BDA - Assignment 8

Anonymous

Contents

Model assessment

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

Use leave-one-out cross-validation (LOO-CV) to assess the predictive performance of the pooled, separate and hierarchical Gaussian models for the factory dataset (see the second exercise in Assignment 7). To read in the data, just use:

```
library(aaltobda)
library(loo)
```

```
## Warning: package 'loo' was built under R version 4.1.3
```

```
## This is loo version 2.5.1

## - 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' ar

## - Windows 10 users: loo may be very slow if 'mc.cores' is set in your .Rprofile file (see https://gi
```

```
data("factory")
```

PSIS-LOO is a recently developed method for approximating the exact LOO and is thus not in BDA3. For more information, see the lecture slides and the original paper.
Use Stan for fitting the models, and the loo R package for computing the approximate LOO-CV given the posterior samples provided by Stan. You can install the package as

```
#install.packages("loo")
```

Python users can use PSIS-LOO implementation in ArviZ library.
The report should include the following parts.

1.

Fit the models with Stan as instructed in Assignment 7. To use the loo or psisloo functions, you need to compute the log-likelihood values of each observation for every posterior draw (i.e. an S-by-N matrix, where S is the number of posterior draws and N = 30 is the total number of observations). This can be done in the generated quantities block in the Stan code; for a demonstration, see the Gaussian linear model lin.stan in the R Stan examples that can be found here.

Separate model

Example priors from assignment 7 is used:
$\mu_j \sim Normal(0, 10)$
$\sigma_j \sim gamma(1, 1)$

```
"
data {
  int<lower=0> N; // number of measurements per machine
  int<lower=0> K; // number of machines
  array[N] vector[K] y; // An array of vectors containing the table data
}

parameters {
  vector[K] mu;
  vector<lower=0>[K] sigma;
}
```

```
model {
  for (k in 1:K) {
    mu[k] ~ normal(0, 10);
    sigma[k] ~ gamma(1, 1);
  }
  for (k in 1:K) {
    y[, k] ~ normal(mu[k], sigma[k]);
  }
}

generated quantities {
  array[K] real ypred;
  array[N] vector[K] log_likelihood; // An array of vectors of the log likelihood
  for (k in 1:K) {
    ypred[k] = normal_rng(mu[k], sigma[k]);
  }
  for (i in 1:N) {
    for (k in 1:K) {
      log_likelihood[i][k] = normal_lpdf(y[i][k] | mu[k], sigma[k]);
    }
  }
}

"
```

Pooled model

Example priors from assignment 7 is used:
$\mu \sim Normal(0, 10)$
$\sigma \sim gamma(1, 1)$

```
"
data {
  int<lower=0> N; // number of measurements per machine
  int<lower=0> K; // number of machines
  array[N] vector[K] y; // An array of vectors containing the table data
}

parameters {
  real mu; // There is only a single mu for all machines
  real<lower=0> sigma; // There is only a single sigma for all machines
}

model {
  // priors
  mu ~ normal(0, 10);
  sigma ~ gamma(1, 1);

  // likelihood
  for (k in 1:K){
    y[,k] ~ normal(mu, sigma);
  }
```

```
}

generated quantities {
  real ypred = normal_rng(mu, sigma);
  array[N] vector[K] log_likelihood; // An array of vectors of the log likelihood
  for (i in 1:N) {
    for (k in 1:K) {
      log_likelihood[i][k] = normal_lpdf(y[i][k] | mu, sigma);
    }
  }
}

"
```

Hierarchical model

Example priors from assignment 7 is used:
$\mu_\tau \sim Normal(0, 10)$
$\sigma_\tau \sim gamma(1, 1)$
$\sigma \sim gamma(1, 1)$

```
"
data {
  int<lower=0> N; // number of measurements per machine
  int<lower=0> K; // number of machines
  array[N] vector[K] y; // An array of vectors containing the table data
}

parameters {
  vector[K] mu;
  real sigma;
  real mu_tau;
  real sigma_tau;
}

model {
  mu_tau ~ normal(0, 10);
  sigma_tau ~ gamma(1, 1);
  sigma ~ gamma(1, 1);
  for (k in 1:K) {
    mu[k] ~ normal(mu_tau, sigma_tau);
  }
  for (k in 1:K) {
    y[, k] ~ normal(mu[k], sigma);
  }
}

generated quantities {
  array[K] real ypred;
  for (k in 1:K) {
    ypred[k] = normal_rng(mu[k] , sigma);
  }
```

```
  array[N] vector[K] log_likelihood; // An array of vectors of the log likelihood
  for (i in 1:N) {
    for (k in 1:K) {
      log_likelihood[i][k] = normal_lpdf(y[i][k] | mu[k], sigma);
    }
  }
  real mu_ypred7 = normal_rng(mu_tau, sigma_tau);
  real ypred7 = normal_rng(mu_ypred7, sigma);
}

"
```

Fitting the models:

```
stan_data <- list(
  y = factory,
  N = nrow(factory),
  K = ncol(factory)
)

file_separate <- file.path("C:/Users/nguye/Desktop/BDA/assignments/assignment 8", "model_separate.stan")
model_separate <- cmdstan_model(file_separate)
model_separate$compile(quiet = FALSE)
result_separate <- model_separate$sample(data = stan_data, show_messages=FALSE)
```

```
## Running MCMC with 4 sequential chains...
##
## Chain 1 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1 Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 1 Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1 Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 1 Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1 Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 1 Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1 Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 1 Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1 Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 1 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1 Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 1 Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1 Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 1 Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1 Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 1 Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1 Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 1 Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1 Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 1 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1 finished in 2.3 seconds.
## Chain 2 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2 Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 2 Iteration:  200 / 2000 [ 10%]  (Warmup)
```

```
## Chain 2 Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 2 Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2 Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 2 Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2 Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 2 Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2 Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 2 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2 Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 2 Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2 Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 2 Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2 Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 2 Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2 Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 2 Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2 Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 2 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2 finished in 2.1 seconds.
## Chain 3 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3 Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 3 Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3 Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 3 Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3 Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 3 Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3 Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 3 Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3 Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 3 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3 Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 3 Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3 Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 3 Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3 Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 3 Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3 Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 3 Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3 Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 3 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3 finished in 2.3 seconds.
## Chain 4 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4 Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 4 Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4 Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 4 Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4 Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 4 Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4 Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 4 Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4 Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 4 Iteration: 1000 / 2000 [ 50%]  (Warmup)
```

```
## Chain 4 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4 Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 4 Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4 Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 4 Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4 Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 4 Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4 Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 4 Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4 Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 4 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4 finished in 2.1 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 2.2 seconds.
## Total execution time: 9.2 seconds.
```

```
file_pooled <- file.path("C:/Users/nguye/Desktop/BDA/assignments/assignment 8", "model_pooled.stan")
model_pooled <- cmdstan_model(file_pooled)
model_pooled$compile(quiet = FALSE)
result_pooled <- model_pooled$sample(data = stan_data, show_messages=FALSE)
```

```
## Running MCMC with 4 sequential chains...
##
## Chain 1 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1 Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 1 Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1 Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 1 Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1 Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 1 Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1 Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 1 Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1 Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 1 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1 Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 1 Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1 Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 1 Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1 Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 1 Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1 Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 1 Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1 Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 1 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1 finished in 0.4 seconds.
## Chain 2 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2 Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 2 Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2 Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 2 Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2 Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 2 Iteration:  600 / 2000 [ 30%]  (Warmup)
```

```
## Chain 2 Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 2 Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2 Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 2 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2 Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 2 Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2 Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 2 Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2 Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 2 Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2 Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 2 Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2 Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 2 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2 finished in 0.4 seconds.
## Chain 3 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3 Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 3 Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3 Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 3 Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3 Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 3 Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3 Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 3 Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3 Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 3 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3 Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 3 Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3 Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 3 Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3 Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 3 Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3 Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 3 Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3 Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 3 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3 finished in 0.4 seconds.
## Chain 4 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4 Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 4 Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4 Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 4 Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4 Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 4 Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4 Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 4 Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4 Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 4 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4 Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 4 Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4 Iteration: 1300 / 2000 [ 65%]  (Sampling)
```

```
## Chain 4 Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4 Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 4 Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4 Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 4 Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4 Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 4 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4 finished in 0.6 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 0.4 seconds.
## Total execution time: 2.2 seconds.
```

```
file_hierarchical <- file.path("C:/Users/nguye/Desktop/BDA/assignments/assignment 8", "model_hierarchica
model_hierarchical <- cmdstan_model(file_hierarchical)
model_hierarchical$compile(quiet = FALSE)
result_hierarchical <- model_hierarchical$sample(data = stan_data, show_messages=FALSE)
```

```
## Running MCMC with 4 sequential chains...
##
## Chain 1 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1 Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 1 Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1 Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 1 Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1 Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 1 Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1 Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 1 Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1 Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 1 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1 Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 1 Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1 Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 1 Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1 Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 1 Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1 Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 1 Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1 Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 1 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1 finished in 1.2 seconds.
## Chain 2 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2 Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 2 Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2 Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 2 Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2 Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 2 Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2 Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 2 Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2 Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 2 Iteration: 1000 / 2000 [ 50%]  (Warmup)
```

```
## Chain 2 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2 Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 2 Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2 Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 2 Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2 Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 2 Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2 Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 2 Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2 Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 2 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2 finished in 1.0 seconds.
## Chain 3 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3 Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 3 Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3 Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 3 Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3 Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 3 Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3 Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 3 Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3 Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 3 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3 Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 3 Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3 Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 3 Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3 Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 3 Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3 Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 3 Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3 Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 3 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3 finished in 1.4 seconds.

## Chain 4 Rejecting initial value:

## Chain 4   Error evaluating the log probability at the initial value.

## Chain 4 Exception: gamma_lpdf: Random variable is -1.58097, but must be positive finite! (in 'C:/Use
## Chain 4 Exception: gamma_lpdf: Random variable is -1.58097, but must be positive finite! (in 'C:/Use

## Chain 4 Rejecting initial value:

## Chain 4   Error evaluating the log probability at the initial value.

## Chain 4 Exception: gamma_lpdf: Random variable is -0.499341, but must be positive finite! (in 'C:/Us
## Chain 4 Exception: gamma_lpdf: Random variable is -0.499341, but must be positive finite! (in 'C:/Us

## Chain 4 Rejecting initial value:

## Chain 4   Error evaluating the log probability at the initial value.
```

```
## Chain 4 Exception: gamma_lpdf: Random variable is -0.881939, but must be positive finite! (in 'C:/Us
## Chain 4 Exception: gamma_lpdf: Random variable is -0.881939, but must be positive finite! (in 'C:/Us

## Chain 4 Rejecting initial value:

## Chain 4   Error evaluating the log probability at the initial value.

## Chain 4 Exception: gamma_lpdf: Random variable is -1.55751, but must be positive finite! (in 'C:/Use
## Chain 4 Exception: gamma_lpdf: Random variable is -1.55751, but must be positive finite! (in 'C:/Use

## Chain 4 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4 Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 4 Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4 Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 4 Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4 Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 4 Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4 Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 4 Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4 Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 4 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4 Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 4 Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4 Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 4 Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4 Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 4 Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4 Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 4 Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4 Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 4 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4 finished in 1.3 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 1.2 seconds.
## Total execution time: 5.3 seconds.

## Warning: 37 of 4000 (1.0%) transitions ended with a divergence.
## See https://mc-stan.org/misc/warnings for details.
```

2.

Compute the Pareto smoothed importance sampling leave-one-out (PSIS-LOO) expected log pointwise predictive density (elpd) values and the $\hat{k}$-values for each of the three models.
Hint! It will be convenient to visualize the $\hat{k}$-values for each model so that you can easily see how many of these values fall in the range $\hat{k} > 0.7$ to assess the reliability of the PSIS-LOO estimate for each model. You can read more about the theoretical guarantees for the accuracy of the estimate depending on $\hat{k}$ from (see the original article, but regarding this assignment, it suffices to understand that if all the $\hat{k}$-values are $\hat{k} \lesssim 0.7$, the PSIS-LOO estimate can be considered to be reliable, otherwise there is a concern that it may be biased (too optimistic, overestimating the predictive accuracy of the model).

Separate model

Example priors from assignment 7 is used:
$\mu_j \sim Normal(0, 10)$
$\sigma_j \sim gamma(1, 1)$

```
loo_separate <- result_separate$loo(variables="log_likelihood",r_eff=TRUE)
```

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

```
cat("The PSIS-LOO elpd value of the separate model is\n")
```

```
## The PSIS-LOO elpd value of the separate model is
```

```
print(loo_separate$estimates[1][1])
```

```
## [1] -196.6803
```

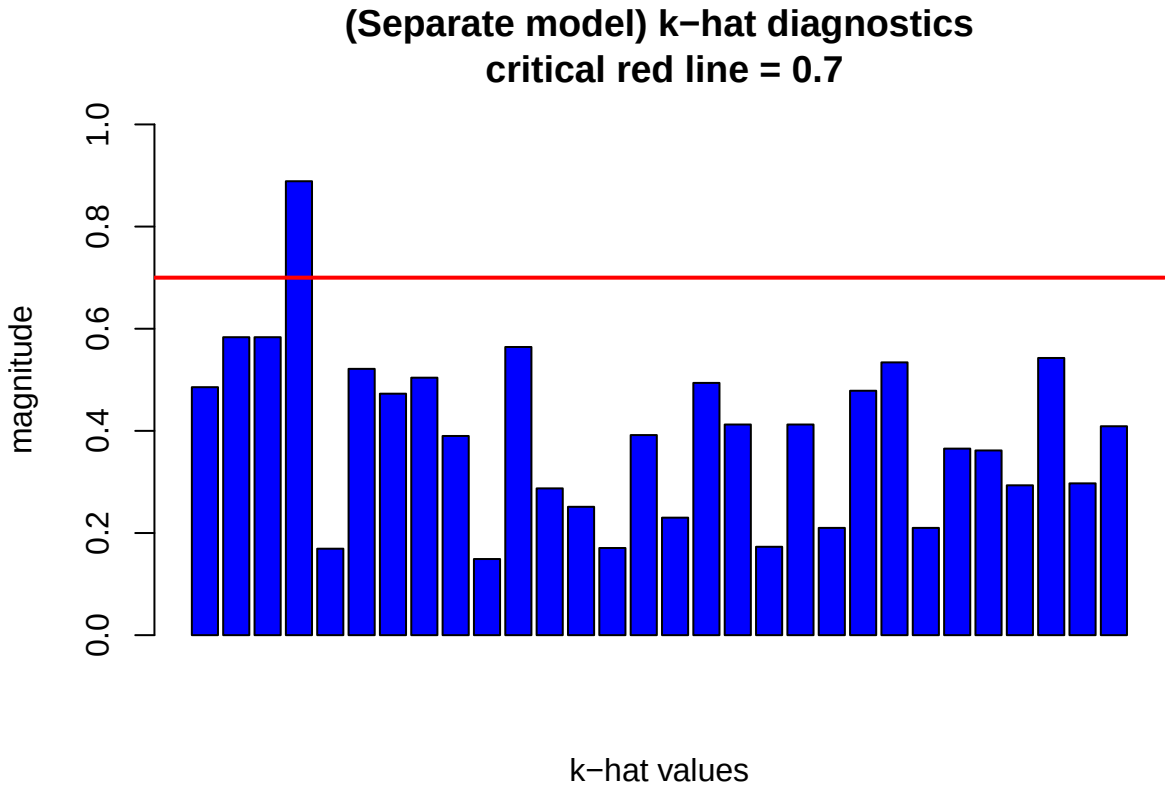
```
pareto_k <- loo_separate$diagnostics$pareto_k

cat("\nThe k-hat values of the separate model is\n")
```

```
##
## The k-hat values of the separate model is
```

```
print(loo_separate)
```

```
##
## Computed from 4000 by 30 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo   -196.7  8.0
## p_loo        20.0  1.4
## looic       393.4 16.0
## ------
## Monte Carlo SE of elpd_loo is NA.
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     22    73.3%   1055
##  (0.5, 0.7]   (ok)        7    23.3%   409
##    (0.7, 1]   (bad)       1     3.3%   343
##    (1, Inf)   (very bad)  0     0.0%   <NA>
## See help('pareto-k-diagnostic') for details.
```

```
barplot(pareto_k, main = "(Separate model) k-hat diagnostics\n critical red line = 0.7", xlab = "k-hat
abline(h=0.7, col="red", lwd=2)
```



So it appears that the separate model is a little biased reliable there is one $\hat{k}$-value larger than 0.7

Pooled model

Example priors from assignment 7 is used:
$\mu \sim Normal(0, 10)$
$\sigma \sim gamma(1, 1)$

```
loo_pooled <- result_pooled$loo(variables="log_likelihood",r_eff=TRUE)
cat("The PSIS-LOO elpd value of the pooled model is\n")
```

```
## The PSIS-LOO elpd value of the pooled model is
```

```
print(loo_pooled$estimates[1][1])
```

```
## [1] -134.603
```

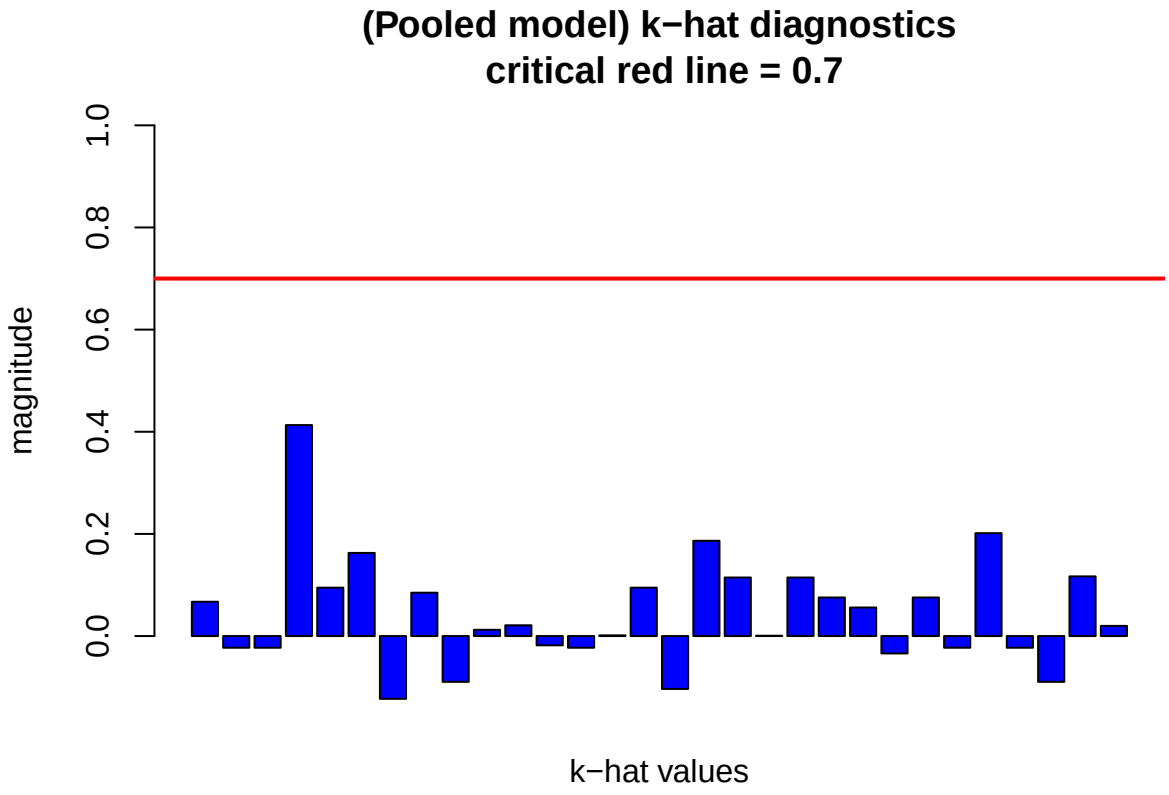
```
pareto_k <- loo_pooled$diagnostics$pareto_k

cat("\nThe k-hat values of the pooled model is\n")
```

```
##
## The k-hat values of the pooled model is
```

```
print(loo_pooled)
```

```
##
## Computed from 4000 by 30 log-likelihood matrix
##
##          Estimate  SE
## elpd_loo   -134.6 4.9
## p_loo         2.5 0.9
## looic       269.2 9.8
## ------
## Monte Carlo SE of elpd_loo is 0.1.
##
## All Pareto k estimates are good (k < 0.5).
## See help('pareto-k-diagnostic') for details.
```

```
barplot(pareto_k, main = "(Pooled model) k-hat diagnostics\n critical red line = 0.7", xlab = "k-hat va
abline(h=0.7, col="red", lwd=2)
```



So it appears that the pooled model is reliable as all $\hat{k}$-values are smaller than 0.7

Hierarchical model

Example priors from assignment 7 is used:
$\mu_\tau \sim Normal(0, 10)$

$\sigma_\tau \sim gamma(1,1)$
$\sigma \sim gamma(1,1)$

```
loo_hierarchical <- result_hierarchical$loo(variables="log_likelihood",r_eff=TRUE)
```

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

```
cat("The PSIS-LOO elpd value of the hierarchical model is\n")
```

```
## The PSIS-LOO elpd value of the hierarchical model is
```

```
print(loo_hierarchical$estimates[1][1])
```

```
## [1] -128.7648
```

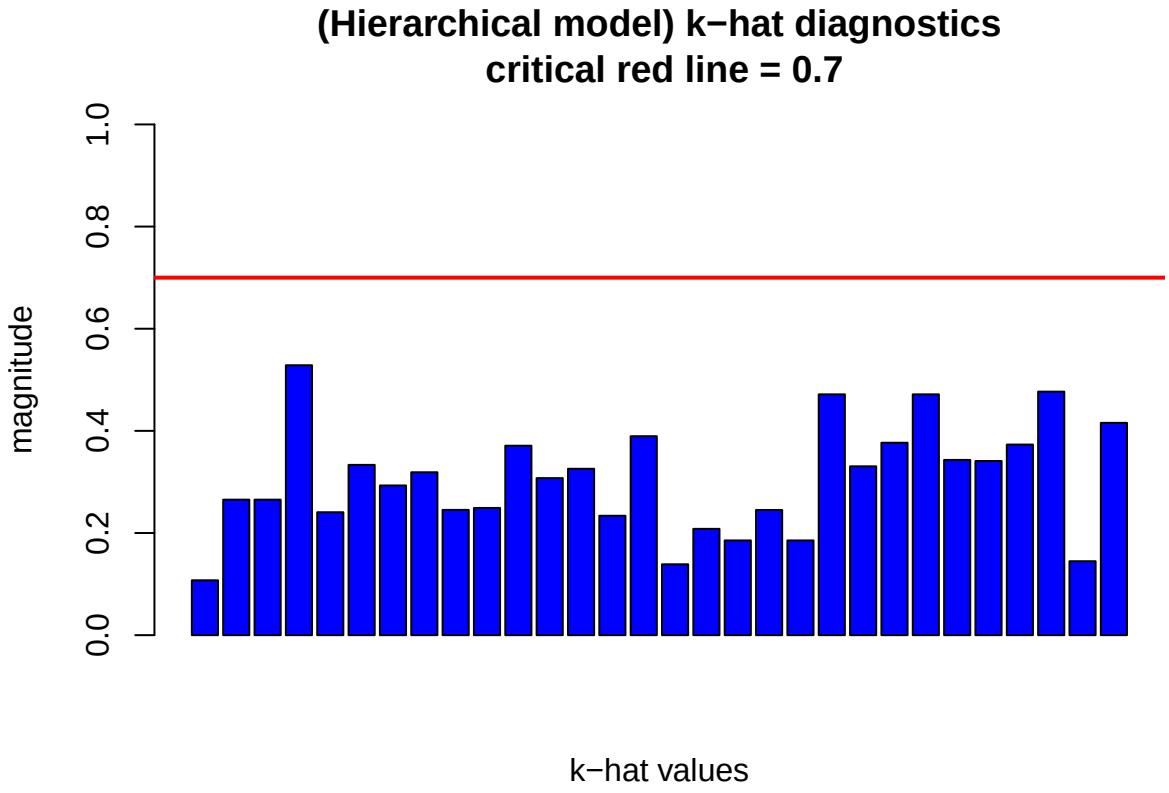
```
pareto_k <- loo_hierarchical$diagnostics$pareto_k

cat("\nThe k-hat values of the hierarchical model is\n")
```

```
##
## The k-hat values of the hierarchical model is
```

```
print(loo_hierarchical)
```

```
##
## Computed from 4000 by 30 log-likelihood matrix
##
##          Estimate  SE
## elpd_loo   -128.8 4.7
## p_loo         8.2 1.7
## looic       257.5 9.5
## ------
## 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%   439
##  (0.5, 0.7]   (ok)        1     3.3%   193
##    (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.
```

```
barplot(pareto_k, main = "(Hierarchical model) k-hat diagnostics\n critical red line = 0.7", xlab = "k-h
abline(h=0.7, col="red", lwd=2)
```



So it appears that the hierachical model is reliable as all $\hat{k}$-values are smaller than 0.7

3.

Compute the effective number of parameters p_{eff} for each of the three models. Hint! The estimated effective number of parameters in the model can be computed from equation (7.15) in the book, where $elpd_{loo-cv}$ is the PSIS-LOO value (sum of the LOO log densities) and lppd is given by equation (7.5) in the book.

Equation (7.15)
$p_{eff} = lppd - lppd_{loo-cv}$

Equation (7.5)
computed lppd = computed log pointwise predictive density
$= \sum_{i=1}^{n} log\left(\frac{1}{S}\sum_{s=1}^{S} p(y_i|\theta^s)\right)$

The priors are similarly used as the parts above

Separate model

```
# A draws matrix of dimension S x N
log_likelihood_separate <- result_separate$draws("log_likelihood", format = "matrix")

S = nrow(log_likelihood_separate) # Number of draws from the model
```

```r
N = ncol(log_likelihood_separate) # Number of observations

lppd = 0 # log pointwise predictive density

for (i in (1:N)){
  col_sum = sum(exp(log_likelihood_separate[, i]))
  lppd = lppd + log((1/S)*col_sum)
}

# lppd_loocv
lppd_loocv = loo_separate$estimates[1][1]

# Number of effective coefficients
p_eff = lppd - lppd_loocv
cat("The p_eff for the separate model:", p_eff)
```

```
## The p_eff for the separate model: 20.03021
```

Pooled model

```r
# A draws matrix of dimension S x N
log_likelihood_pooled <- result_pooled$draws("log_likelihood", format = "matrix")

S = nrow(log_likelihood_pooled) # Number of draws from the model
N = ncol(log_likelihood_pooled) # Number of observations

lppd = 0 # log pointwise predictive density

for (i in (1:N)){
  col_sum = sum(exp(log_likelihood_pooled[, i]))
  lppd = lppd + log((1/S)*col_sum)
}

# lppd_loocv
lppd_loocv = loo_pooled$estimates[1][1]

# Number of effective coefficients
p_eff = lppd - lppd_loocv
cat("The p_eff for the pooled model:", p_eff)
```

```
## The p_eff for the pooled model: 2.531982
```

Hierarchical model

```r
# A draws matrix of dimension S x N
log_likelihood_hierarchical <- result_hierarchical$draws("log_likelihood", format = "matrix")

S = nrow(log_likelihood_hierarchical) # Number of draws from the model
N = ncol(log_likelihood_hierarchical) # Number of observations
```

```
lppd = 0 # log pointwise predictive density

for (i in (1:N)){
  col_sum = sum(exp(log_likelihood_hierarchical[, i]))
  lppd = lppd + log((1/S)*col_sum)
}

# lppd_loocv
lppd_loocv = loo_hierarchical$estimates[1][1]

# Number of effective coefficients
p_eff = lppd - lppd_loocv
cat("The p_eff for the hierarchical model:", p_eff)
```

```
## The p_eff for the hierarchical model: 8.154332
```

4.

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

As concluded from part (2), all of $\hat{k}$-values are lower than 0.7 in the pooled and hierarchical model, making them highly and equally reliable. However, there is some $\hat{k}$-value that is higher than 0.7 in the separate model, making it a little biased.

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.

```
print(loo_compare(x = list(loo_separate=loo_separate, loo_pooled=loo_pooled, loo_hierarchical=loo_hiera
```

```
##                  elpd_diff se_diff
## loo_hierarchical   0.0       0.0
## loo_pooled        -5.8       4.1
## loo_separate     -67.9      10.0
```

```
cat("\nThe PSIS-LOO elpd value of the separate model is\n")
```

```
##
## The PSIS-LOO elpd value of the separate model is
```

```
print(loo_separate$estimates[1][1])
```

```
## [1] -196.6803
```

```
cat("\nThe PSIS-LOO elpd value of the pooled model is\n")
```

```
##
## The PSIS-LOO elpd value of the pooled model is
```

```
print(loo_pooled$estimates[1][1])
```

```
## [1] -134.603
```

```
cat("\nThe PSIS-LOO elpd value of the hierarchical model is\n")
```

```
##
## The PSIS-LOO elpd value of the hierarchical model is
```

```
print(loo_hierarchical$estimates[1][1])
```

```
## [1] -128.7648
```

From the comparison, it is observed that the hierarchical is the best method with regards to both criteria elpd_diff and se_diff. Additionally, the higher the PSIS-LOO elpd value, the better the model. Therefore, the hierarchical model should be chosen as it has the highest PSIS-LOO elpd value.

6.

Both the Stan and R code should be included in your report.
They are already included in this report above.