Iterative Basian Updates, Problem 1.11

Lena S

Coding part for Problem 1.11 + extension to the problem

If something is unclear in the slides/there is a mistake, email me at elenash@stat.ubc.ca

September 20, 2016

Input data for the code

We are going to code the calculations for problem 1.11 in the Tutorial set A. For that, we are going to use pseudocode provided by your instructor in the lecture slides in Conditional probability (slides 53-55). The input values are as shown below:

```
# Input values
pe <- 0.005 #probability of disease P(E)
te <- rep(0.99,5) #prob of positive test given disease
#this is a vector of 5 probabilities
tec <- rep(0.01,5) #prob of positive tests given no disease
test <- c(0,1,1,1,1) #test results part (c1)
```

For loop to compute probabilities

With the input values from previous slide, we are ready to write the for loop to compute probabilities

```
prob <- rep(NA, 6) #empty vector to store the results
prob[1] <- pe #first value is the P(E)

for (i in (1:5)){
if (test[i]==1){p=te[i]; q=tec[i]} #if the test is positive
if (test[i]==0){p=1-te[i]; q=1-tec[i]} #if the test is negativ
#update probability using iterative formula
prob[i+1]=(prob[i]*p)/(prob[i]*p+(1-prob[i])*q)
}

round(prob,5)
[1] 0.00500 0.00005 0.00500 0.33221 0.98010 0.99979
```

Wrapping for loop in a function

We can wrap the for loop in a function to make it easier for us

```
probv <- function(test){
pr <- rep(NA, 6)
pr[1] <- pe
for (i in (1:5)){
if (test[i]==1){p=te[i]; q=tec[i]}
if (test[i]==0){p=1-te[i]; q=1-tec[i]}
pr[i+1]=(pr[i]*p)/(pr[i]*p+(1-pr[i])*q)
}
return(pr) # returns a vector of probabilities
}

test2 <- c(1,1,1,1,0)
probv(test2)
# [1] 0.0050000 0.3322148 0.9801000 0.9997950 0.9999979 0.9997
```

Wrapping for loop in a function

Apply this function to each vector of test results to get the answers

```
> probv(test)
[1] 0.00500 0.00005 0.00500 0.33221 0.98010 0.99979
>
> test2 <- c(1,1,1,1,0)
> probv(test2)
[1] 0.00500 0.33221 0.98010 0.99979 1.00000 0.99979
>
> test3 <- c(1,0,0,0,0)
> probv(test3)
[1] 0.00500 0.33221 0.00500 0.00005 0.00000 0.00000
>
> test4 <- c(0,0,1,0,1)
> probv(test4)
[1] 5e-03 5e-05 0e+00 5e-05 0e+00 5e-05
```

Extension to problem 1.11

Now suppose we set a threshold for the final probability $P(E|I_5)$, meaning in case it is more than α (for example 50%) we give the person treatment and if it's less – we do not. The mistakes that could be made:

- Not giving treatment to a sick patient or
- Giving treatment to a helthy patient

The total probability of mistake can be calculated as follows:

$$P(M) = P(M \cap E) + P(M \cap E^c) = P(M|E)P(E) + P(M|E^c)P(E^c)$$

We will need to simulate 50 000 test results in case person has disease and estimate $P(M|E)$; Then we simulate test results for no disease and estimate $P(M|E^c)$. Then we compute the probability of mistake $P(M)$

Simulating test results given sick

First I make a small adjustment for my function for it to only output the final estimate of probability $P(E|I_5)$. I can just change return(probs) to return(probs[6]). Now we need to simulate test results given patient has disease. It can be done with the following code

```
# Simulating a matrix of test results given sickness
seq1 <- matrix(NA, nrow=50000, ncol=5)
for (j in (1:5)){
seq1[,j] <- rbinom(50000,1,te[j])}
```

This creates a matrix where each row is a result of 5 tests with 50 000 rows. We can calculate our probability estimates and estimate $P(M|E)$ as follows:

```
#Calculate the probability estimate
prob1 <- apply(seq1, MARGIN=1, probv)
alpha=0.5 # this is a treshold we set
#The fraction of wrongly not given treatment
frac1 <- sum(prob1<=alpha)/ length(prob1) #[1] 0.3420767
```

Simulating test results given healthy

Simulating test results given patientis healthy. The code is similar to the one on previous slide

```
#Simulating test results given no sickness
seq2 <- matrix(NA, nrow=50000, ncol=5)
for (j in (1:5)){
seq2[,j] <- rbinom(50000,1,tec[j])
}

#Calculate the probability estimate
prob2 <- apply(seq2, MARGIN=1, probv)

#Fraction of those wrongly given treatment
frac2 <- sum(prob2>=alpha)/ length(prob2)  #[1] 3.333333e-05
```

Now we can calculate $P(M) = 0.002$ for $\alpha = 0.5$. Every cutoff will provide different $P(M)$.

Trying different thresholds

We can try different values of the threshold to calculate the probability of error for each and choose the value that gives the smallest probability of error

```
# Error probabilities for different cutoffs
alpha <- seq(from=0.01, to=0.99, by=0.01)
Total <- rep(NA, length(alpha))
for (k in (1:length(alpha))){
frac1 <- sum(prob1<=alpha[k])/ length(prob1)
frac2 <- sum(prob2>=alpha[k])/ length(prob2)
Total[k] <- 0.995*frac2+0.005*frac1
}
```

Choosing the best option is left as an exercise.