STAT534:Statistical Computing
Homework 4

Po-An, Chen(Andy Chen)

April 24, 2022

1 Bayesian Inference in Univariate Logistic Regression

We assume to have observed the n independent samples

$$\mathcal{D} = \{(y_1, x_1), ..., (y_n, x_n)\}$$

The response variable Y is binary, i.e. $y_i \in \{0, 1\}$ for $i = 1, 2, ..., n$. The explanatory variable X can be continuous or discrete. We consider the univariate logistic regression model

$$\log \frac{P(y = 1|x)}{P(y = 0|x)} = \beta_0 + \beta_1 x. \tag{1}$$

Our model assumptions say that each y_i follows a Bernoulli distribution with probability of success $\pi_i = P(y_i = 1|x_i)$:

$$y_i \sim Ber(\pi_i).$$

Since the sample are assumed to be independent, the likelihood is:

$$L(\beta_0, \beta_1|\mathcal{D}) = \prod_{i=1}^{n}[P(y_i = 1|x_i)]^{y_i}[1 - P(y_i = 1|x_i)]^{1-y_i},$$

where

$$\pi_i = P(y_i = 1|x_i) = logit^{-1}(\beta_0 + \beta_1 x_i) \in (0, 1).$$

We define the *logit* function:

$$logit : (0, 1) \to (-\infty, +\infty), \quad logit(p) = \log \frac{p}{1-p}.$$

Its inverse is:

$$logit^{-1} : (-\infty, +\infty) \to (0, 1), \quad logit^{-1}(x) = \frac{\exp(x)}{1 + \exp(x)}.$$

The log-likelihood is

$$\begin{aligned} l(\beta_0, \beta_1|\mathcal{D}) &= \log L(\beta_0, \beta_1|\mathcal{D}), \\ &= \sum_{i=1}^{n}(y_i \log \pi_i + (1 - y_i) \log[1 - \pi_i]). \end{aligned}$$

Simple calculations show that

$$\begin{aligned}\frac{\partial l(\beta_0,\beta_1|\mathcal{D})}{\partial\beta_0} &= \sum_{i=1}^{n}[y_i = \pi_i],\\ \frac{\partial l(\beta_0,\beta_1|\mathcal{D})}{\partial\beta_1} &= \sum_{i=1}^{n}[y_i x_i - \pi_i x_i].\end{aligned}$$

We assume that the logistic regression coefficients follow independent $N(0,1)$ priors. The joint posterior distribution of β_0 and β_1 is therefore given by

$$P(\beta_0,\beta_1|\mathcal{D}) = \frac{1}{P(\mathcal{D})}\exp(l^*(\beta_0,\beta_1)), \tag{2}$$

where

$$l^*(\beta_0,\beta_1) = -\log(2\pi) - \frac{1}{2}(\beta_0^2+\beta_1^2) + l(\beta_0,\beta_1|\mathcal{D}),$$

and

$$P(\mathcal{D}) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\exp(l^*(\beta_0,\beta_1))d\beta_0 d\beta_1. \tag{3}$$

We call $P(\mathcal{D})$ the marginal likelihood associated with the univariate logistic regression (1). The gradient of $l^*(\beta_0,\beta_1)$ is

$$\nabla l^*(\beta_0,\beta_1) = \begin{pmatrix}\frac{\partial l^*(\beta_0,\beta_1)}{\partial\beta_0}\\ \frac{\partial l^*(\beta_0,\beta_1)}{\partial\beta_1}\end{pmatrix}.$$

The Hessian matrix associated with $l^*(\beta_0,\beta_1)$ is

$$D^2 l^*(\beta_0,\beta_1) = \begin{bmatrix}\frac{\partial^2 l^*(\beta_0,\beta_1)}{\partial\beta_0^2} & \frac{\partial^2 l^*(\beta_0,\beta_1)}{\partial\beta_0\partial\beta_1}\\ \frac{\partial^2 l^*(\beta_0,\beta_1)}{\partial\beta_1\partial\beta_0} & \frac{\partial^2 l^*(\beta_0,\beta_1)}{\partial\beta_1^2}\end{bmatrix}.$$

1.1 The Newton-Raphson Algorithm

We determine the mode of the posterior distribution (2), i.e.

$$(\hat{\beta}_0,\hat{\beta}_1) = \underset{(\beta_0,\beta_1)\in\Re^2}{\arg\max}\, l^*(\beta_0,\beta_1),$$

by employing the Newton-Raphson algorithm. The procedure starts with the initial values $(\beta_0^{(0)},\beta_1^{(0)}) = (0,0)$. At iteration k, we update our current estimate $(\beta_0^{(k-1)},\beta_1^{(k-1)})$ of the mode $(\hat{\beta}_0,\hat{\beta}_1)$ to a new estimate $(\beta_0^{(k)},\beta_1^{(k)})$ as follows:

$$\begin{pmatrix}\beta_0^{(k)}\\ \beta_1^{(k)}\end{pmatrix} = \begin{pmatrix}\beta_0^{(k-1)}\\ \beta_1^{(k-1)}\end{pmatrix} - [D^2 l^*(\beta_0^{(k-1)},\beta_1^{(k-1)})]^{-1}\nabla l^*(\beta_0^{(k-1)},\beta_1^{(k-1)}).$$

The procedure stops when the estimates of the mode do not change after performing a new update, i.e. $|\beta_0^{(k)} - \beta_0^{(k-1)}| < \epsilon$ and $|\beta_1^{(k)} - \beta_1^{(k-1)}| < \epsilon$. Here ϵ is some small positive number, e.g. 0.0001. *Please note that we have already discussed and implemented the $Newton-Raphson$ algorithm for univariate logistic regression.*

1.2 The Laplace Approximation

Since the integral (3) cannot be explicitly calculated, we need to approximate it numerically. We calculate the marginal likelihood $P(\mathcal{D})$ using the Laplace approximation, i.e.

$$\widehat{P(\mathcal{D})} = 2\pi \exp(l^*(\hat{\beta}_0, \hat{\beta}_1))\{\det[-D^2 l^*(\hat{\beta}_0, \hat{\beta}_1)]\}^{-1/2}, \tag{4}$$

where $(\hat{\beta}_0, \hat{\beta}_1)$ is the mode of the posterior distribution (2). We note that you should not actually calculate $\widehat{P(\mathcal{D})}$. Instead, you should calculate the logarithm of the marginal likelihood $\log\widehat{P(\mathcal{D})}$.

1.3 The Metropolis-Hastings Algorithm

Sampling from the posterior distribution (2) can be done using the Metropolis-Hastings algorithm. The procedure starts with the initial values $(\beta_0^{(0)}, \beta_1^{(0)}) = (\hat{\beta}_0, \hat{\beta}_1)$, i.e. we start right at the mode of the distribution (2). We update the current state $(\beta_0^{(k-1)}, \beta_1^{(k-1)})$ of the Markov chain to its next state $(\beta_0^{(k)}, \beta_1^{(k)})$ as follows.
We generate a candidate state $(\tilde{\beta}_0, \tilde{\beta}_1)$ by sampling from the bivariate normal distribution

$$N_2\left(\begin{pmatrix}\beta_0^{(k-1)} \\ \beta_1^{(k-1)}\end{pmatrix}, -[D^2 l^*(\hat{\beta}_0, \hat{\beta}_1)]^{-1}\right) \tag{5}$$

Note that the covariance matrix of the proposal (5) is the negative of the inverse of the Hessian matrix evaluated at the mode of (2). The R function `mvrnorm` might be useful here.
We accept the move to the proposed state, i.e. we set $(\beta_0^{(k)}, \beta_1^{(k)}) = (\tilde{\beta}_0, \tilde{\beta}_1)$ with probability

$$\min\left\{1, \exp\left[l^*(\tilde{\beta}_0, \tilde{\beta}_1) - l^*(\beta_0^{(k-1)}, \beta_1^{(k-1)})\right]\right\}. \tag{6}$$

Otherwise the Markov chain stays at its current state, i.e. we set $(\beta_0^{(k)}, \beta_1^{(k)}) = (\beta_0^{(k-1)}, \beta_1^{(k-1)})$.
We see that the proposal distribution (5) is symmetric, i.e. the probability of proposing $(\tilde{\beta}_0, \tilde{\beta}_1)$ if the chain is currently in $(\beta_0^{(k-1)}, \beta_1^{(k-1)})$ is equal with the probability of proposing $(\beta_0^{(k-1)}, \beta_1^{(k-1)})$ if the chain is currently in $(\tilde{\beta}_0, \tilde{\beta}_1)$. As such, the proposal distribution (5)

cancels when we calculate the acceptance probability (6).

The implementation of an iteration of the Metropolis-Hastings algorithm proceeds as follows. If the proposed state leads to an increase of l^*, i.e.

$$l^*(\tilde{\beta}_0, \tilde{\beta}_1) \geq l^*(\beta_0^{(k-1)}, \beta_1^{(k-1)}),$$

we accept the move to the proposed state and set $(\beta_0^{(k)}, \beta_1^{(k)}) = (\tilde{\beta}_0, \tilde{\beta}_1)$. Otherwise, if the proposed state leads to a decrease in l^*, i.e.

$$l^*(\tilde{\beta}_0, \tilde{\beta}_1) < l^*(\beta_0^{(k-1)}, \beta_1^{(k-1)}),$$

we sample u from a Uniform(0, 1) distribution. If

$$\log(u) \leq l^*(\tilde{\beta}_0, \tilde{\beta}_1) - l^*(\beta_0^{(k-1)}, \beta_1^{(k-1)}),$$

we accept the move to the proposed state and set $(\beta_0^{(k)}, \beta_1^{(k)}) = (\tilde{\beta}_0, \tilde{\beta}_1)$. If

$$\log(u) > l^*(\tilde{\beta}_0, \tilde{\beta}_1) - l^*(\beta_0^{(k-1)}, \beta_1^{(k-1)}),$$

We reject the move and the chain stays st the current state, i.e. we set $(\beta_0^{(k)}, \beta_1^{(k)}) = (\beta_0^{(k-1)}, \beta_1^{(k-1)})$.

2 Homework Problems

2.1 Problem 1

Write a function in R that computes the Laplace approximation (4) of the marginal likelihood (3) associated with a univariate logistic regression.

Answer:

For this question, I build several functions for computing the value about equations, and the functions are below,

```r
l_star <- function(y, x, beta){
  return(-log(2*pi) - 0.5*(beta[1]^2 + beta[2]^2) +
     logisticLoglik(y, x, beta))
}

getGradient_forlstar <- function(y, x, beta){
  gradient = matrix(0,2,1);
  Pi = getPi(x,beta);

  gradient[1,1] = -beta[1] + sum(y-Pi);
  gradient[2,1] = -beta[2] + sum((y-Pi)*x);

  return(gradient);
}

getHessian_forlstar <- function(y,x,beta)
{
  hessian = matrix(0,2,2);
  Pi2 = getPi2(x,beta);
  # put negative sign in the last step
  hessian[1,1] = 1 + sum(Pi2);
  hessian[1,2] = sum(Pi2*x);
  hessian[2,1] = hessian[1,2];
  hessian[2,2] = 1 + sum(Pi2*x^2);

  return(-hessian);
}

getcoefNR_forlstar <- function(response,explanatory,data)
{
  #2x1 matrix of coefficients`
  beta = matrix(0,2,1);  # start of the guess
  y = data[,response];
  x = data[,explanatory];

  #current value of log likelihood
  currentLoglik = logisticLoglik(y,x,beta);

  #infinite loop unless we stop it someplace inside
  while(1)
  {
    newBeta = beta - solve(getHessian_forlstar(y,x,beta))%*%
       getGradient_forlstar(y,x,beta);
    # the function to find the inverse function, same as "inv
       "
    newLoglik = logisticLoglik(y,x,newBeta);

    #at each iteration the log likelihood must increase
    if(newLoglik<currentLoglik)
    {
      cat("CODING ERROR!!\n");
      break;
    }

    beta = newBeta;
    #stop if the log likelihood does not improve by too much
    if(newLoglik-currentLoglik<1e-6)
```

```r
    {
      break;
    }
    currentLoglik = newLoglik;
  }

  return(beta);
}

# version for computing log(PD)
getPD <- function(response, explanatory, data){
  y = data[,response];
  x = data[,explanatory];
  beta_star <- getcoefNR_forlstar(response, explanatory, data
     )
  return(log(2*pi)+ l_star(y, x, beta_star) - 0.5*log(det(-
     getHessian_forlstar(y, x, beta_star))))
}
```

2.2 Problem 2

Write a function in R that estimates the coefficients β_0 and β_1 of the univariate logistic regression (1) as follows:

1. Use the Metropolis-Hastings algorithm to simulate 10000 samples

$$\{(\beta_0^{(k)}, \beta_1^{(k)}) : k = 1, ..., 10000\},$$

   from the posterior distribution (2).

2. The estimates for β_0 and β_1 are given by the sample means:

$$\bar{\beta}_0 = \frac{1}{10000}\sum_{k=1}^{10000} \beta_0^{(k)}, \qquad \bar{\beta}_1 = \frac{1}{10000}\sum_{k=1}^{10000} \beta_1^{(k)}.$$

Answer:

The function I build are below,

```r
MH_AL <- function(response, explanatory, data, iteration){
  y = data[,response]
  x = data[,explanatory]

  # calculate the start beta
  startbeta <- getcoefNR_forlstar(response, explanatory, data
     )
  beta_previous <- startbeta
  beta0_vector <- rep(NA, iteration)
  beta1_vector <- rep(NA, iteration)
  beta0_vector[1] <- beta_previous[1]
  beta1_vector[1] <- beta_previous[2]

  for (i in 2:iteration){
    # sample from the multivariate normal distribution
    sample <- mvrnorm(mu = beta_previous, Sigma = -solve(
       getHessian_forlstar(y, x, beta_previous)))

    # sample a uniform to check if we gonna change the
       current state
    u <- runif(1, 0, 1)

    # check if we stay in the current state
    if (log(u) <= (l_star(y, x, sample) - l_star(y, x, beta_
       previous))){
      beta_previous <- sample
    } # else the beta_previous don't change

    # record the beta for each iterations
    beta0_vector[i] <- beta_previous[1]
    beta1_vector[i] <- beta_previous[2]
  }

  return(list(beta0 = beta0_vector, beta1 = beta1_vector))
}
```

then we can calculate the sample mean for β_0 and β_1.

2.3 Problem 3

The dataset you need to use is contained in the file ”534binarydata.txt”. This file has 148 rows (samples) and 61 columns (variables). The first 60 columns are associated with 60 explanatory variables X, while column 61 (the last column) corresponds with the response binary variable Y.

You need to write a parallel program using the R package snow that computes *in parallel* the Laplace approximation (4) and the estimates of β_0 and β_1 associated with each univariate logistic regression of the response binary variable Y on each of the 60 explanatory variables. You can choose to also obtain the MLEs of β_0 and β_1 to give yourself a way to check your estimates from Problem 2.

Please start from the code below. Your task is to write the function bayesLogistic ad well as all the other functions that are called in bayesLogistic.

```
bayesLogistic = function(apredictor,response,data,
   NumberOfIterations)
{
  ....
}
#PARALLEL VERSION
#datafile = the name of the file with the data
#NumberOfIterations = number of iterations of the Metropolis
   Hastings algorithm
#clusterSize = number of separate processes; each process
   performs one or more
#univariate regressions
main <- function(datafile,NumberOfIterations,clusterSize)
{
  #read the data
  data = read.table(datafile,header=FALSE);
  #the sample size is 148 (number of rows)
  #the explanatory variables are the first 60 columns for
     '534binarydata.txt'
  #the last column is the binary response
  response = ncol(data);
  lastPredictor = ncol(data)-1;
  #initialize a cluster for parallel computing
  cluster <- makeCluster(clusterSize, type = "SOCK")

  #run the MC3 algorithm from several times
  results = clusterApply(cluster, 1:lastPredictor,
     bayesLogistic,
  response,data,NumberOfIterations);
  #print out the results
  for(i in 1:lastPredictor)
  {
    cat('Regression of Y on explanatory variable ',results[[i
       ]]$apredictor,
    ' has log marginal likelihood ',results[[i]]$logmarglik,
    ' with beta0 = ',results[[i]]$beta0bayes,' (',results[[i
       ]]$beta0mle,')',
    ' and beta1 = ',results[[i]]$beta1bayes,' (',results[[i]]
       $beta1mle,')',
    '\n');
  }
  #destroy the cluster
  stopCluster(cluster);
}
#NOTE: YOU NEED THE PACKAGE 'SNOW' FOR PARALLEL COMPUTING
require(snow);
#this is where the program starts
```

```
main('534binarydata.txt',10000,10);
```

Answer:

My bayesLogistic function will be like below,

```
bayesLogistic = function(apredictor,response,data,
   NumberOfIterations)
{
  library(snow)
  library(MASS)
  # import the function we build above
  source('function_for_hw4.R')

  # calculate laplace approximation
  laplace <- getPD(response, apredictor, data)
  beta_bayes <- getcoefNR_forlstar(response, apredictor, data
     )

  # the approximation computing by sample mean from
     Metropolis Hasting Algorithm
  beta_mle <- MH_AL(response, apredictor, data,
     NumberOfIterations)

  return(list(apredictor = apredictor, logmarglik = laplace,
  beta0bayes = beta_bayes[1], beta1bayes = beta_bayes[2],
  beta0mle = sum(beta_mle$beta0)/10000, beta1mle = sum(beta_
     mle$beta1)/10000))
}
```

The main function is not revised, so I skip here. I put my result in the "result.txt" file. I tried two times and found that because Metropolis-Hasting Algorithm has some randomness inside the function, so the result will not be the same but very close to different try. The result from estimation from problem 2 has a small difference from the value we compute from problem 1 function, but they are close to each other.