Suhasini ch

AI DS -A

21110095

Definition

- **Objective:** A news value maximiser: Politically and commercially affiliated media companies are tasked with maximizing the views for certain articles more than others. Build a system that maximises the views for these "aligned" articles.

From my research, When compared to UCB OR Elipson_greedy, I have felt that Thompson sampling concept is a better approach as it uses probability distributions (like the Beta distribution) to model the uncertainty about each option's reward. It adjusts their strategy as they learn more. As the goal is to maximize rewards in a dynamic and uncertain environment.

It Avoids greedy search and ultimately leads to better results.

I have referred this document(https://web.stanford.edu/~bvr/pubs/TS_Tutorial.pdf)

Approach

1. Problem Formulation

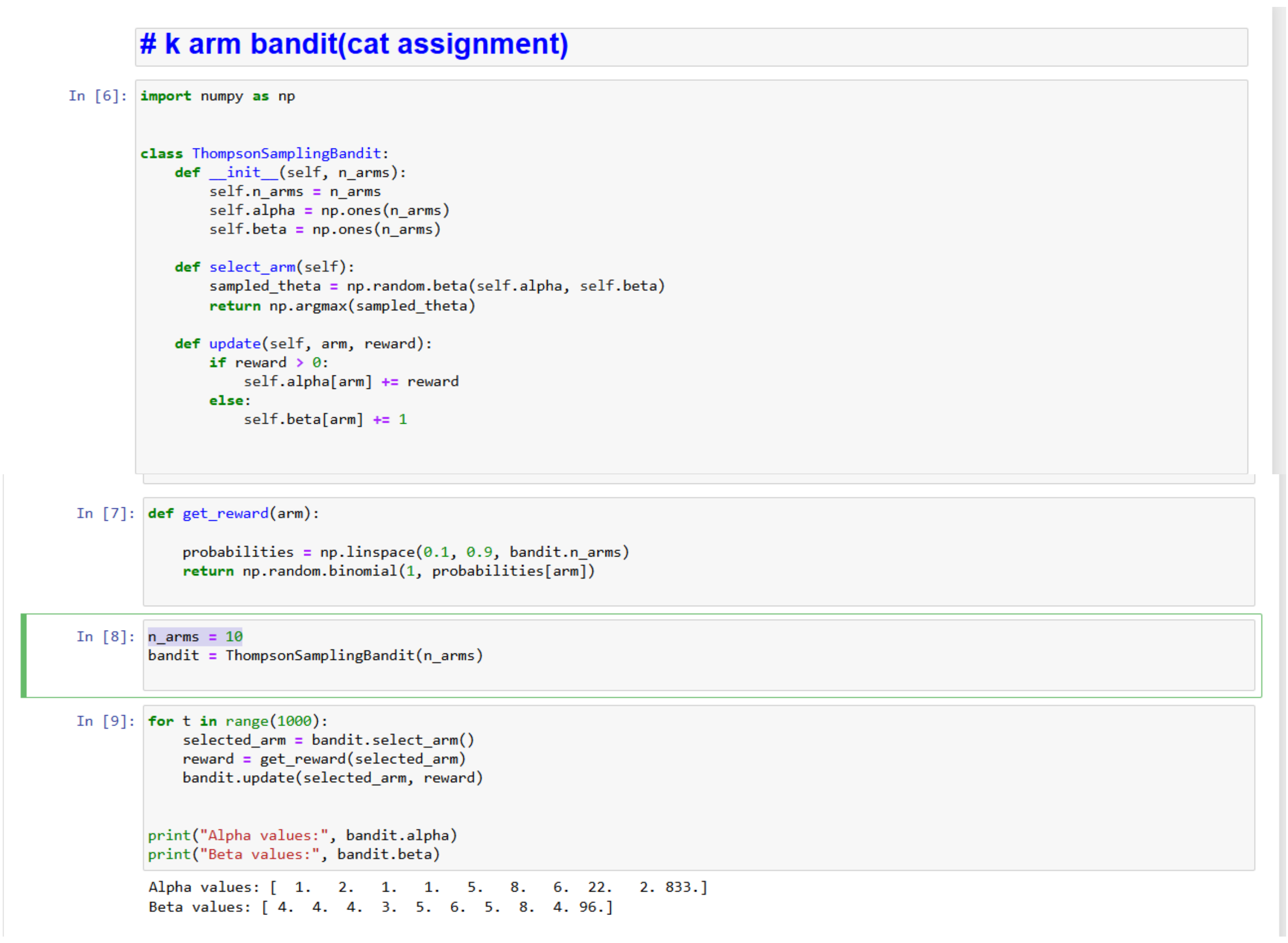
The problem can be formulated as a K-armed bandit problem, where each article is an arm, and the objective is to maximize cumulative rewards (views).

- **Arms (K):** Each arm represents an article available for promotion.
- **Rewards:** The reward is the number of views an article receives after being promoted.
- **Goal:** Maximize the cumulative reward (total views) for aligned articles over time.

2. Thompson Sampling Concept

Thompson Sampling is a method that helps you learn over time which articles to promote by balancing two things:

- **Exploration:** Trying out different articles to gather information.
- **Exploitation:** Using the information you already have to promote the articles that seem to be the best.



k arm bandit(cat assignment)

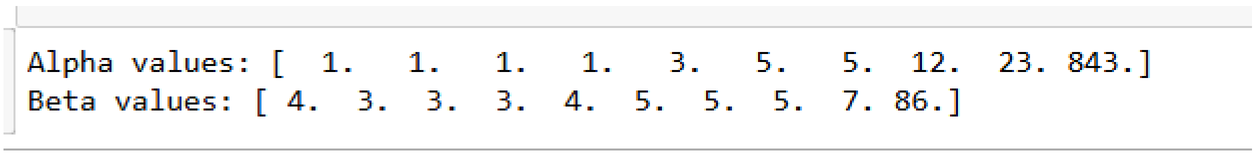
In [6]:
```python
import numpy as np


class ThompsonSamplingBandit:
    def __init__(self, n_arms):
        self.n_arms = n_arms
        self.alpha = np.ones(n_arms)
        self.beta = np.ones(n_arms)

    def select_arm(self):
        sampled_theta = np.random.beta(self.alpha, self.beta)
        return np.argmax(sampled_theta)

    def update(self, arm, reward):
        if reward > 0:
            self.alpha[arm] += reward
        else:
            self.beta[arm] += 1
```

In [7]:
```python
def get_reward(arm):

    probabilities = np.linspace(0.1, 0.9, bandit.n_arms)
    return np.random.binomial(1, probabilities[arm])
```

In [8]:
```python
n_arms = 10
bandit = ThompsonSamplingBandit(n_arms)
```

In [9]:
```python
for t in range(1000):
    selected_arm = bandit.select_arm()
    reward = get_reward(selected_arm)
    bandit.update(selected_arm, reward)


print("Alpha values:", bandit.alpha)
print("Beta values:", bandit.beta)
```

```
Alpha values: [  1.   2.   1.   1.   5.   8.   6.  22.   2. 833.]
Beta values: [ 4.  4.  4.  3.  5.  6.  5.  8.  4. 96.]
```



```
Alpha values: [  1.   1.   1.   1.   3.   5.   5.  12.  23. 843.]
Beta values: [ 4.  3.  3.  3.  4.  5.  5.  5.  7. 86.]
```

Creating the Thompson Sampling Bandit Class

- **ThompsonSamplingBandit Class:** Thompson Sampling to decide which articles to promote.
- **n_arms:** This represents the number of articles you have.
- **alpha and beta:** These are parameters for each article (or arm). They start with a value of 1 for all articles, representing our initial belief about the likelihood of getting views. These values will be updated as the system learns.(initially it is assigned to 1 as we don't know the probability of views of each article).

Selecting an Article to Promote

- **select_arm Function:** This function decides which article to promote based on current knowledge.
- **np.random.beta(self.alpha, self.beta):** For each article, it draws a random value from a probability distribution based on the current alpha and beta values. This gives an estimate of how good each article might be.
- **np.argmax(sampled_theta):** The article with the highest estimated value (the highest sampled number) is chosen to be promoted.

Updating Beliefs Based on Results

- **update Function:** After promoting an article, the system receives feedback (reward). This function updates the system's beliefs about the article's quality.
- **reward > 0:** If the article gets views, it means it was a good choice. The system increases the alpha value for that article, making it more likely to be promoted again.
- **reward = 0:** If the article doesn't get views, the system increases the beta value, making the article less likely to be promoted in the future.

Simulating Rewards

- **get_reward Function:** This simulates whether an article gets views.
- **probabilities:** This is an array of probabilities that represent the chance each article has of getting views. The higher the probability, the more likely the article is to get views.
- **np.random.binomial(1, probabilities[arm]):** This randomly decides whether the article got a view (1) or not (0) based on the probability.

Running the Simulation

- **n_arms = 10:** This sets the number of articles (arms) to 10.
- **bandit = ThompsonSamplingBandit(n_arms):** This creates an instance of the Thompson Sampling system for the 10 articles.
- **for t in range(1000):** This loop simulates promoting articles 1,000 times.
- **selected_arm = bandit.select_arm():** Each time, the system chooses an article to promote.
- **reward = get_reward(selected_arm):** The system simulates whether the promoted article got views.
- **bandit.update(selected_arm, reward):** The system updates its beliefs about that article based on whether it got views.

Finally, it prints the alpha and beta values for each article, which show how confident the system is about each article's potential.

Summary

- **Thompson Sampling** is like a smart decision-maker that learns over time which articles are most likely to get views.
- It tries out different articles (exploration) but also uses what it has learned to focus on the best ones (exploitation).
- Over time, the system becomes better at promoting the articles that will get the most views, helping to maximize the exposure of aligned content.

The system using Thompson Sampling to optimize the selection of articles for promotion, aiming to maximize views. It treats each article as an "arm" in a multi-armed bandit problem, where the algorithm balances between exploring new articles and exploiting those that have previously shown high engagement. By maintaining and updating probability distributions (using alpha and beta parameters) for each article based on the received rewards (views), the system gradually learns which articles are most likely to attract views and prioritizes promoting those, while still occasionally testing less-promoted options to ensure all possibilities are considered.