We consider how to learn through experience to make intelligent decisions. In the generic setting, called the contextual bandits problem, the learner must repeatedly decide which action to take in response to an observed context, and is then permitted to observe the received reward, but only for the chosen action. The goal is to learn to behave nearly as well as the best policy (or decision rule) in some possibly very large and rich space of candidate policies. This talk will describe recent progress on this problem and some of its variants.

Motivating example: Design a website. Site needs to decide on ads or content shown to each user as they arrive.

1. Website visited by user (with profile, browsing history, etc.)
2. Choose ad to present to user
3. User responds (clicks, leaves page, etc.)

Goal: make choices that elicit desired user behavior.

Medical treatment

1. Doctor visited by patient (with symptoms, test results, etc.)
2. Doctor chooses treatment
3. Patient responds (recovers, gets worse, etc.)

Make choices that maximize favorable outcomes.

These are both examples of the contextual bandits problem. The agent, a learner is repeated presented with a context.

1. Learner presented with context.
2. Learner chooses an action.
3. Learner observes reward (but only for chosen action). This is what makes this a bandit problem.

Goal: learn to choose actions to maximize rewards.

This is a general and fundamental problem: how to make intelligence decisions through experience. Goal: learn to choose action to maximize rewards.

Issues:

1. There is a classic dilemma:
   (a) exploit what has already been learned to maximize rewards. You might be missing out on great ways of behaving, new drugs to be explored, etc.
   (b) explore to learn which behaviors give best results.
2. Use context ways effectively. There are mamy choices of behavior possible, and you may never see the same context twice—you need to generalize.
3. Selection bias: if you explore while exploiting, you will tend to get highly skewed data.

4. Space and time efficiency.

We give overview of some algorithms and techniques. We want algorithms that are

- general-purpose and practical
- efficient

I'll first formalize the learning problem, then give algorithms and give an application and a next step.

1 Formal model

Repeat: for $t = 1 : T$,

1. Learner observes context s_t.

   Nature chooses reward vector $r_t \in [0, 1]^K$ (not observed) before the learner selects an action.

2. Learner selects action $a_t \in [K]$.

3. Learner receives observed reward $r_t(a_t)$.

Goal: maximize total reward $\sum_{t=1}^{T} r_t(a_t)$.

Assume pairs (x_t, r_t) are chosen at random iid for different t. Note we expect x_t and r_t to be correlated and x_t to be informative about r_t.

Ex. context is (male, 50,...). Nature selects rewards, learner chooses an action, and only see the reward for that action. Next (female, 18,...). Learner gets sums of rewards.

Learner tries to choose actions based on contexts, a decision rule for selecting an action based on context, a policy. Example: If sex=male, then choose action 2; if else age>45 choose action 1; else cllose action 3.

A policy π maps context x to action a. Before learning begins, as the algorithm designers, we have to choose the general form of policies Π to be used. I.e. we need to decide a policy space. Decision tree, neural network, linear classifier, etc.

We're making a tacit assumption that there exists an (unknown) policy $\pi \in \Pi$ that gives high reward.

Goal: learn through experimentation to do (almost) as well as best $\pi \in \Pi$. For simplicity, assume Π is finite, but typically extremely (exponentially) large, ex. space of all decision trees. Thus we allow policies to be very complex and expressive.

Challenges:

- Π is extremely large.
- We need to be learniner about all policies simultaneously while also performing as well as the best.

- When action selected, we only observed reward for policies that would have chosen same action
- This is exploration vs. exploitation on gigantic scale!

Make the model more precise: The goal is to get high total (or average) reward relative to best policy $\pi \in \Pi$. I.e. we want small regret

$$\max_{\pi \in \Pi} \frac{1}{T} \sum_{t=1}^{T} r_t(\pi(x_t)) - \frac{1}{T} \sum_{t=1}^{T} r_t(a_t).$$

We want regret $\to 0$ as $T \to \infty$, no regret learning algorithm.

2 Algorithms

2.1 Full information setting

Start out by supposing the learner sees the rewards for all actions, not just the selection action. Ex. when the doctor prescribes a drug, finds out what happens with a host of other treatments. This is not realistic, just to build intuition.

We can pick any particular policy and figure out which rewards would have been received from that policy. We can go back and compute what actions would have been selected by that policy and since we can observe all the rewards, figure out what reward would have been received from those actions, and get the total reward from following that policy. The average is a good estimate of the policy's expected reward.

Obvious thing to do: just pick the best policy, Follow-the-Leader. Find the empirically best policy (that's performed the best so far), and use it on the current context to select the next action.

This algorithm gives optimal regret $O\left(\sqrt{\frac{\ln|\Pi|}{T}}\right)$ which goes to 0, and scales as $\sqrt{\ln|\Pi|}$ in the policy space size.

To apply efficiently, we need an oracle: algorithm/subroutine for finding best $\pi \in \Pi$ on observed contexts and rewards. We call it an argmax oracle (ERM, classification, linear oracle).

It does something that's natural in ML. It's solving a standard learning problem, a classification problem, classify objects according to category. If we have a good classification algorithm for Π, we can use it to find a good policy. We can use SVMs, decision tree algorithms, boosting, neural networks, etc. Policy corresponds to classifier, context corresponds to example, action correspond to label.

I highlight techniques that come up.

- Estimate expected reward of each policy.
- Use existing method (oracle) to find best policy.

Proof idea: every policy's empirical average reward close to actual average.

So far we assumed stochastic setting, each (x_t, r_t) in iid. This is not always realistic; you can have correlated or drifting data, or an adversarial environment as in game playing.

In non-stochastic (adversarial) setting, contexts and rewards are arbitrarily, not random, possibly selected by adversary.

FTL does not work: the adversary can force low reward while ensuring one policy gets fairly high reward.

There is another algorithm that works: the Hedge algorithm (Littlestone, Warmuth; Freund, Schapire).

- Maintain one weight for every $\pi \in \Pi$.
- On each rount t
  - choose random policy π with probability proportional to weights
  - use action chosen by π
  - increase weight of each policy according to reward it would have received.

This yields optimal regret even in adversarial setting. But time and space are linear in $|\Pi|$ because it needs to maintain a weight for each policy. This is too slow if $|\Pi|$ is large.

Technique: use weighted combination of policies.

Proof idea: keep track of sum of weights of all policies. Give upper bound in terms of reward of algorithms and lower bound in terms of reward of best policy. Combine to get regret bound.

In summary, FTL is for stochastic setting, and is efficent given access to oracle. Hedge is non-stochastic, inefficient if $|\Pi|$ is large.

Is the best of both possible? This appears impossible (Hazan, Koren).

2.2 Bandit setting

We only see rewards for actions taken. Learner can still compute own total reward. But for any policy π, we only observe π's rewards on subset of rounds—the rounds where the learner picked the same action.

Problems with oracle algorithm:

- only see some rewards
- observed rewards are highly biased (due to skewed choice of actions).

Exploration is very important in bandit setting. Suppose

- drug A is pretty good (cure rate 60%)
- drug B is much better (cure rate 80%)

But in early trials, by chance, A might appear better than B. Follow-the-leader can get stuck in only picking A. Statistics for A get better but statistics for B stay lousy because we're not getting more samples.

Use modified follow-the-leader: ε-greedy/epoch-greedy algorithm. There is explicit exploitation and exploration. On each round,

- choose action according to best policy so far with probability $1-\varepsilon$ (exploitation)
- uniformly at random with probability ε (exploration)

This is simple and fast given oracle with analysis similar to FTL, but you don't get optimal regret, $O\left(\left(\frac{K\ln|\Pi|}{T}\right)^{\frac{1}{3}}\right)$.

Technique: explicit exploration via uniform sampling of actions.

Let's talk about selection bias. Bias is a big problem, but there is an old trick for dealing with bias, inverse propensity weighting. Suppose we want to estimate $\mathbb{E}X$, ex. an unfair coin coming up heads. Suppose with probability p (independent of X) observe X once and with probability $1-p$ don't observe X at all.

Trick: define $\widehat{X} = X/p$ if observed and 0 else. Even if only some of the time we observe X, we can always observe $\widehat{X}$. Then $\mathbb{E}[\widehat{X}] = \mathbb{E}X$; this is unbiased.

We're not done because the variance may be extremely large. To get good estimators we must control variance.

Technique: inverse propensity weighting to get unbiased estimates.

What about bandits in non-stochastic setting? The algorithm is Exp4: contexual bandits algorithm for non-stochastic setting. Combine hedge, uniform sampling of actions, and inverse propensity weighting. Get optimal regret

$$O\left(\sqrt{\frac{K\ln|\Pi|}{T}}\right).$$

This analysis is similar to Hedge, but we must account for variance. Complexity scales as $|\Pi|$.

To compare, epoch greedy is for stochastic setting, not optimal, efficient given access to oracle. Exp4 is non-stochastic setting, achieves optimal regret, inefficient if $|\Pi|$ huge.

The difference in regret is big. To get regret ε, for the first we need $O\left(\frac{1}{\varepsilon^3}\right)$; for the second we need $O\left(\frac{1}{\varepsilon^2}\right)$.

Can we get the best of both? Yes, the "Mini-Monster" algorithm (a.k.a. ILOVETOCONBANDITS), Agarwal, Hsu, Kale, Langford, Li, Schapire.

- Apply all preceding techniques
- Every round, find weighted combination of policies satisfiying explicitly stated properties
    1. Low (estimated) regret, i.e., choose actions we think will give high reward. (exploit)
    2. Low (estimated) variance, i.e., ensure future estimates will be accurate. (Explore)

There are exponentially many constraints, but we can solve using simple and efficient algorithm given oracle. Find violated constraint, fix it, and repeat.

Get nearly optimal $\widetilde{O}\left(\sqrt{\frac{K\ln|\Pi|}{T}}\right)$ and fast: only requires average of $O\left(\sqrt{\frac{K}{T\ln|\Pi|}}\right) \ll 1$ oracle queries per round. (Oddly, you call it less when the policy space is larger.

This is the same approach as RandomizedUCB (a.k.a. Monster) but simpler and much faster. (Dudik, Hsu, Kale, Karampatziakis, Langford, Reyzin, Zhang)

Technique: formulate properties as optimization problem and solve.

Proof ideas:

- For regret bound: regret constraint ensures low regret (if estimates are good enough). Variance constraints ensure that they actually will be good enough.
- Use potential function to measure progress. Every iteration of this numerical algorithm, we make good progress. The total amount of progress we need is bounded.

3 Application

We created a system that implements contextual bandits problem, the multiworld testing decision service. It is a unified system for solving contextual bandit problems:

- general purpose
- modular
- easy to interface with existing systems
- designed to reduce common errors (it's hard to collect data in a reliable way).

It's used at MSN.com, to select news articles.

- No previous learning method has been successful
- There is 25% relative lift in click-through rate
- It handles thousands of requests per second.

Contextual bandits is challenging and interesting problem.

People mainly use ε-greedy at this point; mini-Monster hasn't been deployed.