Final Project

Anonymous Authors[1]

Abstract

We mainly investigate the background, algorithms, application of the algorithms, and difficulties for the multi-armed bandit problem. In addition, theorems and calculations about regret are also included. Beyond the theoretical point of view, we practically applied two algorithms through R code: Thompson Sampling and Upper Confidence Bound and visualized the resulting plots. Based on this, we identify the limitations of these algorithms, and discuss the algorithms and their future advancements.

1. Problem Statement

1.1. Background

Multi-armed bandit problem is a classic problem in machine learning which originates from the early slot machines in the casino. Because the gambler is hard to win rewards from the slot machines which have one arm per machine and usually lose lots of money, the slot machines are called multi-armed bandits in this case. For this specific problem, we assume that a gambler tries to win rewards from lots of slot machines and tries to maximize the reward by choosing the next slot machine or stopping gambling according to the results he got previously. How to maximize the rewards is what this problem is about. Statistically, we have a series of stochastic processes for choosing and we need to archive the maximum rewards in a set of actions where every action we get a different amount of rewards.

In the article that we reviewed, the multi-armed bandit problem is under the stochastic environment. It means that the rewards given by the slot machines are not controlled by the owner to gain profits but are statistically stochastic. This environment gives this problem a simpler simulation instead of the problem in real life where the machines can be controlled depending on the action of the gambler which is under the adversarial environment.

Except for the stochastic environment, the "multi-armed bandit problems with complex actions over a set of basic arms" (Gopalan et al., 2014) is also the background of this article. The complex actions' meaning is related to further exploration of the stochastic multi-armed bandit problem. Take the gambler as an example, the gambler needs to explore the slot machines at first to get the initial data. After the gambler got the initial data, he would act according to the initial data to gain the maximum reward, at the same time this action would be a new data point added to the data set. In this process, the actions which include getting the initial data and obtaining the new data point are considered complex actions that are based on a subset of all the basic arms.

The multi-armed bandit problem with complex actions can also be seen as a "complex online problem" (Gopalan et al., 2014) as shown in the article. The "complex online problem" (Gopalan et al., 2014) is a short term for online machine learning multi-armed bandit problem with complex actions in our understanding. As we mentioned above, the gambler would update the data set and choose the next move according to the updated data set, which fits the characteristic of online machine learning – the data is always updating. Also it is complex because the rewards may not be independent. Then in this case, what does the title of this article mean and the content of it is consistent – how does Thompson sampling behave in the stochastic multi-armed bandit problem with complex actions.

1.2. Algorithms

Many algorithms were created and improved in the past years for the multi-armed bandit problem, including:

- Thompson sampling Algorithm
- Upper Confidence Bound Algorithm (UCB)
- Stochastic and Adversarial Optimal Algorithm (SAO)
- ...

We mainly focus on the Thompson sampling algorithm with respect to the article we reviewed in our report, also we

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

will discuss a bit about the Upper Confidence Bound Algorithm (UCB) and the Stochastic and Adversarial Optimal Algorithm (SAO).

1.3. Practical Meanings

In the meanwhile, the complex multi-armed bandit problem has practical meanings which means its algorithms can be applied to real-life situations. In contrast to our gambler example above, the applications in our daily lives are usually not the customers who use strategy but the merchants who do. As Thompson Sampling for Complex Online Problems (Gopalan et al., 2014) stated, the limitations that rewards of the multi-armed bandit are independent is severe such that some applications may not be solved well. The three examples, respectively, web advertising, job scheduling and routing have shown how the rewards are dependent uniquely and affect the complex action. We conclude these situations into three categories:

- Repeated rewards: People tend to not select the machines that have the same rewards. (Web Advertising: people tend to enter only one of the advertisements that advertise the same product.)
- Finite actions limited: If there exists a limited number of the actions due to the finite resource we own (time, money), people need to calculate the total cost when deciding the complex action. (Job Scheduling: Time limitation need to be considered when execute the complex action.)
- Interactive rewards: Sometimes the rewards given by each machine could affect each other. (Routing: If there are many routes from the start point to the destination, one route have a traffic jam then the others do not have.)

1.4. Intuitions

As we mentioned above, the complex online problem has many applications and what role does Thompson sampling play in those real-life situation is important indeed. The most difficult parts of finding out how the Thompson sampling behaves in those situations are the simulations of the not-independent rewards, what algorithm should we choose for comparison and the changes we need for the Thompson sampling to fit the complex online problem.

2. Main Result

2.1. Conclusion

Overall, the frequentist regret bounds of Thompson sampling is calculated by the new method created in Thompson Sampling for Complex Online Problems (Gopalan et al., 2014) so that it is a strong evidence that Thompson sampling is the optimal option for the stochastic multi-armed bandit problem with complex actions.

2.2. Theorem Contribution

The authors gave one major theorem in this article:

„

Theorem 2.1. *(General Regret Bound for Thompson Sampling) Under Assumptions 1-3, the following holds for the Thompson Sampling algorithm. For* $\delta, \epsilon \in (0, 1)$*, there exists .....* $< \frac{1+\epsilon}{1-\epsilon} logT$

”(Gopalan et al., 2014)

This theorem gives the regret bound for Thompson sampling for the complex online problem. The proof of it uses lots of different material and a recent paper on developed self-normalized concentration inequality (Abbasi-Yadkori et al., 2011). Later we will use a simpler way to demonstrate the goal of this theorem.

Along with several corollaries and propositions, the authors not only demonstrate the regret bounds in different situation where the feedback is full or maximum, but also showed the regret improvement which is related to marginal KL-divergences.

- Corollary 1 (Regret for playing subsets of basic arms, Full feedback).
- Corollary 2 (Regret for playing subsets of basic arms, MAX feedback).
- Proposition 2 (Explicit Regret Improvement Based on Marginal KL-divergences).

3. Examples and Counterexamples

In this part, we apply two algorithms, Thompson Sampling and Upper Confidence Bound, to solve the multi-armed bandit problem by using R. For the counterexample part, these two algorithms also obviously reflect the characteristics that it is difficult to be implemented outside the stochastic environment. The later discussion and future work part will continue to describe it and the improvement of it.

In the simulation step, we simulate 10 bandits which the probability of getting reward is between 0 and 0.4. Each bandit generates 10,000 binomial data according to the preset probability which means actions are finite limited. 0 means no reward, 1 means there is a reward. Out of ten bandits, we set the seventh bandit to have the highest probability, which is 0.4. To further test the algorithm, we set the eighth bandit's probability to be 0.399 which is extremely closed to the seventh bandit. We also set 2 bandits(3-rd, 5-th) has the

```r
# simulate 10 bandit with different probabilities
prob.interact = runif(1,min=0,max=0.35)
bandit_one <- rbinom(10000, size = 1, prob = 0.2)
bandit_two <- rbinom(10000, size = 1, prob = 0.5-prob.interact)
bandit_three <- rbinom(10000, size = 1, prob = 0.3)
bandit_four <- rbinom(10000, size = 1, prob = 0.33)
bandit_five <- rbinom(10000, size = 1, prob = 0.3)
bandit_six <- rbinom(10000, size = 1, prob = prob.interact)
bandit_seven <- rbinom(10000, size = 1, prob = 0.4)
bandit_eight <- rbinom(10000, size = 1, prob = 0.399)
bandit_nine <- rbinom(10000, size = 1, prob = 0.38)
bandit_ten <- rbinom(10000, size = 1, prob = 0.37)
combinedDf <- cbind(bandit_one, bandit_two,bandit_three, bandit_four,
                    bandit_five, bandit_six,bandit_seven, bandit_eight,
                    bandit_nine, bandit_ten)
```

Figure 1. This figure shows the simulation step.

same probability of 0.3 which shows the repeated reward. We randomly generate a probability between 0 and 0.35 for second bandit. And set sixth bandit probability related to second bandit by 0.5 - prob(2-nd bandit).

3.1. Thompson Sampling

```r
# Implementing Thompson Sampling
N = 10000
d = 10
bandit_selected_ts = integer()
count_reward_1 = integer(d)
count_reward_0 = integer(d)
total_reward = 0
regret_global = vector()
total_regret = vector()
for (n in 1:N) {
  bandit_max_random = 0
  max_random = 0
  for (i in 1:d) {
    random_selected = rbeta(n = 1,
                            shape1 = count_reward_1[i] + 1,
                            shape2 = count_reward_0[i] + 1)
    if (random_selected > max_random) {
      max_random = random_selected
      bandit_max_random = i
    }
  }
  bandit_selected_ts = append(bandit_selected_ts, bandit_max_random)
  reward = combinedDf[n, bandit_max_random]
  if (reward == 1) {
    count_reward_1[bandit_max_random] = count_reward_1[bandit_max_random] + 1
  } else {
    count_reward_0[bandit_max_random] = count_reward_0[bandit_max_random] + 1
  }
  total_reward = total_reward + reward
}

#visualization
histogramOfbandit_selected_ts <- function(bandit_selected_ts){

  png("ts_bandit_selected.png")

  hist(bandit_selected_ts,
       col = 'darkgreen',
       main = 'Histogram(TS Algorithm) of bandit selections',
       xlab = 'bandit',
       ylab = 'Count of selected')


  dev.off()

}

histogramOfbandit_selected_ts(bandit_selected_ts)
```

Figure 2. This figure shows the model and visualization parts(SuperDataScience).

- **Step 1**: Simulate 10 bandits which the probability of getting reward is between 0 and 0.4. Each bandit generates 10,000 binomial data according to the preset probability. 0 means no reward, 1 means there is a reward. Out of ten bandits, we set the seventh bandit to have the highest probability, which is 0.4. To further test the algorithm, we set the eighth bandit's probability to be 0.399 which is extremely closed to the seventh bandit.

- **Step 2**: Start construct model based on data from step 1. In round n, we count N_1 which is the number of times the reward is 1 and N_0 which is the number of times the reward is 0 before round n.

- **Step 3**: For each bandit, we draw a random value according to beta distribution: $Beta(N_1 + 1, N_0 + 1)$.

- **Step 4**: We choose the bandit which has the highest value in step 3 among all bandits. Then update the reward in step 2 and beta distribution in step 3 for this selected bandit.

- **Step 5**: Loop the second to the fourth step 10,000 times to calculate which bandit is selected the most times and visualized. The bandit with largest count should be the bandit with highest probability of getting reward.

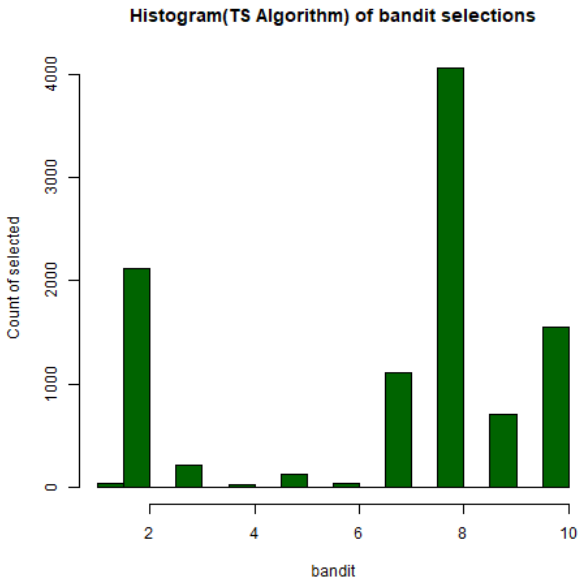

Figure 3. This figure shows that the seventh bandit was chosen the most times by the Thompson Sampling, which is in accordance with our preset probability.

```r
# Implementing UCB
N = 10000
d = 10
bandit_selected = integer(0)
numbers_of_selections = integer(d)
sums_of_rewards = integer(d)
total_reward = 0
for (n in 1:N) {
  bandit_max_random = 0
  max_upper_bound = 0
  for (i in 1:d) {
    if (numbers_of_selections[i] > 0) {
      average_reward = sums_of_rewards[i] / numbers_of_selections[i]
      delta_i = sqrt(3/2 * log(n) / numbers_of_selections[i])
      upper_bound = average_reward + delta_i
    } else {
        upper_bound = 1e400
    }
    if (upper_bound > max_upper_bound) {
      max_upper_bound = upper_bound
      bandit_max_random = i
    }
  }
  bandit_selected = append(bandit_selected, bandit_max_random)
  numbers_of_selections[bandit_max_random] =
    numbers_of_selections[bandit_max_random] + 1
  reward = combinedDf[n, bandit_max_random]
  sums_of_rewards[bandit_max_random] =
    sums_of_rewards[bandit_max_random] + reward
  total_reward = total_reward + reward
}
```

```r
#visualization
histogramOfbandit_selected <- function(bandit_selected){

  png("ucb_bandit_selected.png")

  hist(bandit_selected,
       col = 'darkgreen',
       main = 'Histogram(UCB algorithm) of bandit selections',
       xlab = 'bandit',
       ylab = 'Count of selected')


  dev.off()

}

histogramOfbandit_selected(bandit_selected)
```

Figure 4. This figure shows the model and visualization parts(SuperDataScience).

3.2. Upper Confidence Bound

- **Step 1**: Simulate 10 bandits which the probability of getting reward is between 0 and 0.4. Each bandit generates 10,000 binomial data according to the preset probability. 0 means no reward, 1 means there is a reward. Out of ten bandits, we set the seventh bandit to have the highest probability, which is 0.4. To further test the algorithm, we set the eighth bandit's probability to be 0.399 which is extremely closed to the seventh bandit.

- **Step 2**: Start construct model based on data from step 1. In round n, we count $N_i(n)$ which is the number of i-th bandit was selected and $R_i(n)$ which is the total reward of i-th bandit before round n.

- **Step 3**: Calculate the average reward before round n by: $\overline{r_i(n)} = \frac{R_i(n)}{N_i(n)}$

- **Step 4**: Calculate confidence bound in this round :$[\overline{r_i}(n) - \Delta_i, \overline{r_i}(n) + \Delta_i]$

  where $\Delta_i = \sqrt{\frac{3}{2}\frac{log(n)}{N_i(n)}}$

- **Step 5**: Choose the bandit with the largest upper confidence bound which is $\overline{r_i}(n) + \Delta_i$

- **Step 6**: Loop the second to the fifth step 10,000 times to calculate which bandit is selected the most times and visualized. The bandit with largest count should be the bandit with highest probability of getting reward.

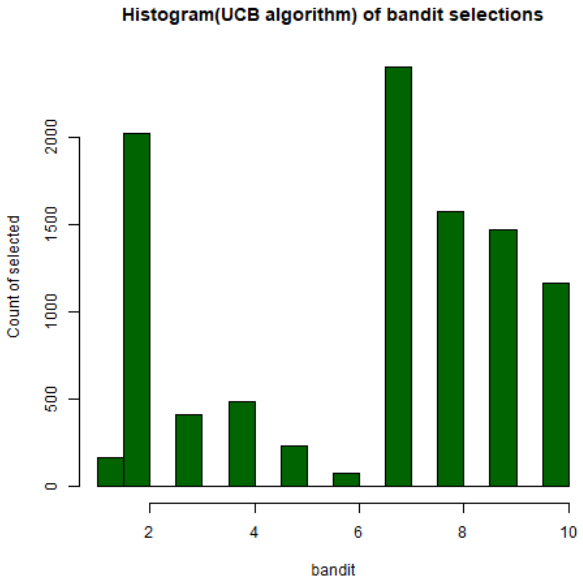

Figure 5. This figure shows that the eighth bandit was chosen the most times by the algorithm, which is in accordance with our preset probability.

From 2 figures, we find that both models fit well in multi-armed bandit problem in stochastic environment.

4. Limitation

Under the stochastic environment, Thompson Sampling seems to be the optimal solution for the multi-armed bandit problem with complex action, while it could be different under the adversarial environment. In another paper, The Best of Both Worlds: Stochastic and Adversarial Bandits (Bubeck et al., 2012), the Stochastic and Adversarial Optimal algorithm (SAO) was introduced. This is an algorithm with optimal regret under both the stochastic and adversarial environment.

Even though the Thompson Sampling algorithm itself is easy to practice on code, the method to prove the regret bound is hard to conduct. The use of a simple particle filter (Ristic et al., 2004) and the experiment of evaluating the performance of the Thompson Sampling algorithm under two kinds of complex bandit scenarios is hard to handle

technically. That's the reason why we chose a much simpler way to show the performance of the Thompson Sampling algorithm in this problem with direct visualization and the easily-understandable explanation.

Back to the adversarial environment, the Thompson Sampling algorithm would not be the optimal method anymore since it need the process of the machines to be statistically stochastic. Otherwise, the Bayesian assumptions we made is no longer righteous.

5. Future Directions

We believe the complex bandits problem would have more categories as we stated above, respectively, Repeated rewards, Finite actions limited and Interactive rewards. At that time, we would have more unique and detailed optimal methods analysis for each one of them instead of the Thompson Sampling algorithm in general. Also, as modern industries growing, more trivial needs can be satisfied with the help of these algorithms, which we believe this direction in meaningful and practical in both ways.

6. References

Gopalan, A., Mannor, S., & Mansour, Y. (2014). Thompson sampling for complex online problems. Retrieved April 16, 2022, from http://proceedings.mlr.press/v32/gopalan14.pdf

Bubeck S. & Slivkins, A. (2012). The Best of Both Worlds: Stochastic and Adversarial Bandits. Retrieved April 16, 2022, from http://jmlr.org/proceedings/papers/v23/bubeck12b/bubeck12b.pdf

Abbasi-Yadkori, Yasin, Pal, David, and Szepesvari, Csaba. Improved algorithms for linear stochastic bandits. In Advances in Neural Information Processing Systems 24, pp.2312–2320, 2011.

Garivier, A. and Capp e, O. The KL-UCB algorithm for bounded stochastic bandits and beyond. Journal of Machine Learning Research - Proceedings Track, 19:359–376, 2011.

SuperDataScience. (n.d.). Retrieved April 17, 2022, from https://www.superdatascience.com/pages/

References

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.