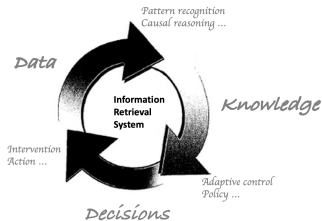


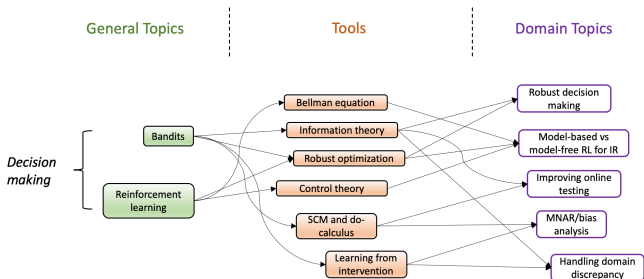
Tutorial: Theoretical Tools for Designing Modern Information Retrieval System (Part2)

Author: *Da Xu, Chuanwei Ruan*

Contact: {daxu5180,ruanchuanwei}@gmail.com



1 Part 2: Decision Making



① Part 2: Decision Making

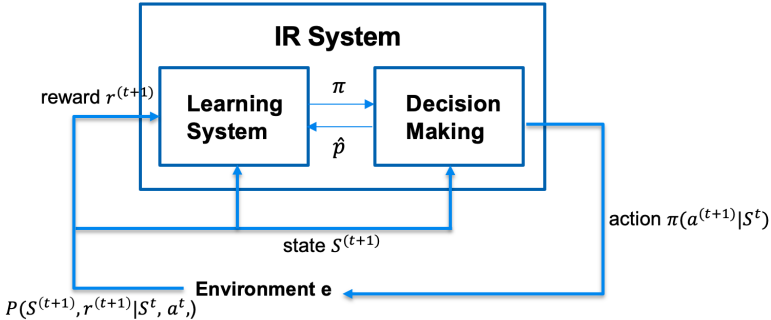
Decision Making

- IR systems interact with real world. Every action we make induces some objective reward/cost from the environment – *everything that is not under our control*.
- Pattern recognition helps us in the sense that we do good if similar scenarios occur.
 - However, the "similar scenario" may consist only a very small part of what might happen when the IR systems keep interacting with the environment.
 - Even with the "similar scenario", there are **uncertainty** associated with what we have seen, as well as the learning procedure we use.
 - We still need to optimize the decision making strategy in face of the unknown environment.
- Interestingly, even if the environment is known, it still takes planning to optimize a long-term goal.

Decision Making

In real-world IR system, the feedback we get are **evaluative**, instead of **instructive**.

- We only know how good an action is.
- We don't know what is the best action (like in supervised learning).



Decision Making

- A distinctive feature of IR is that we can actively **query** the environment. In causal inference term, we can make **intervention** with the environment.
 - CV and NLP can not do that.
- To quantify how well we can do in a (perhaps infinite) sequence of queries, we focus on the cumulative regret.

$$R_n(x) = \mathbb{E} \left[\sum_{i=1}^T r_t \right]$$

where \mathbb{E} is w.r.t the interaction between our strategy π and the environment.

- The key notion is to balance **exploration and exploitation**. General exploration ideas such as *upper confidence bound (UCB)*, *ϵ -greedy*, *Thompson sampling*, *actor-critic* are quite standard.
- What will be a good cumulative regret?
 - $O(T)$ is clearly bad.
 - Generally, $O(\sqrt{T})$ is considered descent.

Bandit

- Bandit is a simplified setting in the sense that the transition of states are ignored – the actions do not change the state.
- The IR system interacts directly with environment, and we get a sequence of $\{(a_i, r_i)\}_{i=1}^t$ or $\{(a_i, x_i, r_i)\}_{i=1}^t$ in the contextual setting.
- We focus on UCB – its idea is one of the greatest in AI history!
- The key to UCB is **confidence bounds**. Suppose the mean reward of an action is $\hat{\mu}_a$, computed from $\{r_1(a), \dots, r_n(a)\}$.
 - **Hoeffding Bound:**
Assume $X_{a,i}, a = 0, \dots, k$ are independent, σ -sub-Gaussian random variables. Then for any $\epsilon > 0$:

$$Pr\left(\frac{1}{n_a} \sum_{i=1}^{n_a} X_{a,i} - \mu_a \geq \epsilon\right) \leq \exp\left(\frac{-n_a \epsilon^2}{2\sigma^2}\right),$$

- UCB calibrates decision making by the uncertainty of the estimation.
"optimism in face of uncertainty."

Bandit

It holds with probability at least $1 - \delta$:

$$\mu_a \in \left[\underbrace{\hat{\mu}_a - \sqrt{2\sigma^2 \log(1/\delta)/n_a}}_{\text{LCB}^{(a)}(n_a, \delta)}, \underbrace{\hat{\mu}_a + \sqrt{2\sigma^2 \log(1/\delta)/n_a}}_{\text{UCB}^{(a)}(n_a, \delta)} \right],$$

In each round, we choose $\text{argmax} \text{UCB}(a)$.

- Why it works? General idea for proving bounds for bandit algorithm:
 - ① differentiate "good" and "bad" events
 - ② bound the regret of "good" events, which is simple
 - ③ show "bad" events happen with small probability
- Exploration with bandit is quite a mature technique.
- But off-policy learning (the policy we aim to improve is different from the policy that generates the feedback) with bandit feedback is actively studied and related to many of our previous topics.

Adaptive Online Testing

For **hypothesis testing**, the inference goal of an online experiment is to test a statistical hypothesis that when H_0 is rejected, it holds:

$$Pr(\text{rejecting } H_0 \text{ for } H_1 \mid H_0 \text{ is true}) \leq \alpha,$$

Practically for IT companies, adaptive online testing means the proportion of traffic guided to each arm during the random assignment can be adjusted based on the performance to date.

For **best-arm identification**, suppose $a^{[i]}$ is the arm with the i^{th} highest reward, and $\Delta_j = \mu_{a^{[1]}} - \mu_{a_j}$ is the *suboptimality gap* of arm a_j . The exploratory algorithms can be described by a stopping time T and a data-adaptive decision rule such that

$$Pr(T < \infty \text{ and } \Delta_{(T)} > 0) \leq \delta,$$

Adaptability to online testing while not costing its statistical rigorousness:

- **hypothesis testing**: *stopping rule*
- **best-arm identification**: *sampling strategy*

Adaptive Online Testing

Pure exploration problem takes the *fixed confidence* setup:

$$\mu_a \in \left[\underbrace{\hat{\mu}_a - \sqrt{2\sigma^2 \log(1/\delta)/n_a}}_{\text{LCB}^{(a)}(n_a, \delta)}, \underbrace{\hat{\mu}_a + \sqrt{2\sigma^2 \log(1/\delta)/n_a}}_{\text{UCB}^{(a)}(n_a, \delta)} \right],$$

- **Action Elimination**: sample from each arm and eliminate according to: $\text{UCB}^{(a_i)} < \text{LCB}^{(a_0)}$. Continue until only one arm is left.
- **Upper Confidence Bound** : sample and stop when there exists an a^* such that: $\text{LCB}^{(a^*)} \geq \text{UCB}^{(a_i)}$ for all $i = 1, \dots, k$.
- **LUCB**: keep sampling from both the current *best* and *second-best* arm (denote by a^* and a^{**}) and stop when: $\text{LCB}^{(a^*)} > \text{UCB}^{(a^{**})}$.

Confidence intervals not only guides how to explore, but also decides when to stop, which brings in hypothesis testing, as confidence intervals can also be related to p-value¹.

¹Xu et al. On the Advances and Challenges of Adaptive Online Testing, WSDM'22

Adaptive Online Testing

A always valid p-value is a stochastic process $\{P^{(t)}\}_{t=1}^{\infty}$ such that for any (random) stopping time T , under any distribution $\mathbb{P}_{\mathcal{H}_0}$ we have:

$$\mathbb{P}_{\mathcal{H}_0}(P^{(T)} \leq \alpha) \leq \alpha.$$

Extends property of p-value of following *uniform distribution* under $\mathbb{P}_{\mathcal{H}_0}$. To control FDR in an online fashion ($R^{(t)} = 1$: a discovery at step t), $\alpha^{(t)}$ should be aware of the history $\{R^{(j)}\}_{j=1}^{(t-1)}$ with a proper decision rule :

$$\alpha^{(t)} := \alpha^{(t)}(\alpha; R^{(1)}, \dots, R^{(t-1)})$$

- 1 Obtain the adjusted significance level $\alpha^{(t)}$ using the history p-values $\{P^{(j)}\}_{j=1}^{t-1}$ and rejections $\{R^{(j)}\}_{j=1}^{t-1}$, with α -investing.
- 2 Follow LCUB: compute $\{UCB^{a_i}(n_i(t), \alpha^{(t)})\}_{i=0}^k$ to find a^* and sample from them. Compute and track the any-time p-value $P^{(t)}$.
- 3 Decide whether to stop based on record $R^{(t)} = 1[P^{(t)} > \alpha^{(t)}]$.

Adaptive Online Testing

Let $T(\mathcal{I}, \delta)$ be the stopping time for applying on problem \mathcal{I} such that it achieves the δ -correctness. It holds that:

$$\inf T(\mathcal{I}, \delta) = \mathcal{O}\left(\sum_{j=2}^k \frac{1}{\Delta_{[j]}^2} \left(\ln \frac{1}{\delta} + H(\mathcal{I})\right) + \frac{\ln \ln \Delta_{[2]}^{-1}}{\Delta_{[2]}^2} \text{polylog}(k, \delta^{-1})\right),$$

where $H(\mathcal{I})$ is an entropy term of the set \mathcal{I} .

Three quantities essential to the lower bound:

- ① $\sum_{j=2}^k 1/\Delta_{[j]}^2$: measures the *overall complexity* (total gap) of the problem instance
- ② $1/\Delta_{[2]}^2$: the gap between the best arm and the strongest contender;
- ③ $H(\mathcal{I})$: the amount of variation (or divergence) in the gaps' distribution

Off-Policy Learning

- Most often, we use reweighting to correct for the difference between the **behavior policy** π and **logging policy** π_0 , e.g.

$$\hat{V}(\pi; \pi_0, X) = \frac{1}{T} \sum_{i=1}^T \frac{\pi(a_i|x_i)}{\pi_0(a_i|x_i)} r_i.$$

Sometimes people use normalized weights, or adding control variables (e.g. $\hat{r}(X_i)$) to reduce variance. These methods are quite mature, if π_0 has a large enough support.


- The goal of off-policy learning is to optimize the behavior policy: $\underset{\pi \in \mathcal{F}}{\operatorname{argmin}} \hat{V}(\pi; \pi_0, X)$ (regret minimization) and analyze the optimal π^* .
- What are the biggest challenges? As a system, there are three major sources of uncertainty that cause robustness concerns:
 - environment uncertainty: the conditional distribution $R_i|X_i$ changes over time as a mechanism of the environment.
 - logging uncertainty: the logging policy is not truthfully recorded
 - deployment uncertainty: the candidate deployment can not be executed as is.

Domain Topic: Robust Off-Policy Learning

To resolve the problem, we follow the principle of: *"optimize the worst-possible scenario to achieve robustness"*

- Define the hypothetical uncertainty set using domain knowledge:
 - ① environment uncertainty: $X \in \mathcal{U}(X; \alpha)$, e.g. $D(\tilde{X}||X) \leq \alpha$
 - ② logging uncertainty: $\pi_0(a|X) \in \mathcal{U}(\pi_0(a|X); \alpha)$, e.g. $D(\tilde{\pi}_0||\pi) \leq \alpha$
 - ③ deployment uncertainty: $\pi(a|X) \in \mathcal{U}(\pi(a|X); \alpha)$
- Derive the min-max optimization objective:
 - ① $\min_{\pi} \max_{D(\tilde{X}||X) \leq \alpha} \hat{V}(\pi; \pi_0, \tilde{X})$
 - ② $\min_{\pi} \max_{D(\tilde{\pi}||\pi_0) \leq \alpha} \hat{V}(\pi; \tilde{\pi}, X)^2$
 - ③ $\min_{\pi} \max_{D(\tilde{\pi}||\pi) \leq \alpha} \hat{V}(\tilde{\pi}; \pi_0, X)^3$

²Xu et al. Counterfactual Adversarial Learning and Evaluation of Recsys, NIPS'20

³Xu et al. Robust Off-policy Learning for Online Uncertainty, AAAI'22 

Domain Topic: Robust Off-Policy Learning

- Analyze the optimization structure:

- is there duality structure to convert the constraint problem to unconstraint with little cost, e.g.

$$\min_{\pi} \max_{D_w(\tilde{X}||X) \leq \alpha} \hat{V}(\pi; \pi_0, \tilde{X}) \leq \min_{\pi} \left\{ \hat{V}(\pi, \pi_0, \tilde{X}) + \lambda D_w(\tilde{X}||X) \right\}$$

- is there efficient solution to the subproblem, so we can proceed in a minorize-maximization fashion:

$$\max_{D(\tilde{\pi}||\pi_0) \leq \alpha} \tilde{V}(\pi; \tilde{\pi}, X) \leq \hat{V}(\pi; X, \alpha), \quad \underset{\pi}{\text{minimize}} \tilde{V}(\pi; X, \alpha).$$

- Theoretical analysis of π^* : "generalizing the empirical improvement of π^* v.s. π_0 to unseen data", e.g.

$$V(\pi_0) - V(\pi^*) \leq \hat{V}(\pi_0) - \hat{V}(\pi^*) + \text{complexity} + \text{slack}$$

- The uncertainty constraint acts just like creating a discrepancy between the observed domain and the hypothetical "uncertainty-maximization" domain:

$$V(\pi_0) - V(\pi^*) \leq \hat{V}(\pi_0) - \hat{V}(\pi^*) + \text{func}(\text{complexity}, \text{slack}, \alpha)$$

Model-Based v.s. Model-Free RL

What if we enter the more complex system-environment world? What is the best way to achieve robust RL control?

Consider reward maximization the dynamic system is generated by the different equation:

$$S_{t+1} = f_t(S_t, a_t, \epsilon_t)$$

We focus on the episodic setting where the policy can depend on the trajectory $\tau_t = (a_1, \dots, a_{t-1}, S_1, \dots, S_{t-1})$

The goal is to:

$$\text{maximize } \mathbb{E} \left[\sum_{t=0}^T R(S_t, a_t) \right]$$

subject to: $S_{t+1} = f(S_t, a_t, \epsilon_t)$ and $a_t = \pi_t(r_t)$

Our policy π_t is the optimization variable.

Model-Based v.s. Model-Free RL

Perhaps the most obvious strategy to solve RL problem is to estimate the dynamical process, and use it in a dynamic programming solution in the following control problem.

Estimating dynamical system is also known as **system identification**, e.g. we can build a ML model g that optimizes:

$$\min_g \sum_{t=0}^T \|s_{t+1} - g(s_t, a_t)\|^2,$$

and then use its estimate \hat{g} to solve the problem. However, we still need a decent estimation of the noise process ϵ_t .

- Clearly, model-based approach depends on the correctness of \hat{g} , as well as the specification of ϵ_t .

Can we directly approximate the optimal control given by the Bellman equation?

Model-Based v.s. Model-Free RL

In the context of episodic RL, *approximate dynamic programming* recursively compute (backward in time):

$$Q_{\gamma}(s, a) = r(s, a) + \gamma \mathbb{E} \left[\max_{a'} Q_{\gamma}(f(s, a, \epsilon), a) \right].$$

Note that the discount factor λ is critical to approximate the infinite-horizon behavior such that **Q-learning** converges:

$$Q_{\gamma}^{new}(s_k, a_k) = (1 - \eta) Q^{new}(s_k, a_k) + \eta \left(r(s_k, a_k) + \gamma_{a'} Q^{new}(s_{k+1}, a') \right)$$

The renown temporal-difference methods also belong to this category. For the purpose of control, one can simply use the resulting optimal action-value function, i.e.

$$a_t = \arg \max_a Q_{\gamma}(s_t, a).$$

We mention that we can parameterize the Q function – being model-free is simply a historical term that says we do not model the environment.

Model-Based v.s. Model-Free RL

Another important family of model-free methods is to **directly** learn from the past experience, without building a model or resort to Bellman equation. The key idea is to transform optimization problem to a **sampling problem**.

Note that unconstraint optimization problem can be converted to *optimizing over distribution*:

$$\max_x R(x) \quad \rightarrow \quad \max_{p(x)} \mathbb{E}_p[R(x)],$$

as long as p contains the family of Dirac distributions.

Suppose we parameter p by θ , then the problem becomes:

$\max_{\theta} \mathbb{E}_{p(x;\theta)}[R(x)]$. Using the log-likelihood trick (basically the one used in REINFORCE), the gradient computes as:

$$\nabla_{\theta} \mathbb{E}_{p(x;\theta)}[R(x)] = \mathbb{E}_{p(x;\theta)}[R(x) \nabla_{\theta} \log p(x; \theta)].$$

This converts a optimization problem to a sampling problem.

Model-Based v.s. Model-Free RL

- By sampling x_0 from $p(x; \theta^{(t)})$ at the current round of optimization, we obtain a *one-sampled-based* gradient:
 $G(x_0; \theta^{(t)}) = R(x_0) \nabla_{\theta} \log p(x_0; \theta^{(t)})$. We then use it to obtain $\theta^{(t+1)}$ in a GD fashion.
- To reduce the variance, we may start with multiple samples $\{x_0, \dots, x_k\}$ and compute the average gradient.
- As compared to solving the Bellman equation, the essence of policy gradient is to use **parameterized randomized policy**. Given a trajectory τ , its probability distribution induced by the policy is:

$$p(\tau; \theta) = \prod_t p(s_{t+1} | s_t, a_t) \pi(a_t | s_t; \theta),$$

and the total reward: $r(\tau) = \sum_t r_t(s_t, a_t)$.

Model-Based v.s. Model-Free RL

- Model-free methods, especially policy gradient, are highly versatile and easy to implement.
- Approximate dynamic programming has roots in Bell equation, and thus has favorable theoretical guarantees when the assumptions hold. But what is a good solution for IR problems?

A distinctive property of IR is that the internal state of the system are of high dimension. Therefore, per update, we try to inject the model with as much information as possible on each dimension.

Model-Based v.s. Model-Free RL

- Note that model-based method, i.e. when solving $\min_g \sum_{t=0}^T \|s_{t+1} - g(s_t, a_t)\|^2$, uses all the dimensions.
- Approximate dynamic programming has cramped all the information into a single dimension with a single equation.
- Also, the guarantee from Bellman equation requires carefully choosing the discount factor, which is non-trivial.
- Finally, the versatility of policy gradient comes at significant costs. Firstly, the target function we care about – r – is only accessed through function evaluations, i.e. $R(x_0) \nabla_{\theta} \log p(x_0; \theta^{(t)})$ (inefficient!). Also, the variance issues remain an open challenge, especially in the sequential learning setting.

Building the Test Bed for Control-based Recsys

**Unfortunately, our tutorial stops here at WSDM'22.
Please join us in our future events where we will present the
complete content of our tutorial.**