Mattias Villani and Per Sidén
Division of Statistics and Machine Learning
Dept. of Computer and Information Science
Linköping university
mattias.villani@liu.se

2018-05-03

Bayesian Learning, 6 hp

Computer lab 4

- You are strongly recommended to use R for solving the labs since the computer exam will be in R.
- You are supposed to work and submit your labs in pairs, but do make sure that both of you are contributing.
- It is OK to discuss the lab with other student pairs in general terms, but `it is not allowed to share exact solutions`.
- Submit your *solutions* via LISAM no later than **May** 24 at **midnight**.

1. *Poisson regression - the MCMC way.*
   Consider the following Poisson regression model

$$y_i|\beta \sim \text{Poisson}\left[\exp\left(\mathbf{x}_i^T\beta\right)\right], \; i = 1, ..., n,$$

   where y_i is the count for the ith observation in the sample and x_i is the p-dimensional vector with covariate observations for the ith observation. Use the data set `eBayNumberOfBidderData.dat`. This dataset contains observations from 1000 eBay auctions of coins. The response variable is **nBids** and records the number of bids in each auction. The remaining variables are features/covariates ($\mathbf{x}$):

   - **Const** (for the intercept)
   - **PowerSeller** (is the seller selling large volumes on eBay?)
   - **VerifyID** (is the seller verified by eBay?)
   - **Sealed** (was the coin sold sealed in never opened envelope?)
   - **MinBlem** (did the coin have a minor defect?)
   - **MajBlem** (a major defect?)
   - **LargNeg** (did the seller get a lot of negative feedback from customers?)
   - **LogBook** (logarithm of the coins book value according to expert sellers. Standardized)
   - **MinBidShare** (a variable that measures ratio of the minimum selling price (starting price) to the book value. Standardized).

(a) Obtain the maximum likelihood estimator of β in the Poisson regression model for the eBay data [Hint: `glm.R`, don't forget that `glm()` adds its own intercept so don't input the covariate Const]. Which covariates are significant?

(b) Let's now do a Bayesian analysis of the Poisson regression. Let the prior be $\beta \sim \mathcal{N}\left[\mathbf{0}, 100 \cdot (\mathbf{X}^T\mathbf{X})^{-1}\right]$ where $\mathbf{X}$ is the $n \times p$ covariate matrix. This is a commonly used prior which is called Zellner's g-prior. Assume first that the posterior density is approximately multivariate normal:

$$\beta|y \;\sim\; \mathcal{N}\left(\tilde{\beta}, J_{\mathbf{y}}^{-1}(\tilde{\beta})\right),$$

where $\tilde{\beta}$ is the posterior mode and $J_{\mathbf{y}}(\tilde{\beta})$ is the negative Hessian at the posterior mode. $\tilde{\beta}$ and $J_{\mathbf{y}}(\tilde{\beta})$ can be obtained by numerical optimization (`optim.R`) exactly like you already did for the logistic regression in Lab 2 (but with the log posterior function replaced by the corresponding one for the Poisson model, which you have to code up.).

(c) Now, let's simulate from the actual posterior of β using the Metropolis algorithm and compare with the approximate results in b). Program a general function that uses the Metropolis algorithm to generate random draws from an *arbitrary* posterior density. In order to show that it is a general function for any model, I will denote the vector of model parameters by θ. Let the proposal density be the multivariate normal density mentioned in Lecture 8 (random walk Metropolis):

$$\theta_p|\theta_c \sim N\left(\theta_c, \tilde{c} \cdot \Sigma\right),$$

where $\Sigma = J_{\mathbf{y}}^{-1}(\tilde{\beta})$ obtained in b) and θ_c is the current draw (hence the subscript c) and is the same as $\theta^{(i)}$ in the slides for Lecture 8. The value $\tilde{c}$ is a tuning parameter and should be an input to your Metropolis function (so that a user can change it). The user of your Metropolis function should be able to supply her own posterior density function, not necessarily for the Poisson regression, and still be able to use your Metropolis function. This is not so straightforward, unless you have come across *function objects* in `R` and the triple dot (`...`) wildcard argument. Let me give you the roadmap. Ok, take a deep breath ... this is good stuff ...

First, one of the input arguments of your Metropolis function should be `logPostFunc` (or some other suitable name). `logPostFunc` is a *function object* that computes the log posterior density at any value of the parameter vector. This is needed when you compute the acceptance probability of the Metropolis algorithm. I suggest always to program the *log* posterior density, since logs are more stable and avoids problems with too small or large numbers (overflow). Note that the ratio of posterior densities in the Metropolis acceptance probability can be written

$$\frac{p(\theta_p|\mathbf{y})}{p(\theta_c|\mathbf{y})} = \exp\left[\log p(\theta_p|\mathbf{y}) - \log p(\theta_c|\mathbf{y})\right]$$

This is smart since the large or small common factors in $p(\theta_p|\mathbf{y})$ and $p(\theta_c|\mathbf{y})$ cancel out before we evaluate the exponential function (which can otherwise overflow).

Second, the first argument of your (log) posterior function should be `theta`, the (vector) of parameters for which the posterior density is evaluated. You can of course use some other name for the variable, but it must be the *first* argument of the posterior density function.

Third, the user's posterior density is also a function of the data and prior hyper-parameters and those need to be supplied to the Metropolis function in some way. A nice way of doing that is to use the triple dot (...) argument which is like a wildcard for *any* parameters supplied by the user. This makes it possible to use the Metropolis function for any problem, even when you as a programmer don't know what the user's posterior density function looks like or what kind of data and hyper-parameters being used in that particular problem. To make all of this very clear (I hope!) I give a simple code below where the log posterior density for the Bernoulli model with a Beta prior is coded. That log posterior density is then used in a useless, but illustrative, function `MultiplyByTwo` that returns 2 times the log posterior density evaluated at $\theta = 0.3$. Note how the `MultiplyByTwo` takes a function object as input and how it uses the triple dot (...) argument to supply the data s and f, and the prior hyper-parameters a and b without explicitly using these symbols inside the function. That makes the `MultiplyByTwo` function applicable for *any* function.
Now, use your new Metropolis function to sample from the posterior of β in the Poisson regression for the eBay dataset. Assess MCMC convergence by graphical methods. The parameters $\phi_j = \exp(\beta_j)$ are usually considered more interpretable than the β_j. Compute the posterior distribution of ϕ_j for all variables.

(d) Use the MCMC draws from c) to simulate from the predictive distribution of the number of bidders in a new auction with the characteristics below. Plot the predictive distribution. What is the probability of no bidders in this new auction?

- **PowerSeller** = 1
- **VerifyID** = 1
- **Sealed** = 1
- **MinBlem** = 0
- **MajBlem** = 0
- **LargNeg** = 0
- **LogBook** = 1
- **MinBidShare** = 0.5

GOOD LUCK!

```
# This is the log posterior density of the beta(s+a,f+b) density
LogPostBernBeta <- function(theta, s, f, a, b){
 logPost <- (s+a-1)*log(theta) + (f+b-1)*log(1-theta)
 return(logPost)
}

# Testing if the log posterior function works
s <- 8;f <- 2;a <- 1;b <- 1
logPost <- LogPostBernBeta(theta = 0.1, s, f, a, b)
print(logPost)

# This is a rather useless function that takes the function myFunction,
# evaluates it at x = 0.3, and then returns two times the function value.
MultiplyByTwo <- function(myFunction, ...){
 x <- 0.3
 y <- myFunction(x,...)
 return(2*y)
}
#Let's try if the MultiplyByTwo function works:
MultiplyByTwo(LogPostBernBeta,s,f,a,b)
```