Leveraging Multi-Armed Bandit Algorithms for Dynamic Decision Making

Tudor Coman | Software Development Engineer

Image source:
Wikipedia

Reinforcement Learning

What is Reinforcement Learning?

- A subset of AI algorithms (like Machine Learning or Deep Learning)
- Learning what action to take in order to maximize a reward metric (mapping situations to actions)
- Unique characteristics
  - Closed-loop (current actions influence later inputs)
  - No direct instructions (no training dataset)
  - The consequences of actions can be longer-term (the result is not to be observed immediately)
- Most common issue that needs to be solved: balancing **exploration** (acquiring new information) and **exploitation** (using the current knowledge to maximize the reward)

Policies in Reinforcement Learning

- A policy π is a mapping from states to actions
- This policy can be either deterministic or stochastic
- Reinforcement learning algorithms can be on-policy and off-policy
- **On-policy**: the agent is learning the policy that if follows
- **Off-policy**: the agent collects data using one policy (the behavior policy) but learns about the optimal policy (target policy)
- Bandits with good exploration rates model stochastic policies

Multi-armed bandits

- A common reinforcement learning problem
- A player has to repeatedly choose between a finite set of actions; each action provides a random reward from a specific distribution to that action
- The final objective is to maximize the total reward obtained after a number of rounds

Adobe

How is this problem helpful?

- Healthcare: finding the best treatment while minimizing the side effects
- Finance: maximizing the yield of a portfolio by allocating funds to a finite set of instruments
- **Digital marketing: maximizing sales/clicks/views in an A/B test – *our study case***

What is A/B testing?

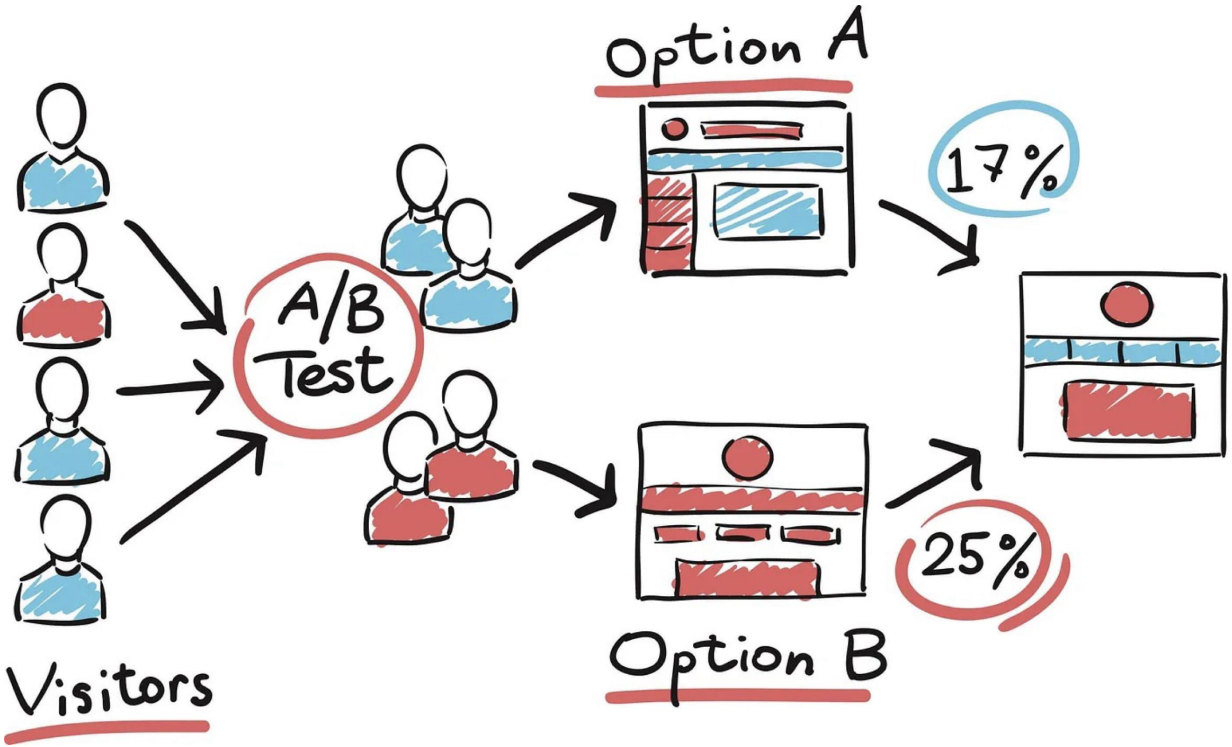


Image source:
https://medium.com/@alejandroAttento/ab-testing-in-real-life-step-by-step-8d455dc5674a/

Our setup

- A/B testing platform, where traffic is normally allocated between treatments with percentages defined by the client
- Adding multi-armed bandits and transitioning the manual allocation to a dynamic, automatic version
- Maximizing conversions in an A/B/C test with the following conversions: 6%, 2% and 4.5%
- The bandits can be considered Bernoulli because actions have only a Yes/No outcome (or “coin-tosses”)

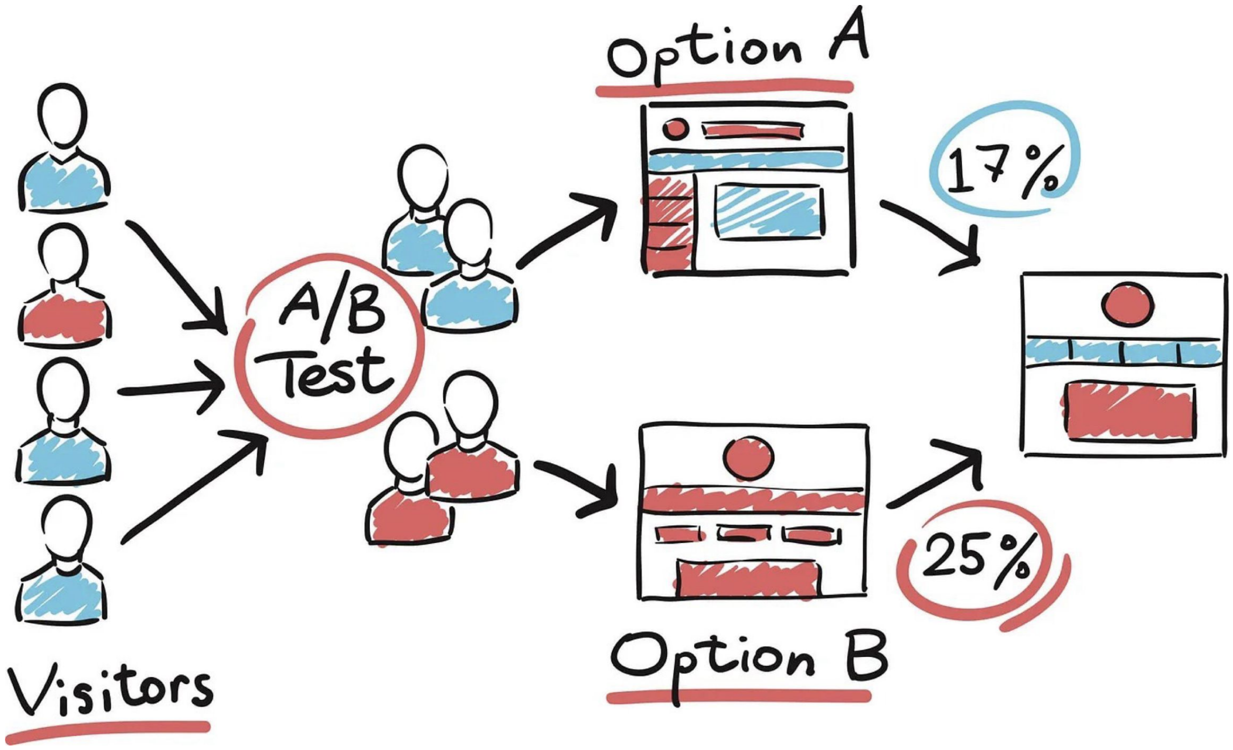


Image source:
https://medium.com/@alejandroAttento/ab-testing-in-real-life-step-by-step-8d455dc5674a/

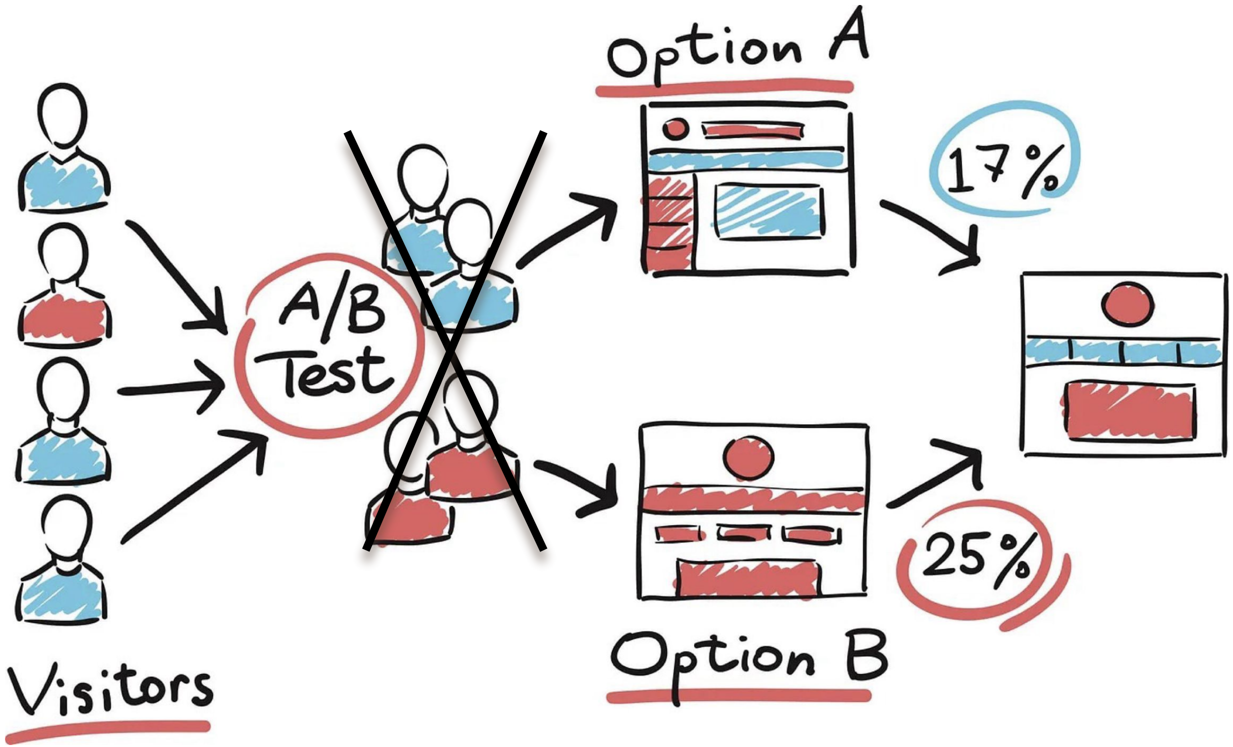


Image source:
https://medium.com/@alejandroAttento/ab-testing-in-real-life-step-by-step-8d455dc5674a/

a “smart” (AI-based) decisioning engine that does online policy evaluation => determining whether a user sees option A or B at runtime

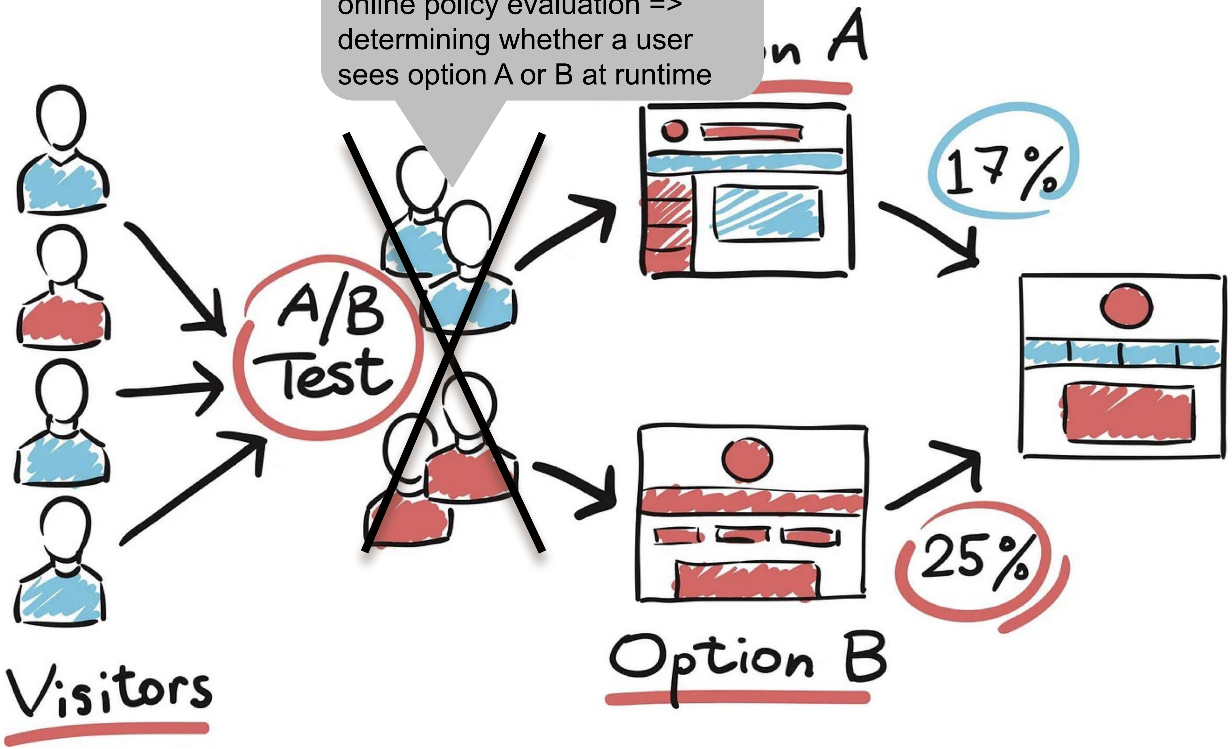
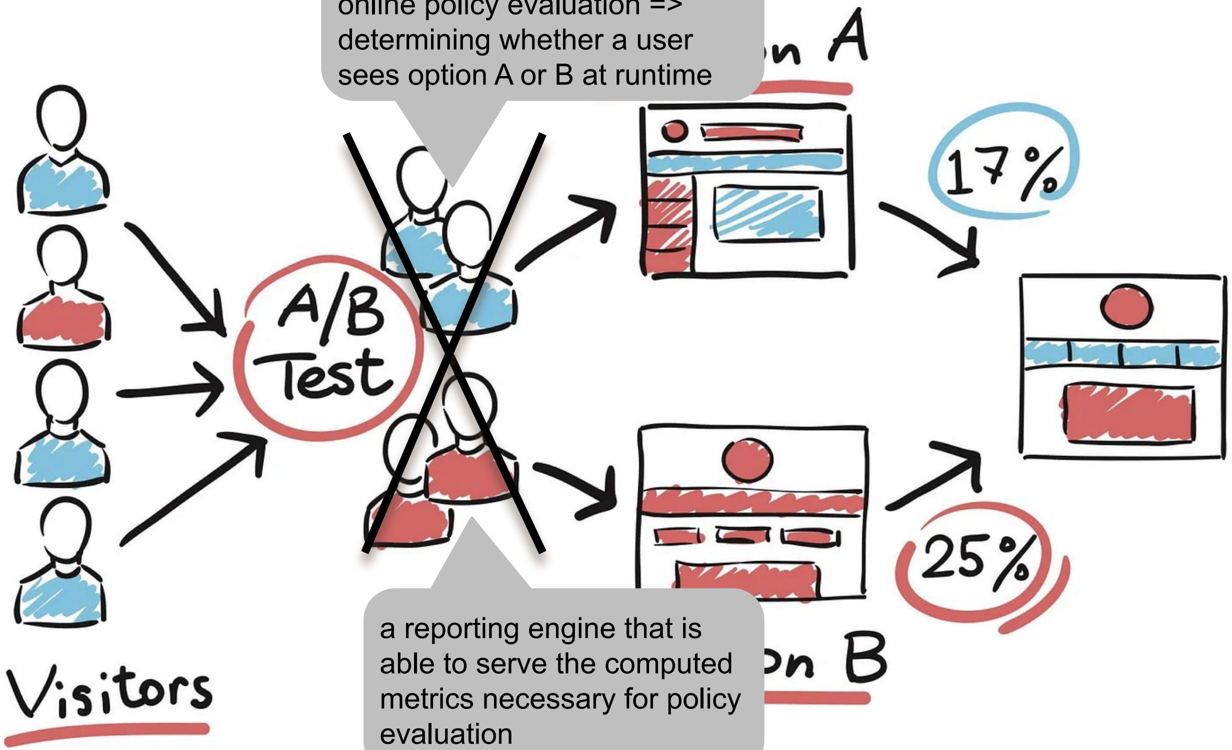


Image source:
https://medium.com/@alejandroAttento/ab-testing-in-real-life-step-by-step-8d455dc5674a/



a “smart” (AI-based) decisioning engine that does online policy evaluation => determining whether a user sees option A or B at runtime

a reporting engine that is able to serve the computed metrics necessary for policy evaluation

Image source:
https://medium.com/@alejandroAttento/ab-testing-in-real-life-step-by-step-8d455dc5674a/

Non-functional requirements for these services

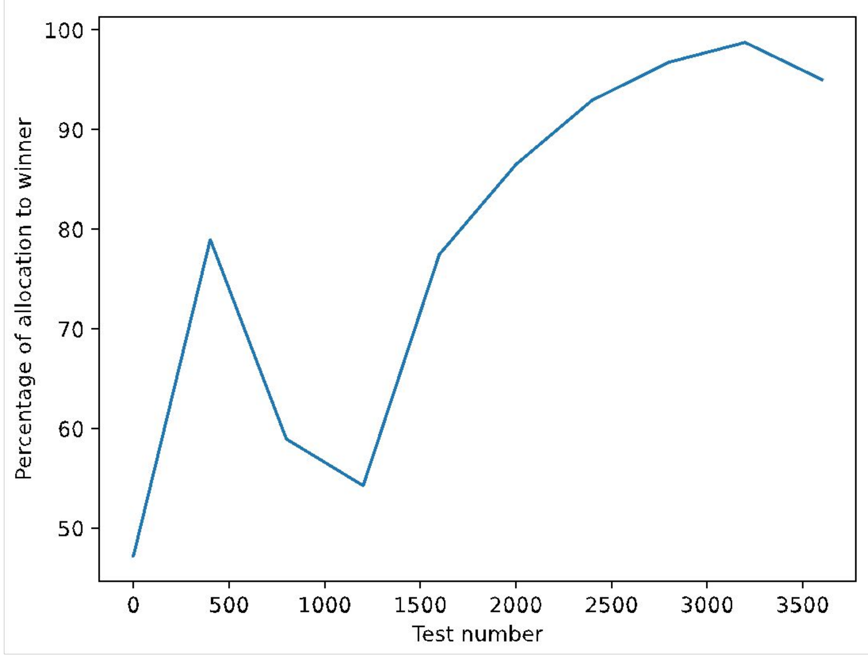
- Millisecond response time (slow policy evaluation will result in a slower user experience)
- Online policy evaluation requires a fast reporting engine
- For websites with a lot of traffic, a Big Data processing pipeline might prove to be necessary

Algorithms used

- **Thompson Sampling**
- **Upper Confidence Bound**
- **Probably Approximately Correct**
- **LilUCB**

Thompson Sampling

- Beta-Bernoulli bandit
- Beta distribution – used to model coin tosses Beta(#heads, #tails)
- During each iteration, all arms gets a score sampled from its Beta distribution (using number of conversions and number of no-conversions). The arm with the maximum score is then used
- Exploring the possible option and gradually exploiting the accumulated information



Thompson Sampling Winner Allocation

Probably Approximately Correct

- This approach uses always-valid confidence bounds
- Great for choosing more than one winning option

Arm conversion variance

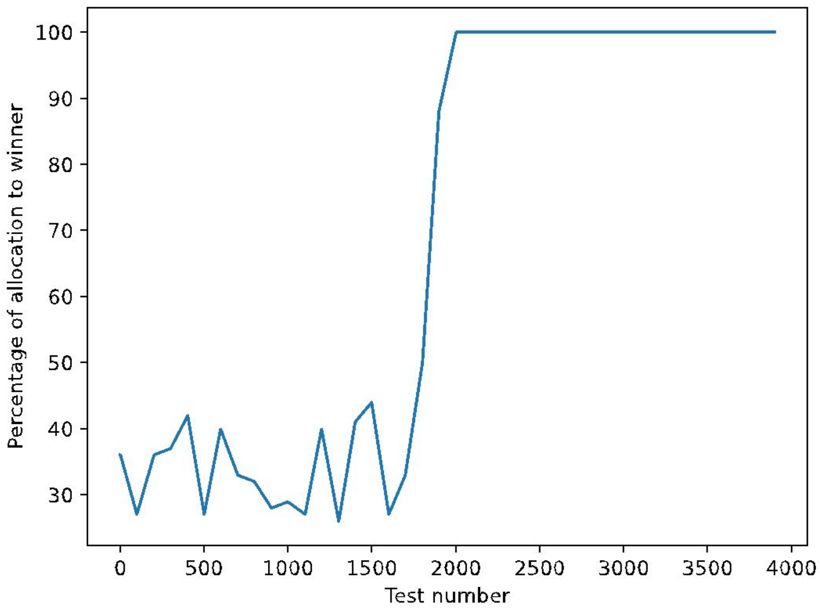
$$CS_{1-\alpha} := \left\{ \overline{x}_k \pm \sigma_k \sqrt{\frac{2(n\rho^2+1)}{n^2\rho^2} \log\left(\frac{\sqrt{n\rho^2+1}}{\alpha}\right)} \right\}$$

Arm conversion rate

Arm number of views

Finetuning parameter

Uncertainty factor



Probably Approximately Correct Winner Allocation

LilUCB

- The better "brother" of Upper Confidence Bound
- Using the Law of Iterated Logarithm (Lil) to optimize the number of attempts to get to the

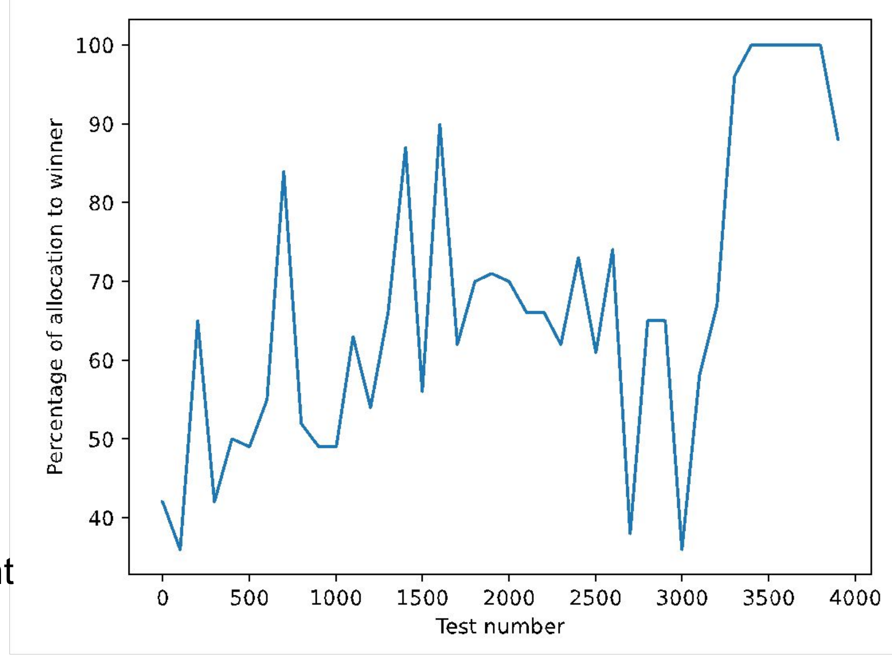
$$U_k = \overline{x_k} + (1+\beta)(1+\sqrt{\epsilon})\sqrt{\frac{2(1+\epsilon)\log\left(\frac{\log(1+\epsilon)n_k}{\delta}\right)}{n_k}}$$

Arm conversion rate

Arm number of views

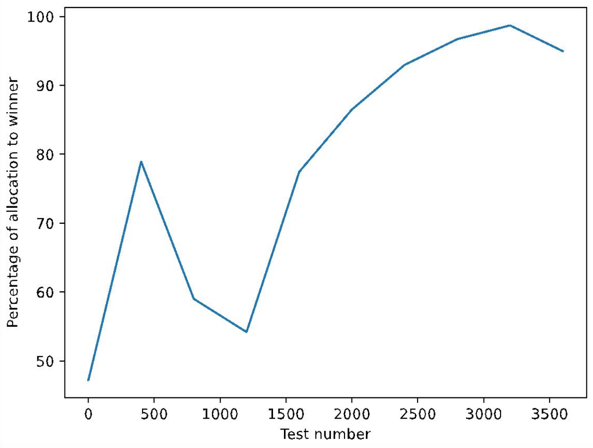
Certaint
y
Factor

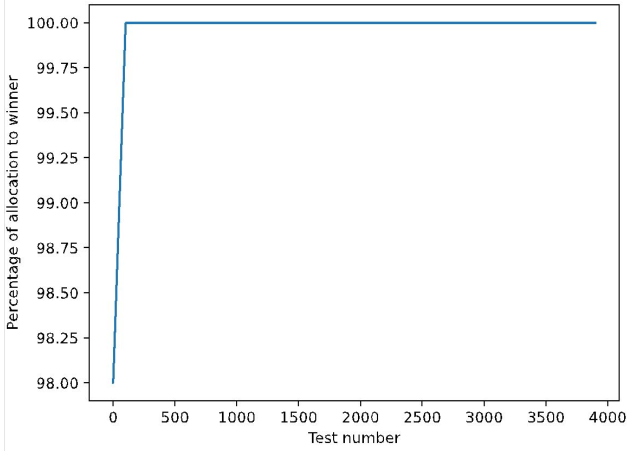
ϵ, α, β > 0
parameters

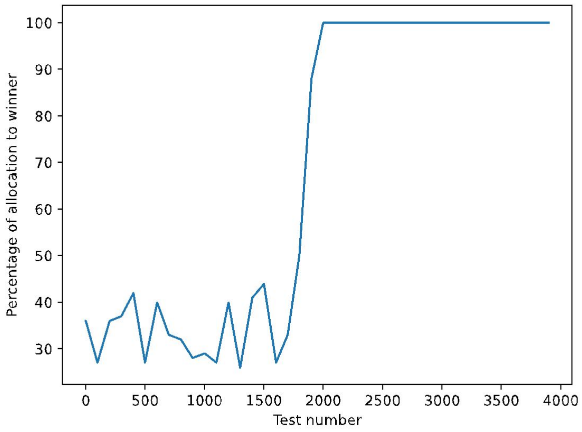


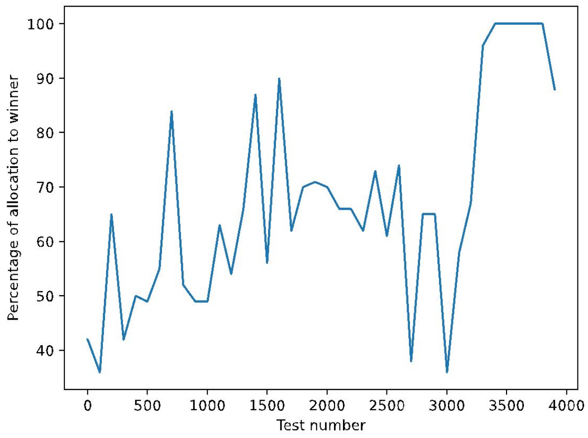
LilUCB Winner Allocation

Results Overview









Conclusions

- Even though currently not popular, a great way to **optimize** decision-making across various sectors using AI and Reinforcement Learning
- A lot of initiatives and research literature on the subject: contextual bandits, collaborative bandit etc.
- The code: https://github.com/tudorcoman/multi-armed-bandits/

More reading on the algorithms…

- Peter Auer, Nicolo Cesa-Bianchi and Paul Fischer, “Finite-time Analysis of the Multiarmed Bandit Problem”, in Machine Learning 47 (2002), pp. 235–256, doi: 10. 1023/A:1013689704352
- Shivaram Kalyanakrishnan, Ambuj Tewari, Peter Auer and Peter Stone, “PAC Subset Selection in Stochastic Multi-armed Bandits”, in ICML’12: Proceedings of the 29th International Conference on Machine Learning, Edinburgh, Scotland: Omnipress, 2012, isbn: 9781450312851
- Kevin Jamieson, Matthew Malloy, Robert Nowak and Sébastien Bubeck, “lil’ UCB : An Optimal Exploration Algorithm for Multi-Armed Bandits”, 2013, doi: 10.48550/ arXiv.1312.7308

Let’s connect!

- LinkedIn: https://linkedin.com/in/tcoman/
- GitHub: https://github.com/tudorcoman/

Q&A

Adobe