# Assignment #7 – Machine Learning – Professor Haugh

## **Jaime Gacitua**

## **jg3499**

## **Due Friday April 22, 2016**

## Question 1

1. We have the following distribution

It can be re-written as,

Characterizing this function as an exponential family of the form

We can identify,

1. Substituting in the expression for the conjugate prior,

In order to transform this expression to the form,

We have to create the following equivalences,

From the first equation,

From the second relationship,

From here, we can state the following equations and solutions,

## Question 2

1. Let x follow a Bernoulli distribution, with parameter.
2. If the conjugate prior for is a beta distribution, it must have the form.
3. Computing the posterior distribution, considering i.i.d realizations of the r.v.

We recognize from here that the posterior is

## Question 3

1. We start with the expression

Using the definition of, and the variance function of correlated values,

Using the definition of the autocorrelation function, over the,

We know that, and, therefore,

From the definition of,

1. If we assume that follows an AR(1) process, , and .

From the definition of, and using this particular case for

From the definition of known series,

As n grows, becomes smaller and smaller.

We will assume n is large, yielding the convenient approximation,

1. We will start on the left hand side of the relationship we want to prove, and using the identity provided in the homework,

Using (2),

Since the are i.i.d. ,

With this result, we can re-arrange the terms

If we consider as data points, we can remove the expectation, yielding an estimator for,

This is an unbiased estimator only if is an unbiased estimator for. This condition must be true because if we take expectation of the right hand side, it will yield zero only if.

1. From part (a) we demonstrated that

I will construct an unbiased estimator for, using realizations of the process (each of length).

Where,

The average of therealization, the overall average value across all the realizations.

To justify (1), we can see that we found an unbiased estimator,

Relating this estimator to (1), we can see that the expressions are in a sense equivalent,

The termis represented by

The term is represented by, but now we will gather the average variance, expanding this term to.

## Question 4

1. The implemented Gibbs sampler can be found in the appendix, in the code for question 4.
2. The Gelman-Rubin diagnostic was implemented and can be found in the appendix, in the code for question 4.

The output of the diagnostic can be found in part (c)

1. A total of chains were created, and. The output table is presented below. The statistic achieved is better than the textbook. It is interesting to see the high dispersion of. It seems to be a parameter more difficult to estimate with certainty.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Parameter | 2.5% | 25% | 50% | 75% | 97.5% | R |
| Theta1 | 56.3 | 60.0 | 61.5 | 62.8 | 66.1 | 1.00002 |
| Theta2 | 61.4 | 64.5 | 65.7 | 67.0 | 70.4 | 1.00000 |
| Theta3 | 63.2 | 66.3 | 67.5 | 68.8 | 72.4 | 1.00001 |
| Theta4 | 56.4 | 59.7 | 60.8 | 61.9 | 64.8 | 1.00001 |
| Mu | 53.4 | 62.0 | 63.9 | 65.8 | 74.5 | 1.00000 |
| Sigma | 2.3 | 2.9 | 3.3 | 3.8 | 5.9 | 1.00004 |
| Tau | 1.6 | 3.5 | 5.4 | 8.7 | 30.6 | 1.00001 |

# Appendix

## Question 4 – Code

rm(list=ls())

require(stats)

require(geoR)

#### Dataset ----

dataA = matrix(data = c(62, 60, 63, 59))

dataB = matrix(data = c(63, 67, 71, 64, 65, 66))

dataC = matrix(data = c(68, 66, 71, 67, 68, 68))

dataD = matrix(data = c(52, 62, 60, 61, 63, 64, 63, 59))

# Calculate total number of data points

n <- length(dataA) + length(dataB) + length(dataC) + length(dataD)

#### Functions ----

calc.sigma2 <- function(dataA, dataB, dataC, dataD, thetas){

# initialize

sigma2 <- 0

for(i in dataA){

sigma2 <- sigma2 + (i-thetas[1])^2

}

for(i in dataB){

sigma2 <- sigma2 + (i-thetas[2])^2

}

for(i in dataC){

sigma2 <- sigma2 + (i-thetas[3])^2

}

for(i in dataD){

sigma2 <- sigma2 + (i-thetas[4])^2

}

n <- length(dataA) + length(dataB) + length(dataC) + length(dataD)

sigma2 <- sigma2 / n

## Generate inv-chi squared to get sigma2

sigma2 <- rinvchisq(n = 1, df = n, scale = sigma2)

return(sigma2)

}

calc.tau2 <- function(thetas.temp, mu.temp){

# initialize

tau2.temp <- 0

for(j in thetas.temp){

tau2.temp <- tau2.temp + (j-mu.temp)^2

}

J <- length(thetas.temp)

tau2.temp <- tau2.temp / (J - 1)

## Generate chi squared

tau2.temp <- rinvchisq(n = 1, df = (J-1), scale = tau2.temp+0.001)

return(tau2.temp)

}

calc.theta.param <- function(data, mu, sigma2, tau2){

theta.param <- ( mu / tau2 + length(data) \* mean(data) / sigma2 ) /

(1 / tau2 + length(data) / sigma2)

return(theta.param)

}

calc.vega.param <- function(data, tau2, sigma2){

vega.param <- 1 / (1 / tau2 + length(data) / sigma2)

return(vega.param)

}

calc.thetas <- function(dataA, dataB, dataC, dataD, mu, sigma2, tau2){

theta <- mat.or.vec(4,1)

# For each theta draw from its distribution

theta.param <- calc.theta.param(dataA, mu, sigma2, tau2)

vega.param <- calc.vega.param(dataA, sigma2, tau2)

theta[1] <- rnorm(1, mean = theta.param, sd = sqrt(vega.param))

theta.param <- calc.theta.param(dataB, mu, sigma2, tau2)

vega.param <- calc.vega.param(dataB, sigma2, tau2)

theta[2] <- rnorm(1, mean = theta.param, sd = sqrt(vega.param))

theta.param <- calc.theta.param(dataC, mu, sigma2, tau2)

vega.param <- calc.vega.param(dataC, sigma2, tau2)

theta[3] <- rnorm(1, mean = theta.param, sd = sqrt(vega.param))

theta.param <- calc.theta.param(dataD, mu, sigma2, tau2)

vega.param <- calc.vega.param(dataD, sigma2, tau2)

theta[4] <- rnorm(1, mean = theta.param, sd = sqrt(vega.param))

return(theta)

}

calc.mu <- function(thetas, tau2){

mu.param <- sum(thetas) / length(thetas)

mu <- rnorm(1, mean = mu.param, sd = sqrt(tau2 / length(thetas)))

return(mu)

}

iterate <- function(thetas, mu, sigma2, tau2, dataA, dataB, dataC, dataD){

thetas <- calc.thetas(dataA = dataA, dataB = dataB, dataC = dataC,

dataD = dataD, mu = mu, sigma2 = sigma2, tau2 = tau2)

mu <- calc.mu(thetas = thetas, tau2 = tau2)

sigma2 <- calc.sigma2(dataA = dataA, dataB = dataB, dataC = dataC,

dataD = dataD, thetas = thetas)

tau2 <- calc.tau2(thetas = thetas, mu = mu)

return.list <- list("sigma2" = sigma2, "tau2" = tau2, "thetas" = thetas, "mu" = mu)

return(return.list)

}

calc.var.psi.chains <- function(psi.chains){

psi.col.means <- colMeans(psi.chains)

psi.mean <- mean(psi.chains)

n <- nrow(psi.chains)

m <- ncol(psi.chains)

B <- n / (m - 1) \* sum((psi.col.means - psi.mean)^2)

W <- 1 / m \* sum( n / (n-1) \* (colMeans(psi.chains\*psi.chains) - colMeans(psi.chains)^2))

var.psi.chains <- (n-1)/n\*W + B/n

R.convergence <- sqrt(var.psi.chains / W)

output = list("var" = var.psi.chains, "B" = B, "W" = W, "R.conv" = R.convergence)

return(output)

}

### Initialization Values ----

m <- 4 # number of chains

n <- 100000 # length of each chain

n0 <- 2\*n # = 2n, 50% of each chain is discarded

chains <- list()

iterations1 <- list()

iterations2 <- list()

psi.sigma2 <- matrix(0, nrow = n, ncol = m)

psi.tau2 <- matrix(0, nrow = n, ncol = m)

psi.mu <- matrix(0, nrow = n, ncol = m)

psi.theta1 <- matrix(0, nrow = n, ncol = m)

psi.theta2 <- matrix(0, nrow = n, ncol = m)

psi.theta3 <- matrix(0, nrow = n, ncol = m)

psi.theta4 <- matrix(0, nrow = n, ncol = m)

chain.number <- 0

### Run the chains ----

for(iter.m in 1:(m/2)){

# Initialize thetas randomly

thetas = mat.or.vec(1,4)

thetas[1] = sample(dataA,1)

thetas[2] = sample(dataB,1)

thetas[3] = sample(dataC,1)

thetas[4] = sample(dataD,1)

# Initialize mu

mu = mean(thetas)

# initial sigma2

sigma2 <- calc.sigma2(dataA, dataB, dataC, dataD, thetas)

# initial tau2

tau2 <- calc.tau2(thetas = thetas, mu = mu)

# collect first iteration parameters

parameters <- list("sigma2" = sigma2, "tau2" = tau2, "thetas" = thetas, "mu" = mu)

# iterate for a total of n0+2n times

counter <- 1

for(iter.n in 1:(n0+2\*n)){

# cat('Iter: ', iter.n, '\n')

# update all parameters

parameters <- iterate(thetas = parameters$thetas,

mu = parameters$mu,

sigma2 = parameters$sigma2,

tau2 = parameters$tau2,

dataA = dataA,

dataB = dataB,

dataC = dataC,

dataD = dataD)

## Save iterations after we have 'burned in' n0

# Save the first m iterations in one chain

if(iter.n == 1){

chain.number <- chain.number + 1

cat('Chain Number: ', chain.number, '\n')

}

if((counter>n0) && (counter<=(n0+n))){

psi.sigma2[counter - n0, chain.number] <- parameters$sigma2

psi.tau2[counter - n0, chain.number] <- parameters$tau2

psi.mu[counter - n0, chain.number] <- parameters$mu

psi.theta1[counter - n0, chain.number] <- parameters$thetas[1]

psi.theta2[counter - n0, chain.number] <- parameters$thetas[2]

psi.theta3[counter - n0, chain.number] <- parameters$thetas[3]

psi.theta4[counter - n0, chain.number] <- parameters$thetas[4]

}

# Save the next m iterations in a second chain

if(iter.n == (n0+n+1)){

chain.number <- chain.number + 1

cat('Chain Number: ', chain.number, '\n')

}

if(counter>(n0+n)){

psi.sigma2[counter - n0 - n, chain.number] <- parameters$sigma2

psi.tau2[counter - n0 - n, chain.number] <- parameters$tau2

psi.mu[counter - n0 - n, chain.number] <- parameters$mu

psi.theta1[counter - n0 - n, chain.number] <- parameters$thetas[1]

psi.theta2[counter - n0 - n, chain.number] <- parameters$thetas[2]

psi.theta3[counter - n0 - n, chain.number] <- parameters$thetas[3]

psi.theta4[counter - n0 - n, chain.number] <- parameters$thetas[4]

}

counter <- counter + 1

}

}

var.psi.theta1 <- calc.var.psi.chains(psi.theta1)

var.psi.theta2 <- calc.var.psi.chains(psi.theta2)

var.psi.theta3 <- calc.var.psi.chains(psi.theta3)

var.psi.theta4 <- calc.var.psi.chains(psi.theta4)

var.psi.sigma2 <- calc.var.psi.chains(psi.sigma2)

var.psi.tau2 <- calc.var.psi.chains(psi.tau2)

var.psi.mu <- calc.var.psi.chains(psi.mu)

var.psi.theta1$R.conv

var.psi.theta2$R.conv

var.psi.theta3$R.conv

var.psi.theta4$R.conv

var.psi.mu$R.conv

var.psi.sigma2$R.conv

var.psi.tau2$R.conv

quants <- c(0.025, 0.25, 0.5, 0.75, 0.975)

quantile(as.vector(psi.theta1, mode = 'numeric'), quants)

quantile(as.vector(psi.theta2, mode = 'numeric'), quants)

quantile(as.vector(psi.theta3, mode = 'numeric'), quants)

quantile(as.vector(psi.theta4, mode = 'numeric'), quants)

quantile(as.vector(psi.mu, mode = 'numeric'), quants)

quantile(as.vector(sqrt(psi.sigma2), mode = 'numeric'), quants)

quantile(as.vector(sqrt(psi.tau2), mode = 'numeric'), quants)