Stochastic Optimization Assignment 3

Aytek Mutlu

December 09, 2019

1 Introduction

In this assignment, 2 Bernoulli arms with uniform priors are considered. The objective is to find the maximum discounted number of successes using those arms. For this purpose, variety of different policies are applied and compared in the following sections.

2 Gittins Index

In order to apply Bayesian policy, Gittins Indices are calculated for all possible states offline. Considering the expensive computation of the mentioned index, the state space is limited to (40,40), which means the maximum number of success and failure is bound to 40. As required in the assignment, discount factor is used as 0.95 i.e. $\alpha = 0.95$

For each and every state in state space, an iteration is looped until value matrix converges (i.e. change in consecutive iterations are less than 10^{-3}). Within this iteration, value of each and every state is updated with the following formula:

$$V_t[k,l] = max(p + \alpha \cdot (p \cdot V_{t-1}[k+1,l] + (1-p) \cdot V_{t-1}[k,l+1]), V_{t-1}[a,b])$$

where $p = k/(k+l)$ and state (a,b) is the state where restarting option applies to. This means (a,b) is the state for which Gittins Index is calculated.

When $V_t[a,b] - V_{t-1}[a,b] < 0.001$, iteration stops and Gittins index for state (a,b) is calculated as follows:

$$m(a,b) = V[a,b] \cdot (1-\alpha)$$

This procedure is applied for all states in state space, and the following Gittins index table is obtained (Only first 10 successes are 10 failures are illustrated for visual concerns):

l \ k	1	2	3	4	5	6	7	8	9	10
1	0.7583	0.8354	0.8710	0.8924	0.9069	0.9175	0.9256	0.9321	0.9374	0.9418
2	0.5570	0.6776	0.7409	0.7812	0.8095	0.8308	0.8473	0.8605	0.8715	0.8807
3	0.4313	0.5591	0.6359	0.6868	0.7246	0.7533	0.7762	0.7947	0.8102	0.8234
4	0.3465	0.4731	0.5527	0.6101	0.6530	0.6864	0.7138	0.7362	0.7550	0.7714
5	0.2872	0.4078	0.4875	0.5465	0.5927	0.6294	0.6593	0.6845	0.7061	0.7246
6	0.2440	0.3567	0.4355	0.4941	0.5413	0.5800	0.6120	0.6391	0.6622	0.6824
7	0.2112	0.3169	0.3926	0.4510	0.4975	0.5369	0.5703	0.5987	0.6233	0.6447
8	0.1857	0.2843	0.3567	0.4141	0.4609	0.4994	0.5333	0.5627	0.5882	0.6106
9	0.1653	0.2576	0.3269	0.3826	0.4287	0.4675	0.5005	0.5303	0.5565	0.5796
10	0.1488	0.2353	0.3015	0.3552	0.4005	0.4390	0.4723	0.5012	0.5278	0.5514

Table 1: Gittins Indices for Bernoulli Arms with k success and l failures

3 Policies

For the sake of clear comparison, all policies are implemented with same sample space of $N = 40$. Each policy calculates the discounted number of successes and in order to come up with the expected discounted number of successes, all policies are run for $N_{simulation} = 1000$ times and results are averaged. Then, the results are compared in Section 4.

3.1 Full Information

This policy is the base policy where it is assumed that the priors of bandits are known and therefore better arm is selected all the time. Of course, a different arm with a different given prior is selected at each simulation. Here are the steps for the policy:

1. Pick two priors for 2 arms (either given manually or use random numbers between 0 and 1)
2. Select the arm with larger prior and use the same selected arm for 40 times
3. Simulate each time based on given prior of the selected arm (i.e. $x = 1$ if trial is a success and vice versa)
4. Update reward with $r = r + \alpha^t \cdot x$

3.2 Bayesian using Gittins Index

Bayesian policy bases its methodology on Gittins Index table calculated offline. Conceptually, it selects the arm with higher Gittins index given its state at each iteration. Here are the steps for the policy:

1. Randomly pick an initial arm
2. Simulate selected arm (initially with 0.5 success probability, and then use posterior distribution in further iterations i.e. no. of successes / no. of trials)

3. Update state of the selected arm (i.e. 1 success 0 failure etc.)
4. Update reward with $r = r + \alpha^t \cdot x$
5. Compare Gittins indices of both arms given their states and then select the arm with higher Gittins index
6. Go to step 2 and iterate for 40 times

3.3 Thompson Sampling

Thompson sampling follows a very similar procedure with Bayesian approach where the only difference is that the arm at each iteration is selected based on their posterior Beta distributions. Here are the steps:

1. Randomly pick an initial arm
2. Simulate selected arm (initially with 0.5 success probability, and then use posterior distribution in further iterations i.e. no. of successes / no. of trials)
3. Update state of the selected arm (i.e. 1 success 0 failure etc.)
4. Update reward with $r = r + \alpha^t \cdot x$
5. Compare posterior Beta distributions of both arms given their states and then select the arm with higher Beta distribution
6. Go to step 2 and iterate for 40 times

3.4 Greedy Q-Learning

Greedy Q-Learning policy is a stateless approach where the policy always exploits the arm with larger Q value. Here are the steps for Greedy Q-Learning:

1. Initialize Q values with 0 for both arms
2. Provide 2 priors for arms
3. Select the arm with larger Q-value
4. Simulate selected arm with its prior
5. Update reward with $r = r + \alpha^t \cdot x$
6. Update Q value of the selected arm with $Q[arm] = (1/t) \cdot \alpha^t \cdot x + (1 - (1/t)) \cdot Q[arm]$
7. Go to step 3 and iterate for 40 times

3.5 Optimistic Q-Learning

Optimistic Q-learning is exactly same with Greedy Q-Learning where the only difference is the initial values for Q. Higher values are used during initialization and the rest of the process is identical.

1. Initialize Q values with 1 for both arms
2. Provide 2 priors for arms
3. Select the arm with larger Q-value
4. Simulate selected arm with its prior
5. Update reward with $r = r + \alpha^t \cdot x$
6. Update Q value of the selected arm with $Q[arm] = (1/t) \cdot \alpha^t \cdot x + (1 - (1/t)) \cdot Q[arm]$
7. Go to step 3 and iterate for 40 times

3.6 ϵ-Greedy Q-Learning

ϵ-Greedy Q-Learning explores with ϵ probability and exploits with the rest of the probability. This means that policy picks a random arm during exploration and picks the arm that has maximum Q value during exploitation.

1. Initialize Q values with 0 for both arms
2. Provide 2 priors for arms
3. Select an arm either randomly with ϵ probability or the arm with larger Q-value with 1- ϵ probability
4. Simulate selected arm with its prior
5. Update reward with $r = r + \alpha^t \cdot x$
6. Update Q value of the selected arm with $Q[arm] = (1/t) \cdot \alpha^t \cdot x + (1 - (1/t)) \cdot Q[arm]$
7. Go to step 3 and iterate for 40 times

4 Conclusion

Only parameter that would affect the outcome of any policy is ϵ in ϵ-Greedy Q-Learning. For this purpose, an experiment is run with different ϵ values and Figure 1 is obtained.

As it can be seen, $\epsilon = 0.3$ is the optimal point for the objective. The comparison of policies are shown in Table 2. Not surprisingly, Full Information provided the maximum number, as it treats on the provided information.

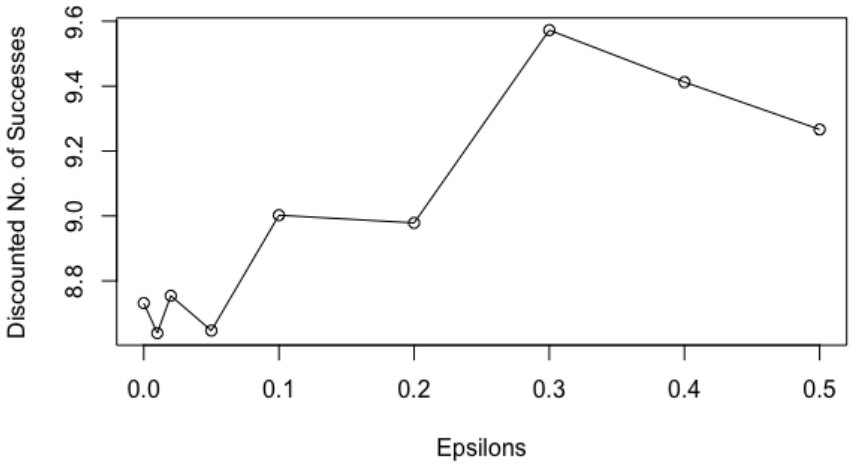

Figure 1: Discounted Number of Successes with changing ϵ values

Bayesian policy with Gittins Index follows the full information, yet its computational hassle makes it less favorable. Then, Thompson Sampling is very successful and computationally not heavy, as well.

Among stateless approaches, Greedy Q-learning is the worst as it focuses too much on exploitation. Its optimistic version offers a better result, as it is initialized with higher values. Finally, ϵ-Greedy Q-Learning performs moderately well as it looks for a balance between exploration and exploitation.

Policy	**Discounted No. of Successes**
Full Information	11.58
Bayesian with Gittins Index	10.72
Thompson Sampling	10.08
Greedy Q-Learning	8.51
Optimistic Q-Learning	9.79
ϵ-Greedy Q-Learning	9.28

Table 2: Objective Results of Policies

5 Code

```
1
2
3  #Gittins index calculation function
4  Gittins ← function(alpha,N){
5    m = matrix(c(0),N,N)
6    for(a in 1:N){
7      for(b in 1:N){
8        V = matrix(c(0),N,N)
9        diff = 1.0
10       while(diff > 0.001){
11         V_temp  = V
12         for(k in a:N){
13           for(l in b:N){
14             p = (k)/(k+l)
15             V[k,l] = max(p + alpha * (p * V_temp[min(k+1,N),
                   l] + (1-p) * V_temp[k,min(l+1,N)]), V_temp[a,
                   b])
16           }
17         }
18         diff = V[a,b] - V_temp[a,b]
19       }
20       m[a,b] = V[a,b]*(1-alpha)
21     }
22   }
23 return(m)
24 }
25
26 FullInformation ← function(alpha,N){
27   p_bandits = runif(2)
28   arm = which.max(p_bandits)
29   r=0
30   for(t in 1:N){
31     p_arm = p_bandits[arm]
32     x = rbinom(1,1,p_arm)
33     r = r + alpha^(t-1) *x
34       }
35   return(r)
36 }
37
38 #Bayesian with Gittins Index
39 Bayesian ← function(alpha,N){
40   arm_states = matrix(c(0),2,2)
41   p_bandits = runif(2)
42   arm = which.max(p_bandits)
43   r=0
44   for(t in 1:N){
45     p_arm = (arm_states[arm,2]+1)/(rowSums(arm_states)[arm
```

```
        ]+2)
    x = rbinom(1,1,p_arm)
    arm_states[arm,x+1] = arm_states[arm,x+1]+1
    r = r + alpha^(t-1) *x
    arm = ifelse(gittins_table[arm_states[1,2]+1,arm_states
        [1,1]+1] > gittins_table[arm_states[2,2]+1,arm_states
        [2,1]+1],1,2)
  }
  return(r)
}

#Thompson Sampling
Thompson <- function(alpha,N){
  arm_states = matrix(c(0),2,2)
  p_bandits = runif(2)
  arm = which.max(p_bandits)
  r=0
  for(t in 1:N){
    p_arm = (arm_states[arm,2]+1)/(rowSums(arm_states)[arm
        ]+2)
    x = rbinom(1,1,p_arm)
    arm_states[arm,x+1] = arm_states[arm,x+1]+1
    r = r + alpha^(t-1) *x
    arm = ifelse(rbeta(n=1,shape1=arm_states[1,2]+1,shape2 =
        arm_states[1,1]+1)>rbeta(n=1,shape1=arm_states
        [2,2]+1,shape2 =arm_states[2,1]+1),1,2)
  }
  return(r)
}

#Greedy Q-Learning
Greedy <- function(alpha,N,epsilon,initial_prob){

  Q = matrix(c(initial_prob),1,2)
  p_bandits = runif(2)
  r=0

  for(t in 1:N){

    if(runif(1)<epsilon)
      arm = rbinom(1,1,0.5)+1
    else
      arm = which.max(Q)

    x = rbinom(1,1,p_bandits[arm])
    r = r + alpha^(t-1) *x

    Q[arm] = (1/t) * alpha^(t-1) * x + (1-(1/t))*Q[arm]
  }
  return(r)
```

```
}

#calculate Gittins Index
alpha = 0.95
N = 40
gittins_table = Gittins(alpha,N)

#Simulation
N_sim = 1000
full_info_reward = 0
bayesian_reward = 0
thompson_reward = 0
greedy_reward = 0
eps_greedy_reward = 0
opt_greedy_reward = 0

for(n in 1:N_sim){

  fi = FullInformation(alpha,N-1)
  ba = Bayesian(alpha,N-1)
  th = Thompson(alpha,N-1)
  gr = Greedy(alpha,N-1,epsilon=0,initial_prob = 0)
  opt_gr = Greedy(alpha,N-1,epsilon=0,initial_prob = 1)
  eps_gr = Greedy(alpha,N-1,epsilon=0.3,initial_prob = 0)


  full_info_reward = full_info_reward + fi
  bayesian_reward = bayesian_reward + ba
  thompson_reward = thompson_reward + th
  greedy_reward = greedy_reward + gr
  opt_greedy_reward = opt_greedy_reward + opt_gr
  eps_greedy_reward = eps_greedy_reward + eps_gr
}

full_info_reward = full_info_reward/N_sim
bayesian_reward = bayesian_reward/N_sim
thompson_reward = thompson_reward/N_sim
greedy_reward = greedy_reward/N_sim
opt_greedy_reward = opt_greedy_reward/N_sim
eps_greedy_reward = eps_greedy_reward/N_sim

print(c("Full info:",full_info_reward))
print(c("Bayesian:",bayesian_reward))
print(c("Thompson Sampling:",thompson_reward))
print(c("Greedy Q-Learning:",greedy_reward))
print(c("Optimistic Greedy Q-Learning:",opt_greedy_reward))
print(c("Epsilon-Greedy Q-Learning:",eps_greedy_reward))

#Find optimal epsilon
epsilons = c(0,0.01,0.02,0.05,0.1,0.2,0.3,0.4,0.5)
```

```
eps_greedy_rewards = c()

for(eps in epsilons){
  eps_greedy_reward = 0
  for(n in 1:N_sim){
    eps_gr = Greedy(alpha,N-1,epsilon=eps,initial_prob = 0)
    eps_greedy_reward = eps_greedy_reward + eps_gr
  }
  eps_greedy_rewards = c(eps_greedy_rewards,
      eps_greedy_reward)
}
eps_greedy_rewards = eps_greedy_rewards/N_sim


plot(epsilons,eps_greedy_rewards,type='o',xlab='Epsilons',
    ylab='Discounted No. of Successes')
```