Lecture 12

Latent Variable Methods

Latent variable methods are a cornerstone of many Bayesian statistical procedures. Typically for modeling convenience a latent (unobservable) normal random variable is assumed to underlie more complicated scenarios. A few common examples are:

Probit Regression

Probit regression is a binary GLM that uses the probit link rather than a logit link. The model can be written as:

Ordinal Regression

Ordinal regression is a specific type of multinomial regression model for categorical data. Likert scale questions are a common example of this type of response.

Censored Data

Censored data is common in many statistical settings including biostatistics and reliability. The basic idea is that all values of the data cannot be measured, and the data has to be censored. In some situations certain values cannot be measured as they are below a devices detection level; however, the values are actually zero. Thus there is some truncation of the variable, for example $ < .1$. This is an example of lower censoring. Upper censoring can also occur, a common example is survival analysis. Typically the response of interest is the time until death. If the subject is still living at the end of the study, the result will be $ > x$, where x is the time the subject was alive during the study. Similar to the previous examples a latent variable is assumed to underlie this process.

State-Space Modeling

Another example of latent variable modeling is state space models. A common example is:

Probit Regression Exercise

Data Simulation

```
set.seed(11192018)
num.pts <- 1000
beta <- c(1,1)
X <- rnorm(num.pts)
X.comb <- cbind(rep(1,num.pts),X)
X.beta <- X.comb %*% beta
probs <- pnorm(X.beta)
Y <- rbinom(num.pts, 1, probs)
```

Model Fitting

```
num.mcmc <- 10000
beta.samples <- matrix(1, nrow = num.mcmc, ncol = 2)
upper <- lower <- rep(0, num.pts)
upper[Y == 1] <- Inf
lower[Y == 0] <- -Inf
cov.beta <- solve(t(X.comb) %*% X.comb + diag(2))

for (iter in 2:num.mcmc){
  # sample latent z
  z <- rtruncnorm(num.pts, a = lower, b = upper, mean = X.comb %*% beta.samples[iter -1,], sd = 1)

  # sample beta
  exp.beta <- cov.beta %*% t(X.comb) %*% z
  beta.samples[iter, ] <- rmnorm(1, mean = exp.beta, varcov = cov.beta)
}


glm(Y~X, family = binomial(link = 'probit'))
```

```
##
## Call:  glm(formula = Y ~ X, family = binomial(link = "probit"))
##
## Coefficients:
## (Intercept)            X
##      0.9843       1.0406
##
## Degrees of Freedom: 999 Total (i.e. Null);  998 Residual
## Null Deviance:       1152
## Residual Deviance: 804.3     AIC: 808.3
```

```
colMeans(beta.samples)
```

```
## [1] 0.9791879 1.0358753
```

Q identify the priors for this model. Do these seem reasonable? **Q** are you satisfied that the sampler is working? **Q** conduct a posterior predictive check for this situation.

Data Analysis

Return to the seattle housing data set and fit a probit model to understand patterns that cause a house to sell for more than $400,000.

```r
library(readr)
seattle <- read_csv('http://www.math.montana.edu/ahoegh/teaching/stat532/data/SeattleBinaryHousing.csv')
```

```
## Parsed with column specification:
## cols(
##   GR.400k = col_integer(),
##   price = col_double(),
##   sqft_living = col_integer(),
##   sqft_lot = col_integer(),
##   zipcode = col_integer()
## )
```

Hierarchical GLMs

Now suppose we are interested in the number of students in each school that past the test rather than overall mean test score. Then consider this following model.

Q: Now what do we need for priors?

Q: How do we take posterior samples?

Now write out the model using a probit link.

Q: Now what do we need for priors?

Q: How do we take posterior samples?

Hierarchical Probit Code Example

Hierarchical Probit Function

```
Hierachical_Probit <- function(Y, X, id, cum_n, num.mcmc = 1000){
  # Gibbs Sampler for Hierarchical Binary Data
  m <- length(unique(id))
  p <- ncol(X)
  N <- length(Y)

  #initialize storage
  beta.samples <- array(0, dim=c(m, p, num.mcmc))
  theta.samples <- matrix(0, num.mcmc, p)
  Sigma.samples <-  array(0, dim=c(p, p, num.mcmc))
  Sigma.samples[,,1] <- diag(p)
  X.beta <- rep(0,N)

  # priors
  Lambda <- diag(p) * 1
  Lambda.inv <- solve(Lambda)
  mu.0 <- rep(0,p)
  nu.0 <- p + 2
  S.0 <- diag(p)

  # bounds for latent variables
  upper <- lower <- rep(Inf, N)
  upper[Y == 0] <- 0
  lower[Y == 0] <- -Inf
  lower[Y == 1] <- 0

  for (iter in 2:num.mcmc){
    ## Sample latent Z
    X.beta[1:cum_n[1]] <- X[(1):(cum_n[1]),] %*% beta.samples[1, , iter - 1]
    for (group.var in 2:m){
      X.beta[(cum_n[group.var - 1] + 1):(cum_n[group.var])] <-
        X[(cum_n[group.var - 1] + 1):(cum_n[group.var]),] %*% beta.samples[group.var, , iter - 1]
    }
    z <- rtruncnorm(N,a = lower, b = upper, mean = X.beta, sd = 1)

    ## sample betas
    Sigma.inv <- solve(Sigma.samples[,,iter - 1])

    x.tmp <- X[(1):(cum_n[1]),]
    z.tmp <- matrix(z[(1):(cum_n[1])], ncol = 1)
    var.beta <- solve(t(x.tmp) %*% x.tmp + Sigma.inv)
    exp.beta <- var.beta %*% (t(x.tmp) %*% z.tmp + Sigma.inv %*% theta.samples[iter-1,] )
    beta.samples[1, , iter] <- rmnorm(n = 1, mean = exp.beta, varcov = var.beta)
    for (group.var in 2:m){
      x.tmp <- X[(cum_n[group.var - 1] + 1):(cum_n[group.var]),]
      z.tmp <- matrix(z[(cum_n[group.var - 1] + 1):(cum_n[group.var])], ncol = 1)
      var.beta <- solve(t(x.tmp) %*% x.tmp + Sigma.inv)
      exp.beta <- var.beta %*% (t(x.tmp) %*% z.tmp + Sigma.inv %*% theta.samples[iter-1,] )
      beta.samples[group.var, , iter] <- rmnorm(n = 1, mean = exp.beta, varcov = var.beta)
    }
```

```
    ## sample theta
    var.theta <- solve(m * Sigma.inv + Lambda.inv)
    exp.theta <- var.theta %*% (Sigma.inv %*% colSums(beta.samples[, , iter]) + Lambda.inv %*% mu.0)
    theta.samples[iter, ] <- rmnorm(n = 1, mean = exp.theta, varcov = var.theta)

    ## sample Sigma
    Sigma.df <- nu.0 + m
    S.theta <- matrix(0, p, p)
    for (group.var in 1:m){
      S.theta <- S.theta + (beta.samples[group.var,,iter] - theta.samples[iter,]) %*%
        t(beta.samples[group.var,,iter] - theta.samples[iter,])
    }
    Sigma.scale <- (S.0 + S.theta)
    Sigma.samples[, , iter] <- riwish(Sigma.df, Sigma.scale)
  }
  colnames(theta.samples) <- colnames(X)
  dimnames(beta.samples)[[2]] <- colnames(X)
  return(list(beta.samples = beta.samples, theta.samples = theta.samples,
              Sigma.samples = Sigma.samples))
}
```

Simulate Hierarchical Binary Data

```
set.seed(11192018)
m = 25 # number of groups
n_m = rep(1000, m) # observations per group
N = sum(n_m) # total observations
cum_n <- cumsum(n_m)

p = 2 # number of predictors
theta <- c(0,1)
sigma <- diag(c(1,1))
beta <- rmnorm(m, mean = theta, varcov = sigma) # individual parameters

### simulate data
X <- matrix(c(rep(1,N),rnorm(N)), nrow=N, ncol=2)
Y <- Xbeta <- probs <- id <- rep(0,N)

Xbeta[1:cum_n[1]] <- X[1:cum_n[1],] %*% beta[1,]
probs[1:cum_n[1]] <- pnorm(Xbeta[1:cum_n[1]])
Y[1:cum_n[1]] <- rbinom(n_m[1],1,probs[1:cum_n[1]])
id[1:cum_n[1]] <- 1
for (group.var in 2:m){
  Xbeta[(cum_n[group.var - 1] + 1):(cum_n[group.var])] <-
    X[(cum_n[group.var - 1] + 1):(cum_n[group.var]),] %*% beta[group.var,]
  probs[(cum_n[group.var - 1] + 1):(cum_n[group.var])] <-
    pnorm(Xbeta[(cum_n[group.var - 1] + 1):(cum_n[group.var])])
  Y[(cum_n[group.var - 1] + 1):(cum_n[group.var])] <-
    rbinom(n_m[group.var],1,probs[(cum_n[group.var - 1] + 1):(cum_n[group.var])])
  id[(cum_n[group.var - 1] + 1):(cum_n[group.var])] <- group.var
}

sim <- data.frame(Y = as.factor(Y), id = factor(id), X=X[,2])
```

```
ggplot(data=sim, aes(Y)) + geom_bar() + facet_wrap(~id,labeller = "label_both") +
  ggtitle('appearance across groups') + xlab('')
```

Analysis

```
tmp.vals <- Hierachical_Probit(Y, X, id, cum_n, num.mcmc = 8000)

beta.samples <- tmp.vals$beta.samples
kable(beta[1:5,])
kable(rbind(rowMeans(beta.samples[1,,]), rowMeans(beta.samples[2,,]), rowMeans(beta.samples[3,,]),
            rowMeans(beta.samples[4,,]), rowMeans(beta.samples[5,,])))

theta.samples <- tmp.vals$theta.samples
plot(theta.samples[1:8000 %% 5 == 0,1], type = 'l', main = 'Trace plot for theta 1')
abline(h = theta[1], col= 'red', lwd = 2)

plot(theta.samples[1:8000 %% 5 == 0,2], type = 'l', main = 'Trace plot for theta 2')
abline(h = theta[2], col= 'red', lwd = 2)
```