there exists a fractionally optimal distribution $\mathcal{D}$ with two additional nice properties:

- $\mathcal{D}$ randomizes over at most d arms,
- $c_i(\mathcal{D}) \leq B_i/T$ for each resource i (*i.e.,* in the fractional relaxation, we run out of all resources simultaneously).

5 "Clean event" and confidence regions

Another essential preliminary step is to specify the high-probability event (*clean event*) and the associated confidence intervals. As for bandits with IID rewards, the clean event specifies the high-confidence interval for the expected reward of each action a:

$$|\bar{r}_t(a) - r(a)| \leq \texttt{conf}_t(a) \quad \text{for all arms } a \text{ and rounds } t,$$

where $\bar{r}_t(a)$ is the average reward from arm a by round t, and $\texttt{conf}_t(a)$ is the confidence radius for arm a. Likewise, the clean event specifies the high-confidence interval for consumption of each resource i:

$$|\bar{c}_{i,t}(a) - c(a)| \leq \texttt{conf}_t(a) \quad \text{for all arms } a, \text{ rounds } t, \text{ and resources } i,$$

where $\bar{c}_{i,t}(a)$ is the average resource-i consumption from arm a so far.

Jointly, these confidence intervals define the "confidence region" on the matrix

$$\mu = (\quad (r(a); c_1(a), \ \ldots \ , c_d(a)) : \quad \text{for all arms } a)$$

such that μ belongs to this confidence region with high probability. Specifically, the confidence region at time t, denoted $\texttt{ConfRegion}_t$, is simply the product of the corresponding confidence intervals for each entry of μ. We call μ the *latent structure* of the problem instance.

Confidence radius. How should we define the confidence radius? The standard definition is

$$\texttt{conf}_t(a) = O\left(\sqrt{\frac{\log T}{n_t(a)}}\right),$$

where $n_t(a)$ is the number of samples from arm a by round t. Using this definition would result in a meaningful regret bound, albeit not as good as (1).

In order to arrive at the optimal regret bound (1), it is essential to use a more advanced version:

$$\texttt{conf}_t(a) = O\left(\sqrt{\frac{\nu \log(T)}{n_t(a)}} + \frac{1}{n_t(a)}\right), \tag{3}$$

where the parameter ν is the average value for the quantity being approximated: $\nu = \bar{r}(a)$ for the reward and $\nu = \bar{c}_{i,t}$ for the consumption of resource i. This version features improves scaling with $n = n_t(a)$: indeed, it is $1/\sqrt{n}$ in the worst case, and essentially $1/n$ when ν is very small. The analysis relies on a technical lemma that (3) indeed defines a confidence radius, in the sense that the clean event happens with high probability.

6 Three algorithms for BwK

Three different algorithms have been designed for BwK, all achieving the optimal regret bound (1). All three algorithms build on the common foundation described above, but then proceed via very different techniques. In the remainder we describe these algorithms and the associated intuition; the analysis of any one of these algorithms is both too complicated and too long for this lecture.

We present the first two algorithms in detail (albeit with some simplification for ease of presentation), and only give a rough intuition for the third one.

Algorithm I: balanced exploration. This algorithm can be seen as an extension of Successive Elimination. Recall that in Successive Elimination, we start with all arms being "active" and permanently de-activate a given arm a once we have high-confidence evidence that some other arm is better. The idea is that each arm that is currently active can potentially be an optimal arm given the evidence collected so far. In each round we choose among arms that are still "potentially

optimal", which suffices for the purpose of exploitation. And choosing *uniformly* (or round-robin) among the potentially optimal arms suffices for the purpose of exploration.

In BwK, we look for optimal *distributions* over arms. Each distribution $\mathcal{D}$ is called *potentially optimal* if it optimizes $\texttt{FR}(\mathcal{D})$ for some latent structure μ in the current confidence region $\texttt{ConfRegion}_t$. In each round, we choose a potentially optimal distribution, which suffices for exploitation. But *which* potentially optimal distribution to choose so as to ensure sufficient exploration? Intuitively, we would like to explore each arm as much as possible, given the constraint that we can only use potentially optimal distributions. So, we settle for something almost as good: we choose an arm uniformly at random, and then explore it as much as possible, see Algorithm 1.

Algorithm 1 Balanced exploration

In each round t,

1: $S_t \leftarrow$ the set of all potentially optimal distributions over arms.
2: Pick arm b_t uniformly at random, and pick a distribution $\mathcal{D} = \mathcal{D}_t$ over arms so as to maximize $\mathcal{D}(b_t)$, the probability of choosing arm b_t, among all potentially optimal distributions $\mathcal{D}$.
3: pick arm $a_t \sim D$.

While this algorithm is well-defined as a mapping from histories to action, we do not provide an efficient implementation for the general case of BwK.

Algorithm II: optimism under uncertainty. For each latent structure μ and each distribution $\mathcal{D}$ we have a fractional value $\texttt{FR}(\mathcal{D}|\mu)$ determined by $\mathcal{D}$ and μ. Using confidence region $\texttt{ConfRegion}_t$, we can define the Upper Confidence Bound for $\texttt{FR}(\mathcal{D})$:

$$\texttt{UCB}_t(\mathcal{D}) = \sup_{\mu \in \texttt{ConfRegion}_t} \texttt{FR}(\mathcal{D}|\mu). \tag{4}$$

In each round, the algorithm picks distribution $\mathcal{D}$ with the highest UCB. An additional trick is to pretend that all budgets are scaled down by the same factor $1-\epsilon$, for an appropriately chosen parameter ϵ, and redefine $\texttt{FR}(\mathcal{D}|\mu)$ accordingly. Thus, the algorithm is as follows:

Algorithm 2 UCB for BwK

1: Rescale the budgets: $B_i \leftarrow (1-\epsilon) \times B_i$ for each resource i
2: In each round t, pick distribution $\mathcal{D} = \mathcal{D}_t$ with highest $\texttt{UCB}_t(\mathcal{D})$
3: pick arm $a_t \sim D$.

The rescaling trick is essential: it ensures that we do not run out of resources too soon due to randomness in the outcomes or to the fact that the distributions $\mathcal{D}_t$ do not quite achieve the optimal value for $\texttt{FR}(\mathcal{D})$.

Choosing a distribution with maximal UCB can be implemented by a linear program. Since the confidence region is a product set, it is easy to specify the latent structure $\mu \in \texttt{ConfRegion}_t$ which attains the supremum in (4). Indeed, re-write the definition of $\texttt{FR}(D)$ more explicitly:

$$\texttt{FR}(\mathcal{D}) = \left(\sum_a \mathcal{D}(a)\, r(a) \right) \left(\min_{\text{resources } i} \frac{B_i}{\sum_a \mathcal{D}(a)\, c_i(a)} \right).$$

Then $\texttt{UCB}_t(\mathcal{D})$ is obtained by replacing the expected reward $r(a)$ of each arm a with the corresponding upper confidence bound, and the expected resource consumption $c_i(a)$ with the corresponding

lower confidence bound. Denote the resulting UCB on $r(\mathcal{D})$ with $r^{\texttt{UCB}}(\mathcal{D})$, and the resulting LCB on $c_i(\mathcal{D})$ with $c_i^{\texttt{LCB}}(\mathcal{D})$. Then the linear program is:

$$\begin{array}{ll} \text{maximize} & \tau \times r^{\texttt{UCB}}(\mathcal{D}) \\ \text{subject to} & \tau \times c_i^{\texttt{LCB}}(\mathcal{D}) \leq B_i(1-\epsilon) \\ & \tau \leq T \\ & \sum_a D(a) = 1. \end{array}$$

Algorithm III: resource costs and Hedge. The key idea is to pretend that instead of budgets on resources we have *costs* for using them. That is, we pretend that each resource i can be used at cost v_i^* per unit. The costs v_i^* have a mathematical definition in terms of the latent structure μ (namely, they arise as the dual solution of a certain linear program), but they are not known to the algorithm. Then we can define the "resource utilization cost" of each arm a as

$$v^*(a) = \sum_i v_i^* \, c_i(a).$$

We want to pick an arm which generates more reward at less cost. One natural way to formalize this intuition is to seek an arm which maximizes the *bang-per-buck ratio*

$$\Lambda(a) = r(a)/v^*(a).$$

The trouble is, we do not know $r(a)$, and we *really* do not know $v^*(a)$.

Instead, we approximate the bang-per-buck ratio $\Lambda(a)$ using the principle of optimism under uncertainty. As usual, in each round t we have upper confidence bound $U_t(a)$ on the expected reward $r(a)$, and lower confidence bound $L_{t,i}(a)$ on the expected consumption $c_i(a)$ of each resource i. Then, given our current estimates $v_{t,i}$ for the resource costs v_i^*, we can optimistically estimate $\Lambda(a)$ with

$$\Lambda_t^{\texttt{UCB}}(a) = r(a)/\textstyle\sum_i v_{t,i} \; L_{t,i}(a),$$

and choose an arm $a = a_t$ which maximizes $\Lambda_t^{\texttt{UCB}}(a)$. The tricky part is to maintain meaningful estimates $v_{t,i}$. Long story short, they are maintained using a version of Hedge.

One benefit of this algorithm compared to the other two is that the final pseudocode is very simple and *elementary*, in the sense that it does not invoke any subroutine such as a linear program solver. Also, the algorithm happens to be extremely fast computationally.

7 Bibliographic notes

The general setting of BwK is introduced in Badanidiyuru et al. (2013), along with the first and third algorithms and the lower bound. The UCB-based algorithm is from Agrawal and Devanur (2014). A more thorough discussion of the motivational examples, as well as an up-to-date discussion of related work, can be found in Badanidiyuru et al. (2013).

References

Shipra Agrawal and Nikhil R. Devanur. Bandits with concave rewards and convex knapsacks. In *15th ACM Conf. on Economics and Computation (ACM EC)*, 2014.

Ashwinkumar Badanidiyuru, Robert Kleinberg, and Aleksandrs Slivkins. Bandits with knapsacks. In *54th IEEE Symp. on Foundations of Computer Science (FOCS)*, 2013.