BDA - Assignment 8

Anonymous

Contents

Model assessment: LOO-CV for factory data with Stan (6p)

In this exercise we are using LOO-CV to assess the predictive performance of the pooled, separate and hierarchical models which were implemented last week in assignment 7.

Read data.

```
library(aaltobda)
library("ggplot2")
library(rstan)
```

```
## Loading required package: StanHeaders
```

```
## 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)
```

```
data("factory")
rstan_options(auto_write = TRUE)
set.seed(123)
```

1. Fit the models with Stan as instructed in Assignment 7.

Here, it is enough to copy to models from assignment 7. I'll also add how I ended up choosing the weakly informative priors for the models.

Weakly informative priors

In the Stan github page (https://github.com/stan-dev/stan/wiki/Prior-Choice-Recommendations) it says that a good general weakly informative prior is normal distribution with mean equal to data mean and standard deviation equal to deviation between the column means:

```
mu_prior_mu <- mean(colMeans(factory))
mu_prior_sigma <- sd(colMeans(factory))

round(mu_prior_mu, 1)
```

```
## [1] 92.9
```

```
round(mu_prior_sigma, 1)
```

```
## [1] 13.4
```

On the other hand, on the course book BDA3, on page 55 it says:

"We characterize a prior distribution as weakly informative if it is proper but is set up so that the information it does provide is intentionally weaker than whatever actual prior knowledge is available."

It is also written on the same page that:

"Rather than trying to model complete ignorance, we prefer in most problems to use weakly informative prior distributions that include a small amount of real-world information, enough to ensure that the posterior distribution makes sense."

Based on this, we could choose a weakly informative prior to be e.g. $\mu_j \sim N(100, 50)$. Here the data mean is close to the actual value, but the second parameter is set, on purpose, higher than we know based on data it is. This way we have enough information to make sure that our inferences are contrained to be reasonable. We started from strongly informative data prior and broadened it to account for uncertainty in our prior beliefs and in the applicability of any historically based prior on new data. (BDA3, p. 55-56)

With this prior the resulting posterior will most likely make sense, but the prior isn't too informative.

Weakly informative prior for σ_j parameter needs to be decided bit differently. On BDA3 p. 130 it says that for variance parameters we should consider the t family of distributions (actually, the half-t, since the scale parameter τ is constrained to be positive) as an alternative class that includes normal and Cauchy as edge cases. For our purposes, it is enough to recognize that the half-Cauchy can be a convenient weakly informative family; the distribution has a broad peak at zero and a single scale parameter, which we shall label A to indicate that it could be set to some large value. It also says in the book that we shall consider half-Cauchy models for variance parameters which are estimated from a small number of groups (so thatinferences are sensitive to the choice of weakly informative prior distribution). In our case the data has only 5 groups so based on this half-Cauchy could be reasonable choice.

Later on pages 131 and 132 of the BDA3 it is also shown via example, that by choosing scale parameter value of the half-Cauchy distribution correctly, a good posterior is achieved and the whole model will perform well. The scale parameter value should be chosen to be a bit higher than we expect for the standard deviation of the underlying data, so that the model will be constrained only weakly. Based on this we choose our weakly informative prior for σ_j to be half-Cauchy with scale parameter 40 because we expect our data to deviate approximately 25 from data mean and we on purpose choose higher value than that.

Models

Separate model

In the separate model, each machine has its own model.

$$y_{ij} \sim N(\mu_j, \sigma_j)$$
$$\mu_j \sim N(100, 50)$$
$$\sigma_j \sim Cauchy(0, 40) > 0$$

```
data {
  int < lower =0 > N; // n of measurements
  int < lower =0 > J; // n of machine
  vector[J] y[N];
}
parameters {
```

```
  vector<lower = 0>[J] mu ;
  vector<lower = 0>[J] sigma ;
}
model {
  // weakly informative priors
  for ( j in 1: J ){
    mu [j] ~ normal (100, 50);
    sigma [j] ~ cauchy(0, 40);
  }
  // likelihood
  for ( j in 1: J )
    y[ ,j ] ~ normal (mu[j], sigma[j]);
}
generated quantities {
  vector[J] ypred ;
  matrix[N, J] log_lik ;
  for (j in 1:J)
    ypred[j] = normal_rng (mu[j], sigma[j]); // Compute predictive distribution for J machines
  for (i in 1:N)
    for (j in 1:J)
      log_lik[i, j] = normal_lpdf(y[i,j] | mu[j], sigma[j]); // Compute the log-likelihood values of ea
}
```

```
stan_data <- list(y = factory, N = nrow(factory), J = ncol(factory))
sm <- rstan::sampling(separate_model, data = stan_data)
```

```
##
## SAMPLING FOR MODEL 'c2ee9008df364b03d1ec290c1cc8f71b' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 1.1e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.11 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.052064 seconds (Warm-up)
## Chain 1:                0.045497 seconds (Sampling)
## Chain 1:                0.097561 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'c2ee9008df364b03d1ec290c1cc8f71b' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 5e-06 seconds
```

```
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.05 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.052052 seconds (Warm-up)
## Chain 2:                0.051048 seconds (Sampling)
## Chain 2:                0.1031 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'c2ee9008df364b03d1ec290c1cc8f71b' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 5e-06 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.05 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.052189 seconds (Warm-up)
## Chain 3:                0.041676 seconds (Sampling)
## Chain 3:                0.093865 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'c2ee9008df364b03d1ec290c1cc8f71b' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 6e-06 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.06 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
```

```
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.054755 seconds (Warm-up)
## Chain 4:                0.039467 seconds (Sampling)
## Chain 4:                0.094222 seconds (Total)
## Chain 4:
```

```
monitor(sm)
```

```
## Inference for the input samples (4 chains: each with iter = 2000; warmup = 0):
##
##                 Q5   Q50   Q95  Mean   SD  Rhat Bulk_ESS Tail_ESS
## mu[1]         57.8  77.0  96.5  77.2 11.9     1     2104     1257
## mu[2]         93.6 106.2 117.1 106.0  7.4     1     2380     1415
## mu[3]         74.7  88.0 101.8  88.2  8.5     1     2211     1433
## mu[4]        102.9 111.5 119.8 111.4  5.4     1     2087     1698
## mu[5]         79.4  90.2 101.1  90.2  6.9     1     3211     1960
## mu[6]         68.3  86.5 104.4  86.5 11.4     1     2342     1358
## sigma[1]      13.5  22.7  45.6  25.4 11.3     1     2487     2011
## sigma[2]       8.3  14.3  30.5  16.2  7.8     1     2542     2164
## sigma[3]       9.0  15.7  34.7  17.9  8.8     1     2160     1474
## sigma[4]       5.4   9.6  21.0  10.9  5.5     1     2024     1661
## sigma[5]       7.6  13.1  28.7  15.2  9.3     1     2519     2119
## sigma[6]      13.1  22.2  43.7  24.6 10.6     1     2494     2050
## ypred[1]      29.0  76.8 123.9  77.0 30.3     1     3589     3115
## ypred[2]      76.6 106.2 136.1 106.0 20.0     1     3559     2872
## ypred[3]      55.3  88.3 121.4  87.9 22.3     1     3241     2720
## ypred[4]      91.2 111.5 132.4 111.6 14.0     1     3638     2657
## ypred[5]      63.6  89.7 118.3  89.9 21.4     1     3695     3261
## ypred[6]      40.3  86.0 132.1  86.0 28.4     1     3671     3124
## log_lik[1,1]  -4.9  -4.2  -3.7  -4.2  0.4     1     1693     1395
## log_lik[2,1]  -5.2  -4.4  -3.9  -4.4  0.4     1     2282     1578
## log_lik[3,1]  -5.2  -4.4  -3.9  -4.4  0.4     1     2282     1578
## log_lik[4,1]  -6.8  -5.1  -4.3  -5.3  0.8     1     3727     2874
## log_lik[5,1]  -5.0  -4.2  -3.7  -4.3  0.4     1     2492     2049
## log_lik[1,2]  -4.9  -4.0  -3.4  -4.1  0.5     1     2840     2562
## log_lik[2,2]  -4.5  -3.7  -3.1  -3.7  0.4     1     1915     1646
## log_lik[3,2]  -4.7  -3.9  -3.3  -3.9  0.4     1     2238     1733
## log_lik[4,2]  -4.5  -3.7  -3.1  -3.7  0.4     1     1923     2147
## log_lik[5,2]  -6.4  -4.6  -3.9  -4.8  0.8     1     4062     3329
## log_lik[1,3]  -5.1  -4.2  -3.6  -4.2  0.5     1     2925     2594
## log_lik[2,3]  -4.7  -3.8  -3.3  -3.9  0.4     1     1616     1404
## log_lik[3,3]  -4.6  -3.8  -3.3  -3.9  0.4     1     1578     1371
## log_lik[4,3]  -4.6  -3.8  -3.2  -3.8  0.4     1     1639     1281
```

```
## log_lik[5,3]  -6.6  -4.7  -4.0  -4.9  0.8     1     4124     3309
## log_lik[1,4]  -4.4  -3.6  -3.0  -3.6  0.4     1     2481     2263
## log_lik[2,4]  -4.6  -3.6  -3.0  -3.7  0.5     1     2692     2783
## log_lik[3,4]  -4.3  -3.4  -2.8  -3.5  0.4     1     1842     1861
## log_lik[4,4]  -5.0  -3.8  -3.2  -3.9  0.6     1     3933     3724
## log_lik[5,4]  -4.3  -3.4  -2.8  -3.5  0.4     1     1842     1861
## log_lik[1,5]  -4.9  -4.0  -3.4  -4.1  0.5     1     4084     3039
## log_lik[2,5]  -4.5  -3.8  -3.2  -3.8  0.4     1     2285     1755
## log_lik[3,5]  -5.2  -4.1  -3.5  -4.2  0.5     1     4161     3424
## log_lik[4,5]  -4.9  -4.0  -3.4  -4.1  0.5     1     4084     3039
## log_lik[5,5]  -4.4  -3.6  -3.0  -3.6  0.4     1     1958     1429
## log_lik[1,6]  -6.7  -5.0  -4.3  -5.2  0.8     1     3450     2916
## log_lik[2,6]  -4.9  -4.2  -3.6  -4.2  0.4     1     1666     1357
## log_lik[3,6]  -5.3  -4.5  -3.9  -4.5  0.4     1     2431     2753
## log_lik[4,6]  -4.9  -4.2  -3.7  -4.2  0.4     1     2474     1995
## log_lik[5,6]  -5.1  -4.3  -3.8  -4.4  0.4     1     1883     1665
## lp__         -59.4 -53.9 -50.3 -54.3  2.8     1      942     1886
##
## For each parameter, Bulk_ESS and Tail_ESS are crude measures of
## effective sample size for bulk and tail quantities respectively (an ESS > 100
## per chain is considered good), and Rhat is the potential scale reduction
## factor on rank normalized split chains (at convergence, Rhat <= 1.05).
```

Pooled model

In the pooled model, all the measurements are combined and no distinction is made between the machines (they are drawn from the same distribution and the paramters do not change between the machines). We are using the same weakly informative priors as earlier in the separate model as there is no need to change them.

$$y_{ij} \sim N(\mu, \sigma)$$
$$\mu \sim N(100, 50)$$
$$\sigma \sim Cauchy(0, 40) > 0$$

```
data {
  int < lower =0 > N; // n of measurements
  int < lower =0 > J; // n of machines
  vector[J] y[N];
}
parameters {
  real<lower = 0> mu ;
  real<lower = 0> sigma ;
}
model {
  // weakly informative priors
  mu ~ normal (100, 50);
  sigma ~ cauchy(0, 40);
  // likelihood
  for ( j in 1: J )
    y[ ,j ] ~ normal (mu, sigma);
}
generated quantities {
  real ypred ;
  matrix[N, J] log_lik ;
```

```
  ypred = normal_rng(mu, sigma); // Compute predictive distribution for a machine as we cannot tell the
  for (i in 1:N)
    for (j in 1:J)
      log_lik[i, j] = normal_lpdf(y[i, j] | mu, sigma); // Compute the log-likelihood values of each obs
}
```

```
stan_data <- list(y = factory,N = nrow(factory),J = ncol(factory))
pm <- rstan::sampling(pooled_model, data = stan_data)
```

```
##
## SAMPLING FOR MODEL '933e1c43e5efdc7375491b582fce9f39' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 8e-06 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.08 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.014703 seconds (Warm-up)
## Chain 1:                0.014464 seconds (Sampling)
## Chain 1:                0.029167 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL '933e1c43e5efdc7375491b582fce9f39' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 4e-06 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.04 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
```

```
## Chain 2:  Elapsed Time: 0.015366 seconds (Warm-up)
## Chain 2:                0.014862 seconds (Sampling)
## Chain 2:                0.030228 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL '933e1c43e5efdc7375491b582fce9f39' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 5e-06 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.05 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.015189 seconds (Warm-up)
## Chain 3:                0.014982 seconds (Sampling)
## Chain 3:                0.030171 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL '933e1c43e5efdc7375491b582fce9f39' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 4e-06 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.04 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.015537 seconds (Warm-up)
## Chain 4:                0.014425 seconds (Sampling)
## Chain 4:                0.029962 seconds (Total)
## Chain 4:
```

```
monitor(pm)
```

```
## Inference for the input samples (4 chains: each with iter = 2000; warmup = 0):
##
##                  Q5   Q50   Q95  Mean   SD  Rhat Bulk_ESS Tail_ESS
## mu             87.4  92.9  98.6  92.9  3.4     1     2751     2068
## sigma          15.1  18.4  23.5  18.7  2.6     1     2698     2133
## ypred          62.3  92.8 125.3  93.2 19.4     1     3845     3914
## log_lik[1,1]  -4.3  -4.0  -3.8  -4.0  0.1     1     2936     2375
## log_lik[2,1]  -4.1  -3.8  -3.6  -3.9  0.1     1     2556     2187
## log_lik[3,1]  -4.1  -3.8  -3.6  -3.9  0.1     1     2556     2187
## log_lik[4,1]  -8.7  -7.1  -5.9  -7.2  0.9     1     2626     2145
## log_lik[5,1]  -5.4  -4.8  -4.5  -4.9  0.3     1     2814     2074
## log_lik[1,2]  -5.2  -4.7  -4.3  -4.7  0.3     1     3000     2632
## log_lik[2,2]  -4.5  -4.2  -4.0  -4.2  0.2     1     2802     2149
## log_lik[3,2]  -4.9  -4.5  -4.2  -4.5  0.2     1     2975     2512
## log_lik[4,2]  -4.3  -4.0  -3.8  -4.0  0.1     1     2581     1959
## log_lik[5,2]  -4.1  -3.9  -3.7  -3.9  0.1     1     2727     2117
## log_lik[1,3]  -4.2  -3.9  -3.7  -4.0  0.1     1     2508     1969
## log_lik[2,3]  -4.1  -3.8  -3.6  -3.9  0.1     1     2531     2173
## log_lik[3,3]  -4.1  -3.8  -3.6  -3.9  0.1     1     2556     2187
## log_lik[4,3]  -4.2  -3.9  -3.7  -3.9  0.1     1     2770     1976
## log_lik[5,3]  -5.4  -4.8  -4.5  -4.9  0.3     1     2814     2074
## log_lik[1,4]  -4.3  -4.1  -3.8  -4.1  0.1     1     2620     2004
## log_lik[2,4]  -5.4  -4.8  -4.5  -4.9  0.3     1     3002     2496
## log_lik[3,4]  -5.1  -4.6  -4.3  -4.7  0.2     1     2992     2637
## log_lik[4,4]  -4.2  -4.0  -3.8  -4.0  0.1     1     2525     1975
## log_lik[5,4]  -5.1  -4.6  -4.3  -4.7  0.2     1     2992     2637
## log_lik[1,5]  -4.4  -4.1  -3.9  -4.1  0.2     1     3045     2577
## log_lik[2,5]  -4.1  -3.9  -3.7  -3.9  0.1     1     2483     1929
## log_lik[3,5]  -4.3  -4.0  -3.8  -4.0  0.1     1     2548     1999
## log_lik[4,5]  -4.4  -4.1  -3.9  -4.1  0.2     1     3045     2577
## log_lik[5,5]  -4.1  -3.8  -3.6  -3.9  0.1     1     2556     2187
## log_lik[1,6]  -6.8  -5.8  -5.1  -5.8  0.5     1     2654     1986
## log_lik[2,6]  -4.1  -3.8  -3.6  -3.9  0.1     1     2556     2187
## log_lik[3,6]  -4.3  -4.0  -3.8  -4.0  0.1     1     2581     1959
## log_lik[4,6]  -4.5  -4.2  -4.0  -4.2  0.2     1     3068     2632
## log_lik[5,6]  -4.2  -3.9  -3.7  -3.9  0.1     1     2493     1909
## lp__         -97.1 -94.7 -94.0 -95.0  1.0     1     1702     2191
##
## For each parameter, Bulk_ESS and Tail_ESS are crude measures of
## effective sample size for bulk and tail quantities respectively (an ESS > 100
## per chain is considered good), and Rhat is the potential scale reduction
## factor on rank normalized split chains (at convergence, Rhat <= 1.05).
```

Hierarchical model

In hierarchical model, as in the model described in the book, use the same measurement standard deviation σ for all the groups in the hierarchical model. Again, there is no need to change our priors for this case. Only thing changing is that we need to use the hyper-priors now in addition to priors.

$$y_j \sim N(\theta_{ij}, \sigma)$$
$$\theta_j \sim N(\mu, \tau)$$

$$\sigma \sim Cauchy(0, 40) > 0$$
$$\mu \sim N(100, 50)$$
$$\tau \sim Cauchy(0, 40) > 0$$

```
data {
  int < lower =0 > N; // n of measurements
  int < lower =0 > J; // n of machines
  vector[J] y[N];
}
parameters {
  real mu; // hyper-parameter 1
  real<lower=0> tau; // hyper-parameter 2
  vector[J] theta; // separate mean parameter theta for each machine
  real<lower=0> sigma; // common sigma parameter for all machines
}
model {
  // weakly informative priors
  mu ~ normal (100, 50); // hyperprior for mu
  tau ~ cauchy(0, 40); // hyperprior for tau
  for ( j in 1: J ){
    theta [j] ~ normal (mu, tau);
  }
  sigma ~ cauchy(0,40);
  // likelihood
  for ( j in 1: J )
    y[ ,j ] ~ normal (theta[j], sigma);
}
generated quantities {
  vector[J] ypred ;
  real theta7;
  matrix[N, J] log_lik ;

  for (j in 1:J)
    ypred[j] = normal_rng(theta[j], sigma);
  theta7 = normal_rng(mu, tau);
  for (i in 1:N)
    for (j in 1:J)
      log_lik[i, j] = normal_lpdf(y[i, j] | theta[j], sigma); // Compute the log-likelihood values of ea
}
```

```
stan_data <- list(y = factory,N = nrow(factory),J = ncol(factory))
hm <- rstan::sampling(hierarchical_model, data = stan_data)
```

```
##
## SAMPLING FOR MODEL '14dba4e514a095cf24a0420bdacd2549' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 1e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.1 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
```

```
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.081023 seconds (Warm-up)
## Chain 1:                0.027932 seconds (Sampling)
## Chain 1:                0.108955 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL '14dba4e514a095cf24a0420bdacd2549' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 5e-06 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.05 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.070451 seconds (Warm-up)
## Chain 2:                0.035003 seconds (Sampling)
## Chain 2:                0.105454 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL '14dba4e514a095cf24a0420bdacd2549' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 5e-06 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.05 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
```

```
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.071337 seconds (Warm-up)
## Chain 3:                0.02902 seconds (Sampling)
## Chain 3:                0.100357 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL '14dba4e514a095cf24a0420bdacd2549' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 4e-06 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.04 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.0757 seconds (Warm-up)
## Chain 4:                0.030107 seconds (Sampling)
## Chain 4:                0.105807 seconds (Total)
## Chain 4:
```

```
## Warning: There were 1 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
```

```
monitor(hm)
```

```
## Inference for the input samples (4 chains: each with iter = 2000; warmup = 0):
##
##                  Q5    Q50    Q95   Mean   SD  Rhat Bulk_ESS Tail_ESS
## mu             80.6   92.9  103.9   92.7  7.2     1     2465     1873
## tau             6.4   13.5   28.3   15.0  7.2     1     1718     1846
## theta[1]       69.2   80.2   90.9   80.1  6.6     1     2828     2813
## theta[2]       92.7  103.3  113.8  103.3  6.4     1     3318     2587
## theta[3]       78.7   89.0   99.3   89.0  6.3     1     3803     2652
## theta[4]       96.3  107.4  117.9  107.3  6.6     1     2643     2438
## theta[5]       80.5   90.7  100.4   90.6  6.0     1     4682     3076
## theta[6]       77.3   87.6   97.8   87.6  6.3     1     3922     2596
## sigma          11.9   14.9   19.3   15.1  2.3     1     3378     2646
```

```
## ypred[1]       52.8   79.7  106.9   79.7 16.4     1     3967     3720
## ypred[2]       76.1  103.8  129.5  103.2 16.4     1     4007     3606
## ypred[3]       62.2   89.5  115.1   88.8 16.4     1     3635     3226
## ypred[4]       80.0  107.6  134.1  107.2 16.6     1     3647     3128
## ypred[5]       63.7   90.8  117.1   90.7 16.3     1     3838     3614
## ypred[6]       60.3   87.5  114.6   87.5 16.6     1     3826     3719
## theta7         63.9   93.0  122.9   92.9 18.4     1     3684     3325
## log_lik[1,1]   -4.1   -3.7   -3.5   -3.7  0.2     1     2930     3201
## log_lik[2,1]   -4.8   -4.0   -3.6   -4.1  0.4     1     3491     3071
## log_lik[3,1]   -4.8   -4.0   -3.6   -4.1  0.4     1     3491     3071
## log_lik[4,1]   -8.4   -6.3   -4.8   -6.4  1.1     1     3652     2740
## log_lik[5,1]   -4.9   -4.0   -3.6   -4.1  0.4     1     2920     2852
## log_lik[1,2]   -4.9   -4.1   -3.6   -4.1  0.4     1     3771     3244
## log_lik[2,2]   -4.2   -3.7   -3.5   -3.8  0.2     1     2678     2845
## log_lik[3,2]   -4.6   -3.9   -3.5   -4.0  0.3     1     3436     3149
## log_lik[4,2]   -4.1   -3.7   -3.4   -3.7  0.2     1     2351     2352
## log_lik[5,2]   -5.3   -4.2   -3.7   -4.3  0.5     1     3893     3087
## log_lik[1,3]   -4.8   -4.0   -3.6   -4.0  0.4     1     4110     3122
## log_lik[2,3]   -4.1   -3.7   -3.5   -3.7  0.2     1     2770     2719
## log_lik[3,3]   -4.1   -3.7   -3.4   -3.7  0.2     1     2590     2652
## log_lik[4,3]   -4.1   -3.7   -3.4   -3.7  0.2     1     2512     2970
## log_lik[5,3]   -6.0   -4.7   -4.0   -4.8  0.7     1     4169     2930
## log_lik[1,4]   -4.1   -3.7   -3.4   -3.7  0.2     1     2864     2731
## log_lik[2,4]   -4.7   -4.0   -3.6   -4.0  0.4     1     2819     2693
## log_lik[3,4]   -4.5   -3.8   -3.5   -3.9  0.3     1     2623     2622
## log_lik[4,4]   -4.2   -3.8   -3.5   -3.8  0.2     1     3435     2851
## log_lik[5,4]   -4.5   -3.8   -3.5   -3.9  0.3     1     2623     2622
## log_lik[1,5]   -4.6   -4.0   -3.6   -4.0  0.3     1     4688     3288
## log_lik[2,5]   -4.2   -3.8   -3.5   -3.8  0.2     1     3464     3218
## log_lik[3,5]   -4.8   -4.0   -3.6   -4.1  0.4     1     4771     3497
## log_lik[4,5]   -4.6   -4.0   -3.6   -4.0  0.3     1     4688     3288
## log_lik[5,5]   -4.0   -3.7   -3.4   -3.7  0.2     1     2433     2681
## log_lik[1,6]   -7.7   -5.8   -4.5   -5.9  1.0     1     4343     2631
## log_lik[2,6]   -4.1   -3.7   -3.5   -3.8  0.2     1     2935     2580
## log_lik[3,6]   -5.3   -4.3   -3.7   -4.4  0.5     1     4340     2830
## log_lik[4,6]   -4.6   -3.9   -3.5   -4.0  0.3     1     4022     3294
## log_lik[5,6]   -4.8   -4.0   -3.6   -4.1  0.4     1     4194     2767
## lp__         -113.6 -108.6 -105.7 -109.0  2.4     1     1512     2121
##
## For each parameter, Bulk_ESS and Tail_ESS are crude measures of
## effective sample size for bulk and tail quantities respectively (an ESS > 100
## per chain is considered good), and Rhat is the potential scale reduction
## factor on rank normalized split chains (at convergence, Rhat <= 1.05).
```

2. Compute the PSIS-LOO elpd values and the $\hat{k}$-values for each of the three models.

Separate model

```
# Convert extracted log_likelihood values to matrix, since loo-function requires matrix as its paramete
sm_ll <- matrix(unlist(extract(sm, pars = paste("log_lik[",rep(1:5, 6),",", rep(1:6, each=5),"]", sep="
loo_sm <- loo(sm_ll)
```

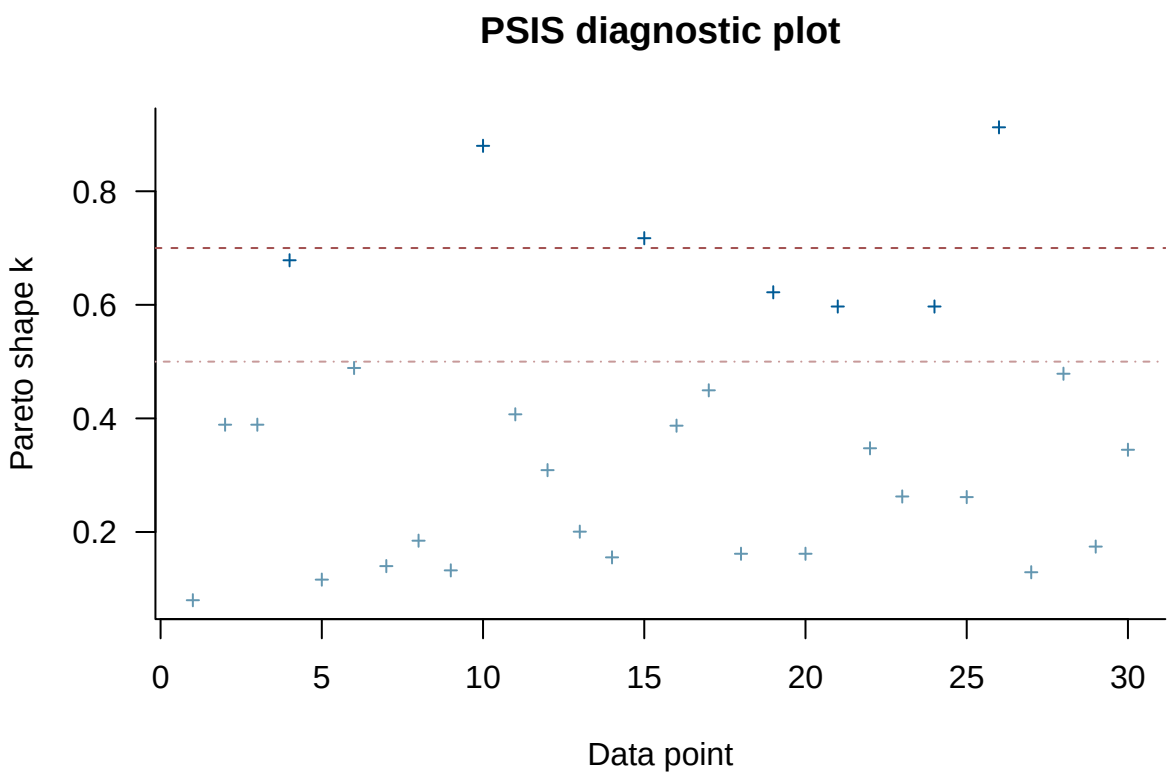
```
## Warning: Relative effective sample sizes ('r_eff' argument) not specified.
## For models fit with MCMC, the reported PSIS effective sample sizes and
## MCSE estimates will be over-optimistic.
```

```
## Warning: Some Pareto k diagnostic values are too high. See help('pareto-k-diagnostic') for details.
```

```
loo_sm
```

```
##
## Computed from 4000 by 30 log-likelihood matrix
##
##          Estimate  SE
## elpd_loo   -129.5 3.4
## p_loo         8.6 1.3
## looic       259.1 6.7
## ------
## Monte Carlo SE of elpd_loo is NA.
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     23    76.7%   2095
##  (0.5, 0.7]   (ok)        4    13.3%   235
##    (0.7, 1]   (bad)       3    10.0%   112
##    (1, Inf)   (very bad)  0     0.0%   <NA>
## See help('pareto-k-diagnostic') for details.
```

```
plot(loo_sm)
```



Pooled model

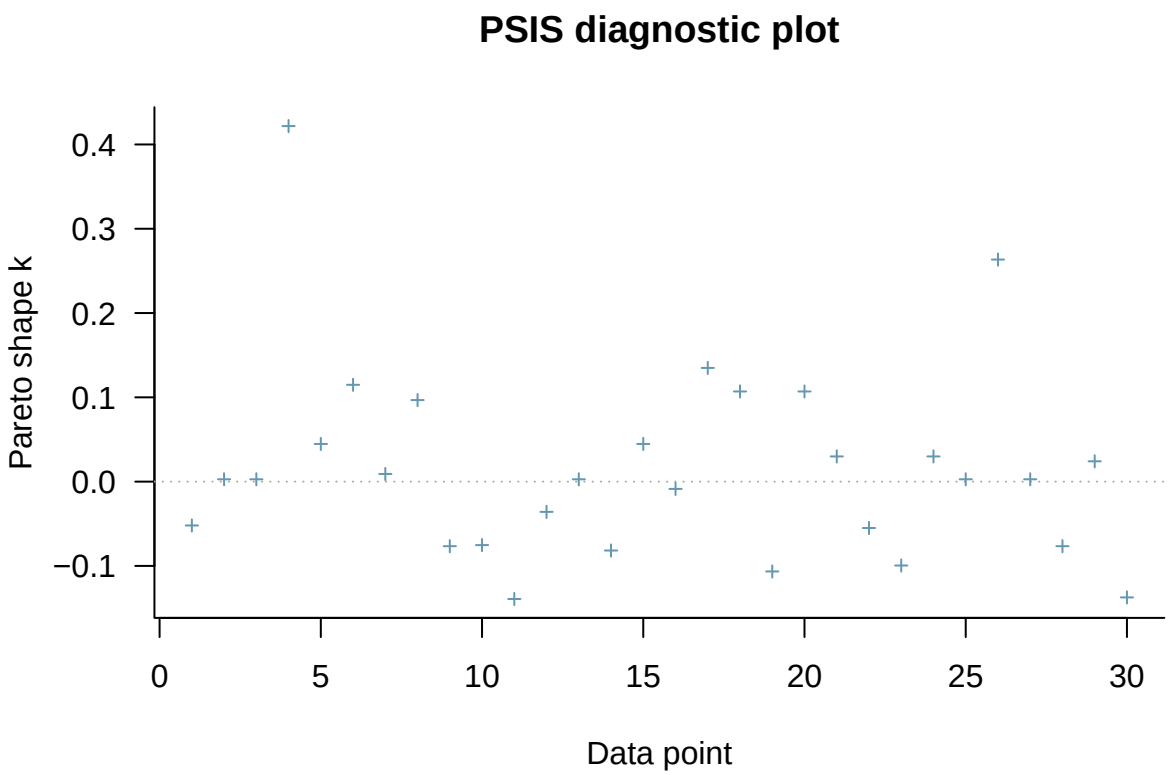
```
# Convert extracted log_likelihood values to matrix, since loo-function requires matrix as its paramete
pm_ll <- matrix(unlist(extract(pm, pars = paste("log_lik[",rep(1:5, 6),",", rep(1:6, each=5),"]", sep="
loo_pm <- loo(pm_ll)
```

```
## Warning: Relative effective sample sizes ('r_eff' argument) not specified.
## For models fit with MCMC, the reported PSIS effective sample sizes and
## MCSE estimates will be over-optimistic.
```

```
loo_pm
```

```
##
## Computed from 4000 by 30 log-likelihood matrix
##
##          Estimate  SE
## elpd_loo   -130.9 4.3
## p_loo         2.0 0.8
## looic       261.8 8.6
## ------
## Monte Carlo SE of elpd_loo is 0.0.
##
## All Pareto k estimates are good (k < 0.5).
## See help('pareto-k-diagnostic') for details.
```

```
plot(loo_pm)
```



Hierarchical model

```
# Convert extracted log_likelihood values to matrix, since loo-function requires matrix as its paramete
hm_ll <- matrix(unlist(extract(hm, pars = paste("log_lik[",rep(1:5, 6),",", rep(1:6, each=5),"]", sep="
loo_hm <- loo(hm_ll)
```

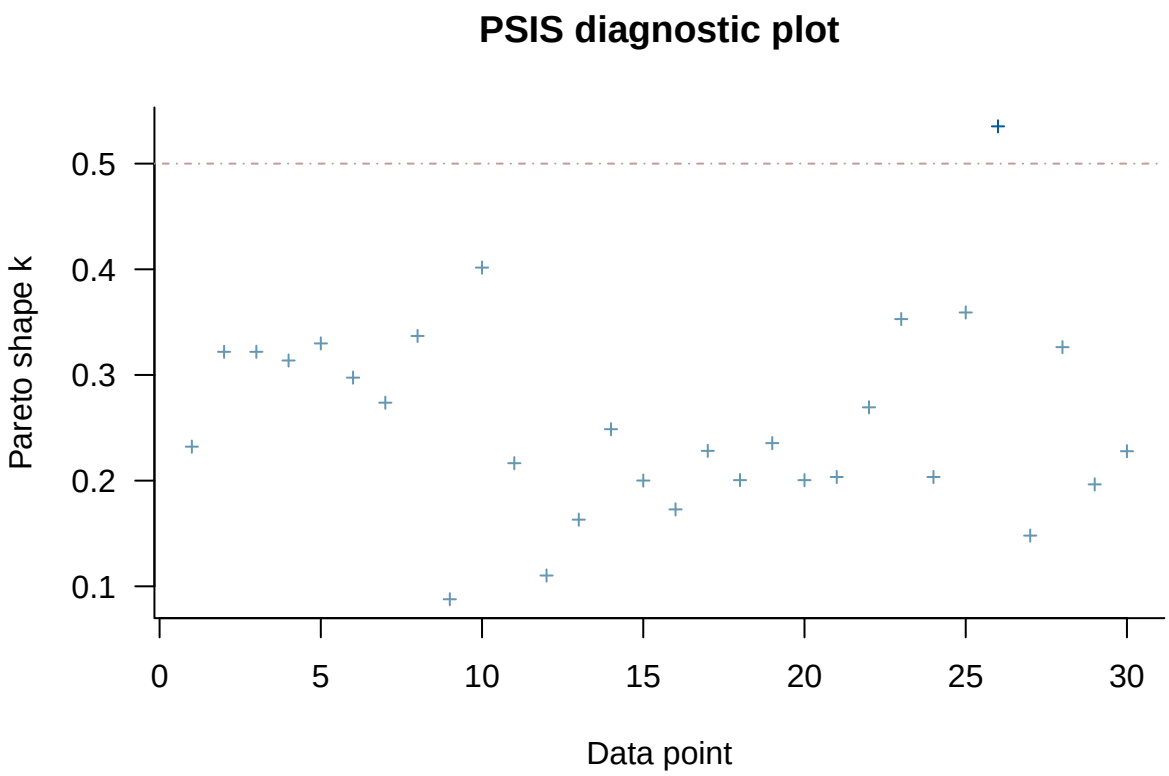
```
## Warning: Relative effective sample sizes ('r_eff' argument) not specified.
## For models fit with MCMC, the reported PSIS effective sample sizes and
## MCSE estimates will be over-optimistic.
```

```
## Warning: Some Pareto k diagnostic values are slightly high. See help('pareto-k-diagnostic') for deta
```

```
loo_hm
```

```
##
## Computed from 4000 by 30 log-likelihood matrix
##
##          Estimate  SE
## elpd_loo   -126.7 4.2
## p_loo         5.6 1.5
## looic       253.3 8.4
## ------
## Monte Carlo SE of elpd_loo is 0.1.
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     29    96.7%   652
##  (0.5, 0.7]   (ok)        1     3.3%   650
##    (0.7, 1]   (bad)       0     0.0%   <NA>
##    (1, Inf)   (very bad)  0     0.0%   <NA>
##
## All Pareto k estimates are ok (k < 0.7).
## See help('pareto-k-diagnostic') for details.
```

```
plot(loo_hm)
```



3. Compute the effective number of parameters p_{eff} for each of the three models.

The estimated effective number of parameters for each model can be solved analytically using the equations 7.15 and 7.5 in the BDA3. In this case, we can actually get the values in a simpler method, since the function used in the previous part 2 computed the p_{eff} estimates.

Separate model

```
### SEPARATE MODEL ###

# p_eff estimate
loo_sm$estimates[2,1]
```

```
## [1] 8.624721
```

```
# p_eff standard errors
loo_sm$estimates[2,2]
```

```
## [1] 1.270785
```

The achieved values are feasible, since the model has in total 12 parameters.

Pooled model

```
### POOLED MODEL ###

# p_eff estimate
loo_pm$estimates[2,1]
```

```
## [1] 1.964756
```

```
# p_eff standard errors
loo_pm$estimates[2,2]
```

```
## [1] 0.7936912
```

These values seem also feasible, since our pooled model had only 2 parameters in total.

Hierarchical model

```
### HIERARCHICAL MODEL ###

# p_eff estimate
loo_hm$estimates[2,1]
```

```
## [1] 5.625189
```

```
# p_eff standard errors
loo_hm$estimates[2,2]
```

```
## [1] 1.484447
```

Also, the estimate for effective number of parameters for hierarchical model seems reasonable, since the original model had in total 7 parameters.

4. Assess how reliable the PSIS-LOO estimates are for the three models based on the $\hat{k}$-values.

The estimates for the hierarchical model and the pooled model were good and all of the $\hat{k}$-values were less or equal than 0.7. Based on this, we can state that the the PSIS-LOO estimates can be considered to be reliable for these two models.

The separate model on the otherhand, had the worst performance when it came to $\hat{k}$-values. Several estimates were greater than 0.7, so there is a concern that the PSIS-LOO estimate for the separate model may be too optimistic. This can lead us to overestimate the predictive accuracy of the model.

5. An assessment of whether there are differences between the models with regard to the $elpd_{loo-cv}$, and if so, which model should be selected according to PSIS-LOO.

The best PSIS-LOO estimate was achieved with hierarchical model. All models had pretty similar performances with estimates ranging from -126.7 to -130.9. Based on the reasoning in the previous part, where the $\hat{k}$-values were compared, we would select our model between pooled and hierachical model. From these two models I would select the hierarchical version, because of the following two reasons:

1. Hierarchical model has the greatest PSIS-LOO estimate with value -126.7.

2. The pooled model has a strong assumption that all the measurements for all the machines follow the same distribution. With hierarchical model, less assumptions are required, which is in general a good feature in statistics.