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.

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 of a paramter of interest by random sampling in a probablistic 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 everything that no new information comes from knowing the history of events

Using the MCMC method, we draw samples from the posterior distribution, and then compute statistics on the samples drawn. We then compare the values of parameters by computing how likely each value is to explain the data, given our prior beliefs. If a randomly generated parameter value is better than the last one, it is added to the chain of parameter values with a certain probability determined by how much better it is.

# 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.

Suppose we have a vector of parameters and we want to estimate the joint posterior distribution .

Suppose we can find and draw random samples from all of the conditional distributions:

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 variables. 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 $\theta\_i = X^{`}\_{i} \beta + Z\_i$, where

We observe if and otherwise, for . Then . The non-informative prior results in a posterior distribution that is proportional to the likelihood function.

In this example, our conditional distributions are:

# 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$Halftime.H.R  
X = cbind(1,x) # m by 2 covariate matrix   
y = modeldata$response1  
theta = 0\*y  
V = solve(t(X)%\*%{X})  
rtV = t(chol(V))   
  
# 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 = 100)  
  
# for 100 iterations   
for (i in 1:100) {  
 # 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  
 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   
}  
  
sampleBetas

## [,1] [,2]  
## [1,] 0.153745111 0.03462860  
## [2,] 0.110831843 0.06267132  
## [3,] 0.064176637 0.07509300  
## [4,] 0.161636586 0.07751821  
## [5,] 0.152921261 0.07682558  
## [6,] 0.368854064 0.07655658  
## [7,] 0.207943169 0.10214817  
## [8,] 0.229336204 0.11238431  
## [9,] 0.259353696 0.11001738  
## [10,] 0.208934514 0.10238137  
## [11,] 0.235076038 0.09400114  
## [12,] 0.252679073 0.07861080  
## [13,] 0.049210238 0.08120212  
## [14,] -0.093699677 0.08120763  
## [15,] 0.077964735 0.07411368  
## [16,] 0.060549348 0.08306523  
## [17,] 0.088361936 0.08448668  
## [18,] 0.083955232 0.09835956  
## [19,] 0.107806231 0.08317686  
## [20,] 0.032645428 0.09196582  
## [21,] 0.098131810 0.09307198  
## [22,] 0.227319672 0.09103652  
## [23,] 0.126433766 0.09334606  
## [24,] 0.202070857 0.09251262  
## [25,] 0.164761494 0.09111699  
## [26,] 0.084138400 0.08675612  
## [27,] 0.168782432 0.09780432  
## [28,] 0.119829317 0.09825334  
## [29,] 0.175911347 0.10933212  
## [30,] 0.048332558 0.10131786  
## [31,] 0.072911814 0.09577894  
## [32,] 0.032323142 0.09247084  
## [33,] 0.074436269 0.09355961  
## [34,] -0.001386073 0.08752161  
## [35,] 0.073794355 0.07980696  
## [36,] 0.122607614 0.07835452  
## [37,] 0.099583962 0.07477660  
## [38,] 0.042702687 0.09922446  
## [39,] 0.113599651 0.09532341  
## [40,] 0.254558525 0.10265737  
## [41,] 0.182192937 0.10667107  
## [42,] -0.008454004 0.10863053  
## [43,] 0.251046496 0.10569256  
## [44,] 0.126753776 0.10644260  
## [45,] 0.186507357 0.09768924  
## [46,] 0.215804314 0.09656963  
## [47,] 0.159072921 0.10117573  
## [48,] 0.163487518 0.10177745  
## [49,] 0.152120402 0.09973018  
## [50,] 0.213141022 0.09660305  
## [51,] 0.122622804 0.10156019  
## [52,] 0.188257048 0.09550694  
## [53,] 0.201004773 0.09063077  
## [54,] -0.081700797 0.08447649  
## [55,] 0.157842089 0.08899607  
## [56,] 0.096951865 0.09130313  
## [57,] 0.120215482 0.09342026  
## [58,] 0.106202740 0.09706669  
## [59,] 0.168719517 0.09692910  
## [60,] 0.186481984 0.10154730  
## [61,] 0.242606691 0.10489033  
## [62,] 0.187298990 0.10614607  
## [63,] 0.195294222 0.10284751  
## [64,] 0.119569487 0.10688583  
## [65,] -0.016515658 0.11020234  
## [66,] 0.117493275 0.11145239  
## [67,] 0.132362714 0.11370568  
## [68,] 0.248152278 0.11490192  
## [69,] 0.224086062 0.10416482  
## [70,] 0.176234607 0.09931985  
## [71,] 0.181398013 0.09264422  
## [72,] 0.172742632 0.09695714  
## [73,] 0.175353479 0.10052499  
## [74,] 0.146823419 0.09903437  
## [75,] 0.028603189 0.10380394  
## [76,] 0.053976558 0.09147684  
## [77,] 0.144914457 0.09174580  
## [78,] 0.090478507 0.09694762  
## [79,] 0.208609817 0.09456148  
## [80,] 0.059512014 0.09127714  
## [81,] 0.158140903 0.09797235  
## [82,] 0.152267188 0.09068163  
## [83,] 0.092589441 0.09015140  
## [84,] 0.101812444 0.08953632  
## [85,] 0.219292192 0.08780029  
## [86,] 0.178387742 0.08638728  
## [87,] 0.123722511 0.09875981  
## [88,] -0.012035540 0.09866045  
## [89,] 0.050604849 0.11472996  
## [90,] 0.048183412 0.10974566  
## [91,] 0.071123435 0.11002668  
## [92,] 0.150336882 0.11335472  
## [93,] 0.225004276 0.11925471  
## [94,] 0.094984657 0.11441063  
## [95,] 0.064774017 0.10837972  
## [96,] 0.215203845 0.10474482  
## [97,] 0.268024510 0.10332231  
## [98,] 0.246699986 0.10514157  
## [99,] 0.150834315 0.10835407  
## [100,] 0.273623775 0.10617009

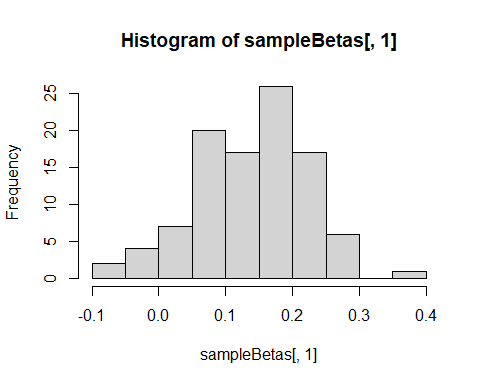
Not that we have out sample of Betas we can see positive correlation between successive values:

cor(sampleBetas[-1,2], sampleBetas[-100,2])

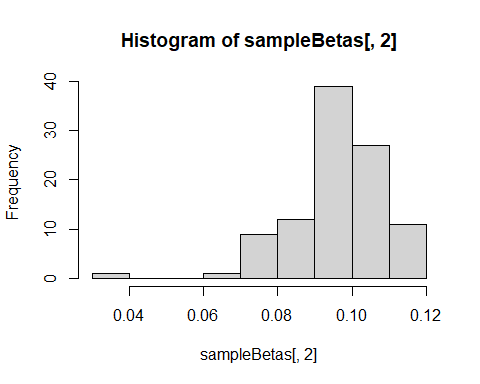
## [1] 0.8119158

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

# historgram 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] 0.1378397 0.0958793

The MLE values for the ’s can be found in the output table after running a general linear model. Comparing these values to the posterior mean estimates we can see they are similar estimates.

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

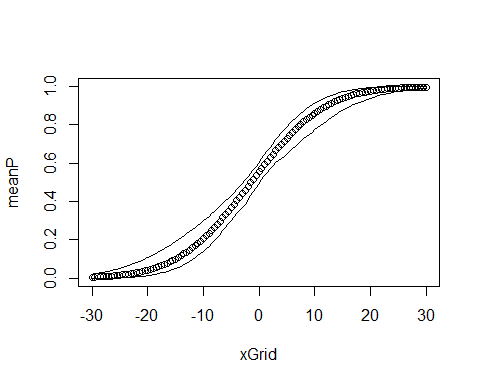
##   
## Call:  
## glm(formula = response1 ~ Halftime.H.R, family = "binomial",   
## data = modeldata)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -2.7455 -0.8333 0.2253 0.8129 2.2836   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) 0.27694 0.15563 1.779 0.0752 .   
## Halftime.H.R 0.16517 0.02244 7.359 1.85e-13 \*\*\*  
## ---  
## 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: 251.48 on 254 degrees of freedom  
## AIC: 255.48  
##   
## Number of Fisher Scoring iterations: 5

# 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.

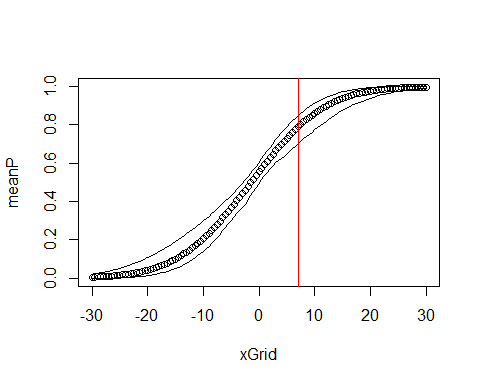
We can generate a graph of the probability curve vs.  with 95% interval estimates drawn in. As we can see, the interval bounds differs based on the lead becoming wider as the and narrower at different parts.

xGrid = seq(-30,30,0.5)  
probMap = matrix(0, ncol = length(xGrid), nrow = 100)  
  
for (i in 1:100){  
 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 lead at halftime, we could use the plot we created above. The red line denotes when the halftime margin is 7. The value at this point represents the probability the home team is going to win given their lead at halftime is 7 points.

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 likelihood for the ’s heads towards 1 as the betas go towards infinity. Because the function never reaches exactly 1, there is not finite area underneath the curve, meaning there is no guarantee the likelihood can be normalized.

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. 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 .

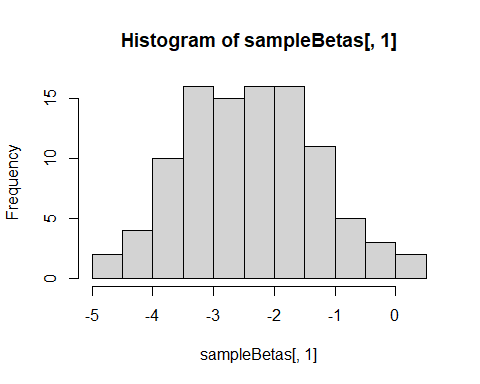
* run the same algorithm with the new x and y should show chaotic behavior with histogram and plot betas against iteration number

The code for the algorithm is as follows:

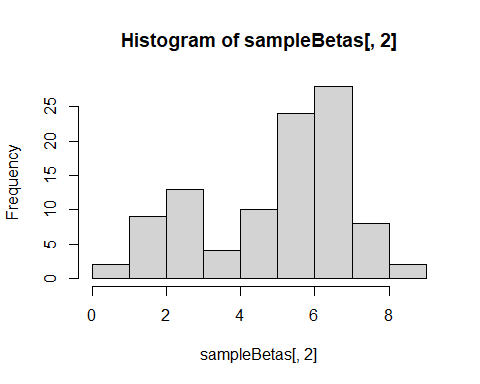
# initialize parameters   
x = c(-2,-1,0,1,2)  
X = cbind(1,x) # m by 2 covariate matrix   
y = c(0,0,0,1,1)  
theta = 0\*y  
V = solve(t(X)%\*%{X})  
rtV = t(chol(V))   
  
# when y is zero want thetas negative  
theta[y == 0] = qnorm(runif(sum(y==0), 0, 0.5))  
# when y is 1 want thetas positives  
theta[y == 1] = qnorm(runif(sum(y==1), 0.5, 1))  
  
# will store the beta values   
sampleBetas = matrix(0,ncol = 2, nrow = 100)  
  
# for 100 iterations   
for (i in 1:100) {  
 # 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  
 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   
}  
  
sampleBetas

## [,1] [,2]  
## [1,] 0.1113604 1.1785758  
## [2,] -0.2628998 1.7040648  
## [3,] -1.0159382 0.6620270  
## [4,] -1.2028519 0.8695037  
## [5,] -1.9359856 1.1500185  
## [6,] -1.2200320 1.2858280  
## [7,] -2.1057016 1.8741723  
## [8,] -1.6069324 2.0000231  
## [9,] -2.2038491 2.0041002  
## [10,] -1.5812219 2.1551781  
## [11,] -1.8352923 1.9943732  
## [12,] -1.2647520 1.6412479  
## [13,] -1.8032213 1.2269293  
## [14,] -1.5856955 1.7307945  
## [15,] -1.1842152 2.0844363  
## [16,] -2.8785755 2.2780370  
## [17,] -2.6668848 2.3953007  
## [18,] -2.1715111 2.3654950  
## [19,] -2.7860521 2.6197679  
## [20,] -2.7380586 2.2728093  
## [21,] -2.2273095 2.5840781  
## [22,] -2.4674825 2.3550967  
## [23,] -3.4449053 2.8055380  
## [24,] -3.2455130 2.6889901  
## [25,] -3.4361733 3.8651515  
## [26,] -2.8123998 4.2558111  
## [27,] -3.0461056 4.1328239  
## [28,] -3.5505037 4.1067091  
## [29,] -3.5463253 3.9445100  
## [30,] -3.9331318 4.7050070  
## [31,] -3.0980904 4.5504436  
## [32,] -2.0787602 4.5686970  
## [33,] -1.3916342 4.7667533  
## [34,] -2.1513870 5.0336983  
## [35,] -3.0691903 5.6534161  
## [36,] -3.9098455 5.4970036  
## [37,] -4.6964089 5.2775367  
## [38,] -4.4850342 5.7362075  
## [39,] -3.9419254 5.5646702  
## [40,] -4.2027226 5.8560211  
## [41,] -4.5993721 6.2427087  
## [42,] -3.2263863 5.5988963  
## [43,] -2.9907602 5.4119605  
## [44,] -3.1117057 5.7088758  
## [45,] -3.0422569 5.5631350  
## [46,] -2.8760198 5.5115713  
## [47,] -3.1947783 5.2779555  
## [48,] -2.2284935 6.0957667  
## [49,] -2.2551008 6.4077056  
## [50,] -2.9434383 6.3122205  
## [51,] -2.7894057 6.3482976  
## [52,] -2.3498745 6.3206373  
## [53,] -2.1032409 6.1884651  
## [54,] -1.9907368 5.9750088  
## [55,] -0.8498377 6.2573598  
## [56,] -0.3116259 6.5841596  
## [57,] 0.2376927 6.2083794  
## [58,] -0.1096735 6.5507347  
## [59,] -2.4178976 6.9420906  
## [60,] -1.7605444 7.4316442  
## [61,] -1.8763137 7.8751723  
## [62,] -1.6894675 8.0754582  
## [63,] -2.2204575 7.7086373  
## [64,] -2.5788636 7.5417789  
## [65,] -1.8777281 8.0657586  
## [66,] -1.2226141 7.2850227  
## [67,] -1.4370582 7.4002311  
## [68,] -2.2596416 7.2444567  
## [69,] -2.7902811 6.9449859  
## [70,] -2.6110399 6.9127552  
## [71,] -1.2306450 6.5747726  
## [72,] -2.4461856 6.4754130  
## [73,] -3.4730843 6.1857748  
## [74,] -3.4834452 6.4100095  
## [75,] -2.4941509 6.4187169  
## [76,] -2.6182505 6.4389381  
## [77,] -3.5834622 6.2556783  
## [78,] -3.7094812 6.3769652  
## [79,] -4.1111468 6.2055661  
## [80,] -4.2753885 6.3459069  
## [81,] -3.8991938 5.6806649  
## [82,] -3.0239002 6.0184367  
## [83,] -3.6291951 5.8199842  
## [84,] -2.9470163 5.7936527  
## [85,] -2.7317437 5.8222935  
## [86,] -3.3497645 5.6337363  
## [87,] -3.6414971 5.8835733  
## [88,] -3.3660877 7.0248790  
## [89,] -3.3702472 6.3290423  
## [90,] -1.9420033 5.9776494  
## [91,] -1.8636743 5.9259821  
## [92,] -1.9101788 6.2726569  
## [93,] -1.3250397 6.4003948  
## [94,] -1.4254440 5.8249932  
## [95,] -0.9591449 5.0575437  
## [96,] -0.9386335 4.9456182  
## [97,] -0.5544921 4.4220340  
## [98,] -1.7741553 4.1807556  
## [99,] -0.9495628 3.7973389  
## [100,] -1.6885438 3.3753600

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



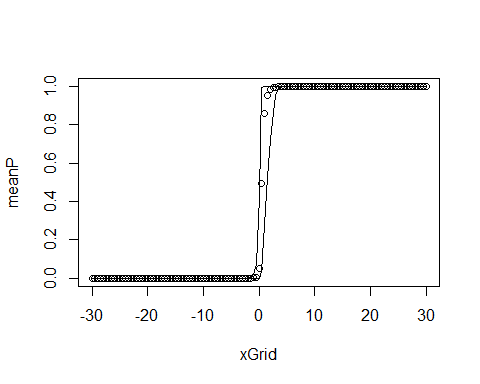
hist(sampleBetas[,2])



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

## [1] -2.408648 4.953070

xGrid = seq(-30,30,0.5)  
probMap = matrix(0, ncol = length(xGrid), nrow = 100)  
  
for (i in 1:100){  
 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)

 # 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>