Multiarmed-Bandit

Lei Shi

11/9/2021

Quick Review

Reference used

1. The Multi-Armed Bandit Problem and Its Solutions: https://lilianweng.github.io/lil-log/2018/01/23/the-multi-armed-bandit-problem-and-its-solutions.html#%CE%B5-greedy-algorithm
2. Multi-Armed Bandit with Thompson Sampling: https://www.r-bloggers.com/2020/09/multi-armed-bandit-with-thompson-sampling/
3. Multi-Armed Bandits as an A/B Testing Solution: https://www.r-bloggers.com/2019/09/multi-armed-bandits-as-an-a-b-testing-solution/
4. Reinforcement Learning: An Introduction: http://incompleteideas.net/book/the-book-2nd.html
5. Exploration vs Exploitation & the Multi Armed Bandit: https://rpubs.com/OttoP/478713

Multi-armed bandit, Exploration & Exploitation

Imagine you are in a casino facing multiple slot machines and each is configured with an unknown probability of how likely you can get a reward at one play. The question is: What is the best strategy to achieve highest long-term rewards?

This is the motivating scenario, which can be generalized to a lot of other interesting real-life problems:

- Clinical trials: how to randomize patients to different treatment(say different drugs) options?
- Recommendation systems: what advertisement should a company serves among several possible options(traditionally termed as A/B test)? What videos (or types of videos) should youtube recommmend so that you can stay for five more minutes?

Some terminology:

- agent : the component that makes the decision of what action to take
- action(variant, policy) : the choice made, e.g. which offer to present from a number of alternatives (impression)
- reward : the result of doing a certain action, i.e. the "outcome" (e.g. a click)
- regret : the loss of not selecting the optimal action
- batch : in oline setting, the data comes in batches.

We focus on the bernoulli steup we used in the lectures: suppose there are K actions, and when played, any action yields either a success or a failure. Action $k \in 1, \cdots, K$ produces a success with probability $\theta_k \in [0, 1]$. The success probabilities $\theta_1, \cdots, \theta_K$ are unknown to the agent, but are fixed over time. Therefore, these probabilities can be learned by experimentation. The objective(reward), roughly speaking, is to maximize the cumulative number of successes over T periods, where T is relatively large compared to the number of arms K.

Now we are faced with the **exploration vs exploitation dilemma**. A nice restaurant I've tried vs a new restaurant I've never visited, which one should I go? We hope to make use of the resources that we have

learned so far and maximize the greedy reward, while leave open the possibility of learning potentially more rewarding actions. In other word, exploitation serves for now, while exploration serves for future.

Solutions for Multi-armed bandit

1. Epsilon-greedy

The ϵ-greedy algorithm takes the best action most of the time, but does random exploration occasionally. According to the ϵ-greedy algorithm, with a small probability ϵ we take a random action, but otherwise (which should be the most of the time, probability 1-ϵ) we pick the best action that we have learnt so far.

2. UCB algorithm(not our focus, will do if we have time)

See https://lilianweng.github.io/lil-log/2018/01/23/the-multi-armed-bandit-problem-and-its-solutions.html#%CE%B5-greedy-algorithm.

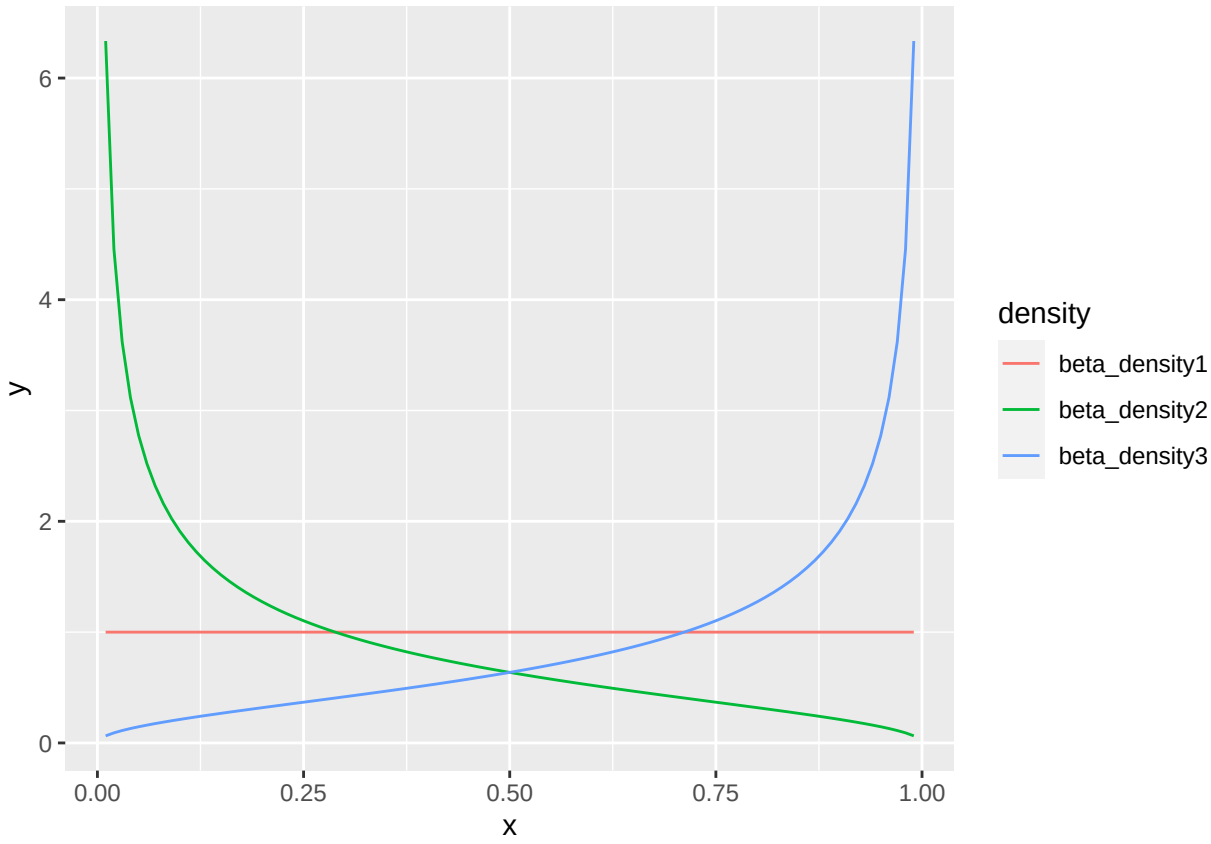
3. Thompson sampling

- Bayesian statistics: prior & posterior distribution, conjugate distribution(https://en.wikipedia.org/wiki/Conjugate_prior?id=%22Table_of_conjugate_distributions"), inference(MAP, Bayesian confidence region)
- Beta distribution

Some basics about beta distribution Beta(α, β):

1. has a ugly pdf and a even worse cdf, though looks nice from plots:

```
Beta_density <- data.frame(
  x = 0.01*(1:99),
  beta_density1 = dbeta(0.01*(1:99), 1, 1),
  beta_density2 = dbeta(0.01*(1:99), 0.5, 1.5),
  beta_density3 = dbeta(0.01*(1:99), 1.5, 0.5)
)

df <- gather(Beta_density, key = density, value = y,
c("beta_density1", "beta_density2", "beta_density3"))

beta_plot <- ggplot(df, aes(x=x, y = y, group = density, colour = density)) +
geom_line()

beta_plot
```



2. If $B \sim \text{Beta}(\alpha, \beta)$,

$$\text{E}(B) = \frac{\alpha}{\alpha+\beta}, \ \text{Var}(B) = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}.$$

Note although the mean is scale invariant, the variance is not. If we visualize the change of Beta distribution:

```
x <- 0.01*1:99
alpha <- 0.2*1:20
beta <- 0.2*1:20

frame <- c()
x_val <- c()
beta_den <- c()
alpha_val <- c()
beta_val <- c()

for (i in 1:length(alpha)){
  frame <- c(frame, rep(i,length(x)))
  x_val <- c(x_val, x)
  alpha_val <- c(alpha_val, rep(alpha[i],length(x)))
  beta_val <- c(beta_val, rep(beta[i],length(x)))
  beta_den <- c(beta_den, dbeta(x, alpha[i], beta[i]))
}

Beta_den <- data.frame(
  frame = frame,
  x_val = x_val,
  alpha_val = alpha_val,
```

```
  beta_val = beta_val,
  beta_den = beta_den
)

Beta_den_plot <- ggplot(Beta_den, aes(x = x_val, y = beta_den)) +
  geom_line() +
  transition_time(frame) +
  ease_aes('linear')

Beta_den_plot
```

3. Uniform distribution is a special Beta distribution, with parameter $(1, 1)$.
4. Beta distribution is the conjugate prior for the bernoulli distribution(more generally the binomial distribution family):

$$\text{Beta}(\alpha, \beta) \to p \to \text{i.i.d Bernoulli(p)} \ X_i \to \text{Beta}(\alpha + \sum X_i, \ \beta + n - \sum X_i) \to \cdots$$

- For Thompson sampling it serves as a prior belief for the distribution of the reward. For simplicity suppose we only have one observation in each batch. The agent samples a $\hat{p}_k \sim \text{Beta}(\alpha_k, \beta_k)$ each time an individual arrives, and assigns her to the treatment corresponding to the highest p_k(say $k^\star$). Then we observe an outcome(success or failure) Y and update the prior:

$$\alpha_{k^\star} \leftarrow \alpha_{k^\star} + Y, \ \beta_{k^\star} \leftarrow \beta_{k^\star} + 1 - Y.$$

Programming perspective

Resources

1. R package: contextual https://www.rdocumentation.org/packages/contextual/versions/0.9.8.4

- If you are using R3.0, then install.packages should work fine.
- If you are using R4.0, install it from github(see the first code chulk). Also need to update Rstudio to the newest version to make the parallel package compatible. this package uses parallel computing.

2. some implementation: https://rpubs.com/OttoP/478713

Implement Thompson sampling

Let's assume that the ground truth success rates of the 4 treatments are:

Trt 1: 10% Trt 2: 11% Trt 3: 12% Trt 4: 13%

```
output <- {}
b_Probs <- c(0.10, 0.11, 0.12, 0.13)
b_Sent <- rep(0, length(b_Probs))
b_Reward <- rep(0, length(b_Probs))

batch_size<-1000
N<-10000
steps<-floor(N/batch_size)
msgs<-length(b_Probs)

for (i in 1:steps) {
  B<-matrix(rbeta(1000*msgs, b_Reward+1, (b_Sent-b_Reward)+1),1000, byrow = TRUE)
  P<-table(factor(max.col(B), levels=1:ncol(B)))/dim(B)[1]
  # tmp are the weights for each time step
```

```
  tmp<-round(P*batch_size,0)

  # Update the Rewards
  b_Reward<-b_Reward+rbinom(rep(1,msgs), size=tmp, prob = b_Probs)

  #Update the Sent
  b_Sent<-b_Sent+tmp

  #print(P)
  output<-rbind(output, t(matrix(P)))
}

# the weights of every step
output
```

```
##         [,1]  [,2]  [,3]  [,4]
##  [1,] 0.266 0.245 0.250 0.239
##  [2,] 0.000 0.475 0.514 0.011
##  [3,] 0.000 0.205 0.791 0.004
##  [4,] 0.000 0.095 0.901 0.004
##  [5,] 0.000 0.074 0.925 0.001
##  [6,] 0.000 0.071 0.924 0.005
##  [7,] 0.000 0.055 0.937 0.008
##  [8,] 0.000 0.033 0.960 0.007
##  [9,] 0.000 0.026 0.972 0.002
## [10,] 0.000 0.052 0.941 0.007
```

As we can see, even from the 5th step the algorithm started to assign more weight to the variant 4 and almost nothing to the variant 1 and variant 2.

Compared with a classical strategy: A/B test

In an A/B test, the customer base is divided into two or more groups, each of which is served a different version of whatever is being tested (such as a special offer, or the layout of an advertising campaign). At the end of the test, whichever variant was most successful is pursued for the customer base at large.

The following example is taken from https://www.r-bloggers.com/2019/09/multi-armed-bandits-as-an-a-b-testing-solution/.

To illustrate, let's use a simplified example to compare a more traditional A/B test to Epsilon Greedy and Thompson Sampling. In this scenario, a customer can be shown one of five variants of an advertisement. For our purposes, we will assume that Ad 1 performs the worst, with a 5% conversion rate. Each ad performs 5% better than the last, with the best performer being Ad 5, at 25% conversion. We'll do 1,000 trials, which means that in an idealized, hypothetical world, the number of conversions we could get by only showing the optimal ad would be 250 (given a 25% conversion rate over 1,000 trials).

```
# A/B TEST
# setup
set.seed(240)
horizon <- 1000L
simulations <- 1000L
conversionProbabilities <- c(0.05, 0.10, 0.15, 0.20, 0.25)
nTestSample <- 0.5 * horizon
clickProb <- rep(NA, simulations)
adDistMatrix <- matrix(NA, nrow = simulations, ncol = length(conversionProbabilities))
adDistMatrixAB <- matrix(NA, nrow = simulations, ncol = length(conversionProbabilities)) # simulation
```

```
for(i in 1:simulations){
  testSample <- sapply(conversionProbabilities,  function(x) sample(0:1, nTestSample, replace = TRUE, p

  testColumns <- (1:length(conversionProbabilities))[-which.max(colSums(testSample))]

  p.values <- sapply(testColumns, function(x) prop.test(x = colSums(testSample[, c(x, which.max(colSums

  adsAfterABTest <- (1:length(conversionProbabilities))[- testColumns[which(p.values < 0.05)]]

  # now just with the best performing ad(s)
  ABSample <- sapply(conversionProbabilities[adsAfterABTest],
                     function(x) sample(0:1, round((horizon - nTestSample)*length(conversionProbabiliti

  clickProbTest <- sum(as.vector(testSample)) / length(unlist(testSample))

  clickProbAB <- sum(as.vector(ABSample)) / length(unlist(ABSample))

  clickProb[i] <- clickProbTest * (nTestSample / horizon) + clickProbAB * (1 - nTestSample / horizon)

  # distribution of which ads were seen over the course of all trials
  adDistMatrix[i,] <- rep(1 / length(conversionProbabilities), length(conversionProbabilities))

  adDistributionAB <- rep(0, length(conversionProbabilities))

  adDistributionAB[adsAfterABTest] <- rep(1 / length(adsAfterABTest), length(adsAfterABTest))

  adDistMatrixAB[i,] <- adDistributionAB
  }
# total payoff
ABPayoff <- (nTestSample * clickProbTest) + (nTestSample * clickProbAB)

ABPayoff
```

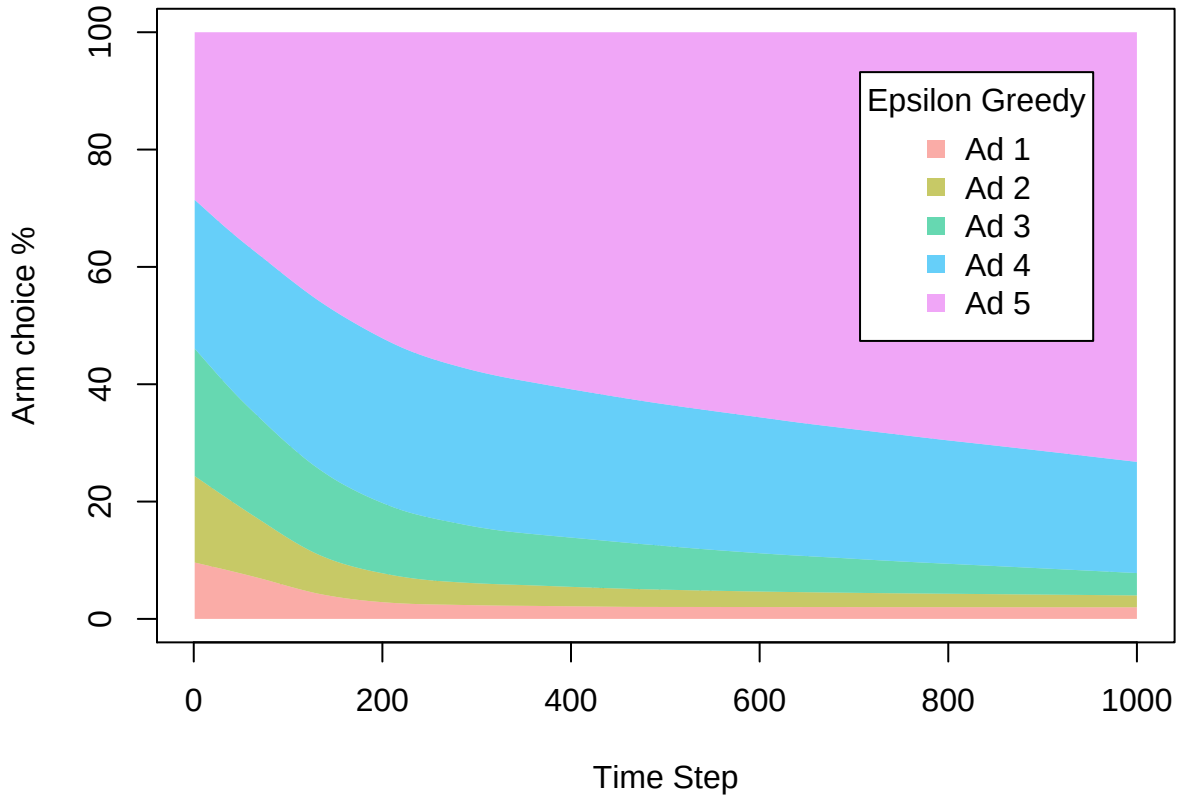
```
## [1] 187.6
```

```
# EPSILON GREEDY
horizon <- 1000L
simulations <- 1000L
conversionProbabilities <- c(0.05, 0.10, 0.15, 0.20, 0.25)
bandit <- BasicBernoulliBandit$new(weights = conversionProbabilities)
policy <- EpsilonGreedyPolicy$new(epsilon = 0.10)
agent <- Agent$new(policy, bandit)
historyEG <- Simulator$new(agent, horizon, simulations)$run()
```

```
## Setting up parallel backend.

## Cores available: 8

## Workers assigned: 7

## Simulation horizon: 1000

## Number of simulations: 1000

## Number of batches: 7

## Starting main loop.
```

```
## Finished main loop.
## Completed simulation in 0:00:12.638
## Computing statistics.
```

```
plot(historyEG, type = "arms",legend_labels = c('Ad 1', 'Ad 2', 'Ad 3', 'Ad 4', 'Ad 5'), legend_title =
```




```
summary(historyEG)
```

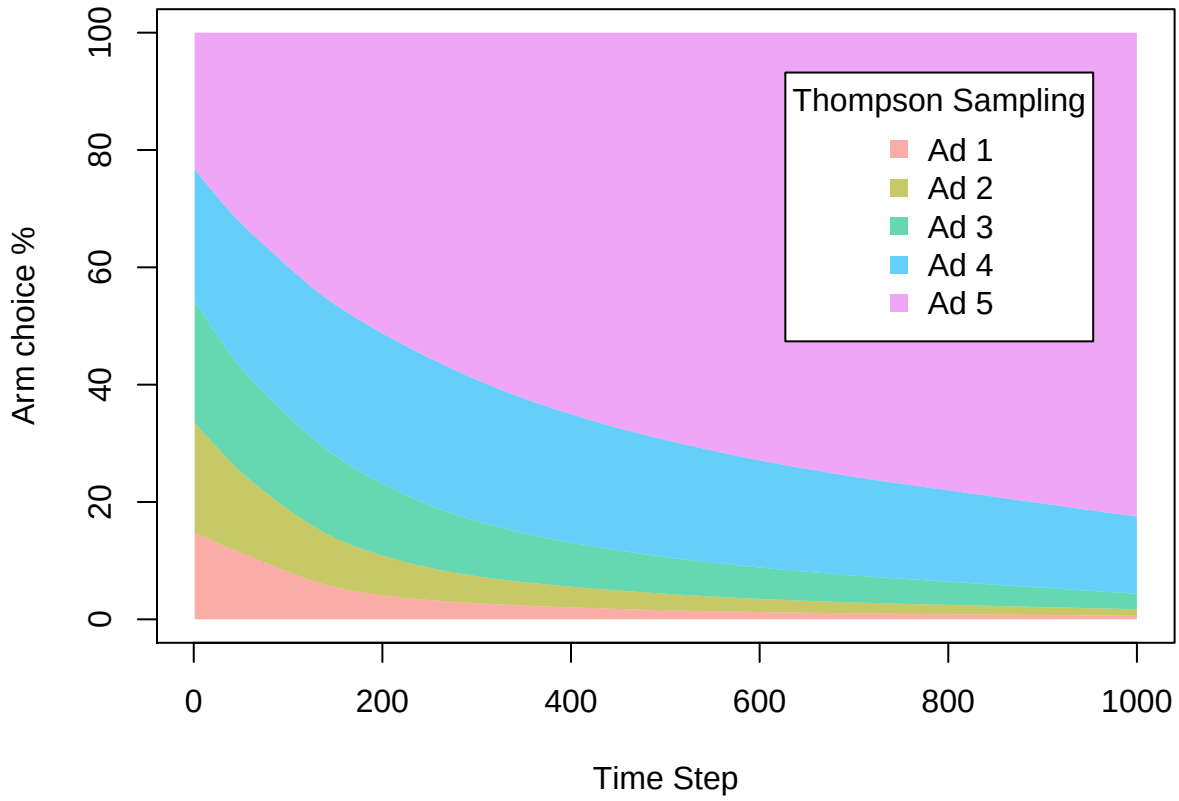
```
##
## Agents:
##
##   EpsilonGreedy
##
## Cumulative regret:
##
##          agent    t sims cum_regret cum_regret_var cum_regret_sd
##  EpsilonGreedy 1000 1000     31.752       592.2287      24.33575
##
##
## Cumulative reward:
##
##          agent    t sims cum_reward cum_reward_var cum_reward_sd
##  EpsilonGreedy 1000 1000    218.111       711.9166      26.68177
##
##
## Cumulative reward rate:
##
##          agent    t sims cur_reward cur_reward_var cur_reward_sd
```

```
##  EpsilonGreedy 1000 1000   0.218111      0.7119166    0.02668177
```

```
# THOMPSON SAMPLING
horizon <- 1000L
simulations <- 1000L
conversionProbabilities <- c(0.05, 0.10, 0.15, 0.20, 0.25)
bandit <- BasicBernoulliBandit$new(weights = conversionProbabilities)
policy <- ThompsonSamplingPolicy$new(alpha = 1, beta = 1)
agent <- Agent$new(policy, bandit)
historyThompson <- Simulator$new(agent, horizon, simulations)$run()
```

```
## Setting up parallel backend.
## Cores available: 8
## Workers assigned: 7
## Simulation horizon: 1000
## Number of simulations: 1000
## Number of batches: 7
## Starting main loop.
## Finished main loop.
## Completed simulation in 0:00:26.083
## Computing statistics.
```

```
plot(historyThompson, type = "arms",  legend_labels = c('Ad 1', 'Ad 2', 'Ad 3', 'Ad 4', 'Ad 5'), legend
```



```
summary(historyThompson)
```

```
##
## Agents:
##
##   ThompsonSampling
##
## Cumulative regret:
##
##             agent    t sims cum_regret cum_regret_var cum_regret_sd
##  ThompsonSampling 1000 1000     30.073       159.1688      12.61621
##
##
## Cumulative reward:
##
##             agent    t sims cum_reward cum_reward_var cum_reward_sd
##  ThompsonSampling 1000 1000    219.968       328.3834      18.12135
##
##
## Cumulative reward rate:
##
##             agent    t sims cur_reward cur_reward_var cur_reward_sd
##  ThompsonSampling 1000 1000   0.219968      0.3283834    0.01812135
```

Drawback of bandit problem & algorithms

The Multi-armed bandit is a simple model for many real-world systems.

Practical issue:

1. Not personalized

Theoretical issue:

1. Highly non-smooth struture
2. Highly probabilistic dependence