MATH 154 - HW4

Zihao Xu

Septemper 25, 2017

assignment

1. Write code to simulate two strategies for the Pass the Pigs game. Which strategy is better (provide empirical evidence for your answer)?

   Hint: think about breaking the problem down and writing separate functions for the different parts.

   a. First strategy should be based on a simple total rule (how many points you have). Consider playing with one other person and pass the pigs to your friend (and they to you) when/if you reach a certain point total. You choose the point total (same number for both individuals).

   b. Second strategy should be based on not only how many points you have, but also how many points your opponent has.

   c. Last, pit the two strategies against each other and see how often strategy 1

```r
# first we need to define the event probabilities
pigprobs <- c(.349, .302, .224, .088, .03, .006)
names(pigprobs) <- c("side.nodot", "side.dot", "razorback", "trotter",
                   "snouter", "lean.jow")
pigprobs.mat <- pigprobs %*% t(pigprobs)   # matrix multiplication
row.names(pigprobs.mat) <- c("side.nodot", "side.dot", "razorback",
                           "trotter", "snouter", "lean.jow")
pigprobs.mat
```

```
##              side.nodot side.dot razorback  trotter snouter lean.jow
## side.nodot     0.121801 0.105398  0.078176 0.030712 0.01047 0.002094
## side.dot       0.105398 0.091204  0.067648 0.026576 0.00906 0.001812
## razorback      0.078176 0.067648  0.050176 0.019712 0.00672 0.001344
## trotter        0.030712 0.026576  0.019712 0.007744 0.00264 0.000528
## snouter        0.010470 0.009060  0.006720 0.002640 0.00090 0.000180
## lean.jow       0.002094 0.001812  0.001344 0.000528 0.00018 0.000036
```

```r
# next we need to account for the number of points in each entry.
pwin.mat <- pigprobs.mat  # make it the right size
pwin.mat[] <- 0         # set all the entries to zero

pwin.mat[1,1] <- pwin.mat[2,2] <- 1
pwin.mat[3,1] <- pwin.mat[3,2] <- pwin.mat[1,3] <- pwin.mat[2,3] <-
  pwin.mat[4,1] <- pwin.mat[4,2] <- pwin.mat[1,4] <- pwin.mat[2,4] <- 5
pwin.mat[5,1] <- pwin.mat[5,2] <- pwin.mat[1,5] <- pwin.mat[2,5] <-
  pwin.mat[4,3] <- pwin.mat[3,4] <- 10
pwin.mat[6,1] <- pwin.mat[6,2] <- pwin.mat[1,6] <- pwin.mat[2,6] <-
  pwin.mat[5,3] <- pwin.mat[5,4] <- pwin.mat[3,5] <- pwin.mat[4,5] <- 15
pwin.mat[6,3] <- pwin.mat[6,4] <- pwin.mat[3,6] <- pwin.mat[4,6] <-
  pwin.mat[3,3] <- pwin.mat[4,4] <- 20
pwin.mat[6,5] <- pwin.mat[5,6] <- 25
pwin.mat[5,5] <- 40
pwin.mat[6,6] <- 60
pwin.mat
```

```
##             side.nodot side.dot razorback trotter snouter lean.jow
## side.nodot           1        0         5       5      10       15
## side.dot             0        1         5       5      10       15
## razorback            5        5        20      10      15       20
## trotter              5        5        10      20      15       20
## snouter             10       10        15      15      40       25
## lean.jow            15       15        20      20      25       60
```

Here is an example of a function you might want to use. In this particular example, we are sampling from the integers 1 to 10 (not what you want) and the probability of each integer is 1:10/55 (P(X=1) = 1/55, . . . P(X=10) = 10/55). Note that the probabilities I've set are *also* not what you want.

```r
one_draw <- function(){
  row_num = sample(c(1:length(pigprobs)), 1, prob = pigprobs, replace = TRUE)
  col_num = sample(c(1:length(pigprobs)), 1, prob = pigprobs, replace = TRUE)
  pwin.mat[row_num, col_num]
}

# (a) Strategy 1: for every turn, when the player accumulates 26 points or more, stop playing
#                 or pig out
strategy_one <- function(increment = 26){
  sum = 0
  while(sum <= increment){
      this_draw <- one_draw()
      if(this_draw == 0){
        sum = 0
        break
      }
      sum = sum + this_draw
  }
  sum
}

# (b) Strategy 2: for every turn, when the player accumulates 26 points or more, stop playing
#                 or pig out;
#                 But when the opponent's score is more than my score, take more risk by
#                 increasing the stopping threshold to 34, i.e. play till 34 stop or pig out
strategy_two <- function(increment = 26, my_score, opponent_score){
    sum = 0
    if (opponent_score > my_score){
        increment <- 34
    }
    while(sum <= increment){
        this_draw <- one_draw()
        if(this_draw == 0){
          sum = 0
          break
        }
        sum = sum + this_draw
    }
    sum
}

# (c) two_player_game function simulates a game played by strategy 1 against strategy 2.
#     the verbose output shows the number of rounds played as well as who own.
#     the normal output returns TRUE if play_one_score > play_two_score ,else false
two_player_game <- function(verbose = FALSE){
    #scores of the two players and number of rounds played
    play_one_score <- 0
    play_two_score <- 0
    number_of_rounds = 0

    while((play_one_score < 100) & (play_two_score < 100)){
        number_of_rounds = number_of_rounds + 1
        play_one_score = play_one_score + strategy_one()
        play_two_score = play_two_score + strategy_two(my_score = play_two_score, opponent_score = play_
    }

    if(verbose){
      print(paste("Number of rounds:", number_of_rounds))
      if(play_one_score > play_two_score){
          print("Strategy 1 won!")
      }else{
          print("Strategy 2 won!")
      }
    }
    play_one_score > play_two_score
}
```

Now, play the game for 10000 and give the result

```r
#Simulate 10000 games:
results <- c()
for(i in 1:10000){
    results <- c(results, two_player_game())
}

print(paste("Number of games won by strategy one:", sum(results)))
```

```
## [1] "Number of games won by strategy one: 4903"
```

```r
print(paste("Number of games won by strategy two:", sum(!results)))
```

```
## [1] "Number of games won by strategy two: 5097"
```

From the result we can tentatively conclude that strategy two is slightly better than strategy one.

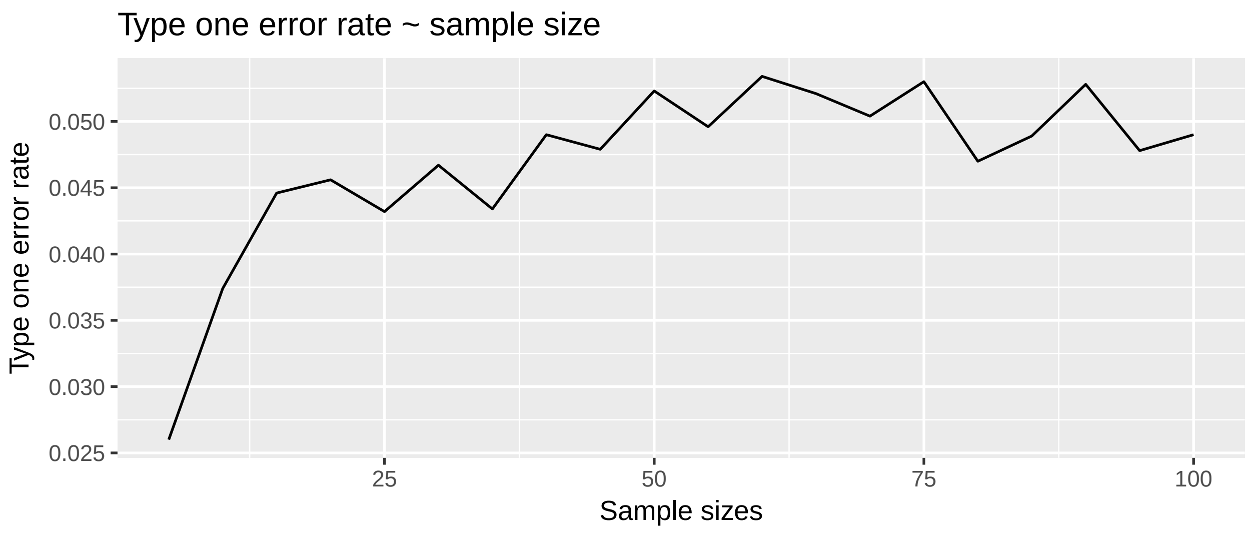
2. One of the t-test assumptions is that the data start off as normally distributed. Using data which are exponentially distributed with a mean of 200 `rexp(sampsize, 1/200)`), test whether the type 1 error rate is correct. Repeat the entire analysis for two samples each of size 5 to 100 by units of 5 `seq(5,100, by=5)`). The steps of the simulation should be:
   a. generate two datasets from the same population with the same sample size.
   b. perform a t.test to see if you reject the null hypothesis (you shouldn't, but sometimes you will). somehow track whether or not you've rejected H_0. You might need to track the p-value using `t.test(samp1, samp2)$p.value`.
   c. repeat the process many many times
   d. record how often you make a type I error (as a proportion) for reps=1000 (at least)
   e. repeat for other sample sizes
   f. plot the empirical type I error rate as a function of sample size
   g. Comment on whether the t-test is robust to the assumption of underlying normal data. [If you are curious, you can also investigate the change in mean of the exponential distribution – more or less

skewed. The skewness also affects the appropriateness of the p-value calculation.]

```r
library(ggplot2)
library(dplyr)

# (a) - (e) Simulation
sample_sizes <- seq(5,100, by = 5)
reps = 10000
type_one_error_rates <- c()
for (s_size in sample_sizes){
    type_one_vector <- c()
    for(i in 1:reps){
        samp1 <- rexp(s_size, 1/200)
        samp2 <- rexp(s_size, 1/200)
        t.test(samp1, samp2)$p.value
        type_one_vector <- c(type_one_vector, t.test(samp1, samp2)$p.value < 0.05)
    }
    type_one_error_rates <- c(type_one_error_rates, sum(type_one_vector)/length(type_one_vector))
}

# (f) plot the empirical type I error rate as a function of sample size
data.frame(sample_sizes, type_one_error_rates) %>%
  ggplot(aes(x = sample_sizes, y = type_one_error_rates)) +
  geom_line() +
  labs(x = "Sample sizes", y = "Type one error rate", title = "Type one error rate ~ sample size")
```



(g) Comments: it appears that, in our case, the t-test is still quite robust except for small sample sizes (5 - around 15). As we can see from the graph, the type I error rate goes from 0.01 to around 0.05 and then fluctuates around 0.05 as the sample size increases from 5 to 100.

3. (Q10 From Nolan and Temple Lang, [Yes, this problem is also part of the homework assignment!])

A major simplification in our blackjack simulation is that we have effectively given the gambler an infinite amount of money. How do things change if we also account for ruin, i.e., the gambler running out of money? (You don't need to re-code anything, but describe, in words, how the situation and then how the code would change.) You may need to glance through the end of the in-class notes on simulating Blackjack, but you do not need to understand every step/function.

Changes: 1. There needs to be a local variable, *money*, that keeps track of the player's money, starting at some user-defined positive initial value (e.g. $money \ < - \ 20$); 2. When doing repeated simulations, after each game, we need to check if play has a ruin (i.e. if($money \ < \ 20$)). If this is the case, terminate the simulation by *break*; otherwise, we could continue playing.