

STAT6011/7611/6111/3317  
COMPUTATIONAL STATISTICS (2016 Fall)  
Midterm Examination

Kei Ikegami (u3535947)

October 30, 2016

1

The code is below.

Listing 1: code

---

```
1  # import packages
2  from scipy.stats import invgamma
3  from scipy.stats import norm
4  from multiprocessing import Pool
5  from datetime import datetime
6  import pandas as pd
7  import numpy as np
8  import matplotlib.pyplot as plt
9  % matplotlib inline
10
11
12 # define functions
13 def integral(estimate, ts):
14     elements = np.ones(len(ts) - 1)
15     for i in range(len(ts) - 1):
16         elements[i] = (ts[i+1] - ts[i])*(estimate[i+1] + estimate[i])/2
17     return sum(elements)
18
19 def sum1(beta):
20     return np.sum(p[:, 0] - beta * p[:, 3])
21
22 def sum2(alpha):
23     return np.sum(p[:, 3] * (p[:, 0] - alpha))
24
25 def sum3(alpha, beta):
26     l = (p[:, 0] - alpha - beta*p[:, 3])
27     return np.sum(l * l)
28
29 def loglike(alpha, beta, sigma):
30     return N*np.log(1/(np.sqrt(2*np.pi*sigma))) - sum3(alpha, beta) / (2*sigma)
31
32 def function1(w):
33
34     if w < m:
35         np.random.seed(datetime.now().microsecond * np.random.randint(20))
36
37         for i in range(n+1):
38             t = ts[i]
39
40             if i == 0:
41                 alphas[0] = 3000
42                 betas[0] = 185
43                 sigmas[0] = 90000
44
45             else:
46                 alphas[0] = np.mean(alpha_sample)
47                 betas[0] = np.mean(beta_sample)
48                 sigmas[0] = np.mean(sigma_sample)
49
```

```

50     for j in range(sample_iter - 1):
51
52         location_alpha = (sigma_alpha*t*sum1(betas[j]) + sigmas[j]*mu_alpha) / (sigma_alpha * N*t + sigmas[
53             j])
54         scale_alpha = np.sqrt((sigma_alpha * sigmas[j]) / (sigma_alpha * N*t + sigmas[j]))
55         alphas[j+1] = norm.rvs(loc = location_alpha, scale = scale_alpha)
56
57         location_beta = (sigma_beta * t * sum2(alphas[j+1]) + sigmas[j] * mu_beta) / (sigma_beta *t* ssx +
58             sigmas[j])
59         scale_beta = np.sqrt((sigmas[j] * sigma_beta) / (sigma_beta *t* ssx + sigmas[j]))
60         betas[j+1] = norm.rvs(loc = location_beta, scale = scale_beta)
61
62         shape = N*t/2 + a
63         invrate = 2*b / (b*t*sum3(alphas[j+1], betas[j+1]) + 2)
64         sigmas[j+1] = invgamma.rvs(a = shape, scale = 1/ invrate)
65
66         alpha_sample = alphas[burn_in:]
67         beta_sample = betas[burn_in:len(betas)]
68         sigma_sample = sigmas[burn_in:len(sigmas)]
69
70         box = np.ones(len(alpha_sample))
71         for k, l in enumerate(alpha_sample):
72             box[k] = loglike(l, beta_sample[k], sigma_sample[k])
73
74         estimates[i] = np.average(box)
75
76     return estimates
77
78 def sum4(beta):
79     return np.sum(p[:, 0] - beta * p[:, 4])
80
81 def sum5(alpha):
82     return np.sum(p[:, 4] * (p[:, 0] - alpha))
83
84 def sum6(alpha, beta):
85     l = (p[:, 0] - alpha - beta*p[:, 4])
86     return np.sum(l * l)
87
88 def loglike2(alpha, beta, sigma):
89     return N*np.log(1/(np.sqrt(2*np.pi*sigma))) - sum6(alpha, beta) / (2*sigma)
90
91 def function2(w):
92     if w < m:
93         np.random.seed(datetime.now().microsecond * np.random.randint(20))
94
95     for i in range(n+1):
96         t = ts[i]
97
98         if i == 0:
99             gammas[0] = 3000
100             deltas[0] = 185
101             taus[0] = 90000
102
103         else:
104             gammas[0] = np.mean(gamma_sample)
105             deltas[0] = np.mean(delta_sample)
106             taus[0] = np.mean(tau_sample)
107
108     for j in range(sample_iter - 1):
109
110         location_alpha = (sigma_alpha*t*sum4(deltas[j]) + taus[j]*mu_alpha) / (sigma_alpha * N*t + taus[j])
111         scale_alpha = np.sqrt((sigma_alpha * taus[j]) / (sigma_alpha * N*t + taus[j]))
112         gammas[j+1] = norm.rvs(loc = location_alpha, scale = scale_alpha)
113
114         location_beta = (sigma_beta * t * sum5(gammas[j+1]) + taus[j] * mu_beta) / (sigma_beta *t* ssz +
115             taus[j])
116         scale_beta = np.sqrt((taus[j] * sigma_beta) / (sigma_beta *t* ssz + taus[j]))
117         deltas[j+1] = norm.rvs(loc = location_beta, scale = scale_beta)
118
119         shape = N*t/2 + a
120         invrate = 2*b / (b*t*sum6(gammas[j+1], deltas[j+1]) + 2)
121         taus[j+1] = invgamma.rvs(a = shape, scale = 1/ invrate)
122
123     gamma_sample = gammas[burn_in:]

```

```

121         delta_sample = deltas[burn_in:len(deltas)]
122         tau_sample = taus[burn_in:len(taus)]
123
124         box2 = np.ones(len(gamma_sample))
125         for k, l in enumerate(gamma_sample):
126             box2[k] = loglike2(l, delta_sample[k], tau_sample[k])
127
128         estimates[i] = np.average(box2)
129
130     return estimates
131
132
133     # setting
134     pine = pd.read_table("pine.txt", delim_whitespace = True)
135     p = pine.values
136     pine['ave_x'] = pine['x'] - np.average(p[:, 1])
137     pine['ave_z'] = pine['z'] - np.average(p[:, 2])
138     p = pine.values
139
140     mu_alpha = 3000
141     sigma_alpha = 10**6
142     mu_beta = 185
143     sigma_beta = 10**4
144     a = 3
145     b = 1/(2*300**2)
146
147     N = np.shape(p)[0]
148     ssx = np.sum(p[:, 3] * p[:, 3])
149     ssz = np.sum(p[:, 4] * p[:, 4])
150
151     n = 10
152     c = 2
153     ts = [(i/n)**c for i in range(n+1)]
154     estimates = np.ones(n+1)
155
156
157     sample_iter = 100000
158     burn_in = 30000
159     m = 100
160     core = 4
161
162     alphas = np.ones(sample_iter)
163     betas = np.ones(sample_iter)
164     sigmas = np.ones(sample_iter)
165
166     gammas = np.ones(sample_iter)
167     deltas = np.ones(sample_iter)
168     taus = np.ones(sample_iter)
169
170
171     # model 1's marginal likelihood
172     if __name__ == '__main__':
173         w = Pool(core)
174         result1 = w.map(function1, range(m))
175
176     expect1 = np.ones(10)
177     for i in range(10):
178         expect1[i] = integral(result1[i], ts)
179
180
181     # model 2's marginal likelihood
182     if __name__ == '__main__':
183         result2 = w.map(function2, range(m))
184
185     expect2 = np.ones(10)
186     for i in range(10):
187         expect2[i] = integral(result2[i], ts)
188
189
190     # computing BF_21
191     bf_21 = []
192     for a,b in zip(expect1, expect2):
193         bf_21.append(np.exp(b-a))
194
195     # the upper left cell in Table1

```

196 bias = np.average(bf.21) - 4862  
 197 std = np.sqrt(np.var(bf.21))

The result is .

The main idea of this paper is the below identity.

$$\begin{aligned}
 \log \{p(\mathbf{y})\} &= \log \left\{ \frac{z(\mathbf{y}|t=1)}{z(\mathbf{y}|t=0)} \right\} = [\log \{z(\mathbf{y}|t)\}]_0^1 = \int_0^1 \frac{1}{z(\mathbf{y}|t)} \frac{d}{dt} z(\mathbf{y}|t) dt \\
 &= \int_0^1 \frac{1}{z(\mathbf{y}|t)} \left( \int_{\theta} \frac{d}{dt} p(\mathbf{y}|\theta)^t p(\theta) \right) dt = \int_0^1 \frac{1}{z(\mathbf{y}|t)} \left( \int_{\theta} \log \{p(\mathbf{y}|\theta)\} p(\mathbf{y}|\theta)^t p(\theta) d\theta \right) dt \\
 &= \int_0^1 \int_{\theta} \log \{p(\mathbf{y}|\theta)\} \frac{p(\mathbf{y}|\theta)^t p(\theta)}{z(\mathbf{y}|t)} d\theta dt = \int_0^1 \int_{\theta} \log \{p(\mathbf{y}|\theta)\} p_t(\theta|\mathbf{y}) d\theta dt \\
 &= \int_0^1 E_{\theta|\mathbf{y},t} [\log \{p(\mathbf{y}|\theta)\}] dt
 \end{aligned}$$

This identity implies that we can approximate the the marginal loglikelihood specific to a model by a numerical integration of the expectation of a loglikelihood fixed at some t. This expectation is gotten by MCMC method, i.e. Gibbs sampling in this example. Given each prior, the full conditional distribution of each parameter is as follows.

$$\begin{aligned}
 p(\alpha|\mathbf{y}, \mathbf{x}, t, \beta, \sigma^2) &\propto p(\mathbf{y}|\alpha, \beta, \sigma^2)^t p(\alpha) \\
 &\propto \exp \left( -\frac{1}{2\sigma^2} \left( t \sum_i (y_i - \beta(x_i - \bar{x}))^2 - 2t\alpha \sum_i (y_i - \beta(x_i - \bar{x})) + \alpha^2 Nt \right) + \frac{1}{2\sigma_{\alpha}^2} (\alpha^2 - 2\mu_{\alpha} \alpha + \mu_{\alpha}^2) \right) \\
 &\propto \exp \left( -\frac{\sigma_{\alpha}^2 Nt + \sigma^2}{\sigma_{\alpha}^2 \sigma^2} \left( \alpha - \frac{\sigma_{\alpha}^2 t \sum_i (y_i - \beta(x_i - \bar{x})) + \sigma^2 \mu_{\alpha}}{\sigma_{\alpha}^2 Nt + \sigma^2} \right)^2 \right) \\
 p(\beta|\mathbf{y}, \mathbf{x}, t, \alpha, \sigma^2) &\propto p(\mathbf{y}|\alpha, \beta, \sigma^2)^t p(\beta) \\
 &\propto \exp \left( -\left( \frac{1}{2\sigma^2} t \sum_i (y_i - \alpha - \beta(x_i - \bar{x}))^2 + \frac{1}{2\sigma_{\beta}^2} (\beta - \mu_{\beta})^2 \right) \right) \\
 &\propto \exp \left( -\left( \frac{\sigma_{\beta}^2 t \sum_i (x_i - \bar{x})^2 + \sigma^2}{2\sigma^2 \sigma_{\beta}^2} \beta^2 - \frac{\sigma_{\beta}^2 t \sum_i (x_i - \bar{x})(y_i - \alpha) + \sigma^2 \mu_{\beta}}{\sigma^2 \sigma_{\beta}^2} \beta \right) \right) \\
 &\propto \exp \left( -\frac{\sigma_{\beta}^2 t \sum_i (x_i - \bar{x})^2 + \sigma^2}{2\sigma^2 \sigma_{\beta}^2} \left( \beta - \frac{\sigma_{\beta}^2 t \sum_i (x_i - \bar{x})(y_i - \alpha) + \sigma^2 \mu_{\beta}}{\sigma_{\beta}^2 t \sum_i (x_i - \bar{x})^2 + \sigma^2} \right)^2 \right) \\
 p(\sigma^2|\mathbf{y}, \mathbf{x}, t, \alpha, \beta) &\propto p(\mathbf{y}|\alpha, \beta, \sigma^2)^t p(\sigma^2) \\
 &= \left\{ \prod_{i=1}^N \left( \frac{1}{\sqrt{2\pi\sigma^2}} \exp \left( -\frac{(y_i - \alpha - \beta(x_i - \bar{x}))^2}{2\sigma^2} \right) \right)^t \right\} \exp \left( -\frac{1}{b\sigma^2} \right) \frac{1}{\gamma(a)} b^{-a} (\sigma^2)^{-(a+1)} \\
 &\propto (\sigma^2)^{-\frac{Nt}{2} + a + 1} \exp \left( -\frac{bt \sum_i (y_i - \alpha - \beta(x_i - \bar{x}))^2 + 2}{2b\sigma^2} \right)
 \end{aligned}$$

By the above,  $\alpha$ 's full conditional distribution is  $N \left( \frac{\sigma_{\alpha}^2 t \sum_i (y_i - \beta(x_i - \bar{x})) + \sigma^2 \mu_{\alpha}}{\sigma_{\alpha}^2 Nt + \sigma^2}, \frac{\sigma^2 \sigma_{\alpha}^2}{\sigma_{\alpha}^2 Nt + \sigma^2} \right)$ .  $\beta$ 's full conditional distribution is  $N \left( \frac{\sigma_{\beta}^2 t \sum_i (x_i - \bar{x})(y_i - \alpha) + \sigma^2 \mu_{\beta}}{\sigma_{\beta}^2 t \sum_i (x_i - \bar{x})^2 + \sigma^2}, \frac{\sigma^2 \sigma_{\beta}^2}{\sigma_{\beta}^2 t \sum_i (x_i - \bar{x})^2 + \sigma^2} \right)$ .  $\sigma^2$  is  $IG \left( \frac{Nt}{2} + a, \frac{2b}{bt \sum_i (y_i - \alpha - \beta(x_i - \bar{x}))^2 + 2} \right)$ .

The above code took about 4 hours. This is due to the long iteration of Gibbs sampling, and I check the MCMC converge very fast in this case. Then I wonder why the authors take such a long chain. If I use some efficient MCMC packages or write the more matrix based code, they must shorten the time. It is, however, better to write a readable code because this is just an assignment.

Anyway, the computation of marginal likelihood is so hard that a lot of tools and methods have been invented. In this example, the number of regression parameters are just three including the precision for each model. Then the model selection problem in high dimension must be a terrible and challenging task.