# Introduction to MABLearning Package

*markcx@stanford.edu*

*March 19, 2017*

## 1. What is MABLearning?

The MABLearning package (short for **M**ulti-**A**rmed **B**andit **L**earning) contains functions to streamline the model bandit learning process for online decision making problems. The package utilizes a number of R packages but tries not to load them all at package. The package provides a flexible class to instantiate the `Bandit` agent, a simulation framework, and three popular learning algorithms. Meanwhile it also provides a simple shiny web interface to illustrate the bandit learning outcome using different algorithms.

**Install the Package**

Assuming your current working directory has the source package, then install **MABLearning** using

```
install.packages("MABLearning_1.0.tar.gz",repo=NULL,type="source",dependencies=TRUE)
```

## 2. Basic Usage

MABLearning has several functions that attempt to streamline the model building and evaluation process. There are three wrapper functions representing three popular algorithms *epsilon greedy*, *ucb*, and *Thompson Sampling*. The default implementation utilized the `parallel` package. Thus load the dependency before calling the simulation algorithm by using

```
library(MABLearning)
library(parallel)
```

### Eps-Greedy

```
set.seed(12345)
STEP_HORIZON = 400
case1 = epsGreedy(steps = STEP_HORIZON, eps = 0.01)
case2 = epsGreedy(steps = STEP_HORIZON, eps = 0.1)
case3 = epsGreedy(steps = STEP_HORIZON, eps = 0.4)
```

To render the plots, see the code in appendix

### UCB

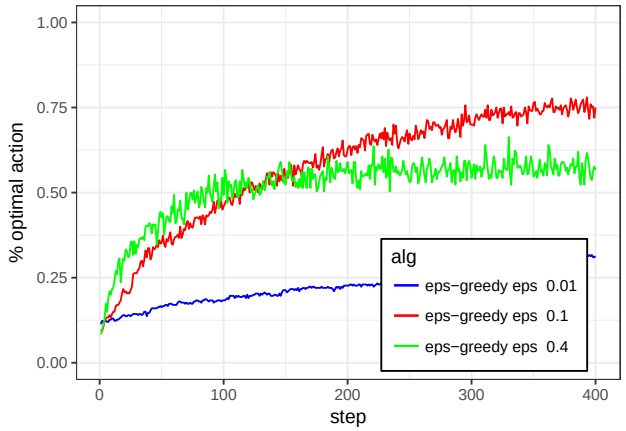
```
STEP_HORIZON = 400
case4 = ucb(steps = STEP_HORIZON, ucbParam = 1)
case5 = ucb(steps = STEP_HORIZON, ucbParam = 2)
```

To render the plots, see the code in appdendix

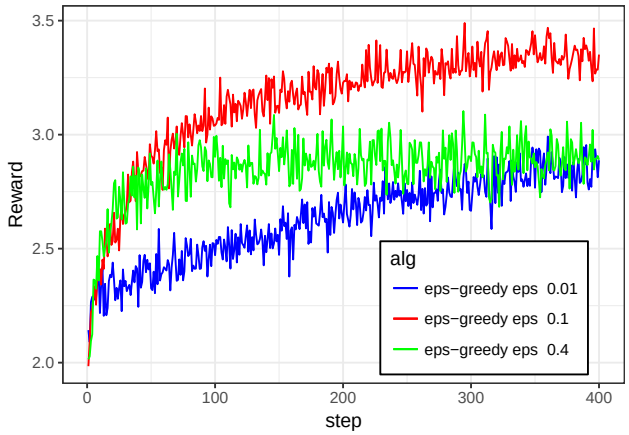
### Thompson Sampling

```
library(MASS)
STEP_HORIZON = 400
case6 = thompsonSampling(steps = STEP_HORIZON )
```

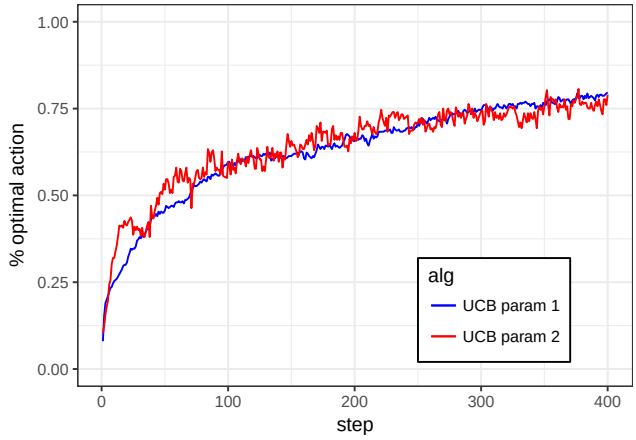
To render the plots, see the code in appdendix



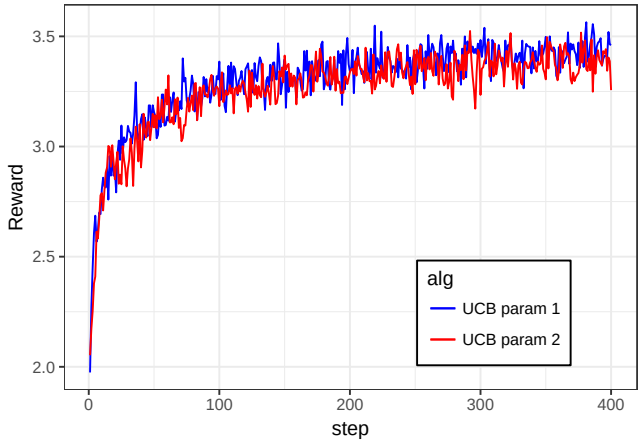

(a) The x-axis is the simulation horizon, y-axis is the probability that the agent pick the best action (i.e. the action associated with highest reward); The faster the algorithm converges to 1, the better it is. In this graph, green curve eps=0.4 converges faster than the other two at early stage. But as time step passed 150, the red curve eps=0.1 converges to higher probability value which indicates a better performance.



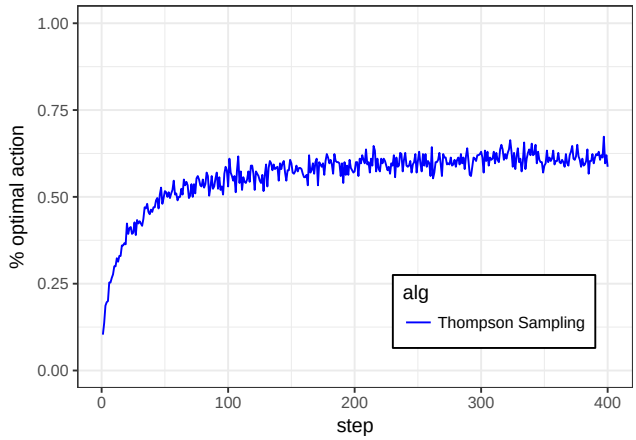

(b) The x-axis is the simulation horizon, y-axis is the sample average reward. This figure shows the cumulative mean rewards that an algorithm can yield. The higher and faster a curve can reach to a certain value, the better the algorithm it is. In this graph, the eps=0.1 algorithm yields the highest cumulative rewards on average.



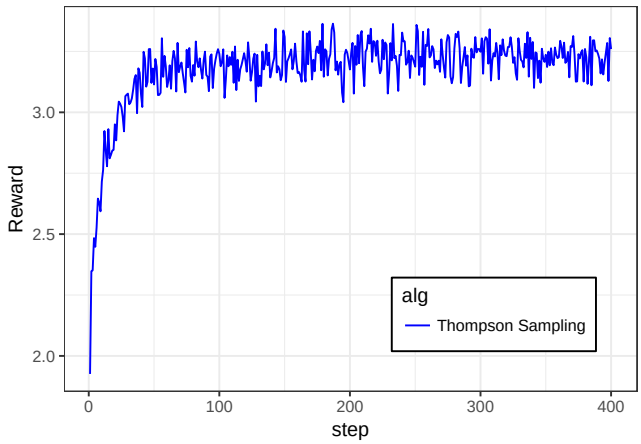

(a) The x-axis is the simulation horizon, y-axis is the probability that the agent pick the best action (i.e. the action associated with highest reward)




(b) The x-axis is the simulation horizon, y-axis is the sample average reward




(a) The x-axis is the simulation horizon, y-axis is the probability that the agent pick the best action (i.e. the action associated with highest reward)




(b) The x-axis is the simulation horizon, y-axis is the sample average reward

### Run a Simple Shiny App Demo

```
MABLearning::mabWebApp()
```

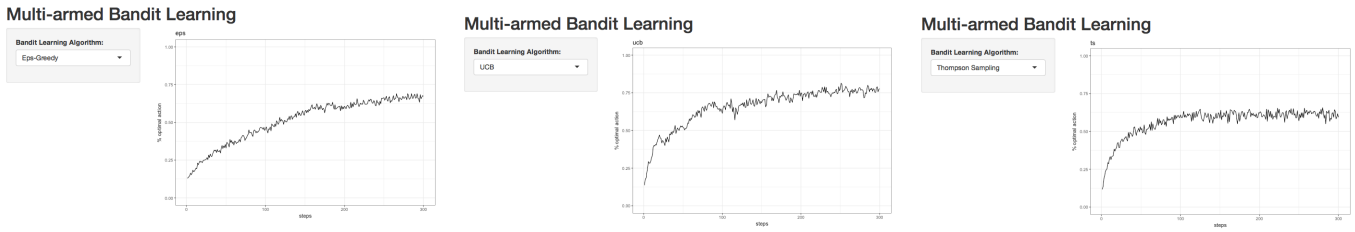



Figure 4: The Shiny web app gives three options, such as *eps, ucb, thompson sampling*, to test the algorithm based the same condition, with 10 arms 400 time-steps and 300 bandit agents simulation

All the above examples apply 300 Bandit agents and then take the average of the results (Monte Carlo method), the simulation time horizon is 400 time steps. The default number of arms is 10, that means the agent tries to pick the best action out of ten candidates.

## 3. Advanced Usage

This package also provides a flexible framework to customized the simulation with adjustable parameters. Here is an example. We can define a customized `banditSimulationCust` function as

```
banditSimulationCust <- function(nBandits, steps, bandits){
  bestActionCounts = rep(0, steps)
  averageRewards = rep(0, steps)

  for (i in c(1:nBandits)) {
      for (t in c(1:steps)){
        action = bandits[[i]]$getAction()
        reward = bandits[[i]]$takeAction(action=action, rewardNoise = expression(rnorm(1, mean=0, sd=1)))
        averageRewards[t] = averageRewards[t] + reward
        if (action == bandits[[i]]$bestAction() ){
          bestActionCounts[t] = bestActionCounts[t] + 1
        }
      }# end loop of steps
  }# end loop of nBandits
  bestActionCounts = bestActionCounts/nBandits
  averageRewards = averageRewards/nBandits

  return(list(bestActionCounts = bestActionCounts,
              averageRewards = averageRewards ))
}
```

Then we can define a customized eps-greedy algorithm as follows

```
epsGreedyCust <- function(nBandits=100, steps=200, eps = 0.1 ){
  bandits = mclapply(seq(nBandits), function(i) {
                  banditInst = Bandit$new(kArm=3, epsilon=eps, sampleAverage=TRUE)
                  banditInst$addRewardNoise(rewardNoise = expression(rnorm(1, mean = 0, sd=1)))
                  return(banditInst) }  )
  res <- banditSimulationCust(nBandits, steps, bandits)
  return(res)
}
```

Here the `Bandit` class instantiate the instance with 3 arms (or actions), the initial value of 2 as an individual expected reward for each arm, and the instance method `addRewardNoise` takes the general random number generator expression as

`rewardNoise` (notice it can also take constant value as a deterministic case ).

Similarly we can specify the customized ucb function `ucbCust` as follows

```
ucbCust <- function(nBandits=100, steps=200, ucbParam=1) {
  bandits = mclapply(seq(nBandits), function(i) {
      banditInst =  Bandit$new(kArm=3, epsilon=0, initial=2, UCBParam=ucbParam, sampleAverage=TRUE )
      banditInst$addRewardNoise(rewardNoise = expression(rnorm(1, mean = 0, sd=1)))
      return(banditInst)
  })
  res <- banditSimulationCust(nBandits, steps, bandits)
  return(res)
}
```

We run the simulation of `epsGreedyCust` and `ucbCust` in 3 arms conditions with 1500 time-steps, and presenting the following results.

```
library(MABLearning)
library(parallel)
library(ggplot2)
STEP_HORIZON = 1500
res1 <- epsGreedyCust(steps = STEP_HORIZON)
res2 <- ucbCust(steps = STEP_HORIZON)

resCust.df = data.frame( step = c(1:STEP_HORIZON) ,
            optActPct1 = res1$bestActionCounts,
            optActPct2 = res2$bestActionCounts )
fig3 <- ggplot2::ggplot(data=resCust.df, aes(x=step)) +
  geom_line(aes(y=optActPct1, color = "optActPct1" )) +
  geom_line(aes(y=optActPct2, color = "optActPct2" )) +
  ylab("% optimal action")  + ylim(0,1) + theme_bw() +
  theme(legend.position = c(0.75, 0.2),
          legend.background = element_rect(colour = "black") ) +
  scale_color_manual(labels = c("eps", "ucb"), values = c("blue", "red") ) +
  guides(color=guide_legend("alg"))
print(fig3)
```

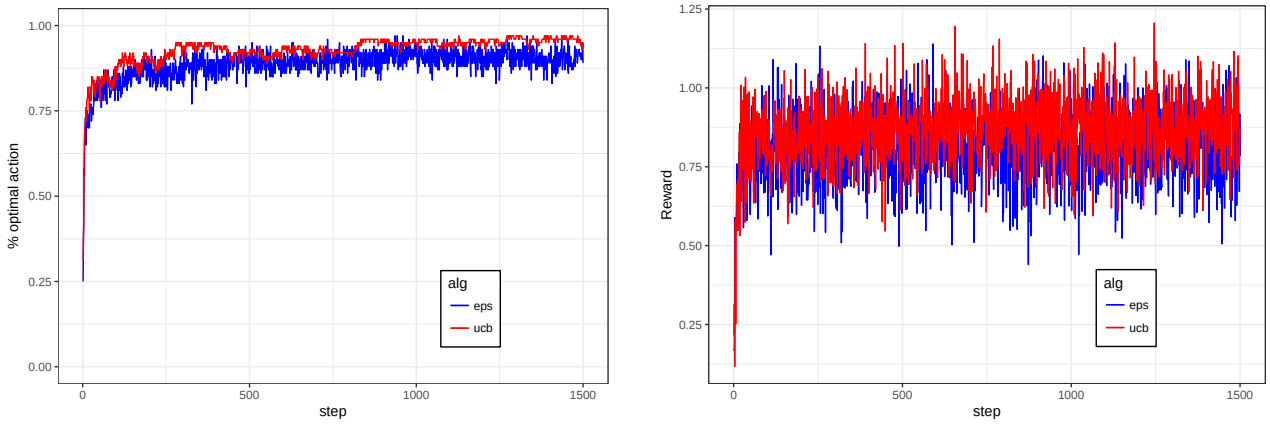



Figure 5: Customized epsilon greedy and upper confidence bound simulation