Bayesian Data Analysis Assignment 2

Benjamin Cox, S1621312

Question 1

A Code for Question 1

A.1 R

../ Q1.R

```r
library(data.table)
library(ggplot2)

library(rstan)
rstan_options(auto_write = TRUE)
#options(mc.cores = parallel::detectCores())
Sys.setenv(LOCAL_CPPFLAGS = '-march=corei7 -mtune=corei7')
options(mc.cores = 1)
library(rstanarm)
library(coda)
library(bayesplot)


#####
#a
avalanches <- fread(file = "data/Avalanches.csv")
avalanches <- avalanches[Rep.events > 0]
avalanches[, ':=' (EADS1 = (Season >= 1994 &
                            Season <= 2003),
                   EADS2 = (Season >= 2004))]

avalanches[Season %in% c(1986, 1994, 2004)]

avalanches[, EWS := 1 + EADS1 + 2 * EADS2]
avalanches[, EWS := as.factor(EWS)]

base_plot <-
  ggplot(data = as.data.frame(avalanches), aes(colour = EWS)) + theme_minimal()
base_plot + geom_line(aes(x = Season, y = Rep.events))
base_plot + geom_line(aes(x = Season, y = Deaths))
base_plot + geom_boxplot(aes(x = EWS, y = Deaths), colour = "black")

cor(avalanches[(EADS1 == FALSE &
                  EADS2 == FALSE), .(Rep.events, Deaths)])
cor(avalanches[EADS1 == TRUE, .(Rep.events, Deaths)])
cor(avalanches[EADS2 == TRUE, .(Rep.events, Deaths)])
#####
#b
to_model <- avalanches[, .(Deaths, Rep.events, EADS1, EADS2)]
model_mat <- model.matrix(Deaths ~ ., data = to_model)#no intercept as cannot have deaths without avalanche

model_mat <- model_mat[,-1]
out_names = colnames(model_mat)
#no need to centre as discrete

#new data
# X_new = matrix(c(1, 20, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1),
#                nrow = 4,
#                byrow = T)
X_new = matrix(c(20, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1),
               nrow = 4,
               byrow = T)
N_new = nrow(X_new)
#check, should be similar
f_glm <-
  glm(Deaths ~ ., data = to_model, family = poisson(link = "log"))
```

```r


stan_poisson_glm <- stan_model(file = "stan/poisson_glm.stan")
stan_poisson_glm_data <-
  list(
    N = nrow(model_mat),
    P = ncol(model_mat),
    y = avalanches$Deaths,
    X = model_mat,
    n_params = c(0, 1e2),
    N_new = N_new,
    X_new = X_new
  )


stan_poisson_glm_s <-
  sampling(
    stan_poisson_glm,
    data = stan_poisson_glm_data,
    chains = 7,
    control = list(adapt_delta = 0.9),
    iter = 3000,
    init_r = 0.1
  )

post_params <- extract(stan_poisson_glm_s, "lambda")[[1]]
colnames(post_params) <- out_names
apply(post_params, 2, summary)

p_pred <- extract(stan_poisson_glm_s, "y_new")[[1]]
mean(p_pred[, 1] < 15)
mean(p_pred[, 2] > 1)
mean(p_pred[, 3] > 1)
mean(p_pred[, 4] > 1)

#####
#dic is bad
#formulae taken from https://en.wikipedia.org/wiki/Deviance_information_criterion
plikrar <- function(x, data) {
  sum(dpois(data, x, log = T))
}
sampling_rates <- extract(stan_poisson_glm_s, "rate")[[1]]
sr_like <- apply(sampling_rates, 1, plikrar, avalanches$Deaths)#calculate log likelihoods of each sampling
sr_like_mean <- mean(sr_like)#calculate mean log likelihood of samples
eap <- colMeans(sampling_rates)#calculate posterior means of rates (not parameters)
p_mean_like <- sum(dpois(avalanches$Deaths, eap, log = T))#calculate log likelihood of EAP
dbar <- -2 * sr_like_mean#expected deviance
pd <- dbar + 2 * p_mean_like#calculate penalty
dic <- pd + dbar#give dic
#####
#prior checking
# dp_av <- avalanches$Deaths/avalanches$Rep.events
# dp_av <- dp_av[!is.nan(dp_av)]
# m_deaths <- mean(dp_av)
# xm <- dp_av - m_deaths
# lnfactor <- 2/(xm)^2
# inffactor <- dp_av / m_deaths
# beta_p <-
# mfc <- exp(xm * inffactor)
# mfc_p <- plnorm(mfc, 0, 2)
avno <- avalanches$Rep.events
avde <- avalanches$Deaths
mede <- mean(avde)
psi <- avde/mede
beta <- log(psi)/(avno - mean(avno))
psi_p <- dlnorm(psi, 0, 2)
beta_p <- dnorm(beta, 0, (avno-mean(avno))^(-2))
#####
stan_poisson_glm_exvar <- stan_model(file = "stan/poisson_glm_exvar.stan")

ym <- data.frame(ym = as.factor(avalanches$Season))
yim <- model.matrix(~ . -1, ym)

stan_poisson_glm_exvar_data <-
  list(
    N = nrow(model_mat),
    P = ncol(model_mat),
    y = avalanches$Deaths,
```

```r
    X = model_mat,
    n_params = c(0, sqrt(10)),
    N_new = N_new,
    X_new = X_new,
    yearindmat = yim,
    N_years = ncol(yim)
  )


stan_poisson_glm_exvar_s <-
  sampling(
    stan_poisson_glm_exvar,
    data = stan_poisson_glm_exvar_data,
    chains = 4,
    control = list(adapt_delta = 0.999),
    iter = 8000,
    init_r = 1
  )

post_params_exvar <- extract(stan_poisson_glm_exvar_s, "lambda")[[1]]
colnames(post_params_exvar) <- out_names
apply(post_params_exvar, 2, summary)

dpp <- extract(stan_poisson_glm_exvar_s, "data_ppred")[[1]]
apply(dpp, 2, summary)
#####
plikrar <- function(x, data) {
  sum(dpois(data, x, log = T))
}
sampling_rates_exv <- extract(stan_poisson_glm_exvar_s, "rate")[[1]]
sr_like_exv <- apply(sampling_rates_exv, 1, plikrar, avalanches$Deaths)#calculate log likelihoods of each sampling
sr_like_mean_exv <- mean(sr_like_exv)#calculate mean log likelihood of samples
eap_exv <- colMeans(sampling_rates_exv)#calculate posterior means of rates (not parameters)
p_mean_like_exv <- sum(dpois(avalanches$Deaths, eap_exv, log = T))#calculate log likelihood of EAP
dbar_exv <- -2 * sr_like_mean_exv#expected deviance
pd_exv <- dbar_exv + 2 * p_mean_like_exv#calculate penalty
dic_exv <- pd_exv + dbar_exv#give dic
#####
```

A.2 Stan

../stan/poisson_glm.stan

```stan
data {
  int<lower=0> N;
  int<lower=0> P;

  int<lower=0> y[N];

  matrix[N, P] X;

  int<lower=0> N_new;
  matrix[N_new, P] X_new;

  vector[2] n_params;
}
transformed data{
}

parameters {
  vector[P] lambda;
}

transformed parameters{
  vector[N] log_rate = X * lambda;
  vector[N_new] log_rate_new = X_new * lambda;
  vector<lower=0>[N] rate = exp(log_rate);
}

model {
  lambda ~ normal(n_params[1], n_params[2]);
  y ~ poisson_log(log_rate);
}

```

```
generated quantities{
  int<lower=0> y_new[N_new] = poisson_log_rng(log_rate_new);
  int<lower=0> data_ppred[N] = poisson_log_rng(log_rate);
}
```

../stan/poisson_glm_exvar.stan

```
data {
  int<lower=0> N;
  int<lower=0> P;

  int<lower=0> y[N];

  matrix[N, P] X;

  int<lower=0> N_new;
  matrix[N_new, P] X_new;

  vector[2] n_params;
}
transformed data{
}

parameters {
  vector[P] lambda;
  real<lower=0,upper=10> theta_hyp;
  real theta;
}

transformed parameters{
  vector[N] log_rate = X * lambda + theta;
  vector[N_new] log_rate_new = X_new * lambda + theta;
  vector<lower=0>[N] rate = exp(log_rate);
}

model {
  theta_hyp ~ uniform(0, 10);
  theta ~ normal(0, theta_hyp);
  lambda ~ normal(n_params[1], n_params[2]);
  y ~ poisson_log(log_rate);
}

generated quantities{
  int<lower=0> y_new[N_new] = poisson_log_rng(log_rate_new);
  int<lower=0> data_ppred[N] = poisson_log_rng(log_rate);
}
```

B R Code for Question 2

../Q2.R

```
library(data.table)
library(ggplot2)
library(dplyr)

library(rstan)
rstan_options(auto_write = TRUE)
#options(mc.cores = parallel::detectCores())
Sys.setenv(LOCAL_CPPFLAGS = '-march=corei7 -mtune=corei7')
options(mc.cores = 1)
library(rstanarm)
library(coda)
library(bayesplot)

#####
#loading and eda
avalanches_prop <- fread(file = "data/Avalanches_part2.csv")
avalanches_prop[, Event_ID := NULL]
avalanches_prop[, Snow_meters := Snow_total / 100]
avalanches_prop[, Snow_fnights := Snow_days / 14]
avalanches_prop[, death_prop := Deaths / Hit]
```

```r
avalanches_prop[, Geo_space := as.factor(Geo_space)]
avalanches_prop[, Rec.station := as.factor(Rec.station)]
cor(avalanches_prop[, .(Season, Snow_meters, Snow_fnights)])
#####
stan_binomial_glm_reff <-
  stan_model(file = "stan/binomial_glm_randomeffects.stan")

submin <- function(x){
  m <- min(x)
  x <- x - m
  attributes(x) <- list("scaled:submin" = m)
  return(x)
}

cont_vars <- c("Snow_meters", "Snow_fnights")#variables to centre
avalanches_prop[,(cont_vars) := lapply(.SD, scale, scale = FALSE), .SDcols = cont_vars]#centre variables
tm_vars <- c("Season")
avalanches_prop[,(tm_vars) := lapply(.SD, submin), .SDcols = tm_vars]


X_fixedeff <-
  model.matrix(death_prop ~ Season + Snow_meters + Snow_fnights - 1, data = avalanches_prop)
X_randomeff <-
  model.matrix(death_prop ~ Geo_space - 1, data = avalanches_prop)
success <- avalanches_prop[, Deaths]
trials <- avalanches_prop[, Hit]


stan_binomial_glm_reff_data <-
  list(
    success = success,
    trials = trials,
    X_f = X_fixedeff,
    X_r = X_randomeff,
    N = length(success),
    P_f = ncol(X_fixedeff),
    P_r = ncol(X_randomeff),
    n_params = c(0, sqrt(10))
  )

stan_binomial_glm_reff_s <-
  sampling(
    stan_binomial_glm_reff,
    data = stan_binomial_glm_reff_data,
    chains = 4,
    control = list(adapt_delta = 0.9),
    iter = 10000#,
    #init_r = 0.1
  )
reff_coda <- As.mcmc.list(stan_binomial_glm_reff_s, pars = c("beta_r", "beta_f"))
gelman.plot(reff_coda, ask = FALSE)

plot_diag_objects <- function(stanfit){
  list(post = as.array(stanfit),
       lp = log_posterior(stanfit),
       np = nuts_params(stanfit))
}

plot_diag <- function(stanfit, pars){
  ps <- vars(starts_with(pars))
  post <- as.array(stanfit)
  lp <- log_posterior(stanfit)
  np <- nuts_params(stanfit)
  p1 <- mcmc_parcoord(post, np = np, pars = ps)
  p2 <- mcmc_pairs(post, np = np, pars = ps)
  p3 <- mcmc_trace(post, pars = ps, np = np)
  p4 <- mcmc_nuts_divergence(np, lp)
  p5 <- mcmc_nuts_energy(np)
  list(p1, p2, p3, p4, p5)
}

#mcmc_trace(stan_binomial_glm_reff_s, pars = vars(starts_with("beta")))

#####
#sans snow fortnights

X_f_nsf <- model.matrix(death_prop ~ Season + Snow_meters - 1, data = avalanches_prop)

```

```r
stan_binomial_glm_reff_nsf_data <-
  list(
    success = success,
    trials = trials,
    X_f = X_f_nsf,
    X_r = X_randomeff,
    N = length(success),
    P_f = ncol(X_f_nsf),
    P_r = ncol(X_randomeff),
    n_params = c(0, sqrt(10))
  )

stan_binomial_glm_reff_nsf_s <-
  sampling(
    stan_binomial_glm_reff,
    data = stan_binomial_glm_reff_nsf_data,
    chains = 4,
    control = list(adapt_delta = 0.9),
    iter = 10000#,
    #init_r = 0.1
  )

#####
#hieratchical on station, sans snow fortnights
X_r_station <- model.matrix(death_prop ~ Rec.station - 1, data = avalanches_prop)

stan_binomial_glm_reff_station_data <-
  list(
    success = success,
    trials = trials,
    X_f = X_f_nsf,
    X_r = X_r_station,
    N = length(success),
    P_f = ncol(X_f_nsf),
    P_r = ncol(X_r_station),
    n_params = c(0, sqrt(10))
  )

stan_binomial_glm_reff_station_s <-
  sampling(
    stan_binomial_glm_reff,
    data = stan_binomial_glm_reff_station_data,
    chains = 4,
    control = list(adapt_delta = 0.9),
    iter = 10000#,
    #init_r = 0.1
  )
```

../stan/binomial_glm.stan

```stan
data {
  int<lower=0> N;
  int<lower=0> P;

  int<lower=0> y[N];

  matrix[N, P] X;

  vector[2] n_params;
}

parameters {
  vector[P] beta;
}

transformed parameters{
  vector[N] lg_p = X * beta;
}

model {
  beta ~ normal(n_params[1], n_params[2]);
  y ~ binomial(1, inv_logit(lg_p));
}
generated quantities{
  int data_ppred[N] = binomial_rng(1, inv_logit(lg_p));
}
```

../ stan/binomial_glm_randomeffects.stan

```
data {
  int<lower=0> N;
  int<lower=0> P_f;
  int<lower=0> P_r;

  int<lower=0> success[N];
  int<lower=1> trials[N];

  matrix[N, P_f] X_f;
  matrix[N, P_r] X_r;

  vector[2] n_params;
}

parameters {
  vector[P_f] beta_f;
  vector[P_r] sn_vec;
  real<lower=0,upper=10> reff_sdv;
}

transformed parameters{
  vector[P_r] beta_r = reff_sdv * sn_vec;
  vector[N] lg_p = X_f * beta_f + X_r * beta_r;
}

model {
  reff_sdv ~ uniform(0, 10);
  sn_vec ~ std_normal(); //hence beta_r ~ normal(0, reff_sdv)
  beta_f ~ normal(n_params[1], n_params[2]);
  success ~ binomial(trials, inv_logit(lg_p));
}
generated quantities{
  int data_ppred[N] = binomial_rng(trials, inv_logit(lg_p));
  vector[N] data_prop = inv_logit(lg_p);
}
```