MCMC for Probit Regression

Brooke Coneeny

# Background Information

As we know, the objective of Bayesian data analysis is to determine the posterior distribution given the prior distribution and likelihood distribution for the data.

Markov Chain Monte Carlo methods allow us to estimate the shape of a posterior distribution in case we cannot compute it directly. In other words, MCMC methods are used to approximate the posterior distribution for a parameter of interest by random sampling in a probabilistic space

Monte Carlo simulations are a way of estimating a fixed parameter by repeatedly generating random numbers.

Markov Chains are sequences of events that are probabilistically related to one another. Each event comes from a set of outcomes, and each outcome determines which outcome occurs next, according to a fixed set of probabilities. An important feature of Markov Chains is that they are memoryless, meaning that no new information comes from knowing the history of past events.

Using the MCMC method, we form a posterior distribution from sampling, and then compute statistics from the sample.

# Gibbs Sampling

Gibbs Sampling is used when the joint distribution is not known explicitly or is difficult to sample from directly, but the conditional distribution of each variable is known and easier to sample from.

For this method, the Markov Chain is constructed by sampling from the conditional distribution for each parameter, conditional on the current values of the other parameters. Once we have iterated over all parameters, we completed one cycle of the Gibbs sampler.

# Gibbs Sampling for Probit Distribution

We can use this MCMC method to carry out Bayesian inference for the Probit regression model. Instead of plugging in the MLE for the coefficient , we assume prior distribution on and simulate from its posterior distribution.

Recall that the Probit model may be expressed in terms of a latent variable

, where

This is a Normal linear regression model for the ’s with given

The form of the probit model probability function is as follows:

Note that is also follows the latent variable representation.

In this example, our conditional distributions are:

Note that these are both draws from Normals which are forced to be the correct sign, determined by the ’s for if then , otherwise for

# Gibbs Sampling Demonstration

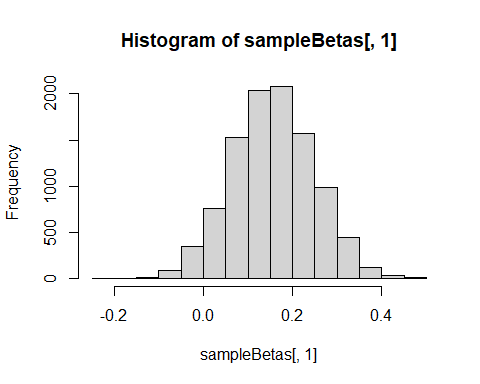
We are going to use the NFL home wins and losses data, using halftime margin as an explanatory variable. This results in two parameters, and .

The code for the algorithm is as follows:

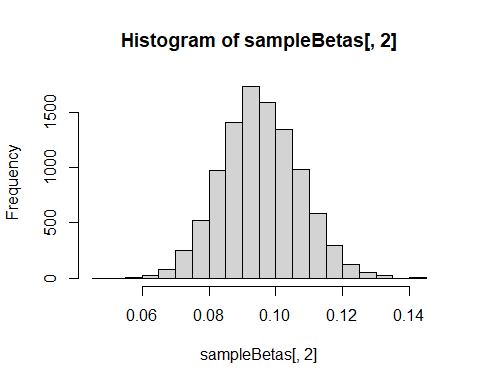
# initialize parameters   
x = modeldata$Halftime.H.R  
X = cbind(1,x) # covariate matrix   
y = modeldata$response1  
theta = 0\*y  
V = solve(t(X)%\*%{X})  
rtV = t(chol(V))   
nsim = 10000  
  
# when y is zero want thetas negative  
theta[y == 0] = qnorm(runif(sum(y==0), 0, 0.5))  
# when y is 1 want thetas positives  
theta[y == 1] = qnorm(runif(sum(y==1), 0.5, 1))  
  
# will store the beta values   
sampleBetas = matrix(0,ncol = 2, nrow = nsim)  
  
# for nsim iterations   
for (i in 1:nsim) {  
 # sample each parameter from its conditional dist  
 # for beta   
 Betahat = V%\*%t(X)%\*%theta  
 Beta = Betahat+rtV%\*%rnorm(2) # taking V ^1/2 times Z is distributed as N(0,V)  
 # for theta   
 mu = X%\*%Beta  
 z = 0\*mu  
 # apply inverse CDF to a random unif resulting in a random value, restricting   
 # the uniform range restricts range of the resulting variable   
 z[y==0] = qnorm(runif(sum(y==0), 0,pnorm(-mu[y==0])))  
 z[y==1] = qnorm(runif(sum(y==1),pnorm(-mu[y==1]),1))  
 theta = mu + z  
 # add sample betas to matrix  
 sampleBetas[i,] = Beta # putting the values of betas in the i'th row of the matrix   
}

As you can see with these two plots, we get distributions for each of the Betas

# histogram of output  
hist(sampleBetas[,1])



hist(sampleBetas[,2])



## Posterior mean VS MLE

The posterior mean estimate can now be found by taking the average of the sample ’s

# getting the means for both Beta 0 and Beta 1  
apply(sampleBetas, 2, mean)

## [1] 0.1559026 0.0954726

The MLE values for the ’s can be found in the output table after running a general linear model.

probit\_model <- glm(response1 ~ Halftime.H.R, data = modeldata, family = "binomial"(link = "probit"))  
summary(probit\_model)

##   
## Call:  
## glm(formula = response1 ~ Halftime.H.R, family = binomial(link = "probit"),   
## data = modeldata)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -2.8699 -0.8548 0.1924 0.8402 2.2861   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) 0.15407 0.09089 1.695 0.0901 .   
## Halftime.H.R 0.09445 0.01183 7.987 1.39e-15 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 351.37 on 255 degrees of freedom  
## Residual deviance: 252.35 on 254 degrees of freedom  
## AIC: 256.35  
##   
## Number of Fisher Scoring iterations: 5

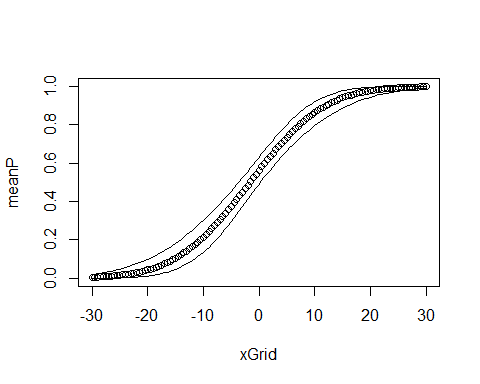
Comparing these values to the posterior mean estimates we can see they are almost the same estimates.

# Interval Estimates

One advantage of Bayesian probit regression is that it is easy to generate interval estimates for the success probabilities. Each simulated implies a probability, and you can find the middle 95% of the simulated probabilities for each to represent 95% credible intervals.

We can generate a graph of the probability curve vs.  with 95% interval estimates drawn in. As we can see, the interval bounds differs based on the home team’s lead at halftime. The interval is wider when the game is still close and narrower when the lead is large.

xGrid = seq(-30,30,0.5)  
probMap = matrix(0, ncol = length(xGrid), nrow = nsim)  
  
for (i in 1:nsim){  
 for (j in 1:length(xGrid)){  
 # regression line with the estimates   
 mu = sampleBetas[i,1] + xGrid[j]\*sampleBetas[i,2]  
 probMap[i,j] = pnorm(mu)   
 }  
}  
  
# get the lower 5 percent of samples  
lower025 = apply(probMap, 2, quantile, 0.025)  
# get the upper 5 percent of values   
upper975 = apply(probMap, 2, quantile, 0.975)  
  
  
meanP = apply(probMap, 2, mean)  
  
plot(xGrid, meanP, ylim = c(0,1))  
lines(xGrid, lower025)  
lines(xGrid, upper975)



If we wanted to report a 95% posterior interval for the probability that a home team will win if they have a 7 point lead at halftime, we could use the plot we created above. The red line denotes when the halftime margin is 7. The interval represents the probability the home team is going to win given their lead at halftime is 7 points, which is between 0.722 and 0.859. This means the probability of the home team winning, if they are leading at half time by 7 points, is between 0.722 and 0.859 (72.2%-85.9%)

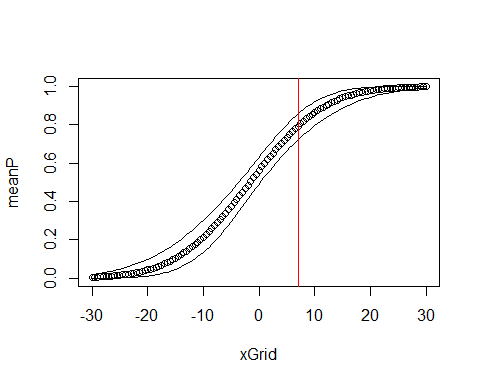
lower\_bound = lower025[75]  
upper\_bound = upper975[75]  
lower\_bound

## [1] 0.7221632

upper\_bound

## [1] 0.8585982

plot(xGrid, meanP, ylim = c(0,1))  
lines(xGrid, lower025)  
lines(xGrid, upper975)  
abline(v = 7, col = "red")



# Improprt Prior Distributions & Proper Posterior Distributions

The improper prior distribution for will not give a proper posterior distribution for data that exhibit perfect separation. If there is perfect separation, y is 0 on one side of a threshold and 1 on the other, creating a step function.

The maximum possibly value for any likelihood function with discrete data is 1, which is achieved only if you can make the probability of the data be 1. It is possible to do this but only in a limit. For our data, the probability increases with x. The curve of the probit probability curve is shaped similar to an S.

However, as the slope parameter increases it becomes close to a step shape. This happens because the intercept will decrease to compensate and keep the curve near 0 on one side of the threshold (where ) and near 1 on the other side of the threshold (where ). The closer this function gets to the step, the fitted probability (the value of the likelihood function) of the ’s get closer to 1. This means that the likelihood function continues to increase towards 1 as and .

Given a constant prior, the posterior density is proportional to the joint likelihood function. If this likelihood function is increasing in its limitis, it cannot be normalized to a joint density function. Therefore, the constant prior is convenient, but when using an improper prior, we have to verify that the posterior will be proper.

A prior distribution (or any other proper distribution) for will yield a proper posterior distribution with these data. This is because any proper prior will combine with a likelihood function and give us a proper posterior distribution. Therefore, it can be known that proper prior distributions lead to proper joint distributions and therefore proper posterior distributions.

### Meaningful Draws

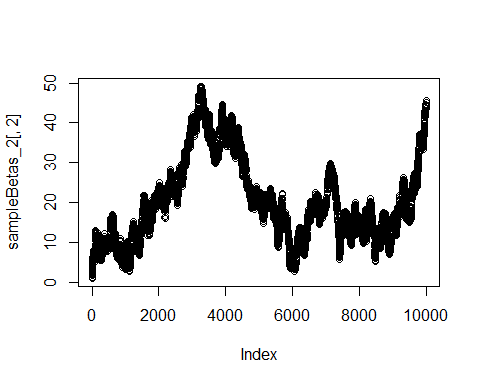
The Gibbs sampler from part c will generate draws, but not meaningful draws (which is dangerous). These draws are not meaningful because the joint posterior is not defined. Using improper prior distributions as done above is therefore risky.

### Example

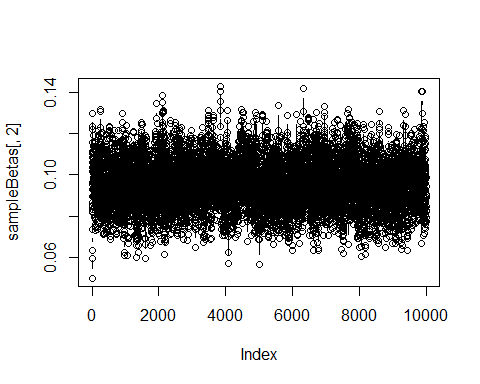
Now we are going to quickly run a Gibbs sampler and report posterior mean and variance estimates for using and . This is an example of perfectly separated data.

The code for the algorithm is as follows:

# initialize parameters   
x\_2 = c(-2,-1,0,1,2)  
X\_2 = cbind(1,x\_2) # m by 2 covariate matrix   
y\_2 = c(0,0,0,1,1)  
theta\_2 = 0\*y\_2  
V\_2 = solve(t(X\_2)%\*%{X\_2})  
rtV\_2 = t(chol(V\_2))   
nsim\_2 = 10000  
  
# when y is zero want thetas negative  
theta\_2[y\_2 == 0] = qnorm(runif(sum(y\_2==0), 0, 0.5))  
# when y is 1 want thetas positives  
theta\_2[y\_2 == 1] = qnorm(runif(sum(y\_2==1), 0.5, 1))  
  
# will store the beta values   
sampleBetas\_2 = matrix(0,ncol = 2, nrow = nsim\_2)  
  
# for 100 iterations   
for (i in 1:nsim\_2) {  
 # sample each parameter from its conditional dist  
 # for beta   
 Betahat\_2 = V\_2%\*%t(X\_2)%\*%theta\_2  
 Beta\_2 = Betahat\_2+rtV\_2%\*%rnorm(2) # taking V ^1/2 times Z is distributed as N(0,V)  
 # for theta   
 mu\_2 = X\_2%\*%Beta\_2  
 z\_2 = 0\*mu\_2  
 z\_2[y\_2==0] = qnorm(runif(sum(y\_2==0), 0,pnorm(-mu\_2[y\_2==0])))  
 z\_2[y\_2==1] = qnorm(runif(sum(y\_2==1),pnorm(-mu\_2[y\_2==1]),1))  
 theta\_2 = mu\_2 + z\_2  
 # add sample betas to matrix  
 sampleBetas\_2[i,] = Beta\_2 # putting the values of betas in the i'th row of the matrix   
}  
  
# beta 1 values plotted against iteration number for both separated and NFL  
plot(sampleBetas\_2[,2], type = 'b')



plot(sampleBetas[,2], type = 'b')



Things to note about this example:

* The results, due to the perfect separation in the data, are not summarizing a proper distribution.
* The draws were not meaningful because they are not coming from any fixed distribution.
* In the plot of vs iteration, we can see the values are continuing to get larger and are not settling into a stable distribution for the separated data set. This is different than the sloped for the NFL data, which vary steadily around 0.09

# Sources

<https://towardsdatascience.com/a-zero-math-introduction-to-markov-chain-monte-carlo-methods-dcba889e0c50>

<https://people.duke.edu/~ccc14/sta-663/mcmc.html>

<https://towardsdatascience.com/gibbs-sampling-8e4844560ae5>