Lecture 21 R scripts

Example: Comparing techniques for fitting logistic regression.

In this example, we will return to the dataset considered in previous lecture. As a reminder this dataset was the result of a study investigating what the optimal dosage of a new medication for influenza should be. A group of influenza patients were recruited, and administered various doses of the experimental medication. One week after administration of medication, the researcher examined the patients to see if symptoms had improved. You were asked to investigate if there is a linear relationship between the log-odds of symptom improvement and medication dose.

Consider a logistic regression,

$$\begin{aligned} y_i &\sim \text{Bin}(n_i, p_i) \\ \log(p_i/(1-p_i)) &= \mathbf{x}_i'\boldsymbol{\beta}. \end{aligned}$$

Assume a flat prior for $\boldsymbol{\beta}$ so that your results are more comparable to a standard logistic GLM.

	Dosage: x						
y_i	0	1	2	3	4	5	6
No improvement: 0	8	5	6	6	3	5	1
Improvement: 1	2	5	4	4	7	5	9

We will compared estimation based on

- Normal approximation based on posterior modes (standard glm)
- Fully Bayesian posterior inference (fitted using the Metropolis-Hasting algorithm)
- Approximate Bayesian inference using the expectation-propagation algorithm.

Code for implementing Expectation-propagation.

```
#Coding for implementing Expectation-propagation for logistic regression.

#Arguments.
#response, response vector
#n: vector of trial sizes
#iter: number of rounds to consider
#epsilon: convergence criteria.

EP.logit<-function(response,n,X,iter,epsilon){
N<-length(response) #size of dataset.
p<-dim(X)[2]
```

```
Sigmainvmu <-matrix(0,p,N) #natural parameter \Sigma^{-1}mu
Sigmainv  <-rep(list(diag(p)),N)  #natural parameter \Sigma^{-1}, stored as list

#Previous parameters of g.
Sigmainvmu0<-rowSums(Sigmainvmu)
Sigmainv0   <-Reduce("+",Sigmainv)

#g_0(\bm \beta) need not be updated and is assumed flat.

#function for tilted distribution
tilt.dist      <- function(x){
gnoti          <-dnorm(x,mean=Mnoti,sd=sqrt(Vnoti))
pr              <-(1+exp(-x))^(-1)
like           <-dbinom(response[i],n[i],pr)
result <-gnoti*like
return(result)
}

#loop for updating g_i(\bm \beta) i= 1, ..., n.
for(j in 1:iter){
for(i in 1:N){
Sigmainvmunoti <-rowSums(Sigmainvmu)  - Sigmainvmu[,i] #Natural parameter \Sigma_{-i}^{-1}\mu_{-i}
Sigmainvnoti   <-Reduce("+",Sigmainv) - Sigmainv[[i]]  #Natural parameter \Sigma_{-i}^{-1}\mu_{-i}
Sigmanoti      <-solve(Sigmainvnoti)                   #parameter \Sigma_{-i}
munoti         <-Sigmanoti%*%Sigmainvmunoti            #parameter \mu_{-i}
Mnoti          <-t(X[i,])%*%munoti                     #M_{-i}
Vnoti          <-t(X[i,])%*%Sigmanoti%*%X[i,]          #V_{-i}

#Moment matching.
E0<-integrate(f=function(x){tilt.dist(x)}, lower=Mnoti-10*sqrt(Vnoti), upper=Mnoti+10*sqrt(Vnoti))
E1<-integrate(f=function(x){x*tilt.dist(x)}, lower=Mnoti-10*sqrt(Vnoti), upper=Mnoti+10*sqrt(Vnoti))
E2<-integrate(f=function(x){x^2*tilt.dist(x)},lower=Mnoti-10*sqrt(Vnoti), upper=Mnoti+10*sqrt(Vnoti))
M <- E1$value/E0$value
V <- E2$value/E0$value - M^2

#Update g_i
MiViinv <- M/V - Mnoti/Vnoti
Viinv   <- 1/V - 1/Vnoti

#transform back to beta scale.
Sigmainvmu[,i] <-X[i,]%*%MiViinv  #natural parameter \Sigma^{-1}mu
Sigmainv[[i]]  <-X[i,]%*%Viinv%*%t(X[i,])
}
#Note by the way the previous lines of code have been written, step six has been implicitly.

#Checking whether to stop iterations.
currentSinvmu <-rowSums(Sigmainvmu)
currentSinv   <-Reduce("+",Sigmainv)

  diff1  <- sqrt((currentSinvmu-Sigmainvmu0)^2)/(abs(currentSinvmu)+0.01)
  diff2  <- sqrt((currentSinv-Sigmainv0)^2)/(abs(currentSinv)+0.01)
  diff.all<-c(diff1,diff2)
  if(max(diff.all) < epsilon) break else Sigmainvmu0 <- currentSinvmu; Sigmainv0 <- currentSinv
```

```
}

#Final mean and variance-covariance matrix of g(\beta)
Sigma <-solve(currentSinv)
mu    <-Sigma%*%currentSinvmu


#Storing and returning results.
param<-list(mu,Sigma,j)
names(param)<-c('betahat','Sigma','iter_break')
return(param)
}
```

Code for implementing the Metropolis algorithm for logistic regression

```
#Run Metropolis sampling.
#Inputs:
#y: vector of responses
#n: vector (or scalar) of trial sizes.
#X: predictor matrix including intercept.
#c: rescaling for variance-covariance matrix, scalar J(theta*|theta(t-1)) = N(theta(t-1), c^2*Sigma)
#Sigma is variance covariance matrix for parameters in J()
#iter: number of iterations
#burnin: number of initial iterations to throw out.
Metropolis.fn<-function(y,n,X,c,Sigma,iter,burnin){
  p <-dim(X)[2]   #number of parameters
  library(mvtnorm)
  theta0<-rnorm(p) #initial values.
  theta.sim<-matrix(0,iter,p) #matrix to store iterations
  theta.sim[1,]<-theta0
  for(i in 1:(iter-1)){
    theta.cand <-rmvnorm(1,mean=theta.sim[i,],sigma=c^2*Sigma) #draw candidate (jointly)
    theta.cand <-as.numeric(theta.cand)
    xbc        <-X%*%theta.cand
    p.c        <-(1+exp(-xbc))^(-1)   #Calculating probability of success for candidates.
    xb         <-X%*%theta.sim[i,]
    p.b        <-(1+exp(-xb))^(-1)     #Calculating probability of success for theta(t-1).
    #difference of log joint distributions.
    r<-sum( dbinom(y,size=n,prob=p.c,log=TRUE)-dbinom(y,size=n,prob=p.b,log=TRUE) )
    #Draw an indicator whether to accept/reject candidate
    ind<-rbinom(1,1,exp( min(c(r,0)) ) )
    theta.sim[i+1,]<- ind*theta.cand + (1-ind)*theta.sim[i,]
  }

  #Removing the iterations in burnin phase
  results<-theta.sim[-c(1:burnin),]
  names(results)<-c('beta0','beta1') #column names
  return(results)
}
```

Fitting logistic regression using outlined techniques

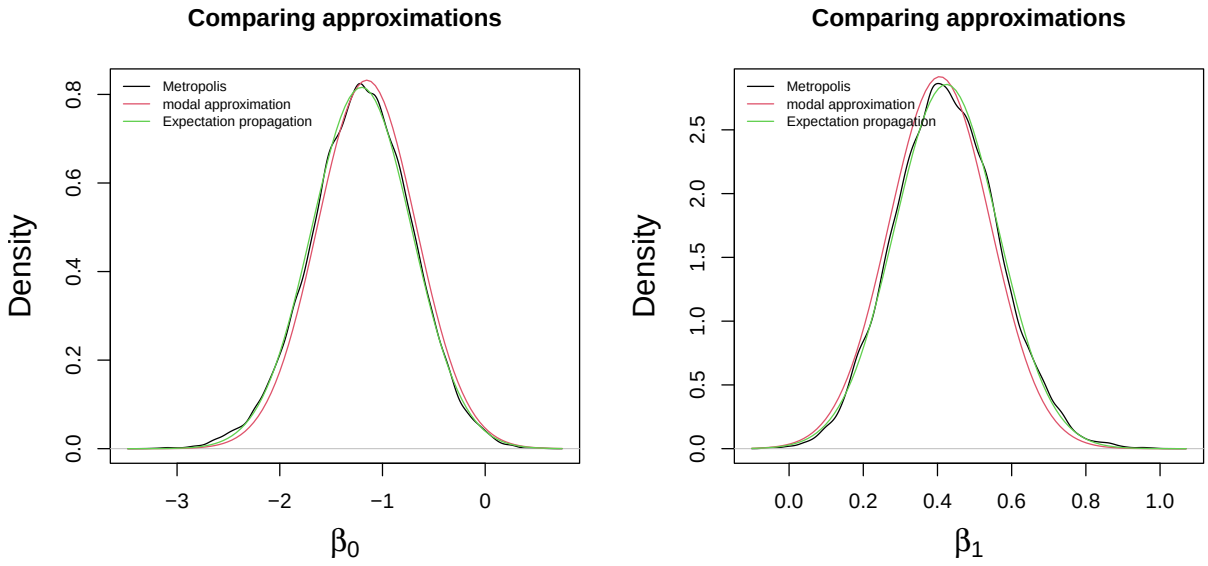
```
#Entering data
x <-0:6
y<-c(2,5,4,4,5,7,9)
N<-10

n<-length(y)
X<-cbind(rep(1,n),x)
#Normal approximation at posterior mode
mod<-glm(cbind(y,N-y)~x,family=binomial('logit'))
#Expectation propagation
mresult<-EP.logit(response=y,n=rep(N,n),X=X,iter=100,epsilon=1e-6)
#Metropolis algorithm
attempt1<-Metropolis.fn(y=y,n=N,X=X,c=2.4/sqrt(2),Sigma=vcov(mod),iter=100000,burnin=50000)
```

```
## Warning: package 'mvtnorm' was built under R version 4.3.1
```

Plots of (estimated) posterior distribution for parameters

```
par(mfrow=c(1,2))
for(i in 1:2){
plot(density(attempt1[,i]),xlab=bquote(beta[.(i-1)]),main='Comparing approximations',cex.lab=1.5)
curve(dnorm(x,mean=mod$coef[i],sd=sqrt(vcov(mod)[i,i])),add=TRUE,col=2 )
curve(dnorm(x,mean=mresult$betahat[i],sd=sqrt(mresult$Sigma[i,i])),add=TRUE,col=3 )
legend('topleft',legend=c('Metropolis','modal approximation','Expectation propagation'),
       col=1:3,lty=1,bty='n',cex=0.7)
}
```



```r
#Credible intervals, 95 %
confint(mod) #Posterior mode
```

```
## Waiting for profiling to be done...
```

```
##                 2.5 %     97.5 %
## (Intercept) -2.150726 -0.2514272
## x            0.150474  0.6916607
```

```r
qnorm(c(0.025,0.975),mean=mresult$betahat[1],sd=sqrt(mresult$Sigma[1,1]))
```

```
## [1] -2.1613758 -0.2453043
```

```r
qnorm(c(0.025,0.975),mean=mresult$betahat[2],sd=sqrt(mresult$Sigma[2,2]))
```

```
## [1] 0.1503045 0.6977256
```

```r
apply(attempt1,2,FUN = function(x) quantile(x,c(0.025,0.975)))
```

```
##             [,1]      [,2]
## 2.5%  -2.2033371 0.1617262
## 97.5% -0.2752511 0.7041893
```