Lecture 23: Contextual Bandits

Instructor: Sergei V. Kalinin

Definitions:

- **Objective:** overall goal that we aim to achieve. Not available during or immediately after experiment.

- **Reward:** the measure of success available at the end of experiment

- **Value:** expected reward. Difference between reward and value is a feedback signal for multiple types of active learning

- **Action:** how can ML agent interact with the system

- **State:** information about the system available to ML agent

- **Policy:** rulebook that defines actions given the observed state

Policies:

- **A/B testing:** separate exploration and exploitation
- **Epsilon-greedy:** dedicate part of the budget to exploration
- **UCB:** balance exploration and exploitation giving preference to selections that are more likely to be positive/are less explored
- **Thompson sampling:** draw from probability distribution

Thompson sampling

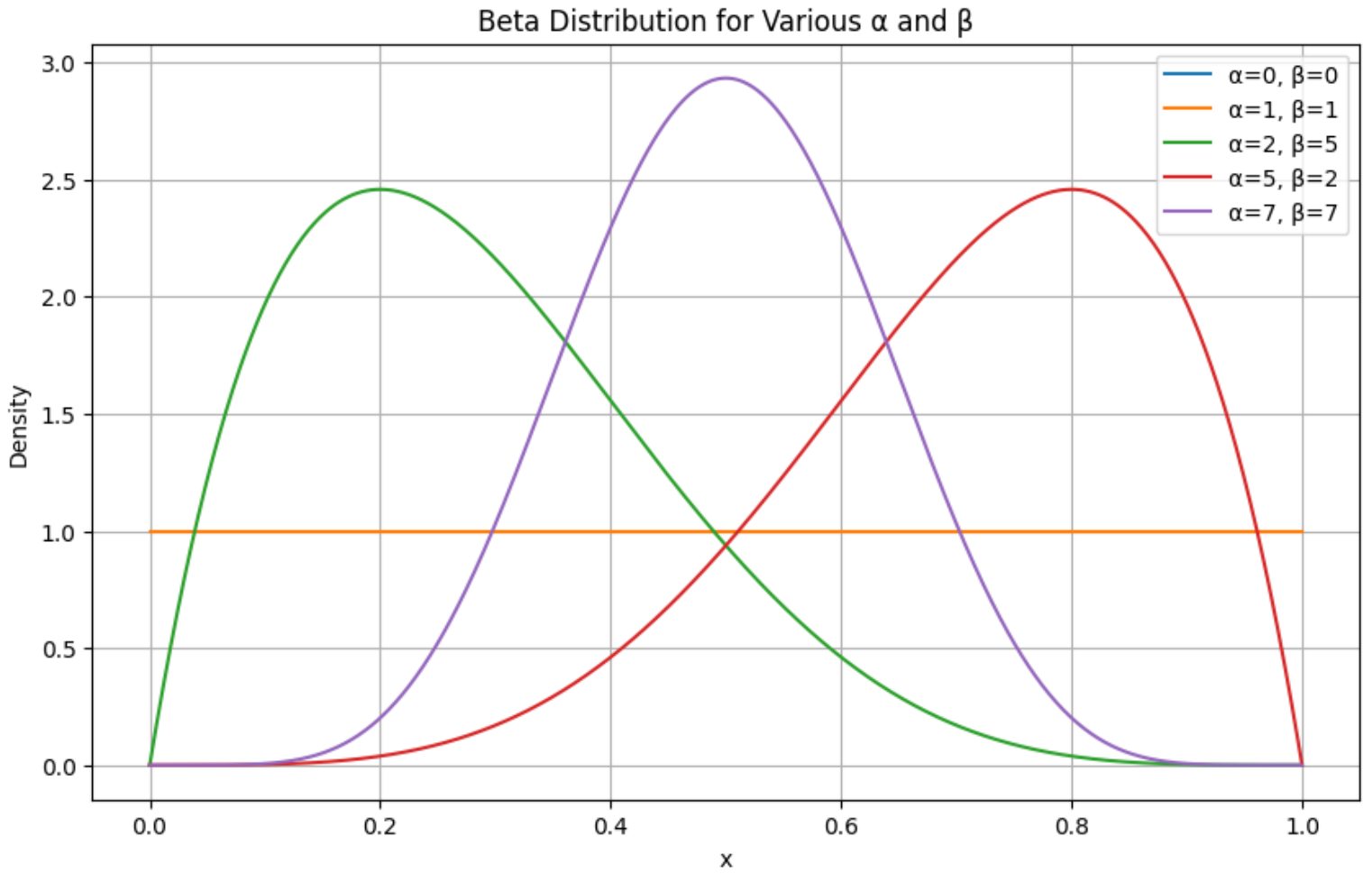
$$p(\theta|X) = \frac{p(X|\theta)p(\theta)}{p(X)}$$

- Initially, we don't have any reason to believe that the parameter is high or low for a given ad. Therefore, it makes sense to assume that $\boldsymbol{\theta_k}$ has a uniform distribution [0,1].
- Assume that we display the ad $\boldsymbol{k}$ and it results in a click. We take this as a signal to update the probability distribution for $\boldsymbol{\theta_k}$ so that the expected value shifts a little bit towards 1.
- As we collect more and more data, we should also see the variance estimate for the parameter shrink. Thompson sampling does it using Bayesian inference.
- This method tells us to take a sample from the posterior distribution of the parameter, $\boldsymbol{p(\theta_k|X)}$ If the expected value of $\boldsymbol{\theta_k}$ is high, we are likely to get samples closer to 1. If the variance is high because ad k has not been selected many times that far, our samples will also have high variance, which will lead to exploration.

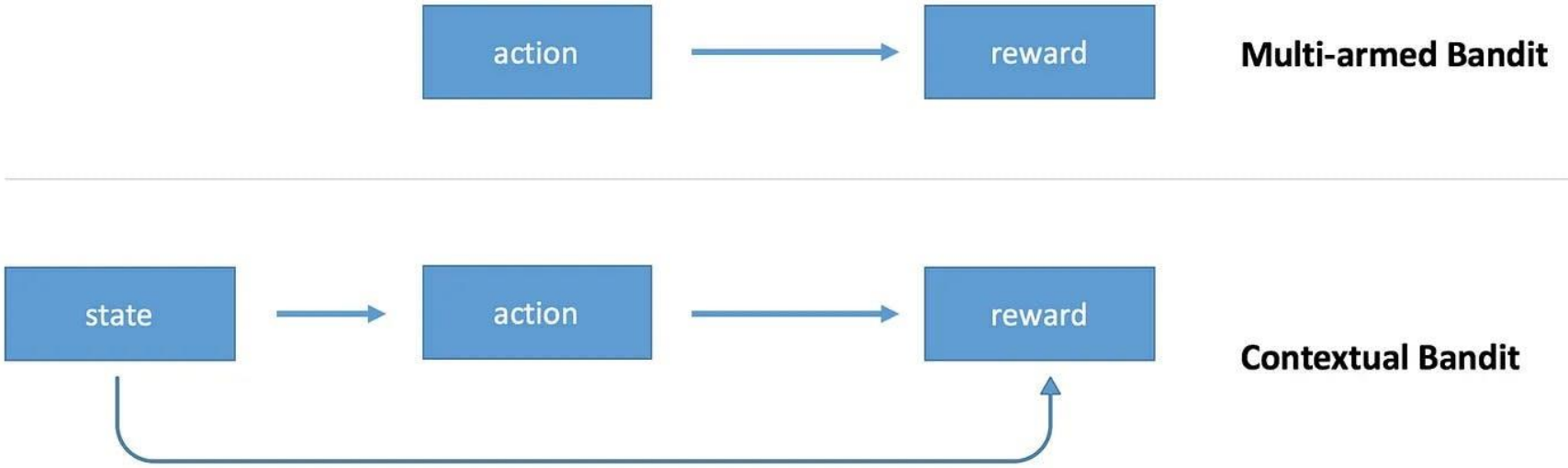
From Enes Bilgin, Mastering Reinforcement Learning with Python, Packt 2020

Thompson sampling: first taste of Bayes

$$p(\theta_k) = \frac{\theta_k^{\alpha_k - 1} \cdot (1 - \theta_k)^{\beta_k - 1}}{B(\alpha_k, \beta_k)}.$$



Contextual Bandits

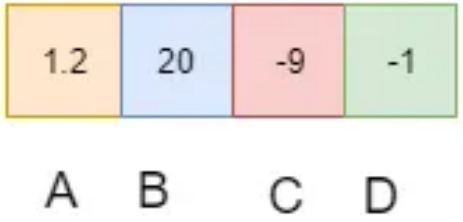


- In contextual bandits the decision-making process is informed by the context.
- The context, in this case, refers to a set of observable variables that can impact the result of the action.
- This addition makes the bandit problem closer to real-world applications, such as personalized recommendations, clinical trials, or ad placement, where the decision depends on specific circumstances.

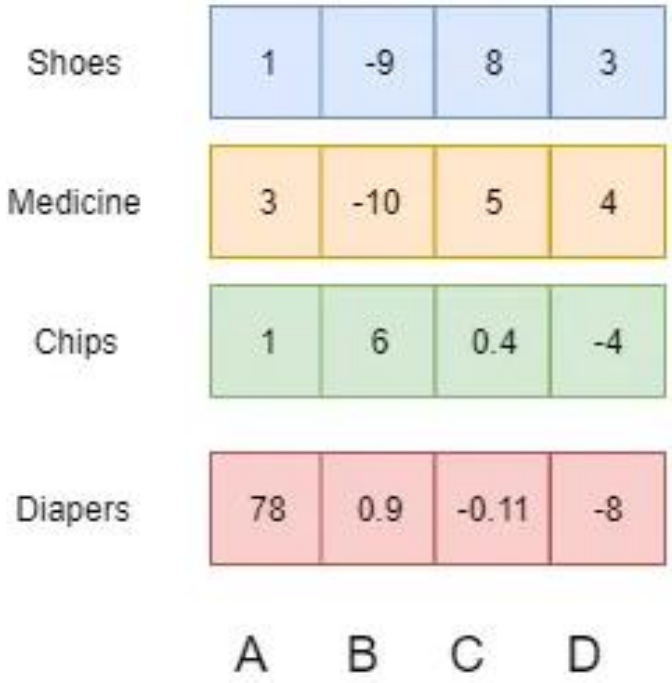
https://hackernoon.com/contextual-multi-armed-bandit-problems-in-reinforcement-learning

Simple example

Bandit



Contextual Bandit



https://medium.com/data-science-in-your-pocket/contextual-bandits-in-reinforcement-learning-explained-with-example-and-codes-3c707142437b

Contextual Bandits

Algorithm 1 Contextual Bandit

Require: Number of actions A
Require: Context feature dimensions d
Require: Context generator
Require: Reward function $\rho(a_t, x_t)$
Require: Learning rate α
Require: Bandit model (e.g., LinUCB, Neural Bandit, Decision Tree Bandit)
  Initialize the bandit model
  for $t = 1$ to T **do**
    Receive the context x_t from the context generator
    Predict the expected reward for each action $\hat{r}(a_t|x_t)$ using the bandit model
    Select the action $a_t = \arg\max_a \hat{r}(a_t|x_t)$
    Receive the reward $r_t = \rho(a_t, x_t)$
    Update the bandit model using the pair (a_t, r_t)
  end for
Ensure: The trained bandit model, The cumulative reward

https://hackernoon.com/contextual-multi-armed-bandit-problems-in-reinforcement-learning

Contextual Bandits

At each time step t, the environment presents a context x_t to the algorithm (often called the agent). Based on this context, the agent chooses an action a_t from a set of possible actions A.

In response to the action, the agent receives a reward r_t. The goal of the agent is to learn a policy π, a mapping from contexts to actions, that maximizes the cumulative reward over time:

$$\max_{\pi} \mathbb{E}\left[\sum_{t=1}^{T} r_t | \pi, D\right] (1)$$

where $D = (x_1, a_1, r_1), ..., (x_T, a_T, r_T)$ is the data (context, action, reward) collected until time T. The expectation is taken over the randomness in the context and rewards.

- The goal of the algorithm is to maximize the cumulative reward over some time horizon.
- In each round, the reward for only the chosen action is observed.
- This is known as partial feedback, which differentiates bandit problems from supervised learning

https://hackernoon.com/contextual-multi-armed-bandit-problems-in-reinforcement-learning

Linear UCB

$$\mathbb{E}[r|x,a] = x^T\theta_a(1)$$

$\mathbb{E}[r|x,a] = x^T\theta_a$ is the expected reward of action a given context x under the linear model
θ_a is the parameter vector for action a.
The objective is to learn θ_a for all actions based on the data.

$\Rightarrow$ At each time step t, given context x_t, LinUCB selects the action a_t that has the maximum UCB:

$$a_t = \arg\max_{a \in A} \left(x_t^T\theta_a + \alpha\sqrt{x_t^T A_a^{-1} x_t} \right)$$

A_a is a matrix that keeps track of the features seen so far for action a,
α is a parameter controlling the trade-off between exploration and exploitation,
the second term is the UCB, which is proportional to the standard deviation of the estimated reward.

Linear UCB Update

Chosen action: $$a_t = \arg\max_{a \in A} \left(x_t^T \theta_a + \alpha \sqrt{x_t^T A_a^{-1} x_t} \right)$$

Model updates: $$A_a = A_a + x_t x_t^T \qquad b_a = b_a + r_t x_t$$

$$\theta_a = A_a^{-1} b_a$$

- LinUCB algorithm is a contextual bandit algorithm that models the expected reward of an action given a context as a linear function
- LinUCB it selects actions based on the Upper Confidence Bound (UCB) principle to balance exploration and exploitation.
- It exploits the best option available according to the linear, but it also explores options that could potentially provide higher rewards, considering the uncertainty in the model's estimates.

Decision Tree cMAB

Formally, a decision tree T defines a partition of the context space X into K disjoint subsets $X_1, X_2, ..., X_K$, each associated with a recommended action a_k.
The reward of action a_k in context x under the decision tree T is:

$$\mathbb{E}[r|x, a_k, T] = \mathbb{E}[r|x \in X_k]$$

$\Rightarrow$ Given a context x, the decision tree selects the action associated with the subset that x falls into.

The objective is to learn the best decision tree that maximizes the expected cumulative reward:

$$\max_T \mathbb{E}\left[\sum_{t=1}^{T} r_t | T, D\right]$$

Where
$D = (x_1, a_1, r_1), ..., (x_T, a_T, r_T)$ is the data (context, action, reward) collected until time T.

Decision Tree Bandit models the reward function as a decision tree, where each leaf node corresponds to an action and each path from the root to a leaf node represents a decision rule based on the context. It performs exploration and exploitation through a statistical framework, making splits and merges in the decision tree based on statistical significance tests.

Neural Network cMAB

- Deep learning models can be used to approximate the reward function in high-dimensional or non-linear cases. The policy is typically a neural network that takes the context and available actions as input and outputs the probabilities of taking each action.

- A popular deep learning approach is to use an actor-critic architecture, where one network (the actor) decides which action to take, and the other network (the critic) evaluates the action taken by the actor.

- For more complex scenarios where the relationship between the context and the reward is not linear, we can use a neural network to model the reward function. One popular method is to use a policy gradient method, such as REINFORCE or actor-critic.

Neural Network cMAB

Let $\pi_\theta(a|x)$ be the policy parameterized by θ, i.e., the probability of selecting action a given context x. The objective is to find θ that maximizes the expected cumulative reward:

$$\max_\theta \mathbb{E}\left[\sum_{t=1}^{T} r_t|\pi_\theta, D\right]$$

$\Rightarrow$ Using the policy gradient theorem, we can update θ iteratively:

$$\theta_{t+1} = \theta_t + \alpha(r_t - b(x_t))\nabla \log \pi_\theta(a_t|x_t)$$

where α is the learning rate, $b(x_t)$ is a baseline function (often the average reward), and the expectation is approximated by sampling actions according to π_θ.

- Neural Bandit uses neural networks to model the reward function, taking into account the uncertainty in the parameters of the neural network.
- It further introduces exploration through policy gradients where it updates the policy in the direction of higher rewards.
- This form of exploration is more directed, which can be beneficial in large action spaces.

LinUCB: The Linear Upper Confidence Bound (LinUCB) model uses linear regression to estimate the expected reward for each action given a context. It also keeps track of the uncertainty of these estimates and uses it to encourage exploration.

Advantages:

- It is simple and computationally efficient.
- It provides theoretical guarantees for its regret bound.

Disadvantages:

- It assumes the reward function is a linear function of the context and action, which may not hold for more complex problems.

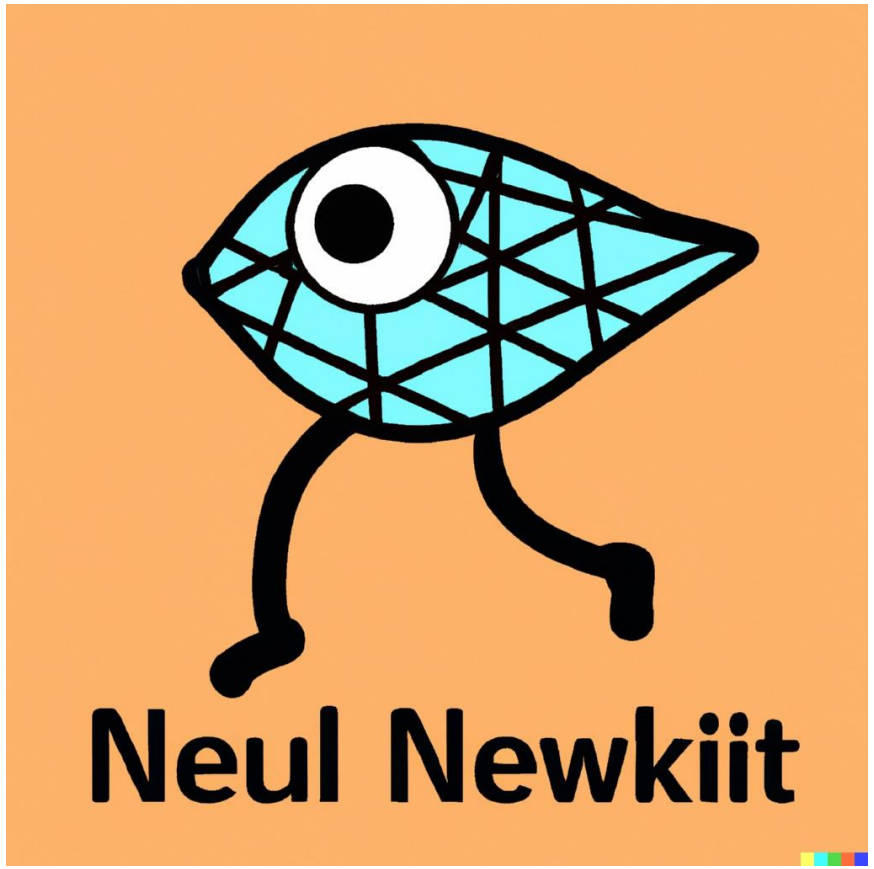
Decision Tree Bandit: The Decision Tree Bandit model represents the reward function as a decision tree. Each leaf node corresponds to an action, and each path from the root to a leaf node represents a decision rule based on the context.

Advantages:

•It provides interpretable decision rules.
•It can handle complex reward functions.

Disadvantages:

•It may suffer from overfitting, especially for large trees.
•It requires tuning hyperparameters such as the maximum depth of the tree.



Neural Bandit: The Neural Bandit model uses a neural network to estimate the expected reward for each action given a context. It uses policy gradient methods to encourage exploration.

Advantages:

- It can handle complex, non-linear reward functions.
- It can perform directed exploration, which is beneficial in large action spaces.

Disadvantages:

- It requires tuning hyperparameters such as the architecture and learning rate of the neural network.
- It may be computationally expensive, especially for large networks and large action spaces.