Probit regression for ordinal data: Categorical features

Setting:

- For each observations, we have 3 associated variables

X1: Nomial variable with 3 possible outcomes {1,2,3} which we will generate from $Mult(1, [0.2, 0.3, 0.5])$ X2: Nomial variable with 3 possible outcomes {1,2,3} which we will generate from $Mult(1, [0.3, 0.5, 0.2])$ Y: Ordinal variable with 5 levels, {1,2,3,4,5} generated using the following process

$\epsilon_i \sim N(0, 1)$ $z_i = \beta^T X_i + \epsilon$ where $X = [X1 == 1, X1 == 2, X2 == 1, X2 == 2]$ i.e. we drop category (1,1) as the baseline $g(z_i) = Y_i$.

Here $\beta = [-3, 2, 2, -4]$ and function g will be the binning function which will bin data into 5 different bins corresponding to 5 possible ordinal outcome.

- We will try to model Y conditioned on other variables (X1, and X2) using probit regression with latent variable Z with rank likelihood on parameter Z.
- We have two unknown parameters β and z_i which will be sampled from the full conditional posterior distribution using blocked gibbs sampling.

Summary on modelling process

$\epsilon_i \sim N(0, 1)$ $z_i = \beta^T X_i + \epsilon$ $z_i \in R(Y)$ where R(Y) = $\{z_i$: $z_i > z_j$ if $Y_i > Y_j$ and $z_i < z_j$ if $Y_i < Y_j\}$

```r
# Data generating process
set.seed(0)
n = 300
beta = c(-3, 2, 2, -4)

# noise term
epsilon = rnorm(n, mean = 0, sd = 1)

# X1
X1 = t(rmultinom(n, size = 1, prob = c(0.2, 0.3, 0.5)))

# X2
X2 = t(rmultinom(n, size = 1, prob = c(0.3, 0.5, 0.2)))

# X
X = cbind(X1[,2:3], X2[,2:3])
colnames(X) <- c('X1_cat2', 'X1_cat3', 'X2_cat2', 'X2_cat3')

# Z
Z = X%*%beta + epsilon

# Cut-off points and Y
g = quantile(Z, probs = c(0.2, 0.4, 0.6, 0.8))
Y = rep(NA, n)
Y[Z<g[1]] = 1
Y[Z>=g[1] & Z<g[2]] = 2
Y[Z>=g[2] & Z<g[3]] = 3
Y[Z>=g[3] & Z<g[4]] = 4
Y[Z>=g[4]] = 5
```

Prior specifications:

$\beta \sim multiN(0, n(X^TX)^{-1})$

Blocked Gibbs Sampling here consists of two major steps

1. Sample new β from its full conditional

$\beta \mid z, X, Y, z \in R(Y) \sim multiN(\frac{n}{n+1}(X^TX)^{-1}X^Tz, \frac{n}{n+1}(X^TX)^{-1})$

2. for each i, using inverse cdf method, sample new z_i from its full conditional which is a truncated normal distribution:

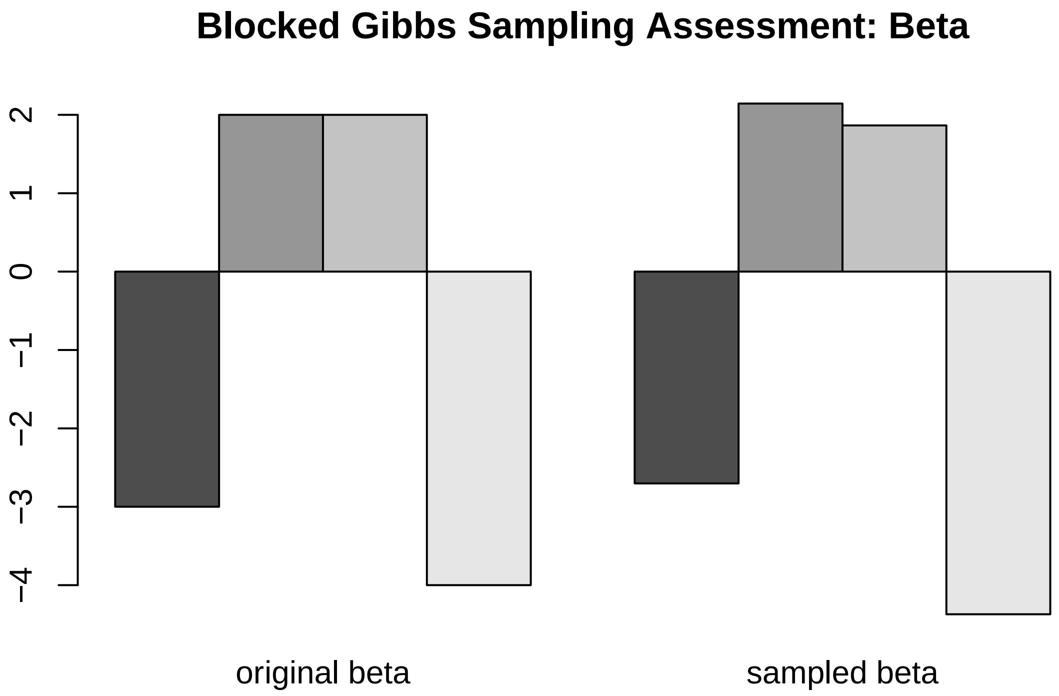
$z_i \mid \beta, X, Y, z_i \in R(Y) \sim N(\beta^T x_i, 1) * I\{z_i \in (a, b)\}$

where a = max(z_j for $Y_j < Y_i$) b = min(z_j for $Y_j > Y_i$)

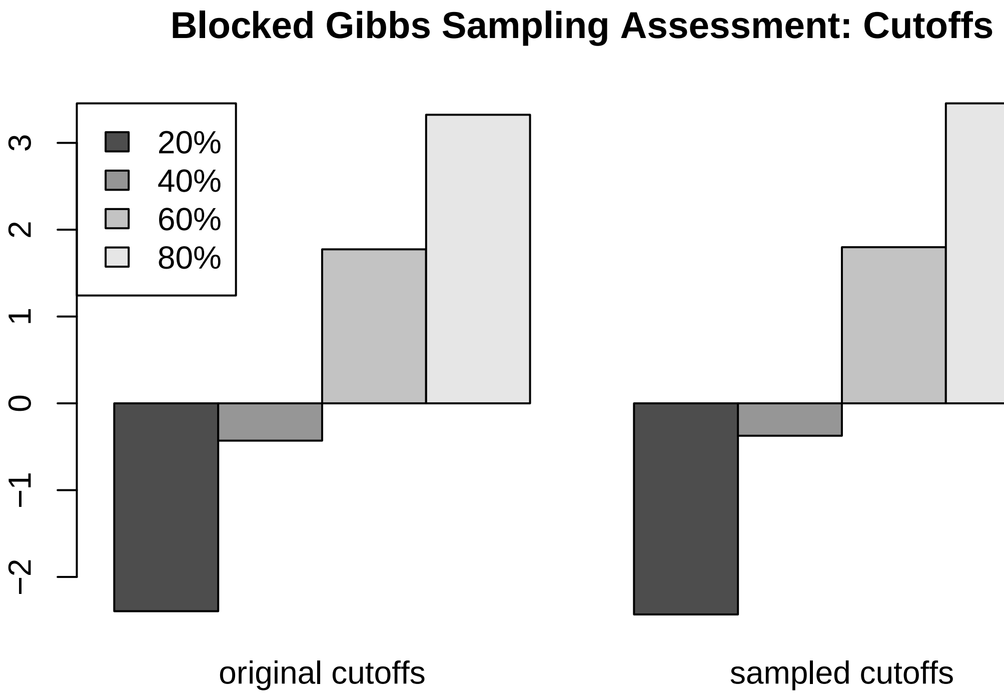
Note that in the gibbs sampling process, we force the first threshold (g1) to be at the true value in the data generating process otherwise the parameters will be unidentifiable (we can definitely fine infinite combinations of weights parameter to order the outcome according to their target Y: scaling, shifting, etc.)

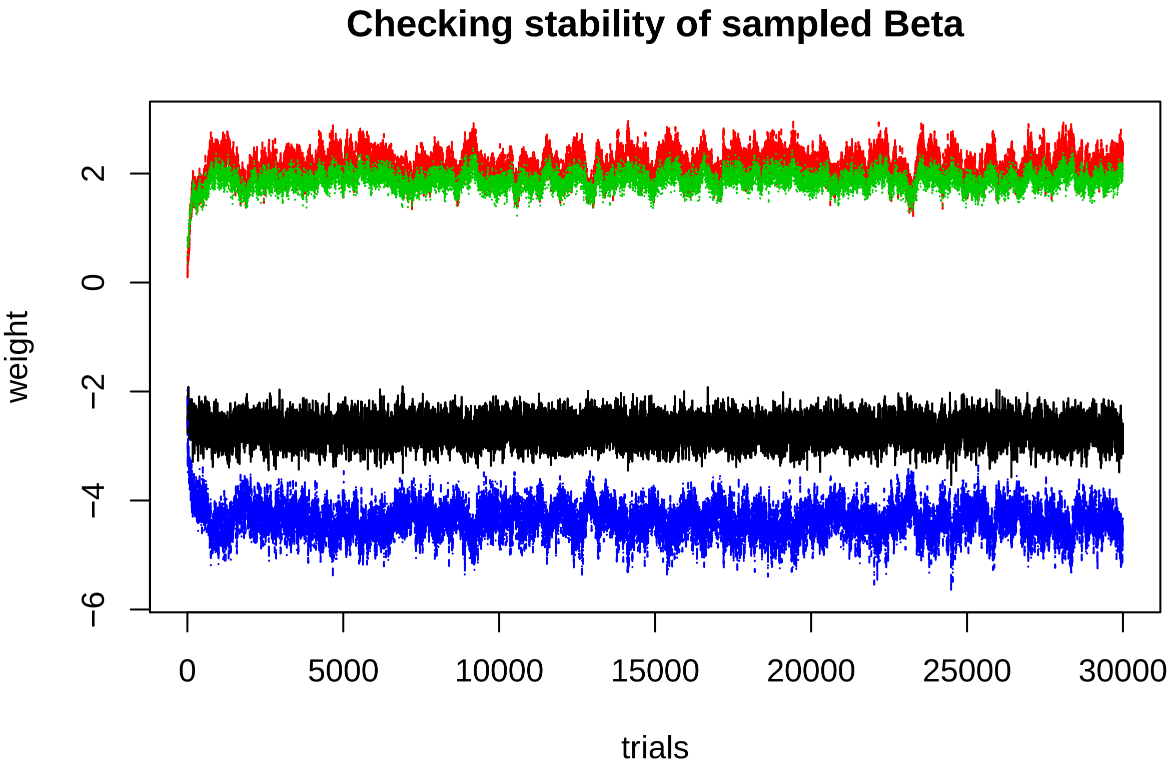
```r
# Blocked Gibbs Sampling
set.seed(1)

# Prior Parameter
g1 = g[1] #Fix g1 the first cutoff
Z_i = g1 + Y - 1.5
variance_beta = (n/(n+1))*solve(t(X)%*%X)
mean_beta_hat = (n/(n+1))*solve(t(X)%*%X)%*%t(X)

# Initialize the sampling matrix
S = 30000
SAMPLED_Z = matrix(nrow=S,ncol=n)
BETA = matrix(nrow=S,ncol=4)

for (round in 1:S) {
  # Step 1: Sample Beta
  mean_beta = mean_beta_hat%*%Z_i
  beta_sampled = dae::rmvnorm(mean = mean_beta,
                              V = variance_beta, method = 'choleski')
  BETA[round,] <- beta_sampled

  # Step 2: Sample Z using inverse cdf appraoch
  for (i in 1:n) {
    # Get the lower and upper bound (a, b) of truncated normal
    a = max(-Inf, Z_i[Y<Y[i]], na.rm = TRUE)
    b = min(Z_i[Y>Y[i]],Inf, na.rm = TRUE)

    # Force the lowest cutoff to be at g1
    if (Y[i] == 1) {
      b = g1
    }else if (Y[i] == 2) {
      a = g1
    }

    # Sample using inverse cdf
    ez = t(beta_sampled)%*%X[i,]
    u = runif(1, pnorm(a - ez), pnorm(b-ez))
    Z_i[i] = ez + qnorm(u)
  }
  SAMPLED_Z[round,] <-  Z_i
}
```

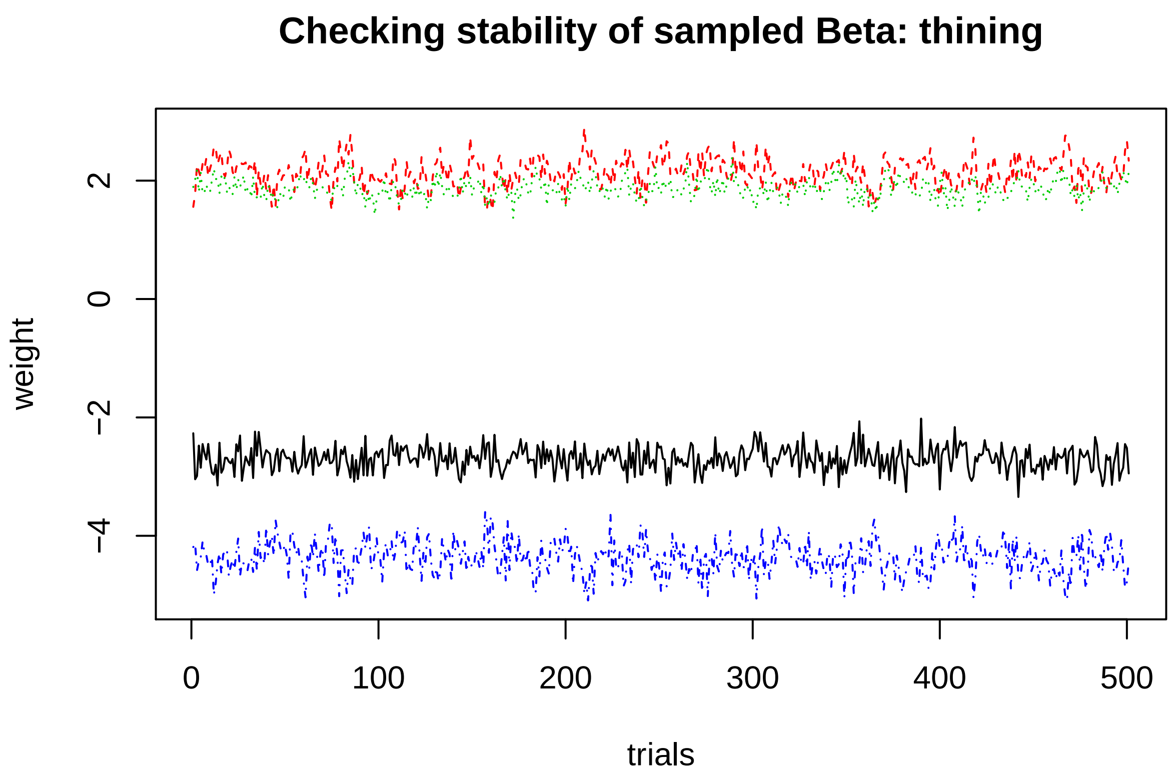
```
burnin = 5000
thining = 50
# Check posterior expectation of beta
sampling_beta = apply(BETA[seq(burnin, dim(BETA)[1], thining),], MARGIN = 2, mean)
df_beta= cbind(beta, sampling_beta)
colnames(df_beta)<- c('original beta', 'sampled beta')
barplot(df_beta, beside = TRUE,
        legend = TRUE, main = 'Blocked Gibbs Sampling Assessment: Beta')
```

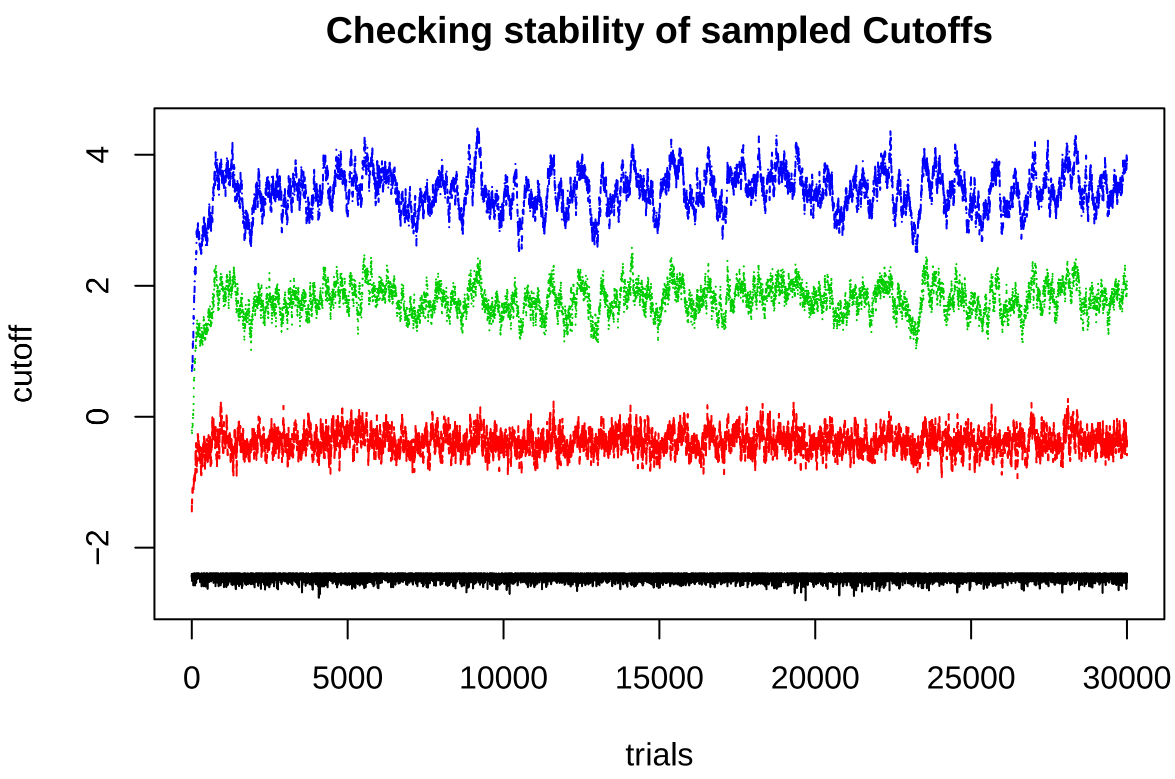


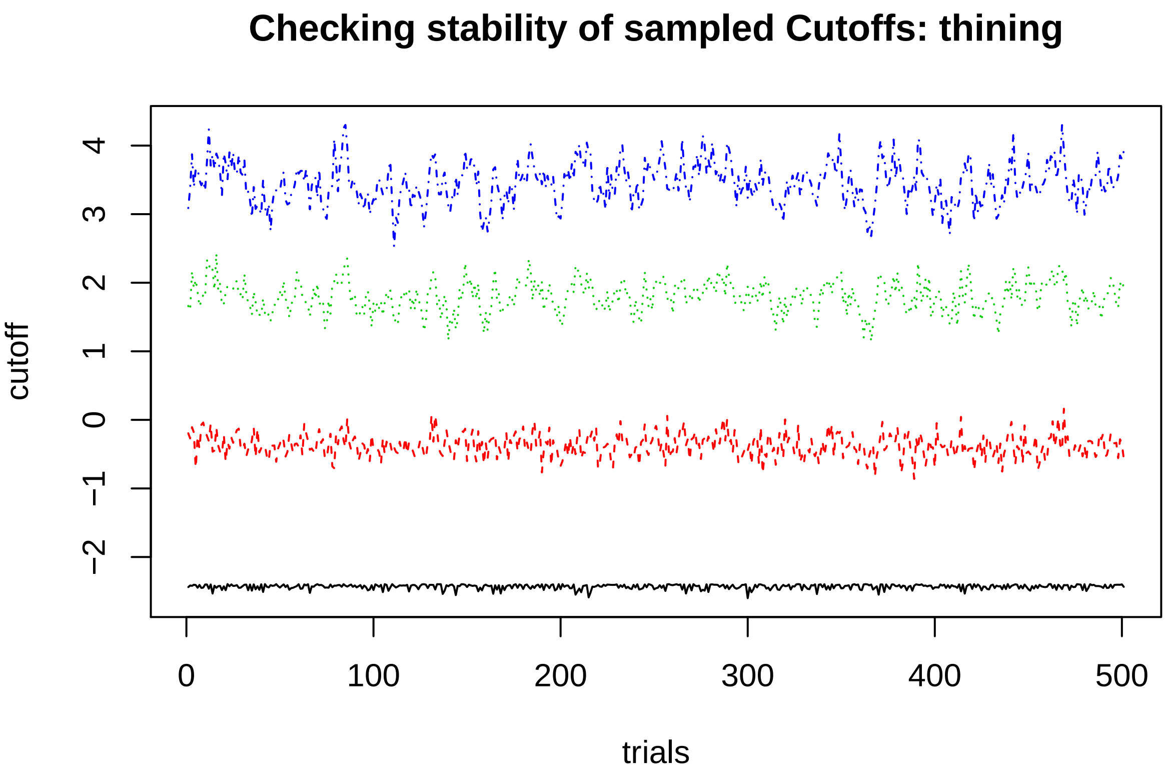
```
# Check cut off points
g1 = apply(SAMPLED_Z[, Y == 1], MARGIN = 1, max)
g2 = apply(SAMPLED_Z[, Y == 2], MARGIN = 1, max)
g3 = apply(SAMPLED_Z[, Y == 3], MARGIN = 1, max)
g4 = apply(SAMPLED_Z[, Y == 4], MARGIN = 1, max)

df_g= cbind(g, c(mean(g1[seq(burnin, dim(BETA)[1], thining)]),
                 mean(g2[seq(burnin, dim(BETA)[1], thining)]),
                 mean(g3[seq(burnin, dim(BETA)[1], thining)]),
                 mean(g4[seq(burnin, dim(BETA)[1], thining)])))
colnames(df_g)<- c('original cutoffs', 'sampled cutoffs')
barplot(df_g, beside = TRUE,
        legend = TRUE, main = 'Blocked Gibbs Sampling Assessment: Cutoffs',
        args.legend=list(x='topleft'))
```


Blocked Gibbs Sampling Assessment: Cutoffs
20%
40%
60%
80%
3
2
1
0
−1
−2
original cutoffs
sampled cutoffs


Checking stability of sampled Beta
weight
2
0
−2
−4
−6
0
5000
10000
15000
20000
25000
30000
trials


Checking stability of sampled Beta: thining
weight
2
0
−2
−4
0
100
200
300
400
500
trials


Checking stability of sampled Cutoffs
cutoff
4
2
0
−2
0
5000
10000
15000
20000
25000
30000
trials


Checking stability of sampled Cutoffs: thining
cutoff
trials
−2
−1
0
1
2
3
4
0
100
200
300
400
500