STA 623 homework 6

Lingyun Shao

Oct. 29, 2018

Problem Statement

- Simulate training data on test scores from $X_i \sim Binomial(100, \theta_i)$ and the latent ability for individual i, $\theta_i \sim Beta(2, 2)$, $i = 1, ..., 100$.
- Suppose our interest is in defining a decision rule which selects the lowest 5% of individuals for extra tutoring.
- Defining an appropriate loss function and estimate the Bayesian expected posterior loss.
- Comment on whether the decision rule can be improved or not.

Hint:

(i) in practice the parameters of the beta density (2,2) won't be known.

(ii) you may want to use MCMC sampling, though you can be successful using other strategies.

Solution

(0) Generating training data

First, we may want to generate some training data as told in the problem statement.

```
set.seed(1)
n = 100
theta = rbeta(n, 2, 2)
X = rbinom(n, 100, theta)
```

(1) Theories

Then, since we are interested in determing whether some individual's ability is in the lowest 5% (i.e. θ below the 0.05 quantile of its distribution) given the observed data. However, we may not know the true parameters a, b in the Beta distribution of $\theta's$, so we need to go one step further and sample from its posterior distribution given the training data to get its MC approximation.

According to Homework 6 guidelines (Thanks for that! Felipe and Micheal)

$$\begin{aligned}
\pi(x) &= Pr(T = 1 | X = x, \underbrace{X^{(n)}}_{training}) \\
&= \int\int Pr(\theta < q_{0.05(a,b)} | x, a, b) \pi(a, b | X^{(n)}) da db \\
&\approx \underbrace{\frac{1}{T} \sum Pr(\theta < |q_{0.05(a_t, b_t)} | x, a_t, b_t)}_{Using\ MC\ approximation}
\end{aligned}$$

The posterior distribution of a, b can be derived from

$$\pi(a, b|X^{(n)}) \propto L(X^{(n)}|a, b)\pi(a, b)$$

a) $\pi(a, b)$

As for the prior, since a, b are supposed to be positive. To make our life easier, I just used two independent Gammas because Gamma distribution has positive support. I followed the example in the guidelines and let

$$a \sim Gamma(3, 2), \quad b \sim Gamma(2, 1)$$

b) $L(X^{(n)}|a, b)$

In terms of the likelihood, as is given in the guidelines, we have

$$\begin{aligned} L(X^{(n)}|a, b) &= \iint_{\boldsymbol{\Theta}} L(X^{(n)}|\boldsymbol{\theta})\pi(\boldsymbol{\theta}|a, b)d\boldsymbol{\theta} \\ &= \prod_{i=1}^{n} \int_{\Theta_i} L(x_i|\theta_i)\pi(\theta_i|a, b)d\theta_i \\ &= \prod_{i=1}^{n} \int_0^1 \binom{100}{x_i} \theta_i^{x_i}(1-\theta_i)^{100-x_i} \times \frac{1}{B(a, b)}\theta_i^{a-1}(1-\theta_i)^{b-1}d\theta_i \\ &= \prod_{i=1}^{n} \binom{100}{x_i} \frac{B(a + x_i, b + 100 - x_i)}{B(a, b)} \end{aligned}$$

Given a and b, the scores x_i are independent and have a Beta-Binomial$(100, a, b)$ distribution.

$$Pr(X = j|a, b) = \binom{100}{j} \frac{B(a + j, b + 100 - j)}{B(a, b)}, \quad j = 1, ..., 100$$

(2) MCMC samples

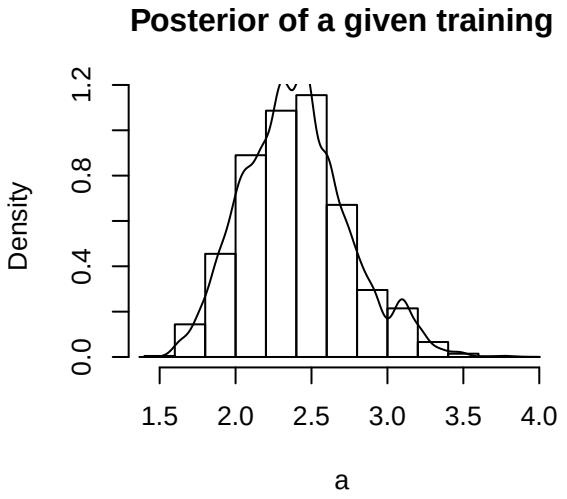
Then with all these theories, we can draw MCMC samples from the posterior distribution $\pi(a, b|X^{(n)})$ using Metropolis algorithm. We choose to use a simple normal distribution with variance 1 as our proposal distribution.

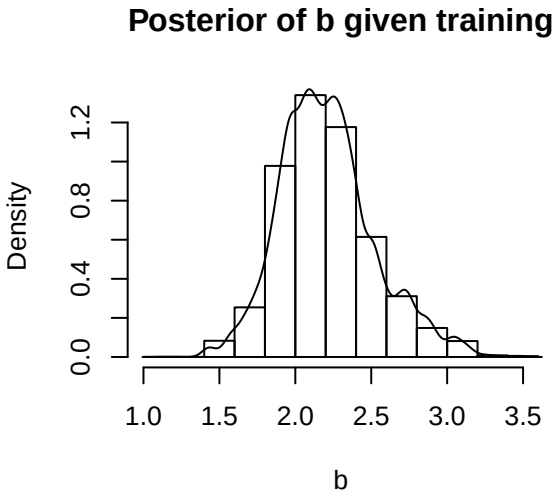
```
B = 500
S = 1e4
a.al = 3; a.be = 2
b.al = 2; b.be = 1
al = c(a.al, b.al); be = c(a.be, b.be)
dlbetabinom = function(j, m, a, b) {
  if(a<=0|b<=0) {
    return(-Inf) # handle exceptional cases
  } else {
    return(lchoose(m,j)+lbeta(a+j,b+m-j)-lbeta(a,b))
  }
}
```

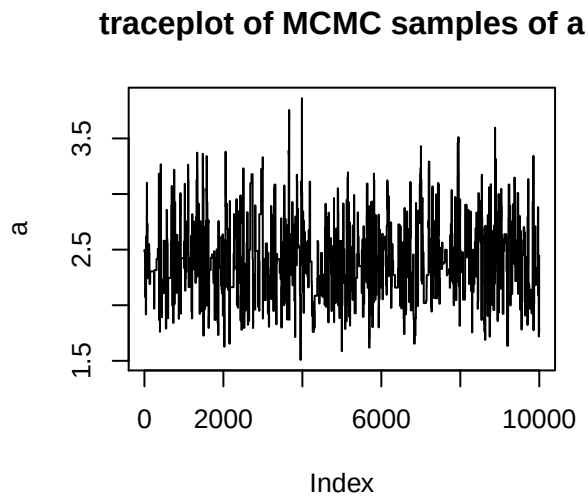
```r
ab = c(1,1) # initial values
AB = ab
accept = rep(0, S)
for(i in 2:(B+S)) {
  ab_prop = rnorm(2, ab, 1)
  log_pr = sum(dgamma(ab_prop, al, be, log = TRUE)) +     # proposed prior
    sum(dlbetabinom(X, 100, ab_prop[1], ab_prop[2])) -    # proposed likelihood
    sum(dgamma(ab, al, be, log = TRUE)) -                 # last prior
    sum(dlbetabinom(X, 100, ab[1], ab[2]))                # last likelihood
  u = runif(1)
  if(u < exp(log_pr)) {
    if(i > B) accept[i-B] = 1
    AB = rbind(AB, ab_prop)
    ab = ab_prop
  } else {
    AB = rbind(AB, ab)
  }
}
# throw away Burn-in
AB = AB[-(1:B),]
rownames(AB) = NULL
colnames(AB) = c('a', 'b')
# acceptance rate
cat('acceptance rate:',mean(accept))
```

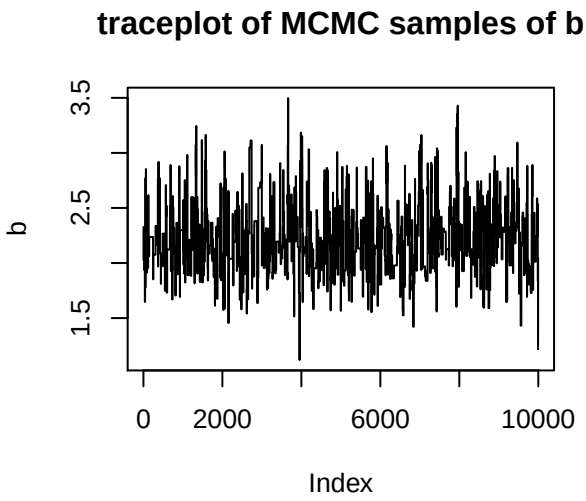
```
## acceptance rate: 0.0893
```

```r
# plots
par(mfrow = c(2,2))
hist(AB[,1], freq=FALSE, main = 'Posterior of a given training', xlab = 'a')
lines(density(AB[,1]))
hist(AB[,2], freq=FALSE, main = 'Posterior of b given training', xlab = 'b')
lines(density(AB[,2]))
plot(AB[,1], type = 'l', main = 'traceplot of MCMC samples of a', ylab = 'a')
plot(AB[,2], type = 'l', main = 'traceplot of MCMC samples of b', ylab = 'b')
```









(3) Monte Carlo Approximation

After obtaining our MCMC samples from the posterior $\pi(a, b|X^{(n)})$, we can plug them in and use Monte Carlo Approximation to get the $\pi(x)$ by simply taking the average of probability over all samples. Here I define a function `pr_tutor` to do the MC approximation and calculate $\pi(x)$.

```
pr_tutor = function(x, sam = AB) {
  nsam = nrow(AB)
  a = sam[,1]
  b = sam[,2]
  a.pos = a + x
  b.pos = b + 100 - x
  q = qbeta(0.05, a, b)
  pr = pbeta(q, a.pos, b.pos)
  return(mean(pr))
}
```

(4) Loss function and Decision rule

Further, we need to propose a loss function to set a decision rule. Since the decision and true state of nature are both binary, then we can use a simple binary 0-1 loss function like

$$Loss(\theta, a) = I_{(\theta \neq a)}$$

where $\theta = \begin{cases} 1, & needs\ tutoring \\ 0, & not\ need\ tutoring \end{cases}$ and $a = \begin{cases} 1, & gets\ tutored \\ 0, & not\ get\ tutored \end{cases}$

So the Bayesian expected loss can be computed as

$$\begin{aligned} B(a) &= Pr(T = 1|X = x)Loss(1, a) + Pr(T = 0|X = x)Loss(0, a) \\ &= \pi(x) \times Loss(1, a) + (1 - \pi(x)) \times Loss(0, a) \end{aligned}$$

```
B_exp_loss = function(a, x) {
  if(a %in% c(0,1)) {
    p = pr_tutor(x)
    return(p * (a==0) + (1-p) * (a==1))
  } else {
    stop('argument a should be either 0 or 1')
  }
}
res = NULL
for(x in 1:100) {
  bloss = NULL
  for(a in c(0,1)) {
    bloss = c(bloss, B_exp_loss(a, x))
  }
  bloss = c(bloss, which.min(bloss)-1)
  res = rbind(res, bloss)
}
rownames(res) = paste0('x = ', 1:100)
colnames(res) = c('a = 0', 'a = 1', 'Bayes action')
kable(res[11:20, ], digits = c(3,3,0), caption = 'Bayes expected loss')
```

Table 1: Bayes expected loss

	a = 0	a = 1	Bayes action
x = 11	0.845	0.155	1
x = 12	0.781	0.219	1
x = 13	0.705	0.295	1
x = 14	0.621	0.379	1
x = 15	0.533	0.467	1
x = 16	0.444	0.556	0
x = 17	0.360	0.640	0
x = 18	0.283	0.717	0
x = 19	0.216	0.784	0
x = 20	0.160	0.840	0

As is shown in the table above (only shows informative results of some values of x's), we can easily find that

the Bayes action (optimizing Bayes expected loss) is 1 for $x = 1, ..., 15$ and 0 for $x = 16, ..., 100$. The result means that if we observed some individual having a test score less than or equal to 15, then we might better choose to have them tutored, otherwise we better decide no to have them tutored.

So the Bayes decision is

$$\delta_B(x) = \begin{cases} 1 \ (gets\ tutored), & x = 1, ..., 15 \\ 0 \ (not\ get\ tutored), & x = 16, ..., 100 \end{cases}$$

Some Comments

Equivalence of using threshold

Unsurprisingly, we can find that here using the 0-1 loss function is exactly the same as using a threshold $\tau = 0.5$ in the setting below:

We can set a decision rule like

$$\delta(x) = \begin{cases} 1, & \pi(x) \geq \tau \\ 0, & \pi(x) < \tau \end{cases}$$

Where $\pi(x)$ is the marginal probability that a student's ability lies below the 5% and needs tutoring ($T = 1$) given the testing score x (i.e. averaging over all possible values of a, b).

$$\begin{aligned} B(0) &= \pi(x) \times Loss(1,0) + (1 - \pi(x)) \times Loss(0,0) = \pi(x) \\ B(1) &= \pi(x) \times Loss(1,1) + (1 - \pi(x)) \times Loss(0,1) = 1 - \pi(x) \\ B(1) &\leq B(0) \iff \pi(x) \geq 0.5 \end{aligned}$$

So we can easily see the equivalence.

Then we may wonder if the threshold of 0.5, i.e. the loss function being 0-1 loss is good enough and whether our decision rule can be improved by choosing them differently.

Here I define a decision function base on our MCMC samples and the choice of threshold to give the optimal decision.

```
d_tutor = function(x, tau = 0.5) {
  p = pr_tutor(x)
  ifelse(p>=tau, 1, 0)
}
```

Generating test data

We can generate more test data and, based on the new data and different threshold, get FPR and FNR ($\theta = 1$ being positive).

To simply this process, I just assume $\theta_i \sim Beta(2, 2)$ and generated 500 $\theta_i's$ and their test score $x_i's$. Then we have their true state of nature of whether θ_i being below the lowest 5% and our decision based on τ and can compute FPR, FNR and Accuracy to check the performance of some thresholds.

```r
set.seed(2)
test.th = rbeta(100, 2, 2)
test.X = rbinom(100, 100, test.th)
q = qbeta(0.05, 2, 2)
state = test.th < q
t = seq(0.1,0.9,by=0.1)
D = NULL
p = progress_estimated(100)
for(x in test.X) {
  d = NULL
  for(tau in t) {
    d = c(d, d_tutor(x, tau))
  }
  D = rbind(D, d)
  p$tick()$print()
}


FNR = apply(D, 2, function(x) sum(state==1&x==0)/sum(state==1))
FPR = apply(D, 2, function(x) sum(state==0&x==1)/sum(state==0))
AC = apply(D, 2, function(x) mean(state==x))
tb = rbind(FNR, FPR, AC)
colnames(tb) = paste0('tau=',t)
rownames(tb)[3] = 'Accuracy'
tb %>% kable(digits = 3)
```

	tau=0.1	tau=0.2	tau=0.3	tau=0.4	tau=0.5	tau=0.6	tau=0.7	tau=0.8	tau=0.9
FNR	0.000	0.000	0.000	0.125	0.125	0.250	0.250	0.375	0.50
FPR	0.109	0.087	0.076	0.076	0.076	0.043	0.043	0.011	0.00
Accuracy	0.900	0.920	0.930	0.920	0.920	0.940	0.940	0.960	0.96

From the table above, we can see that the threshold of $\tau = 0.5$ is not the best regarding FNR and FPR ($\tau = 0.3$ has the same FPR and a lower FNR) or Accuracy.

Thus, we may have some ideas about improving the decision rule. We may choose some uneven loss for the true state of nature being 0 or 1, i.e. choosing a different threshold τ.