STDSR Assignment 2

Konovalova Oksana B21-DS-02

o.konovalova@innopolis.univesity

Report

I. Implementation

1. Implementation of the basic method UCB1:

The UCB1 algorithm is a multi-armed bandit algorithm that balances exploration and exploitation in selecting the best arm. It maintains an estimate of the mean reward of each arm and calculates an Upper Confidence Bound (UCB) value for each arm based on its exploration potential and its mean reward estimate. The arm with the highest UCB value is selected for pulling.

The UCB1 class in this code implements the UCB1 algorithm. It has three main functions: *initialize(), select_arm(), and update()*.

The **initialize()** function initializes the counts and values arrays with zeros. The counts array keeps track of the number of times each arm has been pulled, while the values array stores the estimated mean reward for each arm.

The **select_arm()** function selects the arm to pull based on the UCB1 algorithm. If an arm has never been pulled, it is selected. Otherwise, the UCB value is calculated for each arm using the formula: *UCB value = mean reward + exploration bonus* where the exploration bonus is proportional to the square root of the logarithm of the total number of pulls divided by the number of pulls for the current arm. The arm with the highest UCB value is then selected.

The **update()** function updates the counts and values arrays for the selected arm with the new reward obtained. The counts for the selected arm are incremented by one, and the estimated mean reward for the arm is updated using the formula: *new estimate = ((n - 1) / n) * old estimate + (1 / n) * new reward* where n is the new count for the arm.

Overall, the UCB1 algorithm implemented in this code is a simple and effective way to balance exploration and exploitation in selecting the best arm in a multi-armed bandit problem.

2. Implementation of the Thompson Sampling for Bernoulli reward:

The Thompson Sampling algorithm is another approach to solving the multi-armed bandit problem with Bernoulli-distributed rewards. This algorithm is based on Bayesian inference and uses a Beta distribution to model the uncertainty of each arm's underlying probability of success.

The ThompsonSamplingBernoulli class in this code implements the Thompson Sampling algorithm for the multi-armed bandit problem. It also has three main functions: *initialize(), select_arm(), and update().*

The **initialize()** function initializes the counts and values arrays with zeros, and alpha and beta arrays with ones. The counts array keeps track of the number of times each arm has been pulled, while the values array stores the estimated mean reward for each arm. The alpha and beta arrays represent the prior distribution of the arm's probability of success.

The **select_arm()** function selects the arm to pull based on the Thompson Sampling algorithm. For each arm, a random sample is drawn from a Beta distribution with parameters alpha and beta. The arm with the highest sample value is then selected.

The **update()** function updates the counts, values, alpha, and beta arrays for the selected arm with the new reward obtained. The counts for the selected arm are incremented by one. The estimated mean reward for the arm is updated using the same formula as in the UCB1 algorithm. The alpha and beta values for the arm are also updated based on the reward obtained. If the reward is 1, the alpha value is incremented by 1. If the reward is 0, the beta value is incremented by 1.

Overall, the Thompson Sampling algorithm implemented in this code is a Bayesian approach to solving the multi-armed bandit problem with Bernoulli-distributed rewards. It provides a probabilistic estimate of the underlying probability of success for each arm and updates this estimate as new rewards are obtained. This algorithm can be particularly useful in situations where the rewards are sparse and exploration is important.

3. Implementation of the Thompson Sampling for Gaussian reward:

The Thompson Sampling algorithm is another multi-armed bandit algorithm used for balancing exploration and exploitation in selecting the best arm. The ThompsonSamplingGaussian class in this code implements the Thompson Sampling algorithm for Gaussian-distributed rewards.

The ThompsonSamplingGaussian class has three main functions: *initialize(), select_arm(), and update().*

The **initialize()** function initializes the counts and values arrays with zeros, and the mu and sigma arrays with a default value of 1 for each arm, where the counts array keeps track of the number of times each arm has been pulled, while the values array stores the estimated mean reward for each arm.

The **select_arm()** function selects the arm to pull based on the Thompson Sampling algorithm. It samples a value from a normal distribution for each arm with mean and variance estimates based on the current observations. The arm with the highest sampled value is then selected.

The **update()** function updates the counts, values, mu, and sigma arrays for the selected arm with the new reward obtained. The counts for the selected arm are incremented by one, and the estimated mean reward for the arm is updated using the formula: *new estimate = ((n - 1) / n) * old estimate + (1 / n) * new reward* where n is the new count for the arm. The mu and sigma estimates are also updated using the formulas: *new mu = (old mu * old sigma^2 + new reward * new sigma^2) / (old sigma^2 + new sigma^2)* and *new sigma = sqrt(1 / (1 / old sigma^2 + 1))*

Overall, the Thompson Sampling algorithm implemented in this code is a useful alternative to UCB1 for multi-armed bandit problems with Gaussian-distributed rewards. It provides a simple way to balance exploration and exploitation by sampling from a normal distribution for each arm, using the current observations to estimate the mean and variance parameters.

4. Implementation of the UCB for Gaussian reward:

The UCB for Gaussian algorithm is a variant of the UCB algorithm that is designed for multi-armed bandit problems where the rewards are Gaussian-distributed. Like UCB, it balances exploration and exploitation in selecting the best arm to pull.

The UCB for Gaussian class in this code implements the UCB for Gaussian algorithm. It has three main functions: *initialize(), select_arm(), and update().*

The **initialize()** function initializes the counts and values arrays with zeros, where the counts array keeps track of the number of times each arm has been pulled, while the values array stores the estimated mean reward for each arm.

The **select_arm()** function selects the arm to pull based on the UCB for Gaussian algorithm. If an arm has never been pulled, it is selected. Otherwise, the UCB value is calculated for each arm using the formula: *UCB value = mean reward + exploration bonus* where the exploration bonus is proportional to the square root of the logarithm of the total number of pulls divided by the number of pulls for the current arm, and a constant c that can be set by the user. The arm with the highest UCB value is then selected.

The **update()** function updates the counts and values arrays for the selected arm with the new reward obtained. The counts for the selected arm are incremented by one, and the estimated mean reward for the arm is updated using a weighted average of the previous estimate and the new reward.

Overall, the UCB for Gaussian algorithm implemented in this code is a powerful tool for balancing exploration and exploitation in selecting the best arm in a multi-armed bandit problem with Gaussian-distributed rewards.