Bayesian Multivariate Linear Regression

Onur Boyar

8/12/2020

Introduction

In this work, I apply Bayesian Multiple Linear Regression with Blocked Gibbs Sampling. Different Bayesian models are obtained with using different priors on regression coefficients and covariance matrix.

The Data

The data used in this work is S&P returns along with 5 other stock returns. The objective is to predict S&P returns using other variables.

The Bayesian Model

Let our target variable, S&P prices, be denoted by y with length n. The Bayesian Multivariate Regression assumes that this vector is drawn from a multivariate normal distribution where the mean vector is $X\beta$ and covariance matrix ϕI. Here, X is and observed nxp matrix of covariates where p is equal to number of features. The first column of the X_{nxp} is a vector of 1's.

The parameter vector β is $px1$, ϕ is a common variance parameter, and I is the nxn identity matrix. By using the identity matrix, we are assuming independet observations. Formally,

$$y \sim N_n(X\beta, \phi I)$$

So far, this is identical to the multivariate normal regression seen in the frequentist setting. Assuming X is full column rank, maximizing the likelihood yields the solution:

$$\hat{\beta} = (X^{'}X)^{-1}X^{'}y$$

We will obtain a Bayesian model by specifying a prior distribution for β and ϕ. For this example, we will use a flat, improper prior for β and an inverse gamma prior for ϕ.

$$\beta \sim f(\beta) \propto 1\phi \sim IG(\alpha, \gamma)$$

We can also put priors on the parameters of Inverse Gamma distribution which will lead us to a Hierarchical Model. For simplicity, we assume these hyperparameters are known.

Joint Posterior Distribution

Now that we have defined our model, we can calculate the Joint Posterior distribution. We want to infer ϕ and β given our data, y and X. We have already stated that y is coming from a distribution with parameters β, ϕ and the matrix X.

The joint posterior distribution is proportional to

$$p(\beta, \phi | y, X) \propto p(y|\beta, \phi, X) p(\beta|X) p(\phi|X)$$

The likelohood function of multivariate normal distribution is

$$(2\pi\phi)^{-n/2} exp\bigg(-\frac{1}{2\phi} \sum (x_j - u)^2 \bigg)$$

where $\phi = \sigma^2$.

For Inverse gamma distribution, the probability density function is

$$\frac{\gamma^\alpha}{\Gamma(\alpha)} (1/x)^{\alpha+1} exp(-\gamma/x)$$

where $x = \phi$ for our case.

Another assumption of our model is prior independence. Prior distributions on β and ϕ are independent.

$$p(\beta, \phi | X) = p(\beta|X) p(\phi|X)$$

We can substitute the distributions and obtain the final posterior distribution now.

$$p(\beta, \phi | y, X) \propto \phi^{n/2} e^{-1/2\phi (y-XB)^{'} (y-XB) \phi - (\alpha+1) e^{-y/\phi}}$$

Gibbs Sampling

Gibbs sampling is a MCMC algorithm that repeatedly samples from the conditional distribution of one variable of the target distribution p, given all of the other variables.

Gibbs sampling assumes we can compute conditional distributions of one variable conditioned on all of the other variables and sample exactly from these distributions.

Another sampling method used in Bayesian statistics is Metropolis-Hastings algorithm. In Metropolis-Hastings algorithm we have a proposal distribution and acceptance probabilities for each draws. Gibbs sampling can be considered as a special case of Metropolis-Hastings where the proposed moves are aloways accepted, the acceptance probability is 1.

One variant of Gibbs sampling is blocked Gibbs sampling, where we group two or more variables together in a block and update this block by sampling from the joint distribution of these variables conditioned on all of the other variables. Updating more variables at a time in blocks is helpful. Consider the limiting case where if we could sample from a block containing all of the variables, then we could sample directly from p(x).

In this work, we use Blocked Gibbs Sampling and sample from each feature in every step. We also perform the parameter updates in each step of the chain. We know that inverse gamma distribution is a conjugate prior for covariance values.

We will iteratively update parameters of β and ϕ. We will first derive conditional posteriors of β and ϕ.

The conditional posterior of ϕ is found by dropping factors independent of ϕ from the joint posterior.

$$p(\phi|\alpha,\gamma,\beta,y,X) \propto \phi^{n/2} e^{-1/2\phi(y-XB)^{'}(y-XB)\phi-(\alpha+1)e^{-y/\phi}}$$

$$\propto \phi^{-(\alpha+n/2+1)} e^{1/\phi}[1/2(y-X\beta)^{'}(y-X\beta)+\gamma]$$
$$= InverseGamma(shape = a + n/2, rate = 1/2(y-X\beta)^{'}(y-X\beta)+\gamma)$$

To calculate the conditional posterior of β we can drop initial $\phi^{n/2}$, $\phi^{-(a+1)}$ and $e^{-y/\phi}$.

$$p(\beta|\phi,\gamma,\alpha,y,X) \propto e^{-1/2\phi(y-X\beta)^{'}(y-X\beta)}$$
$$\propto e^{-1/2\phi(y^{'}y-2\beta^{'}X^{'}y+\beta^{'}X^{'}X\beta)}$$
$$\propto e^{-1/2\phi(\beta^{'}X^{'}X\beta-2\beta^{'}X^{'}y)}$$
$$= N_p((X^{'}X)^{-1}X^{'}y, \phi(X^{'}X)^{-1})$$

The reason that our distribution for β is centered around the maximum likelihood estimate for $\hat{\beta}$ is that we are using a flat prior. Besides, the covariance matrix of the conditional posterior is also the frequentist estimate of the covariance matrix, $\mathrm{Cov}(\hat{\beta}) = \phi(X^{'}X)^{-1}$.

Now that we are creating a Multivariate Regression model, β will be a vector of coefficients. In each iteration of the Gibbs sampler, we will draw a whole vector, or “block”, from the posterior. Drawing all of the coefficients at once is what makes our Gibbs Sampler a Blocked Gibbs Sampler. It is much more efficient than drawing from the conditional distribution of each parameter conditional on the others, one at a time.

Blocked Gibbs Sampler: Implementation

```r
library(mvtnorm)
library(invgamma)
library(ggplot2)
library(dplyr)
library(tidyr)
library(xtable)

block_gibbs<-function(y, x, iter, burnin, trim,
                      inv_gamma_shape, inv_gamma_scale){
  # block_gibbs is a function that applies blocked gibbs
  # sampler.
  # ...PARAMETERS
  # ...y -> target variable
  # ...x -> feature matrix
  # ...iter -> number of iterations
  # ...burnin -> specifies the index that simulated values becomes usable values.
  # ...trim -> value that determines the step size when we selecting simulated values
  #            from (burnin, iter). For instance, we select the value at index burnin
  #            and the second value is at the index burnin+trim
  # ...inv_gamma_shape -> shape parameter for inverse gamma prior
  # ...inv_gamma_scale -> scale parameter for inverse gamma prior
  # ...RETURNS
  # ...joint_post_log -> joint posterior distribution
  # initialize gibbs
  xprimex_inv<-solve(t(x)%*%x) # calculate once for repeated use in sampler
  phi<-numeric(iter) # shell for phi
  b<-matrix(nrow=iter, ncol = ncol(x)) # shell for betas
  phi[1]<-6 # initial phi value to start sampler

  # phi hyperparameters
  a<-inv_gamma_shape
  g<-inv_gamma_scale

  # gibbs sampling
  for(i in 2:iter ){
    b[i,]<-rmvnorm(n = 1,
                   mean = ((xprimex_inv%*%t(x))%*%y),
                   sigma = phi[i-1]*xprimex_inv )

    phi[i]<-rinvgamma(n = 1,
                      shape = (n/2 + a),
                      rate = .5*( t((y - x%*%t(t(b[i,])) ))%*%(y - x%*%t(t(b[i,])) ) ) + g)
  }

  # apply burnin and trimming
  keep_draws<-seq(burnin,iter,trim)
  phi<-phi[keep_draws]
  b<-b[keep_draws,]

  # format and output
  joint_post<-data.frame(b=b,phi=phi)
  colnames(joint_post)[1:(ncol(x))]<-paste0('B',0:(ncol(x)-1) )

  joint_post_long<-gather(joint_post,keep_draws) %>%
    rename(param=keep_draws, draw=value) %>%
    mutate(iter=rep(keep_draws,ncol(joint_post)))

  return(joint_post_long)
}
```

In *block_gibbs* function, we run the Blocked Gibbs Sampler given data and other parameters and return formatted data where each parameter has its column and each row is different draw from the Gibbs Sampler.

To run the simulation, first thing we need to do is to read our data and feed it into the *block_gibbs* function.

```
data <- read.csv("/Users/Scoutium/Desktop/dataset.csv")[,-1]
## 75% of the sample size
smp_size <- floor(0.75 * nrow(data))
## set the seed to make your partition reproducible
set.seed(123)
train_ind <- sample(seq_len(nrow(data)), size = smp_size)
train <- data[train_ind, ]
test <- data[-train_ind, ]
lm_model <- lm(train$SP~.,data=train[,-c(1,7)])
summary(lm_model)
```

```
##
## Call:
```

```
## lm(formula = train$SP ~ ., data = train[, -c(1, 7)])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.5605 -1.6465 -0.1678  1.5680  7.4284
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 732.42804   10.89127   67.25   <2e-16 ***
## V2            3.78618    0.07853   48.22   <2e-16 ***
## V143         -1.27807    0.06353  -20.12   <2e-16 ***
## V315          4.39701    0.14132   31.11   <2e-16 ***
## V338          8.27594    0.15288   54.13   <2e-16 ***
## V463          1.93037    0.09339   20.67   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.569 on 2994 degrees of freedom
## Multiple R-squared:  0.9332, Adjusted R-squared:  0.9331
## F-statistic:  8372 on 5 and 2994 DF,  p-value: < 2.2e-16
```

```r
beta <- lm_model$coefficients
tphi <- 1000
n <- nrow(train)
I <- diag(1,n,n)

y <- train[,7]
x <- train[,-7]
x <- as.matrix(x)
```

After reading the data and partitioning it into train and test set, we created frequentist regression model to compare with Bayesian model after applying Gibbs Sampler. The data included column that consists of 1's in the first place. This is why we excluded it when using *lm* function.

Next step is applying the Gibbs Sampler.

```r
system.time(post_dist<-block_gibbs(y = y,
                                   x = x,
                                   iter = 500000,
                                   burnin = 100000,
                                   trim = 50,
                                   inv_gamma_shape = 5,
                                   inv_gamma_scale = 10000))
```

```
##    user  system elapsed
## 371.962  57.901 447.465
```

Visualization of the Results

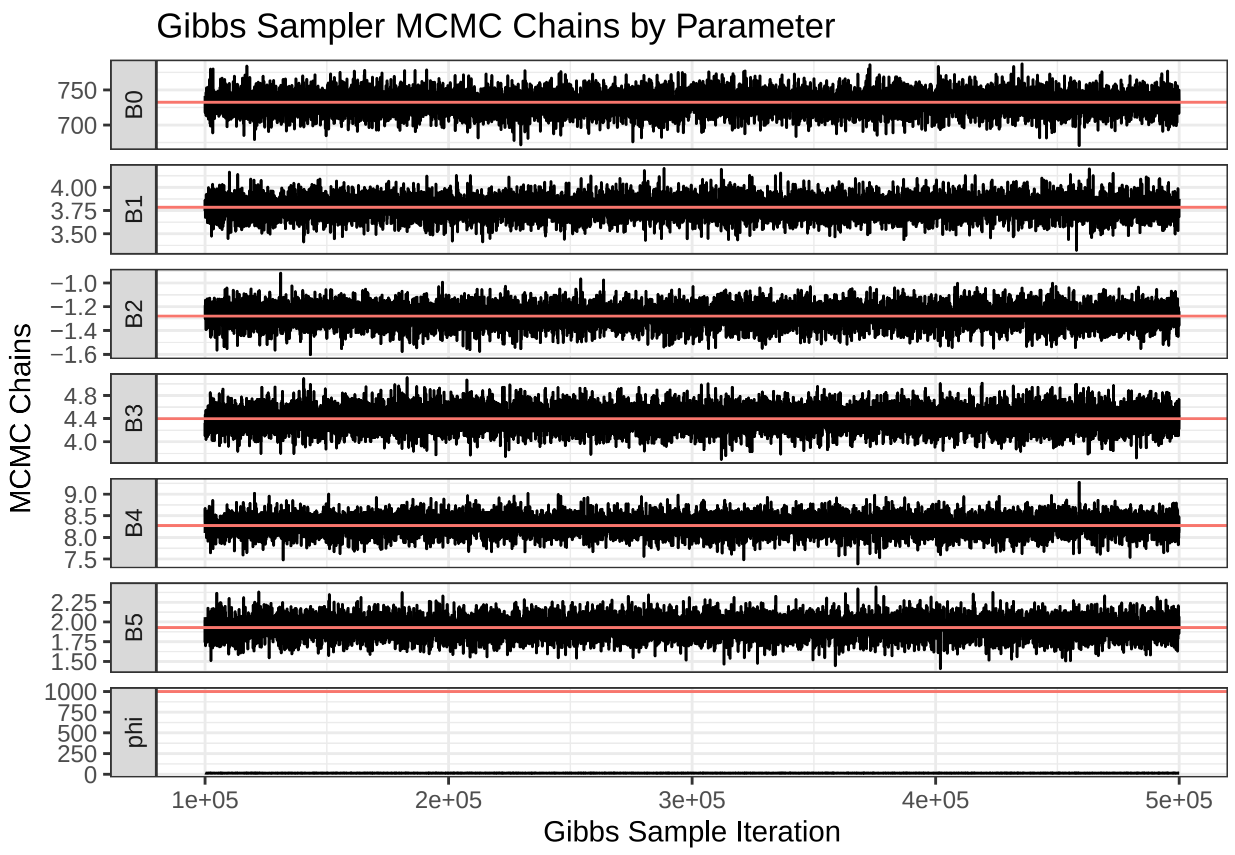
First we will calculate summary statistics of our posterior distribution. These are MAP estimation of the posterior, which is median, mean, 97.5 quantile and 2.5 quantile.

```
# calculate posterior summary statistics
post_sum_stats<-post_dist %>%
  group_by(param) %>%
  summarise(mean = mean(draw),
            median=median(draw),
            lwr=quantile(draw,.025),
            upr=quantile(draw,.975)) %>%
  mutate(initial_coefficients=c(beta,tphi))

# merge on summary statistics
post_dist <- post_dist %>%
  left_join(post_sum_stats, by='param')

knitr::kable(head(post_dist))
```

param	draw	iter	mean	median	lwr	upr	initial_coefficients
B0	741.3394	100000	732.142	732.1537	701.5844	762.1307	732.428
B0	719.8501	100050	732.142	732.1537	701.5844	762.1307	732.428
B0	717.2837	100100	732.142	732.1537	701.5844	762.1307	732.428
B0	740.1158	100150	732.142	732.1537	701.5844	762.1307	732.428
B0	732.8334	100200	732.142	732.1537	701.5844	762.1307	732.428
B0	728.1863	100250	732.142	732.1537	701.5844	762.1307	732.428

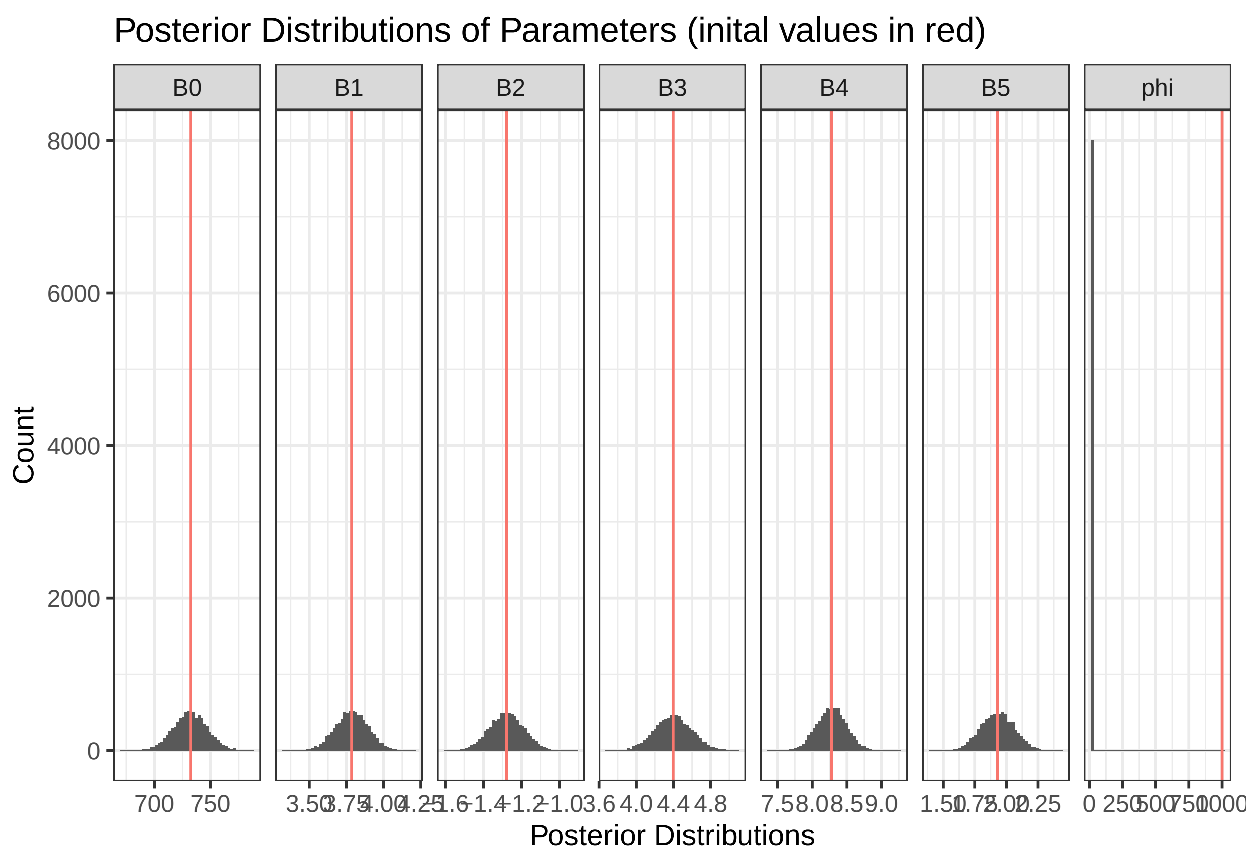
Next, we plot the MCMC chains and highlight the initial coefficient values, beta coefficients that are from frequentist estimation and the initial phi value.

```
# plot MCMC Chains
ggplot(post_dist,aes(x=iter,y=draw)) +
  geom_line() +
  geom_hline(aes(yintercept=initial_coefficients, col='red'), show.legend=FALSE)+
  facet_grid(param ~ .,scale='free_y',switch = 'y') +
  theme_bw() +
  xlab('Gibbs Sample Iteration') + ylab('MCMC Chains') +
  ggtitle('Gibbs Sampler MCMC Chains by Parameter')
```



Finally, we plot the posterior distributions of parameters.

```
# plot Posterior Distributions
ggplot(post_dist,aes(x=draw)) +
  geom_histogram(aes(x=draw),bins=50) +
  geom_vline(aes(xintercept = initial_coefficients,col='red'), show.legend = FALSE) +
  facet_grid(. ~ param, scale='free_x',switch = 'y') +
  theme_bw() +
  xlab('Posterior Distributions') + ylab('Count') +
  ggtitle('Posterior Distributions of Parameters (inital values in red)')
```



Obtaining Predictions

Fitted coefficient values from Frequentist Linear Regression are

```
beta
```

```
## (Intercept)          V2        V143        V315        V338        V463 
##  732.428044    3.786182   -1.278070    4.397006    8.275945    1.930372
```

MAP estimation of the values of the Bayesian Linear Regression are

```
MAP <- post_sum_stats$median[-7]
MEAN <- post_sum_stats$mean[-7]
print(c(MAP, MEAN))
```

```
##  [1] 732.153655   3.786911  -1.278086   4.397726   8.280904   1.930496
##  [7] 732.142003   3.786260  -1.278573   4.398522   8.278752   1.929786
```

Now, lets apply these values to our test set and obtain MSE values.

```
test <- as.matrix(test)
frequentist_model_predictions <- test[,-7]%*%beta
bayesian_model_mean_estimation_predictions <- test[,-7]%*%MEAN
```

```
bayesian_model_map_estimation_predictions <- test[,-7]%*%MAP


frequentist_model_mse <- mean((frequentist_model_predictions - test[,7])^2)
bayesian_model_map_estimation_mse <- mean((bayesian_model_map_estimation_predictions - test[,7])^2)
bayesian_model_mean_estimation_mse <- mean((bayesian_model_mean_estimation_predictions - test[,7])^2)


mse_vals = c(frequentist_model_mse, bayesian_model_map_estimation_mse, bayesian_model_mean_estimation_m
names(mse_vals) <- c("Frequentist Model MSE", "Bayesian Model MSE with MAP Estimation", "Bayesian Modell
print(mse_vals)
```

```
##                    Frequentist Model MSE
##                                 6.691021
##   Bayesian Model MSE with MAP Estimation
##                                 6.833318
## Bayesian Modell MSE with Mean Estimation
##                                 6.691143
```

We saw that Frequentist modell outperformed Bayesian Models for this example. The most surprising part is the difference between mean estimation and map estimation for coefficient values in Bayesian Model.

We can further create cross-validation sets and look at the MSE values at these sets and compare Frequentist and Bayesian Models.