A Guided Tour of **Chapter 15**: Multi-Armed Bandits: Exploration versus Exploitation

Ashwin Rao

ICME, Stanford University

January 11, 2023
Many situations in business (& life!) present dilemma on choices

**Exploitation:** Pick choices that *seem* best based on past outcomes

**Exploration:** Pick choices not yet tried out (or not tried enough)

Exploitation has notions of “being greedy” and being “short-sighted”

Too much Exploitation $\Rightarrow$ Regret of missing unexplored “gems”

Exploration has notions of “gaining info” and being “long-sighted”

Too much Exploration $\Rightarrow$ Regret of wasting time on “duds”

How to balance Exploration and Exploitation so we combine *information-gains* and *greedy-gains* in the most optimal manner

Can we set up this problem in a mathematically disciplined manner?
Examples

- Restaurant Selection
  - **Exploitation:** Go to your favorite restaurant
  - **Exploration:** Try a new restaurant

- Online Banner Advertisement
  - **Exploitation:** Show the most successful advertisement
  - **Exploration:** Show a new advertisement

- Oil Drilling
  - **Exploitation:** Drill at the best known location
  - **Exploration:** Drill at a new location

- Learning to play a game
  - **Exploitation:** Play the move you believe is best
  - **Exploration:** Play an experimental move
Multi-Armed Bandit is a spoof name for “Many Single-Armed Bandits”

A Multi-Armed Bandit problem is a 2-tuple \((\mathcal{A}, \mathcal{R})\)

\(\mathcal{A}\) is a known set of \(m\) actions (known as “arms”)

\(\mathcal{R}^a(r) = \mathbb{P}[r|a]\) is an unknown probability distribution over rewards

At each step \(t\), the AI agent (algorithm) selects an action \(A_t \in \mathcal{A}\)

Then the environment generates a reward \(R_t \sim \mathcal{R}^{A_t}\)

The AI agent’s goal is to maximize the Cumulative Reward:

\[
\sum_{t=1}^{T} R_t
\]

Can we design a strategy that does well (in Expectation) for any \(T\)?

Note that any selection strategy risks wasting time on “duds” while exploring and also risks missing untapped “gems” while exploiting
Is the MAB problem a Markov Decision Process (MDP)?

- Note that the environment doesn’t have a notion of \textit{State}.
- Upon pulling an arm, the arm just samples from its distribution.
- However, the agent might maintain a statistic of history as it’s \textit{State}.
- To enable the agent to make the arm-selection (action) decision.
- The action is then a \textit{(Policy)} function of the agent’s \textit{State}.
- So, agent’s arm-selection strategy is basically this \textit{Policy}.
- Note that many MAB algorithms don’t take this formal MDP view.
- Instead, they rely on heuristic methods that don’t aim to \textit{optimize}.
- They simply strive for “good” Cumulative Rewards (in Expectation).
- Note that even in a simple heuristic algorithm, $A_t$ is a random variable simply because it is a function of past (random) rewards.
Regret

- The Action Value $Q(a)$ is the (unknown) mean reward of action $a$
  
  $$Q(a) = \mathbb{E}[r|a]$$

- The Optimal Value $V^*$ is defined as:
  
  $$V^* = Q(a^*) = \max_{a \in A} Q(a)$$

- The Regret $l_t$ is the opportunity loss on a single step $t$
  
  $$l_t = \mathbb{E}[V^* - Q(A_t)]$$

- The Total Regret $L_T$ is the total opportunity loss
  
  $$L_T = \sum_{t=1}^{T} l_t = \sum_{t=1}^{T} \mathbb{E}[V^* - Q(A_t)]$$

- Maximizing Cumulative Reward is same as Minimizing Total Regret
Counting Regret

- Let $N_t(a)$ be the (random) number of selections of $a$ across $t$ steps
- Define $Count_t(a)$ (for given action-selection strategy) as $\mathbb{E}[N_t(a)]$
- Define Gap $\Delta_a$ of $a$ as the value difference between $a$ and optimal $a^*$
  \[
  \Delta_a = V^* - Q(a)
  \]
- Total Regret is sum-product (over actions) of Gaps and Counts $T$
  \[
  L_T = \sum_{t=1}^{T} \mathbb{E}[V^* - Q(A_t)]
  = \sum_{a \in A} \mathbb{E}[N_T(a)] \cdot (V^* - Q(a))
  = \sum_{a \in A} Count_T(a) \cdot \Delta_a
  \]
- A good algorithm ensures small Counts for large Gaps
- Little problem though: We don’t know the Gaps!
If an algorithm *never* explores, it will have linear total regret.
If an algorithm *forever* explores, it will have linear total regret.
Is it possible to achieve sublinear total regret?
We consider algorithms that estimate $\hat{Q}_t(a) \approx Q(a)$

Estimate the value of each action by rewards-averaging

$$\hat{Q}_t(a) = \frac{1}{N_t(a)} \sum_{s=1}^{t} R_s \cdot 1_{A_s = a}$$

The Greedy algorithm selects the action with highest estimated value

$$A_t = \arg \max_{a \in A} \hat{Q}_{t-1}(a)$$

Greedy algorithm can lock onto a suboptimal action forever

Hence, Greedy algorithm has linear total regret
The $\epsilon$-Greedy algorithm continues to explore forever

At each time-step $t$:
- With probability $1 - \epsilon$, select $A_t = \arg \max_{a \in A} \hat{Q}_{t-1}(a)$
- With probability $\epsilon$, select a random action (uniformly) from $A$

Constant $\epsilon$ ensures a minimum regret proportional to mean gap

$$l_t \geq \frac{\epsilon}{|A|} \sum_{a \in A} \Delta_a$$

Hence, $\epsilon$-Greedy algorithm has linear total regret
Optimistic Initialization

- Simple and practical idea: Initialize $\hat{Q}_0(a)$ to a high value for all $a \in \mathcal{A}$
- Update action value by incremental-averaging
- Starting with $N_0(a) \geq 0$ for all $a \in \mathcal{A}$,
  \[
  N_t(A_t) = N_{t-1}(A_t) + 1
  \]
  \[
  \hat{Q}_t(A_t) = \hat{Q}_{t-1}(A_t) + \frac{R_t - \hat{Q}_{t-1}(A_t)}{N_t(A_t)}
  \]

- Encourages systematic exploration early on
- One can also start with a high value for $N_0(a)$ for all $a \in \mathcal{A}$
- But can still lock onto suboptimal action
- Hence, Greedy + optimistic initialization has linear total regret
- $\epsilon$-Greedy + optimistic initialization also has linear total regret
Decaying $\epsilon_t$-Greedy Algorithm

- Pick a decay schedule for $\epsilon_1, \epsilon_2, \ldots$
- Consider the following schedule

\[ c > 0 \]

\[ d = \min_{a | \Delta_a > 0} \Delta_a \]

\[ \epsilon_t = \min(1, \frac{c|A|}{d^2t}) \]

- Decaying $\epsilon_t$-Greedy algorithm has logarithmic total regret
- Unfortunately, above schedule requires advance knowledge of gaps
- Practically, implementing some decay schedule helps considerably
- **Educational Code** for decaying $\epsilon$-greedy with optimistic initialization
Lower Bound

- Goal: Find an algorithm with sublinear total regret for any multi-armed bandit (without any prior knowledge of $R$)
- The performance of any algorithm is determined by the similarity between the optimal arm and other arms
- Hard problems have similar-looking arms with different means
- Formally described by KL-Divergence $KL(R^a || R^{a^*})$ and gaps $\Delta_a$

Theorem (Lai and Robbins)

Asymptotic Total Regret is at least logarithmic in number of steps, i.e., as $T \to \infty$,

$$L_T \geq \log T \sum_{a | \Delta_a > 0} \frac{1}{\Delta_a} \geq \log T \sum_{a | \Delta_a > 0} \frac{\Delta_a}{KL(R^a || R^{a^*})}$$
Which action should we pick?

- The more uncertain we are about an action-value, the more important it is to explore that action.
- It could turn out to be the best action.
After picking blue action, we are less uncertain about the value
And more likely to pick another action
Until we home in on the best action
Upper Confidence Bounds

- Estimate an upper confidence \( \hat{U}_t(a) \) for each action value
- Such that \( Q(a) \leq \hat{Q}_t(a) + \hat{U}_t(a) \) with high probability
- This depends on the number of times \( N_t(a) \) that \( a \) has been selected
  - Small \( N_t(a) \) \( \Rightarrow \) Large \( \hat{U}_t(a) \) (estimated value is uncertain)
  - Large \( N_t(a) \) \( \Rightarrow \) Small \( \hat{U}_t(a) \) (estimated value is accurate)
- Select action maximizing Upper Confidence Bound (UCB)

\[
A_{t+1} = \arg \max_{a \in A} \{ \hat{Q}_t(a) + \hat{U}_t(a) \}
\]
Hoeffding’s Inequality

**Theorem (Hoeffding’s Inequality)**

Let $X_1, \ldots, X_n$ be i.i.d. random variables in $[0, 1]$, and let

$$\bar{X}_n = \frac{1}{n} \sum_{i=1}^{n} X_i$$

be the sample mean. Then for any $u \geq 0$,

$$\mathbb{P}[\mathbb{E}[\bar{X}_n] > \bar{X}_n + u] \leq e^{-2nu^2}$$

- Apply Hoeffding’s Inequality to rewards of $[0, 1]$-support bandits
- Conditioned on selecting action $a$ at time step $t$, setting $n = N_t(a)$ and $u = \hat{U}_t(a)$,

$$\mathbb{P}[Q(a) > \hat{Q}_t(a) + \hat{U}_t(a)] \leq e^{-2N_t(a) \cdot \hat{U}_t(a)^2}$$
Calculating Upper Confidence Bounds

- Pick a small probability $p$ that $Q(a)$ exceeds UCB $\{\hat{Q}_t(a) + \hat{U}_t(a)\}$
- Now solve for $\hat{U}_t(a)$

$$e^{-2N_t(a) \cdot \hat{U}_t(a)^2} = p$$

$$\Rightarrow \hat{U}_t(a) = \sqrt{-\log p} \cdot \frac{1}{2N_t(a)}$$

- Reduce $p$ as we observe more rewards, eg: $p = t^{-\alpha}$ (for fixed $\alpha > 0$)
- This ensures we select optimal action as $t \to \infty$

$$\hat{U}_t(a) = \sqrt{\frac{\alpha \log t}{2N_t(a)}}$$
Yields UCB1 algorithm for arbitrary-distribution arms bounded in [0, 1]

\[
A_{t+1} = \arg \max_{a \in \mathcal{A}} \{ \hat{Q}_t(a) + \sqrt{\frac{\alpha \log t}{2N_t(a)}} \}
\]

**Theorem**

The UCB1 Algorithm achieves logarithmic total regret asymptotically, i.e., as \( T \to \infty \),

\[
L_T \leq \sum_{a | \Delta_a > 0} \frac{4\alpha \cdot \log T}{\Delta_a} + \frac{2\alpha \cdot \Delta_a}{\alpha - 1}
\]

**Educational Code** for UCB1 algorithm
Bayesian Bandits

- So far we have made no assumptions about the rewards distribution \( \mathcal{R} \) (except bounds on rewards)
- *Bayesian Bandits* exploit prior knowledge of rewards distribution \( \mathbb{P} [\mathcal{R}] \)
- They compute posterior distribution of rewards \( \mathbb{P} [\mathcal{R} | h_t] \) where \( h_t = A_1, R_1, \ldots, A_t, R_t \) is the history
- Use posterior to guide exploration
  - Upper Confidence Bounds (Bayesian UCB)
  - Probability Matching (Thompson sampling)
- Better performance if prior knowledge of \( \mathcal{R} \) is accurate
Bayesian UCB Example: Independent Gaussians

- Assume reward distribution is Gaussian, \( \mathcal{R}^a(r) = \mathcal{N}(r; \mu_a, \sigma_a^2) \)
- Compute Gaussian posterior over \( \mu_a, \sigma_a^2 \) (details in book Appendix)

\[
P[\mu_a, \sigma_a^2 | h_t] \propto P[\mu_a, \sigma_a^2] \cdot \prod_{t|A_t=a} \mathcal{N}(R_t; \mu_a, \sigma_a^2)
\]

- Pick action that maximizes Expectation of: “c std-errs above mean”

\[
A_{t+1} = \arg\max_{a \in \mathcal{A}} \mathbb{E}_{P[\mu_a, \sigma_a | h_t]}[\mu_a + \frac{c \cdot \sigma_a}{\sqrt{N_t(a)}}]
\]
Probability Matching

- **Probability Matching** selects action $a$ according to probability that $a$ is the optimal action

$$\pi(A_{t+1}|h_t) = P_{D_t \sim \mathbb{P}[R|h_t]}[\mathbb{E}_{D_t}[r|A_{t+1}] > \mathbb{E}_{D_t}[r|a], \forall a \neq A_{t+1}]$$

- Probability matching is optimistic in the face of uncertainty
- Because uncertain actions have higher probability of being max
- Can be difficult to compute analytically from posterior
Thompson Sampling

- **Thompson Sampling** implements probability matching

\[
\pi(A_{t+1}|h_t) = P_{D_t \sim P[R|h_t]}[E_{D_t}[r|A_{t+1}] > E_{D_t}[r|a], \forall a \neq A_{t+1}]
\]

\[
= E_{D_t \sim P[R|h_t]}[\mathbb{1}_{A_{t+1}=\arg \max_{a \in A} E_{D_t}[r|a]}]
\]

- Use Bayes law to compute posterior distribution \(P[R|h_t]\)
- **Sample** a reward distribution \(D_t\) from posterior \(P[R|h_t]\)
- Estimate Action-Value function with sample \(D_t\) as \(\hat{Q}_t(a) = E_{D_t}[r|a]\)
- Select action maximizing value of sample

\[
A_{t+1} = \arg \max_{a \in A} \hat{Q}_t(a)
\]

- Thompson Sampling achieves Lai-Robbins lower bound!
- **Educational Code** for Thompson Sampling for Gaussian Distributions
- **Educational Code** for Thompson Sampling for Bernoulli Distributions
Gradient Bandit Algorithms

- Gradient Bandit Algorithms are based on Stochastic Gradient Ascent
- We optimize $Score$ parameters $s_a$ for $a \in A = \{a_1, \ldots, a_m\}$
- Objective function to be maximized is the Expected Reward

$$J(s_{a_1}, \ldots, s_{a_m}) = \sum_{a \in A} \pi(a) \cdot \mathbb{E}[r|a]$$

- $\pi(\cdot)$ is probabilities of taking actions (based on a stochastic policy)
- The stochastic policy governing $\pi(\cdot)$ is a function of the $Scores$:

$$\pi(a) = \frac{e^{s_a}}{\sum_{b \in A} e^{s_b}}$$

- $Scores$ represent the relative value of actions based on seen rewards
- Note: $\pi$ has a Boltzmann distribution (Softmax-function of $Scores$)
- We move the $Score$ parameters $s_a$ (hence, action probabilities $\pi(a)$) such that we ascend along the direction of gradient of objective $J(\cdot)$
Gradient of Expected Reward

To construct Gradient of $J(\cdot)$, we calculate $\frac{\partial J}{\partial s_a}$ for all $a \in A$

$$\frac{\partial J}{\partial s_a} = \frac{\partial}{\partial s_a} \left( \sum_{a' \in A} \pi(a') \cdot \mathbb{E}[r|a'] \right) = \sum_{a' \in A} \mathbb{E}[r|a'] \cdot \frac{\partial \pi(a')}{\partial s_a}$$

$$= \sum_{a' \in A} \pi(a') \cdot \mathbb{E}[r|a'] \cdot \frac{\partial \log \pi(a')}{\partial s_a} = \mathbb{E}_{a' \sim \pi, r \sim R_{a'}} [r \cdot \frac{\partial \log \pi(a')}{\partial s_a}]$$

We know from standard softmax-function calculus that:

$$\frac{\partial \log \pi(a')}{\partial s_a} = \frac{\partial}{\partial s_a} \left( \log \frac{e^{s_{a'}}}{\sum_{b \in A} e^{s_b}} \right) = \mathbb{I}_{a = a'} - \pi(a)$$

Therefore $\frac{\partial J}{\partial s_a}$ can we re-written as:

$$= \mathbb{E}_{a' \sim \pi, r \sim R_{a'}} [r \cdot (\mathbb{I}_{a = a'} - \pi(a))]$$

At each step $t$, we approximate the gradient with $(A_t, R_t)$ sample as:

$$R_t \cdot (\mathbb{I}_{a = A_t} - \pi_t(a)) \text{ for all } a \in A$$
Score updates with Stochastic Gradient Ascent

- $\pi_t(a)$ is the probability of $a$ at step $t$ derived from score $s_t(a)$ at step $t$
- Reduce variance of estimate with baseline $B$ that’s independent of $a$:

$$(R_t - B) \cdot (\mathbb{1}_{a=A_t} - \pi_t(a)) \text{ for all } a \in A$$

- This doesn’t introduce bias in the estimate of gradient of $J(\cdot)$ because

$$E_{a' \sim \pi}[B \cdot (\mathbb{1}_{a=a'} - \pi(a))] = E_{a' \sim \pi}[B \cdot \frac{\partial \log \pi(a')}{\partial s_a}]$$

$$= B \cdot \sum_{a' \in A} \pi(a') \cdot \frac{\partial \log \pi(a')}{\partial s_a} = B \cdot \sum_{a' \in A} \frac{\partial \pi(a')}{\partial s_a} = B \cdot \frac{\partial}{\partial s_a} \left( \sum_{a' \in A} \pi(a') \right) = 0$$

- We can use $B = \bar{R}_t = \frac{1}{t} \sum_{s=1}^{t} R_s = \text{average rewards until step } t$
- So, the update to scores $s_t(a)$ for all $a \in A$ is:

$$s_{t+1}(a) = s_t(a) + \alpha \cdot (R_t - \bar{R}_t) \cdot (\mathbb{1}_{a=A_t} - \pi_t(a))$$

- Educational Code for this Gradient Bandit Algorithm
Value of Information

- Exploration is useful because it gains information
- Can we quantify the value of information?
  - How much would a decision-maker be willing to pay to have that information, prior to making a decision?
  - Long-term reward after getting information minus immediate reward
- Information gain is higher in uncertain situations
- Therefore it makes sense to explore uncertain situations more
- If we know value of information, we can trade-off exploration and exploitation *optimally*
We have viewed bandits as *one-step* decision-making problems.
Can also view as *sequential* decision-making problems.
At each step there is an *information state* $\tilde{s}$
- $\tilde{s}$ is a statistic of the history, i.e., $\tilde{s}_t = f(h_t)$
- summarizing all information accumulated so far
Each action $a$ causes a transition to a new information state $\tilde{s}'$ (by adding information), with probability $\tilde{P}^a_{\tilde{s},\tilde{s}'}$
This defines an MDP $\tilde{M}$ in information state space

$$\tilde{M} = (\tilde{S}, A, \tilde{P}, R, \gamma)$$
Consider a Bernoulli Bandit, such that $R^a = B(\mu_a)$

For arm $a$, reward = 1 with probability $\mu_a$ (= 0 with probability $1 - \mu_a$)

Assume we have $m$ arms $a_1, a_2, \ldots, a_m$

The information state is $\tilde{s} = (\alpha_{a_1}, \beta_{a_1}, \alpha_{a_2}, \beta_{a_2}, \ldots, \alpha_{a_m}, \beta_{a_m})$

$\alpha_a$ records the pulls of arms $a$ for which reward was 1

$\beta_a$ records the pulls of arm $a$ for which reward was 0

In the long-run, $\frac{\alpha_a}{\alpha_a + \beta_a} \rightarrow \mu_a$
We now have an infinite MDP over information states.

This MDP can be solved by Reinforcement Learning.

Model-free Reinforcement learning, eg: Q-Learning (Duff, 1994)

Or Bayesian Model-based Reinforcement Learning
  - eg: Gittins indices (Gittins, 1979)
  - This approach is known as Bayes-adaptive RL
  - Finds Bayes-optimal exploration/exploitation trade-off with respect of prior distribution
Bayes-Adaptive Bernoulli Bandits

Start with $Beta(\alpha_a, \beta_a)$ prior over reward function $R^a$.

Each time $a$ is selected, update posterior for $R^a$ as:
- $Beta(\alpha_a + 1, \beta_a)$ if $r = 1$
- $Beta(\alpha_a, \beta_a + 1)$ if $r = 0$

This defines transition function $\tilde{P}$ for the Bayes-adaptive MDP.

$(\alpha_a, \beta_a)$ in information state provides reward model $Beta(\alpha_a, \beta_a)$.

Each state transition corresponds to a Bayesian model update.
Bayes-adaptive MDP can be solved by Dynamic Programming
The solution is known as the Gittins Index
Exact solution to Bayes-adaptive MDP is typically intractable
Guez et al. 2020 applied Simulation-based search
  - Forward search in information state space
  - Using simulations from current information state
Summary of approaches to Bandit Algorithms

- Naive Exploration (eg: $\epsilon$-Greedy)
- Optimistic Initialization
- Optimism in the face of uncertainty (eg: UCB, Bayesian UCB)
- Probability Matching (eg: Thompson Sampling)
- Gradient Bandit Algorithms
- Information State Space MDP, incorporating value of information
A Contextual Bandit is a 3-tuple \((\mathcal{A}, \mathcal{S}, \mathcal{R})\)

- \(\mathcal{A}\) is a known set of \(m\) actions ("arms")
- \(\mathcal{S} = \mathbb{P}[s]\) is an **unknown** distribution over states ("contexts")
- \(\mathcal{R}_s^a(r) = \mathbb{P}[r|s,a]\) is an **unknown** probability distribution over rewards

At each step \(t\), the following sequence of events occur:
- The environment generates a state \(S_t \sim \mathcal{S}\)
- Then the AI Agent (algorithm) selects an action \(A_t \in \mathcal{A}\)
- Then the environment generates a reward \(R_t \in \mathcal{R}^A_{S_t}\)

The AI agent’s goal is to maximize the Cumulative Reward:

\[
\sum_{t=1}^{T} R_t
\]

- Extend Bandit Algorithms to Action-Value \(Q(s,a)\) (instead of \(Q(a)\))