finite_mixture.R

wnarifin

Thu Aug 9 01:10:01 2018

```
# R in Medicine Meetup
# 20180809
# Wan Nor Arifin
```

Finite mixture

```
# A good introduction can be found here:
# http://www.mas.ncl.ac.uk/~nmf16/teaching/mas8391/slides8303-4.pdf
# by Malcolm Farrow
```

About

```
# - Let say we have a continuous variable e.g. height that comes from
#   two groups of people. And we have no data on the grouping. Can we extract
#   the mean height for each groups?
# - Finite mixture model (FMM) is a statistical modeling approach to do that.
# - Usually applied to continuous data.
# - In medicine, it can be used to come up with subgroups of disease.
#   e.g. severity of depression etc.
# - Latent class analysis (LCA) is the statistical analog of FMM for categorical
#   data.
```

Library

```
library(rjags)  # must also install jags in your computer before rjags
```

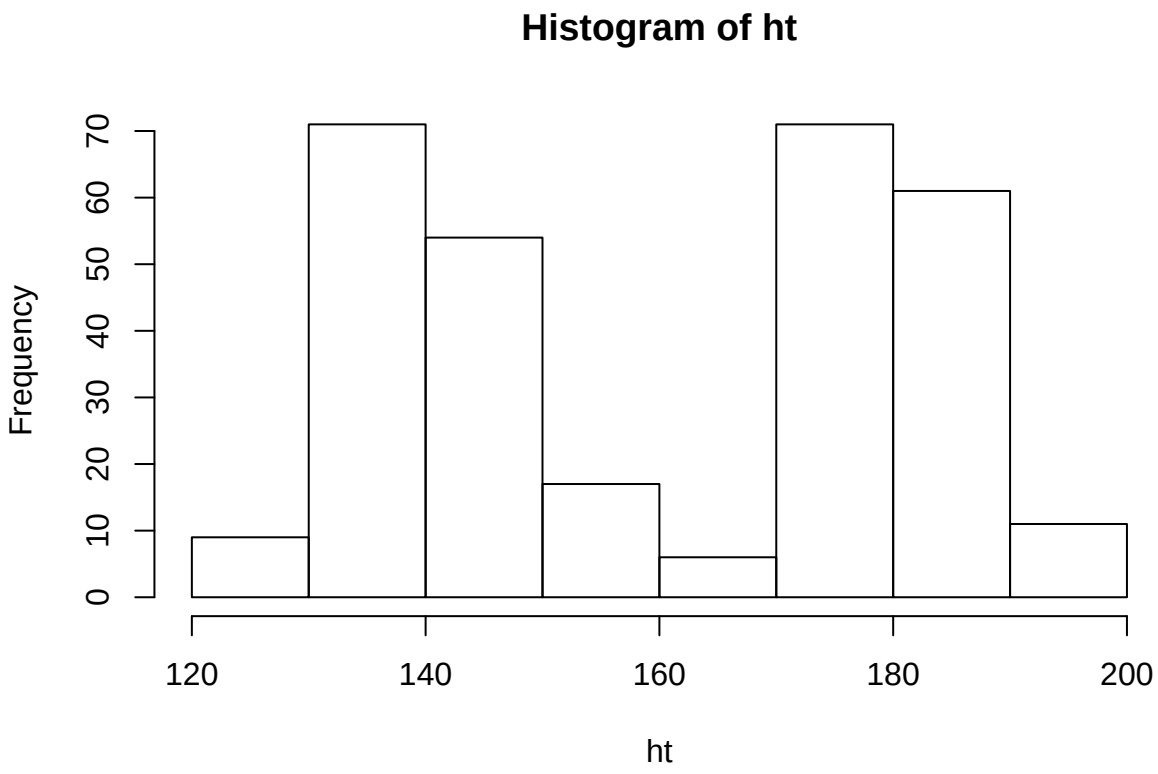
```
## Loading required package: coda
```

```
## Linked to JAGS 4.3.0
```

```
## Loaded modules: basemod,bugs
```

Data

```
# Generate data, say height in cm
# - male = N(mean = 180, SD = 7)
# - female = N(mean = 140, SD = 7)
ht = c(rnorm(150, 180, 7), rnorm(150, 140, 7))
hist(ht)
```




```
jagsdata = list(y = ht, n = length(ht))  # jagsdata must be in list
jagsdata
```

```
## $y
##   [1] 187.9801 179.5344 173.6630 179.5610 194.7649 175.4724 183.3635
##   [8] 181.9980 185.9846 178.5274 174.9786 184.4468 179.1233 187.6272
##  [15] 175.7303 186.9208 176.2030 175.7777 182.6723 182.7194 188.9642
##  [22] 186.4649 178.7951 187.9631 180.4248 180.8432 178.9852 176.9083
##  [29] 179.5499 185.8686 172.6834 176.4804 170.9970 180.5032 184.2338
##  [36] 169.2551 174.4468 173.2643 185.7570 171.2333 184.2945 179.0492
##  [43] 169.5589 189.9524 172.5411 170.6425 184.3090 187.2868 172.7926
##  [50] 184.7690 184.2780 179.7848 180.4047 181.8985 178.9059 188.4654
##  [57] 182.4136 182.4603 182.7380 172.2179 187.9481 176.0188 185.7216
##  [64] 179.9553 175.3605 177.7315 176.4349 189.5832 170.7340 184.8550
##  [71] 174.7332 183.9090 184.0322 186.4006 187.9281 174.1510 183.1360
##  [78] 174.7516 179.8061 178.7326 180.9624 175.0438 187.8006 172.5781
##  [85] 169.3368 176.3915 177.5336 178.3410 190.9609 182.8172 169.8879
##  [92] 172.0410 172.9857 178.2121 173.8908 177.5016 187.9252 176.1709
##  [99] 175.9139 177.7953 179.0104 190.4783 195.4615 178.4251 170.7173
## [106] 190.9149 154.5967 189.9206 172.0945 180.9869 172.8958 176.2980
## [113] 177.7759 180.0254 182.3917 190.7606 189.3420 178.4186 172.6412
## [120] 171.7959 181.0017 189.2966 192.6979 180.8075 195.6410 169.4223
## [127] 183.8429 176.1146 183.9519 182.9355 179.9955 188.6836 191.3429
## [134] 178.4600 174.9950 189.3618 178.7299 178.6129 183.0972 163.8359
## [141] 193.6267 173.9423 175.3842 185.3476 174.6944 196.0981 183.2992
## [148] 178.3389 181.2192 188.5961 143.5645 135.8487 146.4256 140.7066
```

```
## [155] 151.7071 139.5292 135.0694 147.2219 135.9902 144.4030 136.3335
## [162] 131.6753 138.1338 132.3022 137.8214 147.2259 143.2911 152.8547
## [169] 141.7362 141.5953 132.7211 143.1806 133.4260 148.4406 156.5447
## [176] 130.0494 143.8702 154.9471 132.5076 149.4078 128.2357 144.5508
## [183] 146.3535 152.6761 145.5641 140.4483 135.8352 128.8589 158.8111
## [190] 138.1826 136.8327 130.9499 149.3204 150.1447 142.7021 137.6660
## [197] 139.5103 139.5564 135.6449 136.2971 136.5364 150.5705 138.6079
## [204] 133.9000 136.7560 139.5498 132.8757 134.7431 144.2176 138.4482
## [211] 144.5694 137.7523 141.8737 143.6217 129.7053 151.2550 146.1428
## [218] 134.2338 139.9686 139.5207 141.7402 143.3264 139.0749 135.5524
## [225] 137.4557 143.6149 144.8074 136.4749 148.3828 145.0877 146.9717
## [232] 158.7471 130.9168 140.1953 122.1353 139.3849 143.5003 136.1236
## [239] 139.0770 136.6726 139.6417 141.1637 155.5301 137.9144 141.5449
## [246] 140.5486 145.3702 141.8654 140.9139 150.6771 133.0058 153.8336
## [253] 144.2923 141.5553 135.4609 133.6810 132.0150 137.5847 142.3173
## [260] 134.0637 143.8433 138.6847 132.0522 143.7547 137.5190 133.8500
## [267] 133.5313 141.1851 123.4936 133.6462 148.6449 132.6947 132.4196
## [274] 134.2757 141.9830 130.9075 138.5483 156.1880 144.6793 133.5734
## [281] 138.0828 128.0330 157.8212 151.0785 142.3179 129.3662 142.0847
## [288] 135.1320 143.3771 143.9313 136.6793 135.8942 140.5858 130.8232
## [295] 149.6126 133.4104 128.1848 124.0339 133.4951 137.7996
##
## $n
## [1] 300
```

FMM

Finite mixture model,

$$p(\boldsymbol{y}_i|\boldsymbol{\vartheta}) = \sum_{k=1}^{K} \eta_k p(\boldsymbol{y}_i|\boldsymbol{\theta}_k)$$

where $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K$ = component parameters, $\boldsymbol{\eta} = (\eta_1, \ldots, \eta_K)$, $\boldsymbol{\vartheta} = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K, \boldsymbol{\eta})$ = mixture model parameters.

```
# Aim of FMM: Can we "reverse engineer" the process? i.e we obtain the male/female grouping?
#
# Suppose we assume:
# 1. two groups, k = 2
# 2. prior mean = 160, SD = 7, var = 7^2 = 49; precision, tau = 1/var = 1/49
#    mu ~ N(160, 49); in jags mu ~ N(mean, precision) ~ N(160, 1/49)
# 3. prior gamma, mean precision = 1/49,
#    shape, a = 3; rate, b = a/mean = 3/(1/49) = 147
#    tau ~ gamma(3, 147)
# 4. prior beta, pi ~ beta(3, 3) -- gives 50:50 ratio between the groups
```

JAGS code

```
writeLines("
model {
  # data
  for (i in 1:n) {
    c[i] ~ dcat(p[1:2])
```

```
    y[i] ~ dnorm(mu[c[i]], tau[c[i]])
  }

  # tau
  for (j in  1:2) {
    tau[j] ~ dgamma(3, 147) # precision, tau
    sigma[j] = sqrt(1/tau[j]) # transform back to SD
  }

  # mu
  mu0 ~ dnorm(160, 1/50) # hierarchical mean, mu
  for (j in 1:2) {
    mu_mu0[j] ~ dnorm(mu0, 1/50) # sample mu
  }
  mu[1:2] = sort(mu_mu0)  # set order constraint, mu1 < mu2

  # pi
  pi ~ dbeta(3, 3)
  p[1] = pi # weight for grp 1
  p[2] = 1 - pi # weight for grp 2
}
", con = "finmix.bug")
```

Fit JAGS model

```
jagsmodel = jags.model("finmix.bug", jagsdata, n.chains = 2)
```

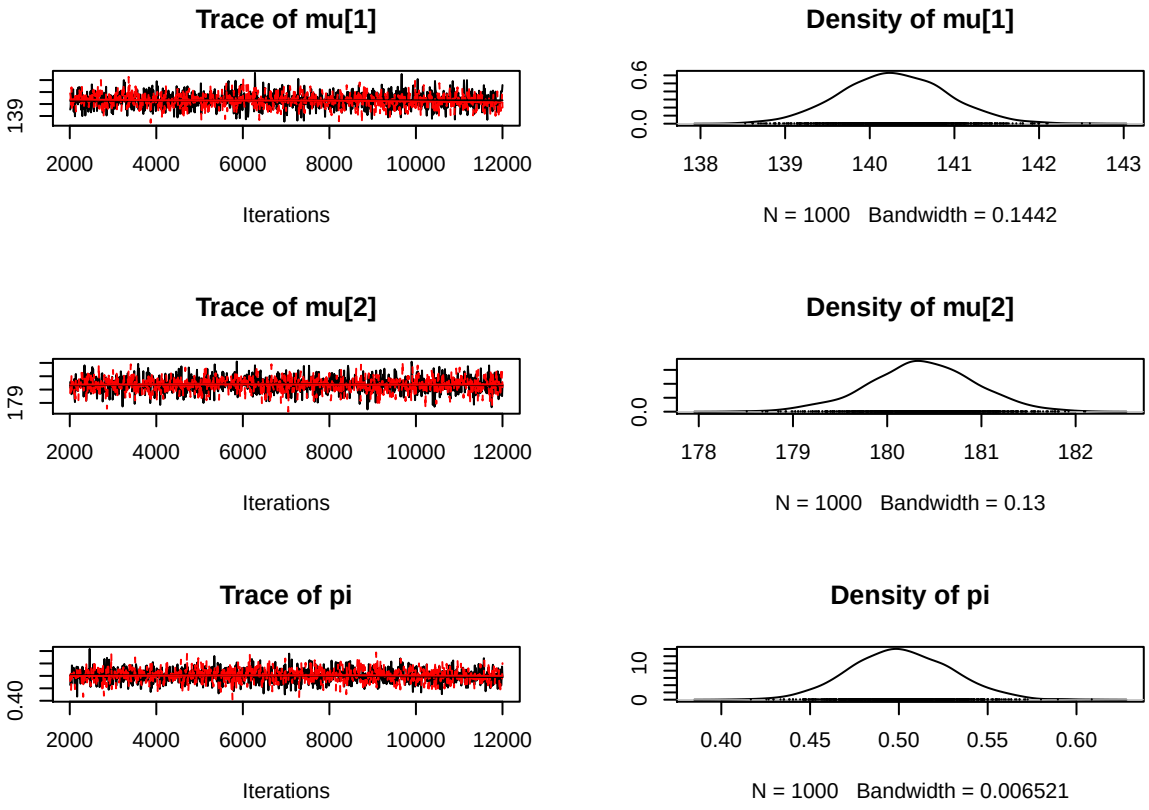
```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 300
##    Unobserved stochastic nodes: 306
##    Total graph size: 1223
##
## Initializing model
```

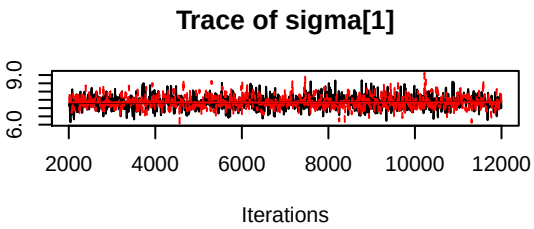
```
update(jagsmodel, 1000)  # burnin = 1000
jagspars = c("mu", "sigma", "pi")
jagssamples = coda.samples(jagsmodel, jagspars, n.iter = 10000, thin = 10)
summary(jagssamples)
```

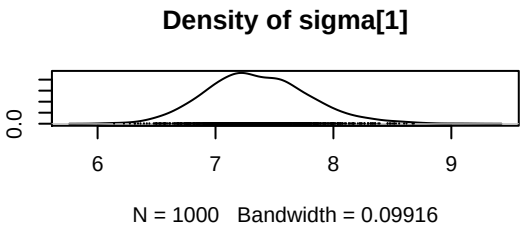
```
##
## Iterations = 2010:12000
## Thinning interval = 10
## Number of chains = 2
## Sample size per chain = 1000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##              Mean      SD  Naive SE Time-series SE
## mu[1]    140.2867 0.62205 0.0139094      0.0139094
```

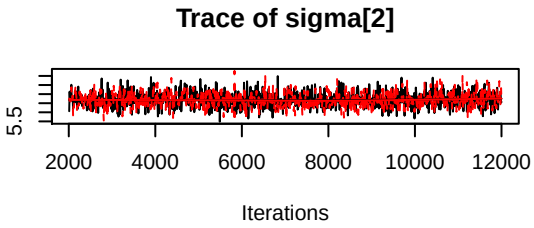
```
## mu[2]    180.3696 0.56706 0.0126797      0.0126824
## pi         0.5012 0.02813 0.0006291      0.0006088
## sigma[1]   7.3596 0.43174 0.0096539      0.0099450
## sigma[2]   6.6845 0.41979 0.0093868      0.0093884
##
## 2. Quantiles for each variable:
##
##              2.5%      25%      50%      75%    97.5%
## mu[1]    139.0780 139.8502 140.2726 140.7087 141.5187
## mu[2]    179.1871 179.9978 180.3685 180.7495 181.4671
## pi         0.4472   0.4819   0.5004   0.5209   0.5568
## sigma[1]   6.5679   7.0568   7.3250   7.6301   8.2933
## sigma[2]   5.9095   6.3950   6.6663   6.9490   7.5475
```

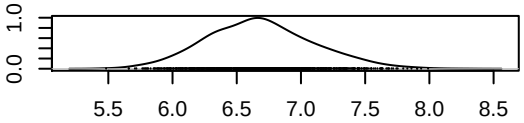
```
plot(jagssamples)
```




Trace of sigma[1]
6.0
9.0
2000
4000
6000
8000
10000
12000
Iterations


Density of sigma[1]
0.0
6
7
8
9
N = 1000 Bandwidth = 0.09916


Trace of sigma[2]
5.5
2000
4000
6000
8000
10000
12000
Iterations


Density of sigma[2]
0.0
1.0
5.5
6.0
6.5
7.0
7.5
8.0
8.5
N = 1000 Bandwidth = 0.09584