BDA-Assignment 9

Anonymous

Set up libraries and data:

```
library(loo)
```

```
## This is loo version 2.4.1

## - Online documentation and vignettes at mc-stan.org/loo

## - As of v2.0.0 loo defaults to 1 core but we recommend using as many as possible. Use the 'cores' arg
```

```
library(rstan)
```

```
## Loading required package: StanHeaders

## Loading required package: ggplot2

## rstan (Version 2.21.2, GitRev: 2e1f913d3ca3)

## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)
```

```
library(aaltobda)
data("factory")
data = list(y = factory, N = nrow(factory), M = ncol(factory))
data
```

```
## $y
##   V1  V2  V3  V4  V5  V6
## 1 83 117 101 105  79  57
## 2 92 109  93 119  97  92
## 3 92 114  92 116 103 104
## 4 46 104  86 102  79  77
## 5 67  87  67 116  92 100
##
## $N
## [1] 5
##
## $M
## [1] 6
```

From the last assignment, the hierarchical model is the model with the highest PSIS-LOO value. Therefore, hierarchical is the best model for this dataset.
Hierarchical model:

```
data {
  int<lower=0> N; // number of data points
  int<lower=0> M; // number of machines
  vector[M] y[N];
}
parameters {
  real mu_0;
  vector[M] mu;
  real sigma_0;
  real sigma;
}
model {
  mu_0 ~ normal(0, 100);
  sigma_0 ~ inv_chi_square(0.1);
  sigma ~ inv_chi_square(0.1);
  for (i in 1:M){
    mu[i] ~ normal(mu_0, sigma_0);
  }

  for (i in 1:M) {
    y[, i] ~ normal(mu[i], sigma);
  }
}
generated quantities {
  real mu7 = normal_rng(mu_0, sigma_0);
  vector[M+1] ypred;
  for (i in 1:(M)) {
      ypred[i] = normal_rng(mu[i], sigma);
  }
  ypred[M+1] = normal_rng(mu7, sigma);
}
```

Fit the model:

```r
hierarchical = stan(file = "hierarchical.stan", data = data, refresh = 0)
```

```
## Warning in readLines(file, warn = TRUE): incomplete final line found on '/Users/
## nguyenlinh/Macadamia/BDA/hierarchical.stan'

## Warning: There were 237 divergent transitions after warmup. See
## http://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
## to find out why this is a problem and how to eliminate them.

## Warning: Examine the pairs() plot to diagnose sampling problems

## Warning: Bulk Effective Samples Size (ESS) is too low, indicating posterior means and medians may be
## Running the chains for more iterations may help. See
## http://mc-stan.org/misc/warnings.html#bulk-ess
```

```
## Warning: Tail Effective Samples Size (ESS) is too low, indicating posterior variances and tail quant
## Running the chains for more iterations may help. See
## http://mc-stan.org/misc/warnings.html#tail-ess
```

```
print(hierarchical)
```

```
## Inference for Stan model: 7d569f233ef55a22b0242906be2c3a8e.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##             mean se_mean    sd    2.5%     25%     50%     75%   97.5% n_eff
## mu_0       92.50    0.16  6.40   79.24   89.10   92.67   96.15  104.88  1673
## mu[1]      81.43    0.31  6.74   68.44   76.71   81.38   86.04   94.56   484
## mu[2]     101.76    0.26  6.72   88.91   97.14  101.78  106.31  115.06   666
## mu[3]      89.42    0.13  5.91   76.96   85.61   89.66   93.32  100.74  2054
## mu[4]     105.57    0.34  7.19   91.10  100.75  105.66  110.45  119.24   455
## mu[5]      90.86    0.12  5.62   79.59   87.28   91.06   94.60  101.42  2331
## mu[6]      87.99    0.13  5.77   76.23   84.24   88.22   91.83   98.88  1893
## sigma_0    12.48    0.34  7.18    2.54    7.88   11.12   15.40   30.59   453
## sigma      15.23    0.10  2.47   11.31   13.45   14.91   16.65   20.81   651
## mu7        92.99    0.28 16.33   61.28   85.23   92.83  100.51  128.11  3358
## ypred[1]   81.73    0.41 16.89   48.98   70.46   81.40   92.61  116.13  1715
## ypred[2]  101.72    0.32 16.77   67.25   90.85  102.05  112.72  134.80  2776
## ypred[3]   89.47    0.26 16.12   59.04   78.58   88.85  100.00  122.26  3737
## ypred[4]  105.19    0.39 17.22   69.62   94.19  105.51  116.47  137.79  1936
## ypred[5]   90.85    0.26 16.57   56.82   79.60   91.01  101.69  123.28  3936
## ypred[6]   87.59    0.27 16.29   54.56   77.40   87.39   98.32  119.38  3644
## ypred[7]   92.91    0.37 22.43   50.68   78.90   92.82  106.88  137.38  3769
## lp__     -119.21    0.15  2.96 -126.06 -120.88 -118.88 -117.21 -114.40   411
##          Rhat
## mu_0     1.00
## mu[1]    1.01
## mu[2]    1.01
## mu[3]    1.00
## mu[4]    1.01
## mu[5]    1.00
## mu[6]    1.00
## sigma_0  1.01
## sigma    1.01
## mu7      1.00
## ypred[1] 1.00
## ypred[2] 1.00
## ypred[3] 1.00
## ypred[4] 1.00
## ypred[5] 1.00
## ypred[6] 1.00
## ypred[7] 1.00
## lp__     1.01
##
## Samples were drawn using NUTS(diag_e) at Wed Feb  9 15:42:56 2022.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

1/

Extract the ypred values:

```
y_pred = extract(hierarchical)$ypred
```

Implement utility:

```
utility = function(draws = y_pred) {
  result = 0
  for (i in 1:length(draws)) {
    if (draws[i] < 85) {
      result = result - 106
    }
    else {
      result = result + (200-106)
    }
  }
  result = result/length(draws)
  return(result)
}
utility(draws = c(123.80, 85.23, 70.16, 80.57, 84.91))
```

```
## [1] -26
```

Calculate expected utility of each machine:

```
utility_list = c(1:7)
for (i in 1: ncol(y_pred)) {
    draw = y_pred[, i]
    expected_utility = utility(draws = draw)
    utility_list[i] = expected_utility
    cat("\nMachine ", i, " expected utility: ", expected_utility)
}
```

```
##
## Machine  1  expected utility:  -23.8
## Machine  2  expected utility:  64.1
## Machine  3  expected utility:  14.5
## Machine  4  expected utility:  70.65
## Machine  5  expected utility:  22.95
## Machine  6  expected utility:  6.25
## Machine  7  expected utility:  24.6
```

```
utility_dict = vector(mode="list", length=7)
names(utility_dict) = utility_list
for (i in 1:7) {
  utility_dict[[i]] = c("1", "2", "3", "4", "5", "6", "7")[i]
}
```

2/

Rank the machines based on the expected utilities from worst to best:

```
sorted = sort(names(utility_dict))
for (i in 1:7) {
    machine = utility_dict[sorted[i]]
    value = names(machine)
    name = machine[[value]]
    cat("\nMachine ", name, " expected utility: ", value)
}
```

```
##
## Machine  1  expected utility:  -23.8
## Machine  3  expected utility:  14.5
## Machine  5  expected utility:  22.95
## Machine  7  expected utility:  24.6
## Machine  6  expected utility:  6.25
## Machine  2  expected utility:  64.1
## Machine  4  expected utility:  70.65
```

The utility value of each machine can be interpreted as profit that each machine makes. Machine 1 utility value is negative which means loss while other machine makes profit.

3/

7th machine expected utility:

```
cat("\nMachine 7 expected utility: ", utility_list[7])
```

```
##
## Machine 7 expected utility:  24.6
```

4/

The 7th machine utility value is positive which mean it is profitable to buy it.