Assignment2_Liang Xuesen (G1700221K)

JL

17 November 2018

Problem Description

Below R program is aimed to solve the classic problem of finding optimal split solution for selecting the best Prince from a group of Princes.

The logic of solving the problem:

Step 1.

- Randomly generate N number of Princes from 1 to N by using "sample(1:N,N,replace = FALSE)". Name it "Input_List".

Step 2.

- Iterate the split position from 0 to N to split the "Input_List" into "evaluation_group" and "selection_group" for each split.

Step 3.

- For each split position, record the maximum number from the "evaluation_group", and compare the item in position 1 to length of "selection_group" in "selection_group" to the maximum number. If any item is greater than or equals to the maximum number, return the item value, stop the comparasion process, and store the value into vector "split_best_choice_sub".
  - If split = 0, return the first item of the "Input_List".
  - If split = N, return the last item of the "Input_List".

Step 4.

- For each split position, repeat the **Step 3** 200 times to get a vector of "split_best_choice_sub" containing 200 best choices result from each process of **Step 3**. Calculate the probability of getting the best Princes (the largest number in "Input_List") from "Input_List" for the 200 times simulation in **Step 4** by using sum(split_best_choice_sub==N)/200.

Step 5.

- For each split position, repeat **Step 4** 200 times and calculate the mean of the 200 probabilities for each split position, and then store the 101 mean values as well as the 101 split position into data frame "bestChoiceSetsMeans".

Step 6.

- Plot the "bestChoiceSetsMeans" and observe the optimal split position for different number of Princes N (N=3, 10, 100).

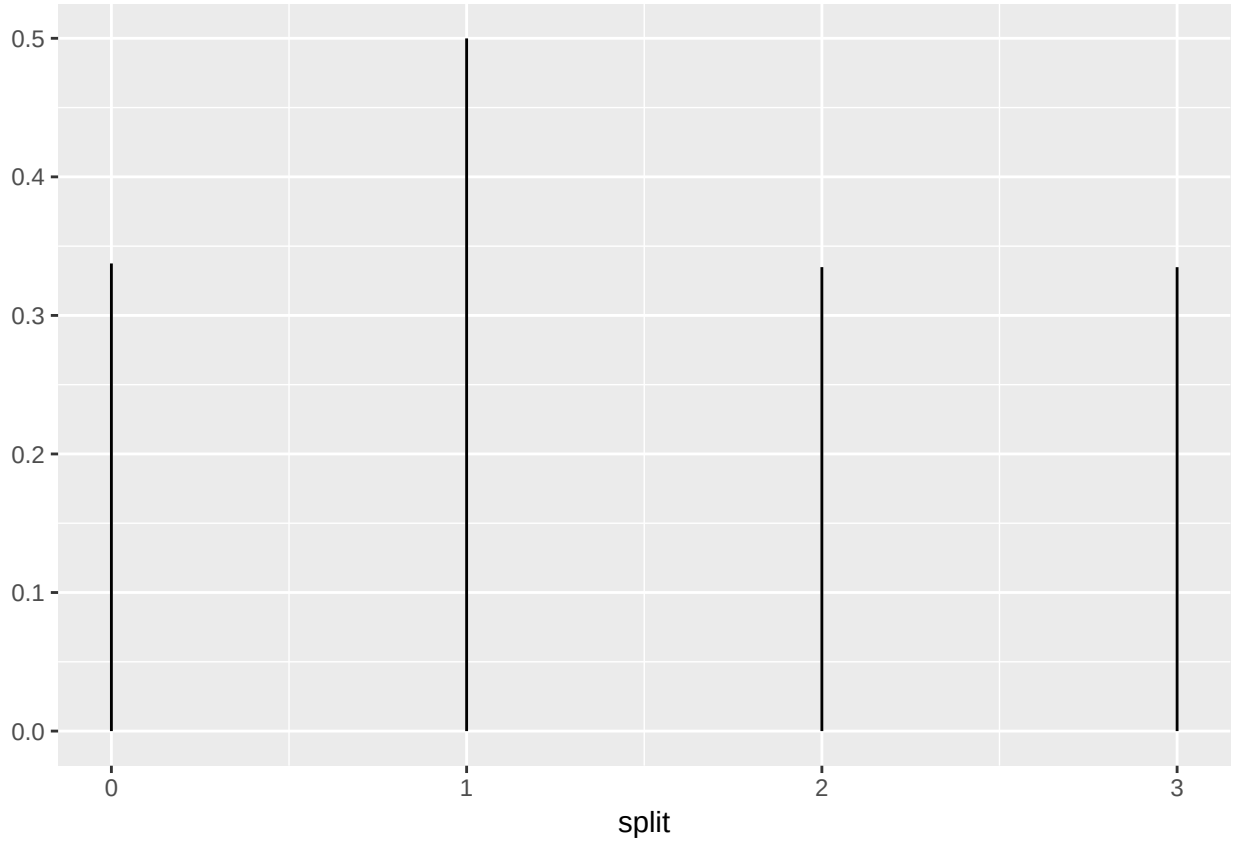
The full code:

```
get_best_choice_set <- function(N, iterationCount){
  best_choice_set_means <- vector()

  for(splitNum in 0:N){
    split_best_choice <- vector()

    for(iterationNum in 1:iterationCount){
      split_best_choice_sub <- vector()
      for(iterationNum in 1:iterationCount){
        split_best_choice_sub <- c(split_best_choice_sub, make_choice(N,splitNum))
        next()
      }
      split_best_choice <- c(split_best_choice, sum(split_best_choice_sub==N)/iterationCount)
      next()
    }

    best_choice_set_means <- c(best_choice_set_means, mean(split_best_choice))
    next()
  }
  #cat(best_choice_set_means)

  df <- data_frame(split=c(0:N), split_mean=best_choice_set_means)

  return(df)
}


make_choice <- function(N, splitNum){
  input_list <- sample(1:N,N,replace = FALSE)

  if(splitNum==0){
    return(input_list[1])
  }
  else if(splitNum==N){
    return (input_list[N])
  }
  else{
    evaluation_group<- head(input_list,splitNum)
    selection_group<-  tail(input_list,N-splitNum)
    evaluationGP_best <- max(evaluation_group)

    for(i in 1:length(selection_group)){
      if(selection_group[i] > evaluationGP_best){
        #cat("return",selection_group[i])
        return(selection_group[i])
      }else{
        if(i == length(selection_group)) {
          #cat("return",selection_group[i])
          return(selection_group[i])
        }else next()
      }
```

```
    }
  }
}
```

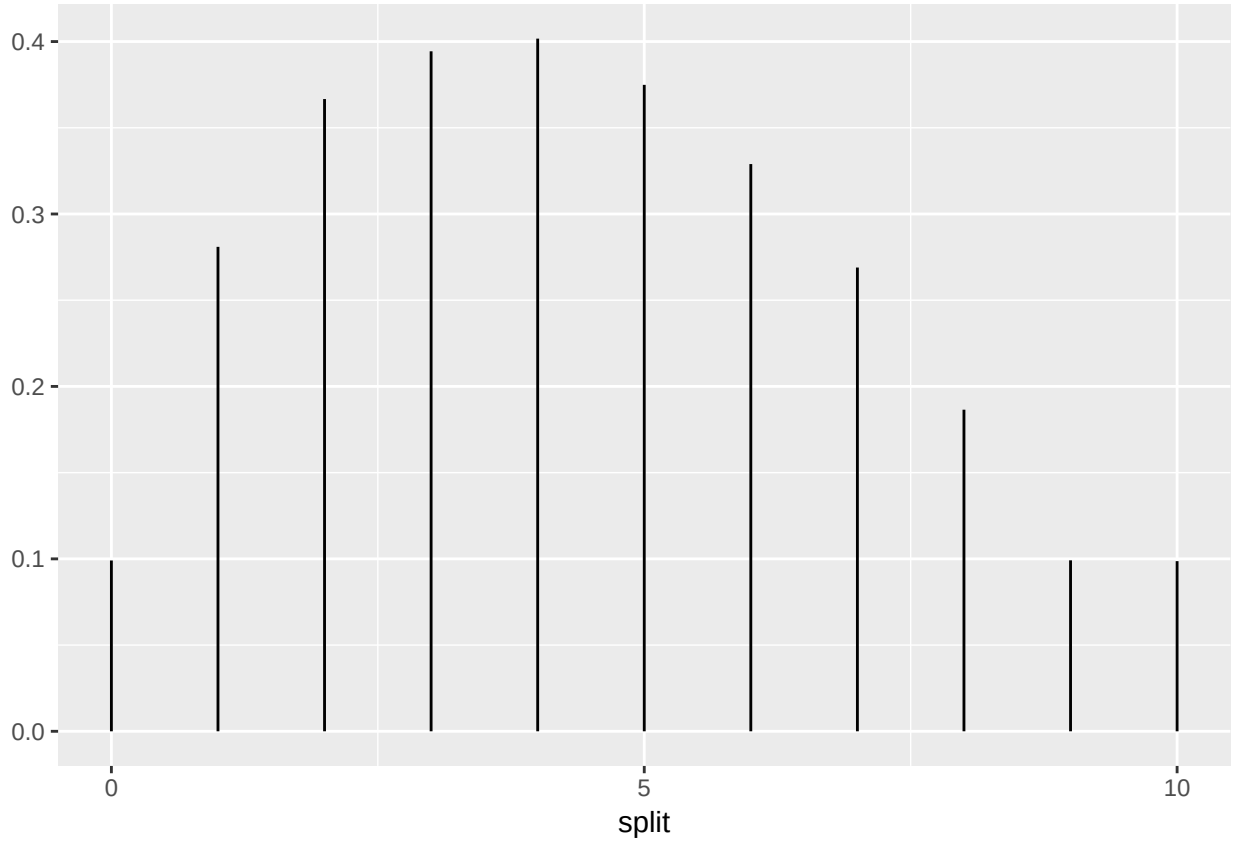
Simulation Plot for N = 3, 10, 100

Plot for N = 3



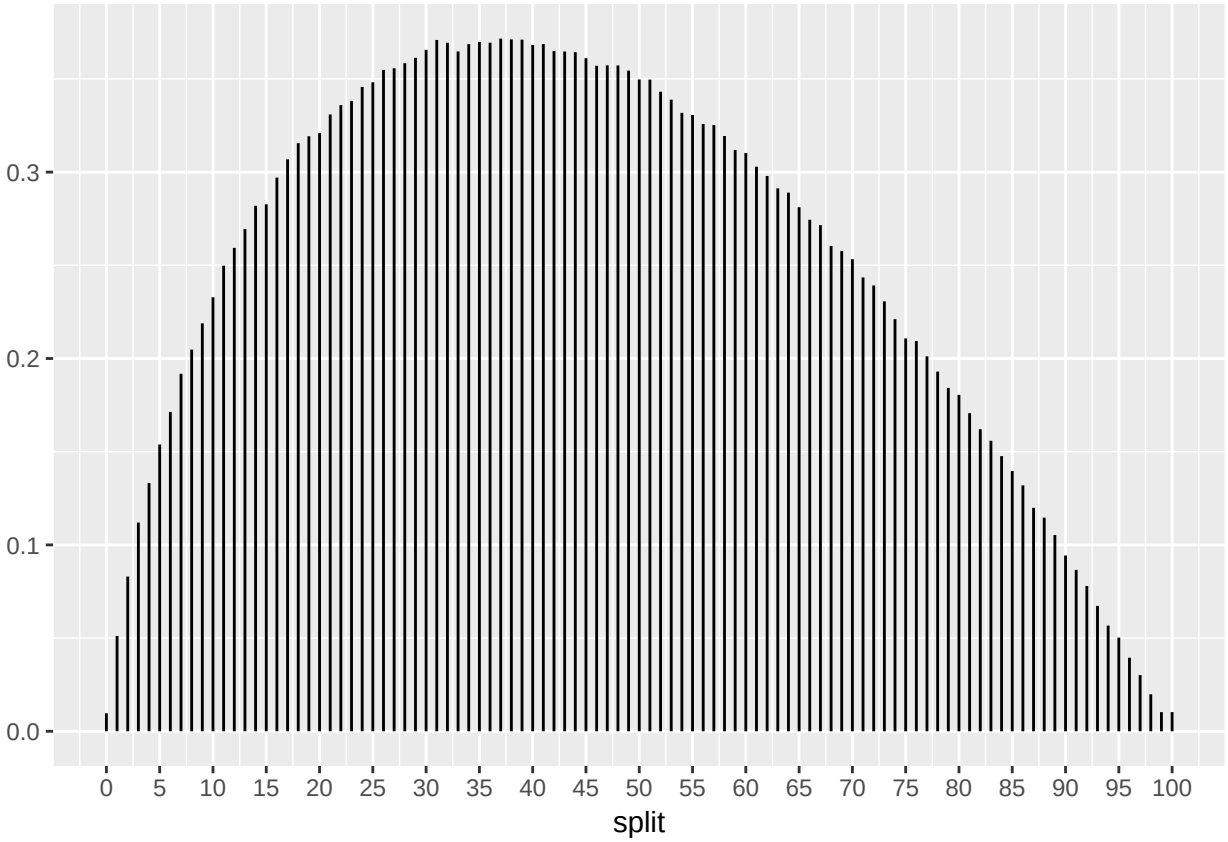

```
## From the plot for N = 3, we can observe that the optimal split position is at split = 1 ,
##  with expected probability of getting the best Prince from the population group = 0.499925
```

Plot for N = 10




```
## From the plot for N = 10, we can observe that the optimal split position is at split = 4 ,
##  with expected probability of getting the best Prince from the population group = 0.4017
```

Plot for N = 100




```
## From the plot for N = 100, we can observe that the optimal split position is at split = 37 ,
##  with expected probability of getting the best Prince from the population group = 0.3716
```

Conclusion

Ideally, the optimal split position should be at around 37% of the total number of Princes N. But due to the computation limitation of my PC, I only run 200*200 iterations to get the probability mean, hence there is some small deviation from 37%.

Total execution time is

```
## Time difference of 4.661851 mins
```