Monty Hall was awesome

Greg Gloor

The Monty Hall problem

The Monty Hall problem shows the danger of not thinking about a probability vector properly. In this problem, you are given a choice of 3 doors, one of which is a winner the other two of which are losers. After you choose, the Monty reveals one of the doors that you did not choose as being a loser. You are given the choice of switching after the reveal. Should you?

The usual response by those not familiar is to believe that the odds of choosing right are the same prior and post reveal. This is wrong as can be demonstrated by thinking about the prior and post reveal data as a probability vector.

Prior to the reveal, there $P = [\frac{1}{3}, \frac{1}{3}, \frac{1}{3}]$ likelihood of being correct. However, post reveal, the probability changes to $P = [0, \frac{2}{3}, \frac{1}{3}]$ or $P = [\frac{2}{3}, 0, \frac{1}{3}]$ depending on whether Monty chose door 1 or door 2. Observe that the door that Monty chooses has its prior probability added to the probability of the only non-chosen door to give a new probabily vector post-reveal. The open door does not disappear, it is just that the probability of that door being a winner has been reduced to 0.

The `R` code below shows the result of simulating the problem with a 3 door setup, assuming you choose door 3 each time. It does not matter which door you choose for this to work.

```
# generalized to any number of elements in the vector
n.elements <- 3

# make a random vector where the largest value is the winner

make.rn <- function(){
  rn <- runif(n.elements)
  rn[rn == max(rn)] <- 1
  rn[rn < 1] <- 0
  return(rn)
}

# if the last element is the winner, choose first or second at random
# does not matter since all are losers
# otherwise choose the one feature that is a winner and keep the last
# which must be a loser
be.Monty <- function(rn){
  if(rn[n.elements] == 1) {
    rnum <- runif(1)
     oneOrTwo <- 2
     if(rnum < 0.5) oneOrTwo =1
     new.rn <- c(rn[c(oneOrTwo,n.elements)])
  }else{
    new.rn <- c(rn[rn == 1], rn[n.elements])
  }
  return(new.rn)
}

output <- matrix(data=NA, nrow=1000, ncol=2)
for(i in 1:1000){
```

```
  rn <- make.rn()
  output[i,] <- be.Monty(rn)
}
```

So in this simulation changing doors gives a winning percentage of 0.649, and not changing gives a winning percentage of 0.351.

So what?

So what does this have to do with high throughput sequencing? It is common practice in any high throughput sequencing experiment to collect data on *all* features in the dataset and then to filter the data to exclude low count reads that are likely to fall below the *significance* threshold. So a dataset of 8000 features, may be reduced to fewer than 1000 features because the vast majority are *sparse* and hence of no interest.

At this point, the remaining 1000 features are tested for *relative difference* between groups, often using a multiple test correction procedure. Lets think about this from the perspective of Monty Hall.

The analyst has collected 8000 doors worth of data, and reduced that to 1000 by having Monty remove 7000 known losers from the dataset. So the remaining 1000 gain the probability of the removed 7000 - meaning that they become more likely to be winners simply because they remain in the dataset. This may not be a large problem is the likelihood that the ones being removed being different or distinguishing is truly close to 0.

Worse, the multiple test correction is done assuming a number of tests of 1000, not 8000! Thus, the multiple test correction will always under-estimate the false positive rate.

So is it any wonder that we have many false positive results?