MTH 511a - 2020: Lecture 31

Instructor: Dootika Vats

A popular Bayesian model is the Bayesian logistic regression model. In this lecture we will present the model and analyze the Titanic dataset from the exam.

1 Bayesian logistic regression

Consider a Bayesian logistic regression model. For $i = 1, \ldots, n$, let

$$x_i = (1, x_{i2}, \ldots, x_{i(p-1)})^T$$

be the vector of covariates for the ith observation and $\beta \in \mathbb{R}^p$ be the corresponding vector of regression coefficients. Suppose response y_i is a realization of Y_i with

$$Y_i | x_i, \beta \sim \text{Bern}\,(p_i) \quad \text{where} \quad p_i = \frac{\exp(x_i^T\beta)}{1 + \exp(x_i^T\beta)}\,.$$

Since this is a Bayesian model, we also assume that β has the following prior distribution

$$\beta \sim N_p(0,\, I_p)\,.$$

Our goal is to find the posterior distribution and report the posterior mean and credible intervals of β. In order to do this, the first thing we do is write down the posterior distribution.

$$\begin{aligned}\pi(\beta|y) &\propto \pi(\beta) \prod_{i=1}^{n} f(y_i|\beta) \\ &\propto e^{-\beta^T\beta/2} \prod_{i=1}^{n} (p_i)^{y_i} \,(1-p_i)^{1-y_i}\end{aligned}$$

The posterior is p dimensional, so here we need to sample from a p dimensional distribution. Our proposal distribution will be

$$q(\beta^* \mid \beta, \sigma^2) = \prod_{k=1}^{p} q(\beta^* | \beta) \, .$$

So each component is given it's own proposal value, independent of each other. We will use all normal distributions, with different step sizes $h_1, \ldots, h_p$. So we propose from

$$N_p \left(\beta_t \, , \begin{bmatrix} h_1 & \ldots & 0 & 0 \\ 0 & h_2 & 0 & 0 \\ \vdots & & \ddots & 0 \\ 0 & 0 & 0 & h_p \end{bmatrix} \right)$$

Note that this is a symmetric proposal, so the MH ratio is simplifies. Since we already know the MLE of the logistic regression model, we can start from the MLE solution!

```
##########################################
## Bayesian logistic regression
## with MH implementation
##########################################
# log posterior
logf <- function(beta)
{
  one.minus.yx <- (1 - y)*X
  -sum(beta^2)/2 - sum(log(1 + exp(-X%*%beta))) - sum(one.minus.yx%*%beta)
}

bayes_logit_mh <- function(y, X, N = 1e4, prop.sd = .35)
{
  p <- dim(X)[2]
  one.minus.yx <- (1 - y)*X

  # starting value is the MLE
  foo <- glm(y ~X - 1, family = binomial("logit"))$coef
  beta <- as.matrix(foo, ncol = 1)
  beta.mat <- matrix(0, nrow = N, ncol = p)
  beta.mat[1, ] <- as.numeric(beta)
  accept <- 0

  for(i in 2:N)
  {
    #symmetric density
    prop <- rnorm(p, mean = beta, sd = prop.sd)

    # log of the MH ratio
```

```
    log.rat <- logf(prop) - logf(beta)
    if(log(runif(1)) < log.rat)
    {
      beta <- prop
      accept <- accept + 1
    }
    beta.mat[i, ] <- beta
  }
  print(paste("Acceptance Prob = ", accept/N))
  return(beta.mat)
}
```

When we run the above function, it automatically prints the acceptance probability. We now load the dataset.

```
titanic <- read.csv("https://dvats.github.io/assets/titanic.csv")

y <- titanic[,1]
X <- as.matrix(titanic[, -1])
```

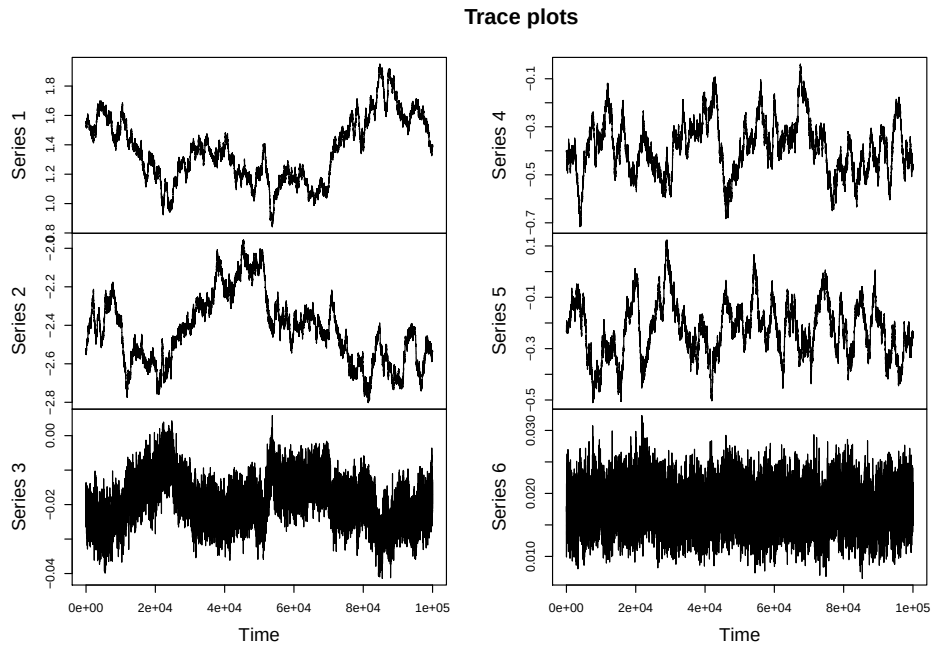
First we will try to find a proposal variance h that words reasonably well to give about 23% acceptance. We will do this by running the sampler for short (10^3) runs. First we let all proposal variance to be the same.

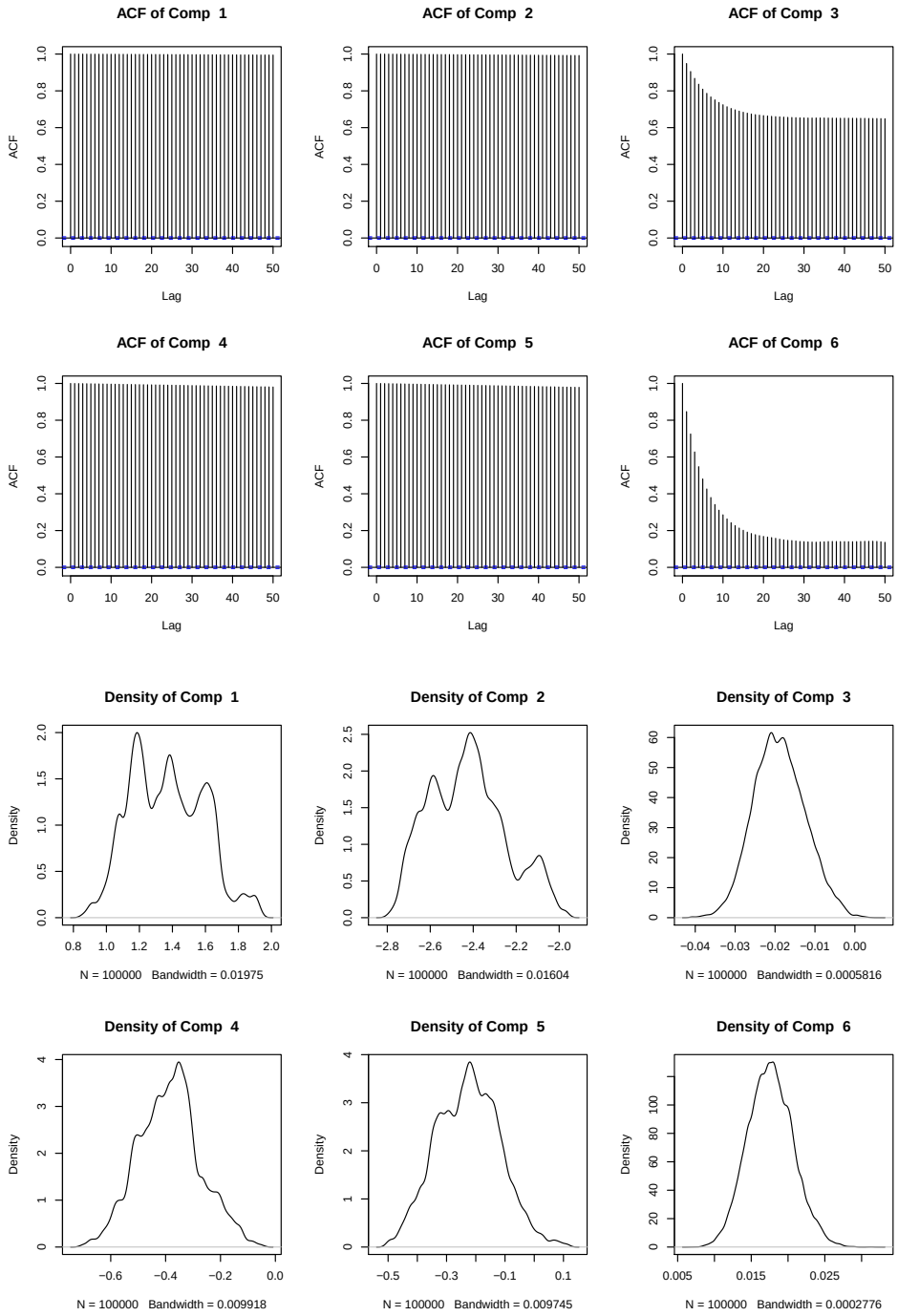
```
## acceptance is too low. we want 23%
## so decrease proposal variance
chain <- bayes_logit_mh(y = y, X = X, N = 1e3, prop.sd = .35)
#[1] "Acceptance Prob = 0"

# still too low
chain <- bayes_logit_mh(y = y, X = X, N = 1e3, prop.sd = .1)
#[1] "Acceptance Prob = 0"

# now its better
chain <- bayes_logit_mh(y = y, X = X, N = 1e3, prop.sd = .0065)
#[1] "Acceptance Prob = 0.218"
```

Notice our first two runs did not work well since our acceptance rate was too low. This means we were proposing large jumps and it would be better to take smaller jumps to increase acceptance. When we reduced the proposal sd to .0065 we got a decent acceptance rate. Now we will run this same run for longer (10^5) and print diagonstics.

```
# will now run the chain much longer for 10^5
# takes a few seconds
chain <- bayes_logit_mh(y = y, X = X, N = 1e5, prop.sd = .0065)

# all trace plots
```

```r
plot.ts(chain)

par(mfrow = c(2,3))
# all ACF plots
for(i in 1:dim(chain)[2])
{
  acf(chain[,i], main = paste("ACF of Comp ", i))
}

# all density plots plots
for(i in 1:dim(chain)[2])
{
  plot(density(chain[,i]), main = paste("Density of Comp ", i))
}
```

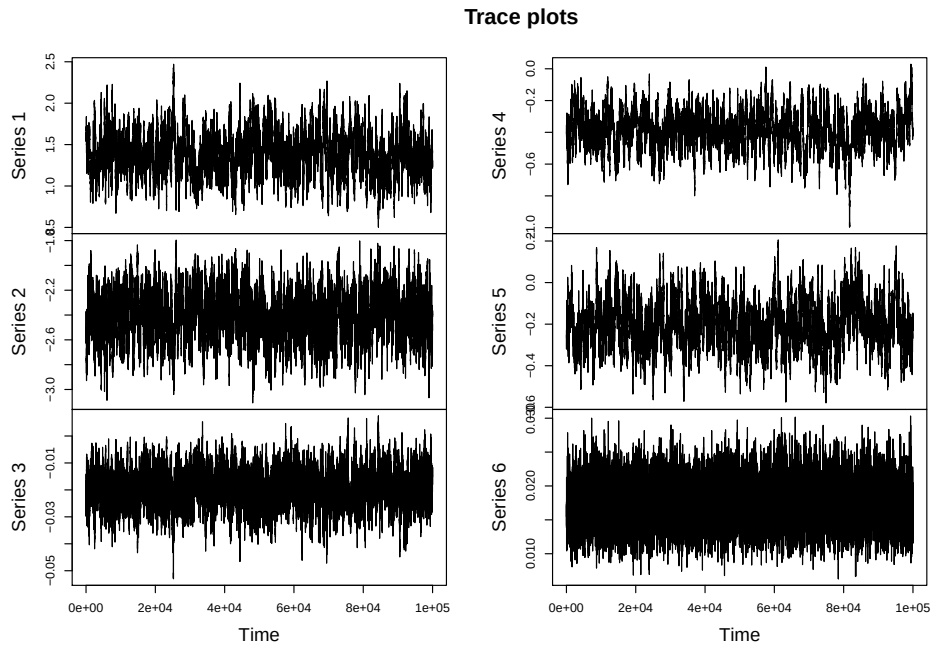


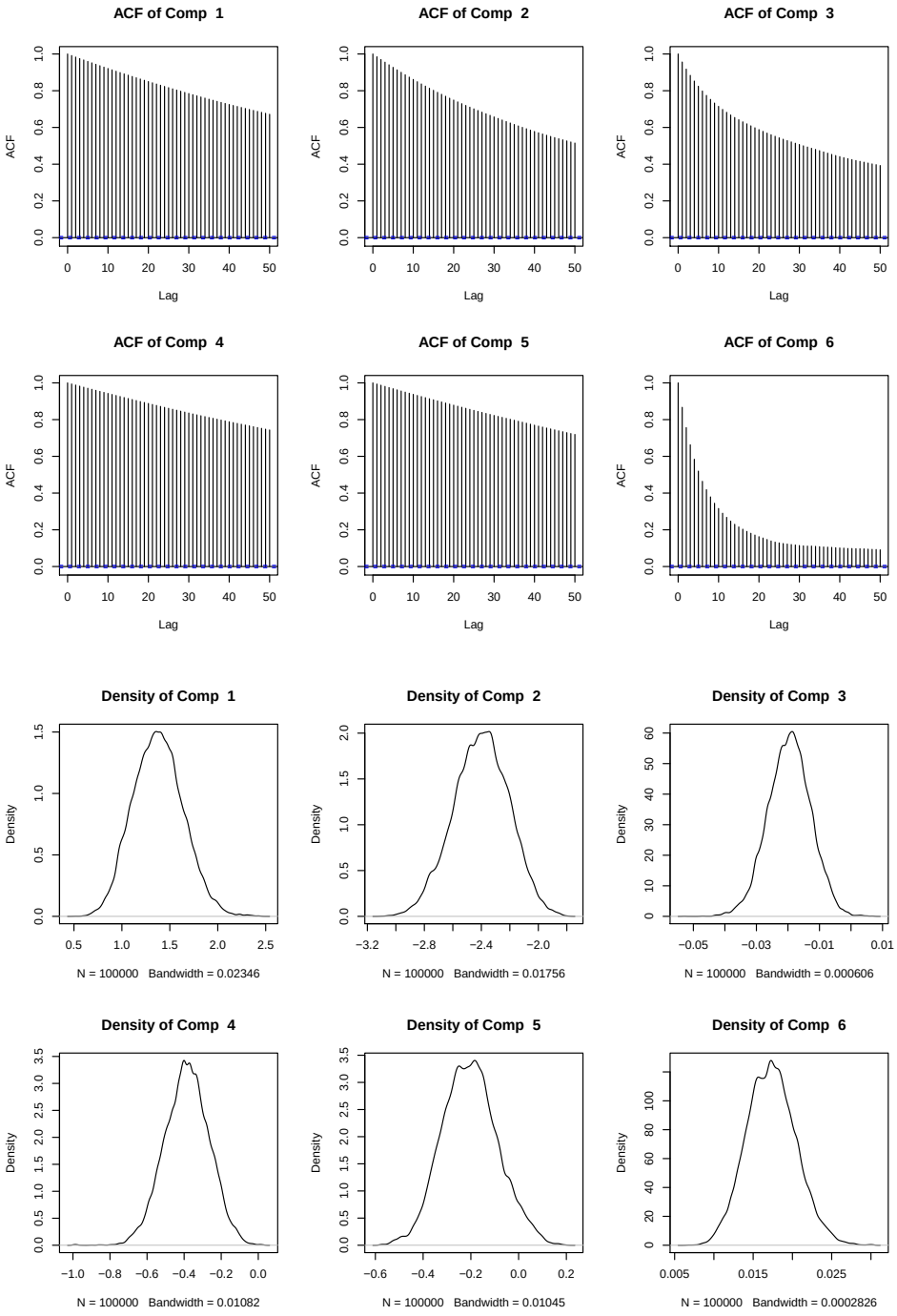


What we see is that although components 3 and 6 are well estimated with sample size 10^5, the other four are very poorly moving. This is because we choose the same proposal variance for each component which is not ideal here. We will now give each component a different proposal variance and run the sampler again for 10^5 steps.

```
# we see above that some components are ok, but 4 components are
```

```
# moving very slowly. This is because we are using the same proposal
# variance for each component, which is not adequate here.
# Below now I use different proposal variances for different
#components.


chain <- bayes_logit_mh(y = y, X = X, N = 1e5, prop.sd = c(.08, .08, .0065,
    .03, .03, .0065))

# all trace plots
plot.ts(chain, main = "Trace plots")

par(mfrow = c(2,3))
# all ACF plots
for(i in 1:dim(chain)[2])
{
  acf(chain[,i], main = paste("ACF of Comp ", i))
}

# all density plots plots
for(i in 1:dim(chain)[2])
{
  plot(density(chain[,i]), main = paste("Density of Comp ", i))
}
```





The estimated density plots, acfs, and trace plots are much better!

Thus, we see that MCMC, although powerful, can be difficult to tune. However, once you can make it work, it works reasonable well.

2 Questions to think about

- Try and implement MCMC for the Bayesian regression model.
- Obtain posterior mean and quantiles for the above implemented example. How do the final estimates compare to the MLE estimates?