Monty Hall Problem with n Doors in R

Boyie Chen

10/25/2019

Monty Hall Problem

Recall that 柏宇助教 had taught about the Monty Hall Problem with the following interactive website:

https://www.mathwarehouse.com/monty-hall-simulation-online

The most exciting thing is that we're going to write down of own function that give us the result of playing a Monty Hall Game once. Of course, since we can easily use `replicate()` to repeat the game for thousands of times, we can do the Monte Carlo Simulation for Monty Hall Problem too!

Original Version of Monty Hall Problem

We're going to define a function of playing a game. The input is `T` or `F`, representing "change the choice" or "keep the choice" respectively. The output is either `T` or `F`, representing whether you get the car or not.

```
#Monty Hall (Original)
# 把問題寫成換與不換為 input，有沒有得到 car 為 output 的 fcn
monty_hall = function(switch = logical()){
  #randomly arrange the door
  doors = c(1,2,3)
  names(doors) = sample(c("goat", "car", "goat"))
  prize_door = which(names(doors)=='car') #find the index of car
  #Now Guess
  guess = sample(doors, 1)
  if(guess == prize_door){
    revealed_door = sample(doors[doors != prize_door], 1) #randomly open one of the goat door
  }else{
    revealed_door = doors[doors != prize_door & doors != guess] #open the other goat door
  }
  #Show the return
  if(switch){
    switched_door = doors[doors != guess & doors != revealed_door]
    return(prize_door == switched_door)
  } else {
    return(prize_door == guess)
  }
}
```

After defining the function that plays the game, we can use `replicate()` and our function `monty_hall()` to simulate the win probability.

The win probability of "change" is:

```
mean(replicate(monty_hall(T), n = 10000))
```

```
## [1] 0.668
```

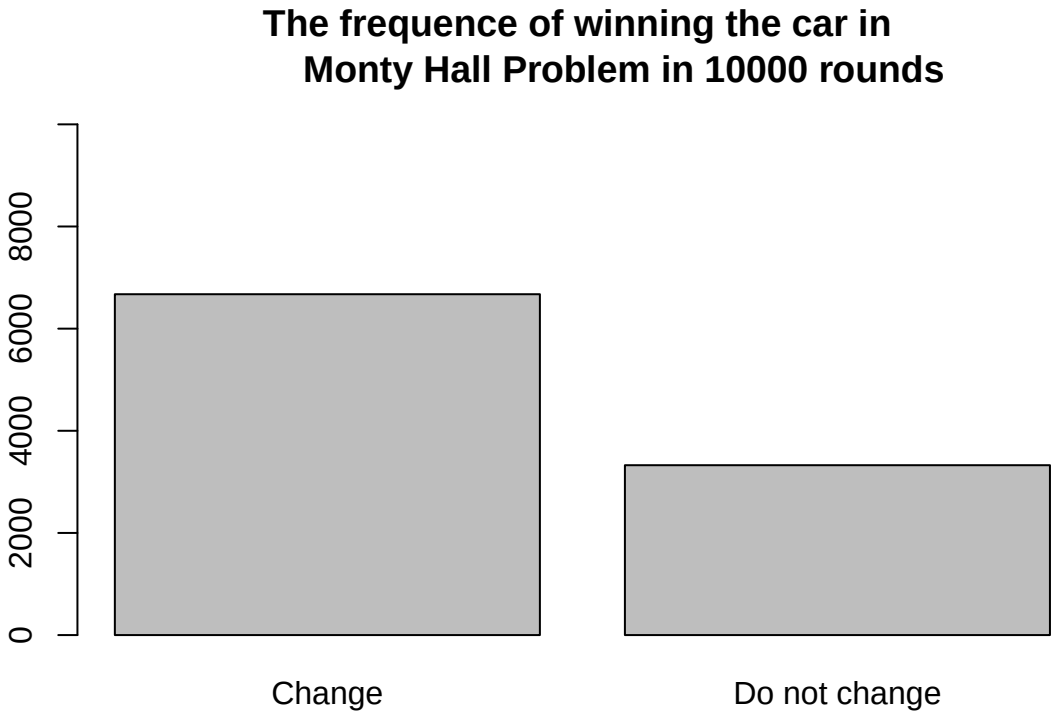
The win probability of "keep the choice" is:

```
mean(replicate(monty_hall(F), n = 10000))
```

```
## [1] 0.334
```

We can draw the barplot of the win probability of the two strategy.

```
result = replicate(monty_hall(T), n = 10000)
barplot(c(sum(result), length(result)-sum(result)),
        main = 'The frequence of winning the car in
        Monty Hall Problem in 10000 rounds',
        names.arg = c('Change', 'Do not change'),
        ylim = c(0,10000))
```



The graph tells that changing the choice gives you a higher win probability which is nearly twice of that of keeping the choice. The above result is nearly the same as the result we've talked in previous TA session. But we do not use any Bayes Theorem here to calculate the win probability. We just set up a function and let the computer do the work. As long as our function is defined in the right way, we can get the win probability by simulation.

Extend to a n door problem (Optional)

Now we look at a extented version of Monty Hall Problem. Say there are n doors. After choosing 1 door, the host open $n-2$ doors and ask you whether you decide to change the choice or not.

Similarly, we can define the following function. The inputs are your decision and how many doors are in this game. The output is whether you get the car or not.

```
monty_hall_n = function(switch = logical(), n){
  #randomly arrange the door
  doors = 1:n
  names(doors) = sample(rep(c("goat", "car"), c(n-1,1)))
  prize_door = which(names(doors)=='car') #find the index of car
```

```
  #Now Guess
  guess = sample(doors, 1)
  if(guess == prize_door){
    revealed_door = sample(doors[doors != prize_door], n-2) #randomly open one of the goat door
  }else{
    #revealed_door = sample(doors[doors != prize_door & doors != guess], n-2)
    revealed_door = doors[!doors %in% c(prize_door, guess)] #open the other goat door
  }
  #Show the return
  if(switch){
    switched_door = doors[!doors %in% c(guess, revealed_door)]#'%in%' is value matching
    return(prize_door == switched_door)
  } else {
    return(prize_door == guess)
  }
}
```

We shall expect that this function gives us the same result when $n = 3$

```
monty_hall_n(T, 3)
```

```
##  goat
## FALSE
```

```
mean(replicate(monty_hall_n(T, 3), n = 10000))
```

```
## [1] 0.6634
```

```
mean(replicate(monty_hall_n(F, 3), n = 10000))
```

```
## [1] 0.3387
```

Slightly add up the number of doors. From 3 to 4.

```
mean(replicate(monty_hall_n(T, 4), n = 10000))
```

```
## [1] 0.7467
```

```
mean(replicate(monty_hall_n(F, 4), n = 10000))
```

```
## [1] 0.2492
```

If there are 100 doors, the difference between the win probability of "change" and "keep" is huge.

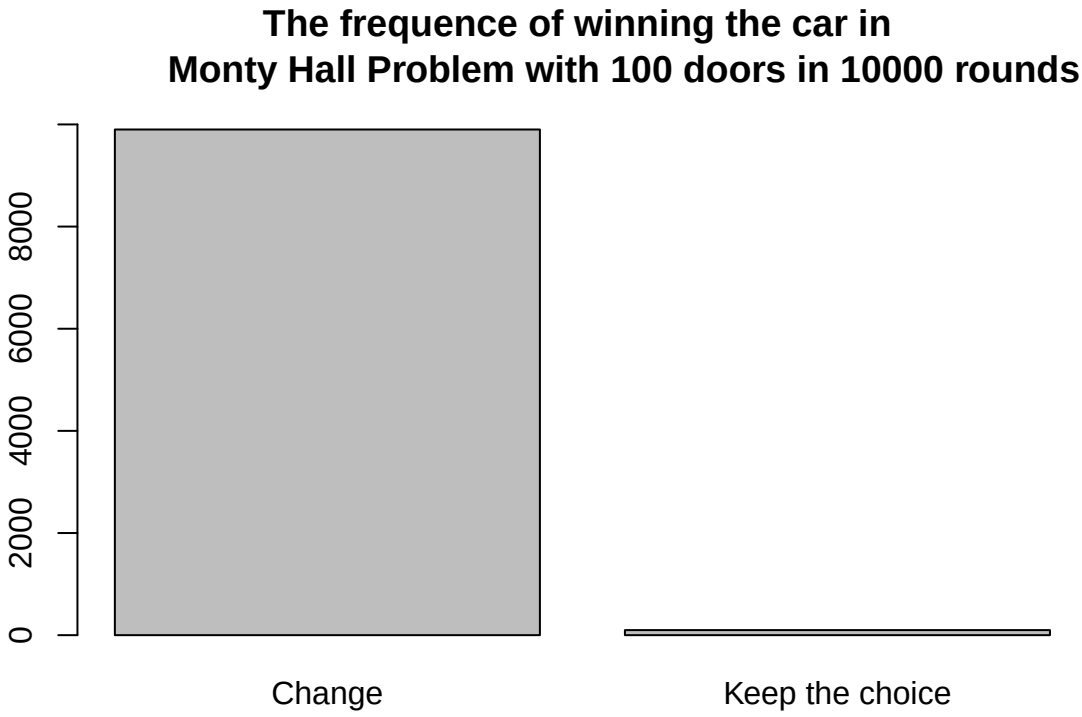
```
mean(replicate(monty_hall_n(T, 100), n = 10000))
```

```
## [1] 0.9902
```

```
mean(replicate(monty_hall_n(F, 100), n = 10000))
```

```
## [1] 0.0115
```

```
result = replicate(monty_hall_n(T, 100), n = 10000)
barplot(c(sum(result), length(result)-sum(result)),
        main = 'The frequence of winning the car in
        Monty Hall Problem with 100 doors in 10000 rounds',
        names.arg = c('Change', 'Keep the choice'),
        ylim = c(0,10000))
```



Further Issue : What if the decision of 'change' is decided randomly?

The above function `monty_hall()` and `monty_hall_n()` gives me the win probability when the decion is fixed. What about a mixed strategy? Or what if my decision of 'change the choice' is decided randomly?

We can simply use `sample()` to decide the input randomly.

```
monty_hall(sample(c(T, F), 1))
```

```
##  goat
## FALSE
```

```
mean(replicate(expr = monty_hall(sample(c(T, F), 1)), 10000))
```

```
## [1] 0.4994
```

```
#the prob of win becomes 0.5
```

The intuition behind this result is that your decision does not depends on new information that the host gives you. Because you do not use new information inflow at all, you won't gain from the information inflow.

Mixed Strategy

Your win probability will fall between the win probability of 'change' and 'keep the choice'.

```
mean(replicate(expr = monty_hall(sample(c(T, F), 1, prob = c(0.9, 0.1))), 10000))
```

```
## [1] 0.6344
```

```
mean(replicate(expr = monty_hall(sample(c(T, F), 1, prob = c(0.1, 0.9))), 10000))
```

```
## [1] 0.3632
```

```r
mean(replicate(expr = monty_hall(sample(c(T, F), 1, prob = c(0.4, 0.6))), 10000))
```

```
## [1] 0.4701
```

Further issue for n doors (Optional)

```r
monty_hall_n(sample(c(T, F), 1), n = 100)
```

```
##  goat
## FALSE
```

```r
mean(replicate(expr = monty_hall_n(sample(c(T, F), 1), n = 100), 10000))
```

```
## [1] 0.5056
```

```r
#the prob of win becomes 0.5

mean(replicate(expr = monty_hall_n(sample(c(T, F), 1, prob=c(0.9,0.1)), n = 100), 10000))
```

```
## [1] 0.8943
```

```r
mean(replicate(expr = monty_hall_n(sample(c(T, F), 1, prob=c(0.1,0.9)), n = 100), 10000))
```

```
## [1] 0.1063
```