RA Exercise

Yan Bo Zeng

March 9, 2020

```r
library(DescTools)
library(arrangements)
library(dplyr)
library(haven)
library(readr)
```

Probability of an individual rating, given the true quality

This function calculates $P(q_{rt}|q_j^*)$ straightforwardly from Normal CDF and the ordered probit. It takes three inputs:

- `qrt`, q_{rt}, integer between 1 and 5, inclusive, representing an individual rating a reviewee can possibly give
- `cut_point`, cut points for the ordered probit, default at $(1, 2, 3, 4)$, must be of length 4
- `qj_true`, q_j^*, the true underlying quality

The vectorized version of this function takes a vector or series as input and returns the answers in a vector

```r
P_qrt_qj <- function(qrt, cut_point = 1:4, qj_true) {

  if(qrt == 1) {
    return(pnorm(cut_point[1], mean = qj_true, sd = 1))
  } else if (qrt == 2) {
    return(pnorm(cut_point[2], mean = qj_true, sd = 1) -
             pnorm(cut_point[1], mean = qj_true, sd = 1))
  } else if(qrt == 3) {
    return(pnorm(cut_point[3], mean = qj_true, sd = 1) -
             pnorm(cut_point[2], mean = qj_true, sd = 1))
  } else if(qrt == 4) {
    return(pnorm(cut_point[4], mean = qj_true, sd = 1) -
             pnorm(cut_point[3], mean = qj_true, sd = 1))
  } else if(qrt == 5) {
    return(1 - pnorm(cut_point[4], mean = qj_true, sd = 1))
  } else {
    return(0)
  }

}

## Vectorize input and output
P_vector <- function(qrt, cut_point = 1:4, qj_true) {

  result <- rep(0, length(qrt))
  for(i in 1:length(qrt)) {
    result[i] <- P_qrt_qj(qrt[i], cut_point, qj_true)
  }

  return(result)
}
```

Test the vectorized function

```r
0:6
```

```
## [1] 0 1 2 3 4 5 6
```

```r
P_vector(0:6, qj_true = 1)
```

```
## [1] 0.000000000 0.500000000 0.341344746 0.135905122 0.021400234 0.001349898
## [7] 0.000000000
```

```r
P_vector(0:6, qj_true = 3)
```

```
## [1] 0.00000000 0.02275013 0.13590512 0.34134475 0.34134475 0.15865525 0.00000000
```

```r
P_vector(0:6, qj_true = 5)
```

```
## [1] 0.000000e+00 3.167124e-05 1.318227e-03 2.140023e-02 1.359051e-01
## [6] 8.413447e-01 0.000000e+00
```

All possible combinations yielding a specific rating, given the total number of reviews

Utilizing the functionality of R, we are able to get all $\binom{5+N_{jt}-1}{N_{jt}}$ possible combinations in a matrix. Each row of the matrix is a possible combination. An index is computed after taking the row-wise averages and then rounding them to the nearest half integer.

One important thing to mention here is the way we round the average ratings. We round down at the .25 but round up at the .75. For example, $round(1.25, 0.5) = 1$ and $round(1.75, 0.5) = 2$

The index identifies the combinations which yield the desired rounded average rating. A submatrix containing all the desired combinations is returned.

`combinatoric` takes two arguments:

- `qjt`, $\tilde{q}_{jt}$, the observable, rounded average ratings
- `N`, N_{jt}, the number of reviewers who arrived prior to t

```r
## Returns a matrix where each row is a possible combination
combinatoric <- function(qjt, N) {

  comb_mat <- combinations(1:5, replace = TRUE, k = N)

  rounded <- RoundTo(apply(comb_mat, MARGIN = 1, FUN = mean), 0.5)

  index <- rounded == qjt

  if(!any(index)) {
    return(0)
  }

  return(comb_mat[index, ])
}
```

Take a look at a sample submatrix

```
combinatoric(qjt = 3.5, N = 20)[sample(1:27, 8, replace = FALSE), ]
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
## [1,]    1    1    1    1    1    1    1    2    4     4     4     5     5     5
## [2,]    1    1    1    1    1    1    1    2    3     4     5     5     5     5
## [3,]    1    1    1    1    1    1    1    2    4     5     5     5     5     5
## [4,]    1    1    1    1    1    1    1    4    4     4     4     5     5     5
## [5,]    1    1    1    1    1    1    1    1    4     5     5     5     5     5
## [6,]    1    1    1    1    1    1    1    1    4     4     5     5     5     5
## [7,]    1    1    1    1    1    1    1    3    3     3     5     5     5     5
## [8,]    1    1    1    1    1    1    1    4    4     4     4     4     4     5
##      [,15] [,16] [,17] [,18] [,19] [,20]
## [1,]     5     5     5     5     5     5
## [2,]     5     5     5     5     5     5
## [3,]     5     5     5     5     5     5
## [4,]     5     5     5     5     5     5
## [5,]     5     5     5     5     5     5
## [6,]     5     5     5     5     5     5
## [7,]     5     5     5     5     5     5
## [8,]     5     5     5     5     5     5
```

Number of ways of a specific combination

`num_ways` takes a given combination and calculates the total number of arrangements this specific combination can fit. It first calls a frequency table over the unique numbers in the combination, and then calculates

$$\prod_{i=1}^{n} \binom{N_{jt} - \sum_{k<i} f_k}{f_i}$$

where $n \leq 5$ is the total number of unique numbers, f_i is the frequency of the i-th unique number, and $f_0 = 0$. For example, the number of arrangements for $[1, 1, 1, 2, 3, 3, 3, 5]$ is $\binom{8}{3}\binom{5}{1}\binom{4}{3}\binom{1}{1} = 1120$

```
## Please make sure the input contains only one combination
num_ways <- function(combination) {

  ## Frequency of each number in the combination
  freq <- table(combination)

  ## Number of ways left to choose from
  num_ways_left <- c(length(combination),
                     length(combination) - cumsum(freq)[-length(freq)])

  return(choose(num_ways_left, freq) %>% prod())
}
```

Test the example above

```
num_ways(c(1,1,1,2,3,3,3,5))
```

```
## [1] 1120
```

Checkpoint

`checkpoint` calculates $p(\tilde{q}_{jt}|N_{jt}, q_j^*)$. It takes four arguments:

- `qjt`, $\tilde{q}_{jt}$, the observable, rounded average ratings
- `N`, N_{jt}, the number of reviewers who arrived prior to t
- `cut_point`, cut points for the ordered probit, default at $(1, 2, 3, 4)$, must be of length 4
- `qj_true`, q_j^*, the true underlying quality

This function puts everything together. Given the true underlying quality and a desired rounded average rating, it calculates the probability of each possible combinations by multiplying the probabilities of all individual ratings in and the total number of arrangements of that combination. Then it returns the sum of the probabilities across all possible combinations yielding the given rounded average rating.

```
checkpoint <- function(qjt, N, qj_true, cut_point = 1:4) {

  comb_mat <- combinatoric(qjt, N)

  ## Deal with the case where only 1 possible combination exists
  if(is.null(nrow(comb_mat))) {
    result <- num_ways(comb_mat) *
      prod(P_vector(comb_mat, cut_point, qj_true))
  } else {
    result <- rep(0, nrow(comb_mat))

    for(i in 1:nrow(comb_mat)) {
      result[i] <- num_ways(comb_mat[i, ]) *
        prod(P_vector(unique(comb_mat[i, ]), cut_point, qj_true) ^
               table(comb_mat[i, ]))
    }
  }

  return(sum(result))
}
```

Conditional expectation function

According to the note, estimating $\int f(q_j^*)dP(q_j^*)$ can be done by simply taking an average of $f(\cdot)$ over a vector of random draws from the prior. If I am not misunderstanding the meaning of the notation $dP(q_j^*)$ here, it tells that the marginal probability $\int p(\tilde{q}_{jt}|N_{jt}, q_j^*)dP(q_j^*)$ can be computed by plugging the prior into $p(\tilde{q}_{jt}|N_{jt}, q_j^*)$ (the `checkpoint` function) and then taking an average.

This further implies that

$$\begin{aligned}\mathbb{E}[q_j^*|\tilde{q}_{jt}, N_{jt}] &= \int q_j^* \cdot p(q_j^*|\tilde{q}_{jt}, N_{jt})dq_j^* \\ &= \int q_j^* \cdot \frac{p(\tilde{q}_{jt}|N_{jt}, q_j^*)p(q_j^*)}{marginal}dq_j^* \\ &= \int \frac{q_j^* \cdot p(\tilde{q}_{jt}|N_{jt}, q_j^*)}{marginal}dP(q_j^*)\end{aligned}$$

Therefore, $\mathbb{E}[q_j^*|\tilde{q}_{jt}, N_{jt}]$ can be computed by plugging the prior into $\frac{q_j^* \cdot p(\tilde{q}_{jt}|N_{jt}, q_j^*)}{marginal}$ and then taking an average.

```
expectation <- function(qjt, N, prior, cut_point = 1:4) {

  ## Weed out invalid inputs
  if(qjt < 1 | 5*N < qjt*N | N > qjt*N) {
```

```r
    return("Invalid inputs")
  }

  ## Make sure ratings are in half integers
  if (qjt%%0.5 != 0) {
    return("Average ratings are rounded to the nearest half integer")
  }


  ## The marginal probability
  part <- rep(0, length(prior))

  for(i in 1:length(prior)) {
    part[i] <- checkpoint(qjt, N, qj_true = prior[i], cut_point)
  }

  marginal <- mean(part)

  if(marginal == 0) {
    return(paste("Not a possible rating given that N =", N))
  }

  return(mean(part * prior / marginal))
}
```

Prior and cut points

```r
set.seed(127888)

prior <- read_dta("true_qualities.dta")$q_star[
  sample(1:6000, 500, replace = FALSE)] %>% as.vector()

cutoffs <- read_csv("cutoffs.csv")$cuts1 %>% as.vector()
```

Results

```r
answer <- c(1,3,5,15,20)
series <- c(1,3,5,15,20)

for(i in c(1.5, 2.5, 3, 4, 5)) {

  for(j in series) {

    cat("\nRating =", i, "N =", j, "\n")

    answer[series == j] <- expectation(qjt = i, N = j,
                                       prior, cutoffs)
    print(answer[series == j])
  }
}
```

```
##
## Rating = 1.5 N = 1
## [1] "Not a possible rating given that N = 1"
##
## Rating = 1.5 N = 3
## [1] "-0.34179718028676"
##
## Rating = 1.5 N = 5
## [1] "-0.482589378380818"
##
## Rating = 1.5 N = 15
## [1] "-0.698849968281155"
##
## Rating = 1.5 N = 20
## [1] "-0.750698711252714"
##
## Rating = 2.5 N = 1
## [1] "Not a possible rating given that N = 1"
##
## Rating = 2.5 N = 3
## [1] "-0.175194266998728"
##
## Rating = 2.5 N = 5
## [1] "-0.227261030399465"
##
## Rating = 2.5 N = 15
## [1] "-0.349539434927337"
##
## Rating = 2.5 N = 20
## [1] "-0.375427852142619"
##
## Rating = 3 N = 1
## [1] "-0.020660502550676"
##
## Rating = 3 N = 3
## [1] "-0.0596854387626051"
##
## Rating = 3 N = 5
## [1] "-0.0784703276041668"
##
## Rating = 3 N = 15
## [1] "-0.114309931306992"
##
## Rating = 3 N = 20
## [1] "-0.11986453160997"
##
## Rating = 4 N = 1
## [1] "0.0137247462876837"
##
## Rating = 4 N = 3
## [1] "0.230740885651626"
##
## Rating = 4 N = 5
## [1] "0.282942263238824"
##
## Rating = 4 N = 15
## [1] "0.391788239628503"
##
## Rating = 4 N = 20
## [1] "0.414274610973013"
##
## Rating = 5 N = 1
## [1] "0.252136605045583"
##
## Rating = 5 N = 3
## [1] "0.592895864949939"
##
## Rating = 5 N = 5
## [1] "0.764278558755437"
##
## Rating = 5 N = 15
## [1] "1.30396782935169"
##
## Rating = 5 N = 20
## [1] "1.34445213317985"
```