

# Bayesian Data Analysis Assignment 2

Benjamin Cox, S1621312

## Question 1

### A Code for Question 1

#### A.1 R

../Q1.R

```
1 library(data.table)
2 library(ggplot2)
3
4 library(rstan)
5 rstan_options(auto_write = TRUE)
6 #options(mc.cores = parallel::detectCores())
7 Sys.setenv(LOCAL_CPPFLAGS = '-march=corei7 -mtune=corei7')
8 options(mc.cores = 1)
9 library(rstanarm)
10 library(coda)
11 library(bayesplot)
12
13
14 #####
15 #a
16 avalanches <- fread(file = "data/Avalanches.csv")
17 avalanches <- avalanches[Rep.events > 0]
18 avalanches[, ']:= (EADS1 = (Season >= 1994 &
19                        Season <= 2003),
20                        EADS2 = (Season >= 2004))]]
21
22 avalanches[Season %in% c(1986, 1994, 2004)]
23
24 avalanches[, EWS := 1 + EADS1 + 2 * EADS2]
25 avalanches[, EWS := as.factor(EWS)]
26
27 base_plot <-
28   ggplot(data = as.data.frame(avalanches), aes(colour = EWS)) + theme_minimal()
29 base_plot + geom_line(aes(x = Season, y = Rep.events))
30 base_plot + geom_line(aes(x = Season, y = Deaths))
31 base_plot + geom_boxplot(aes(x = EWS, y = Deaths), colour = "black")
32
33 cor(avalanches[(EADS1 == FALSE &
34                  EADS2 == FALSE), .(Rep.events, Deaths)])
35 cor(avalanches[EADS1 == TRUE, .(Rep.events, Deaths)])
36 cor(avalanches[EADS2 == TRUE, .(Rep.events, Deaths)])
37 #####
38 #b
39 to_model <- avalanches[, .(Deaths, Rep.events, EADS1, EADS2)]
40 model_mat <- model.matrix(Deaths ~ ., data = to_model) #no intercept as cannot have deaths without avalanche
41
42 model_mat <- model_mat[,-1]
43 out_names = colnames(model_mat)
44 #no need to centre as discrete
45
46 #new data
47 # X_new = matrix(c(1, 20, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1),
48 #               nrow = 4,
49 #               byrow = T)
50 X_new = matrix(c(20, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1),
51               nrow = 4,
52               byrow = T)
53 N_new = nrow(X_new)
54 #check, should be similar
55 f_glm <-
56   glm(Deaths ~ ., data = to_model, family = poisson(link = "log"))
```

```

57
58
59 stan_poisson_glm <- stan_model(file = "stan/poisson_glm.stan")
60 stan_poisson_glm_data <-
61   list(
62     N = nrow(model_mat),
63     P = ncol(model_mat),
64     y = avalanches$Deaths,
65     X = model_mat,
66     n_params = c(0, 1e2),
67     N_new = N_new,
68     X_new = X_new
69   )
70
71
72 stan_poisson_glm_s <-
73   sampling(
74     stan_poisson_glm,
75     data = stan_poisson_glm_data,
76     chains = 7,
77     control = list(adapt_delta = 0.9),
78     iter = 3000,
79     init_r = 0.1
80   )
81
82 post_params <- extract(stan_poisson_glm_s, "lambda")[[1]]
83 colnames(post_params) <- out_names
84 apply(post_params, 2, summary)
85
86 p_pred <- extract(stan_poisson_glm_s, "y_new")[[1]]
87 mean(p_pred[, 1] < 15)
88 mean(p_pred[, 2] > 1)
89 mean(p_pred[, 3] > 1)
90 mean(p_pred[, 4] > 1)
91
92 #####
93 #dic is bad
94 #formulae taken from https://en.wikipedia.org/wiki/Deviance_information_criterion
95 plikrar <- function(x, data) {
96   sum(dpois(data, x, log = T))
97 }
98 sampling_rates <- extract(stan_poisson_glm_s, "rate")[[1]]
99 sr_like <- apply(sampling_rates, 1, plikrar, avalanches$Deaths)#calculate log likelihoods of each sampling
100 sr_like_mean <- mean(sr_like)#calculate mean log likelihood of samples
101 eap <- colMeans(sampling_rates)#calculate posterior means of rates (not parameters)
102 p_mean_like <- sum(dpois(avalanches$Deaths, eap, log = T))#calculate log likelihood of EAP
103 dbar <- -2 * sr_like_mean#expected deviance
104 pd <- dbar + 2 * p_mean_like#calculate penalty
105 dic <- pd + dbar#give dic
106 #####
107 #prior checking
108 # dp_av <- avalanches$Deaths/avalanches$Rep.events
109 # dp_av <- dp_av[!is.nan(dp_av)]
110 # m_deaths <- mean(dp_av)
111 # xm <- dp_av - m_deaths
112 # infactor <- 2/(xm)^2
113 # infactor <- dp_av / m_deaths
114 # beta_p <-
115 # mfc <- exp(xm * infactor)
116 # mfc_p <- plnorm(mfc, 0, 2)
117 avno <- avalanches$Rep.events
118 avde <- avalanches$Deaths
119 mede <- mean(avde)
120 psi <- avde/mede
121 beta <- log(psi)/(avno - mean(avno))
122 psi_p <- dlnorm(psi, 0, 2)
123 beta_p <- dnorm(beta, 0, (avno-mean(avno))^(2))
124 #####
125 stan_poisson_glm_exvar <- stan_model(file = "stan/poisson_glm_exvar.stan")
126
127 ym <- data.frame(ym = as.factor(avalanches$Season))
128 yim <- model.matrix(~ . -1, ym)
129
130 stan_poisson_glm_exvar_data <-
131   list(
132     N = nrow(model_mat),
133     P = ncol(model_mat),
134     y = avalanches$Deaths,

```

```

135     X = model_mat,
136     n_params = c(0, sqrt(10)),
137     N_new = N_new,
138     X_new = X_new,
139     yearindmat = yim,
140     N_years = ncol(yim)
141   )
142
143
144   stan_poisson_glm_exvar_s <-
145     sampling(
146       stan_poisson_glm_exvar,
147       data = stan_poisson_glm_exvar_data,
148       chains = 4,
149       control = list(adapt_delta = 0.999),
150       iter = 8000,
151       init_r = 1
152     )
153
154   post_params_exvar <- extract(stan_poisson_glm_exvar_s, "lambda")[[1]]
155   colnames(post_params_exvar) <- out_names
156   apply(post_params_exvar, 2, summary)
157
158   dpp <- extract(stan_poisson_glm_exvar_s, "data_ppred")[[1]]
159   apply(dpp, 2, summary)
160   #####
161   plikrar <- function(x, data) {
162     sum(dpois(data, x, log = T))
163   }
164   sampling_rates_exv <- extract(stan_poisson_glm_exvar_s, "rate")[[1]]
165   sr_like_exv <- apply(sampling_rates_exv, 1, plikrar, avalanches$Deaths)#calculate log likelihoods of each sampling
166   sr_like_mean_exv <- mean(sr_like_exv)#calculate mean log likelihood of samples
167   eap_exv <- colMeans(sampling_rates_exv)#calculate posterior means of rates (not parameters)
168   p_mean_like_exv <- sum(dpois(avalanches$Deaths, eap_exv, log = T))#calculate log likelihood of EAP
169   dbar_exv <- -2 * sr_like_mean_exv#expected deviance
170   pd_exv <- dbar_exv + 2 * p_mean_like_exv#calculate penalty
171   dic_exv <- pd_exv + dbar_exv#give dic
172   #####

```

## A.2 Stan

../stan/poisson\_glm.stan

```

1  data {
2    int<lower=0> N;
3    int<lower=0> P;
4
5    int<lower=0> y[N];
6
7    matrix[N, P] X;
8
9    int<lower=0> N_new;
10   matrix[N_new, P] X_new;
11
12   vector[2] n_params;
13 }
14 transformed data{
15 }
16
17 parameters {
18   vector[P] lambda;
19 }
20
21 transformed parameters{
22   vector[N] log_rate = X * lambda;
23   vector[N_new] log_rate_new = X_new * lambda;
24   vector<lower=0>[N] rate = exp(log_rate);
25 }
26
27 model {
28   lambda ~ normal(n_params[1], n_params[2]);
29   y ~ poisson_log(log_rate);
30 }
31

```

```

32 generated quantities{
33   int<lower=0> y_new[N_new] = poisson_log_rng(log_rate_new);
34   int<lower=0> data_ppred[N] = poisson_log_rng(log_rate);
35 }

```

../stan/poisson\_glm\_exvar.stan

```

1 data {
2   int<lower=0> N;
3   int<lower=0> P;
4
5   int<lower=0> y[N];
6
7   matrix[N, P] X;
8
9   int<lower=0> N_new;
10  matrix[N_new, P] X_new;
11
12  vector[2] n_params;
13 }
14 transformed data{
15 }
16
17 parameters {
18   vector[P] lambda;
19   real<lower=0,upper=10> theta_hyp;
20   real theta;
21 }
22
23 transformed parameters{
24   vector[N] log_rate = X * lambda + theta;
25   vector[N_new] log_rate_new = X_new * lambda + theta;
26   vector<lower=0>[N] rate = exp(log_rate);
27 }
28
29 model {
30   theta_hyp ~ uniform(0, 10);
31   theta ~ normal(0, theta_hyp);
32   lambda ~ normal(n_params[1], n_params[2]);
33   y ~ poisson_log(log_rate);
34 }
35
36 generated quantities{
37   int<lower=0> y_new[N_new] = poisson_log_rng(log_rate_new);
38   int<lower=0> data_ppred[N] = poisson_log_rng(log_rate);
39 }

```

## B R Code for Question 2

../Q2.R

```

1 library(data.table)
2 library(ggplot2)
3 library(dplyr)
4
5 library(rstan)
6 rstan_options(auto_write = TRUE)
7 #options(mc.cores = parallel::detectCores())
8 Sys.setenv(LOCAL_CPPFLAGS = '-march=corei7 -mtune=corei7')
9 options(mc.cores = 1)
10 library(rstanarm)
11 library(coda)
12 library(bayesplot)
13
14 #####
15 #loading and eda
16 avalanches_prop <- fread(file = "data/Avalanches_part2.csv")
17 avalanches_prop[, Event_ID := NULL]
18 avalanches_prop[, Snow_meters := Snow_total / 100]
19 avalanches_prop[, Snow_fights := Snow_days / 14]
20 avalanches_prop[, death_prop := Deaths / Hit]

```

```

21 avalanches_prop[, Geo_space := as.factor(Geo_space)]
22 avalanches_prop[, Rec.station := as.factor(Rec.station)]
23 cor(avalanches_prop[, .(Season, Snow_meters, Snow_fights)])
24 #####
25 stan_binomial_glm_reff <-
26   stan_model(file = "stan/binomial_glm_ranomeffects.stan")
27
28 submin <- function(x){
29   m <- min(x)
30   x <- x - m
31   attributes(x) <- list("scaled:submin" = m)
32   return(x)
33 }
34
35 cont_vars <- c("Snow_meters", "Snow_fights")#variables to centre
36 avalanches_prop[, (cont_vars) := lapply(.SD, scale, scale = FALSE), .SDcols = cont_vars]#centre variables
37 tm_vars <- c("Season")
38 avalanches_prop[, (tm_vars) := lapply(.SD, submin), .SDcols = tm_vars]
39
40
41 X_fixedeff <-
42   model.matrix(death_prop ~ Season + Snow_meters + Snow_fights - 1, data = avalanches_prop)
43 X_ranomeff <-
44   model.matrix(death_prop ~ Geo_space - 1, data = avalanches_prop)
45 success <- avalanches_prop[, Deaths]
46 trials <- avalanches_prop[, Hit]
47
48
49 stan_binomial_glm_reff_data <-
50   list(
51     success = success,
52     trials = trials,
53     X_f = X_fixedeff,
54     X_r = X_ranomeff,
55     N = length(success),
56     P_f = ncol(X_fixedeff),
57     P_r = ncol(X_ranomeff),
58     n_params = c(0, sqrt(10))
59   )
60
61 stan_binomial_glm_reff_s <-
62   sampling(
63     stan_binomial_glm_reff,
64     data = stan_binomial_glm_reff_data,
65     chains = 4,
66     control = list(adapt_delta = 0.9),
67     iter = 10000#,
68     #init_r = 0.1
69   )
70 reff_coda <- As.mcmc.list(stan_binomial_glm_reff_s, pars = c("beta_r", "beta_f"))
71 gelman.plot(reff_coda, ask = FALSE)
72
73 plot_diag_objects <- function(stanfit){
74   list(post = as.array(stanfit),
75         lp = log_posterior(stanfit),
76         np = nuts_params(stanfit))
77 }
78
79 plot_diag <- function(stanfit, pars){
80   ps <- vars(starts_with(pars))
81   post <- as.array(stanfit)
82   lp <- log_posterior(stanfit)
83   np <- nuts_params(stanfit)
84   p1 <- mcmc_parcoord(post, np = np, pars = ps)
85   p2 <- mcmc_pairs(post, np = np, pars = ps)
86   p3 <- mcmc_trace(post, pars = ps, np = np)
87   p4 <- mcmc_nuts_divergence(np, lp)
88   p5 <- mcmc_nuts_energy(np)
89   list(p1, p2, p3, p4, p5)
90 }
91
92 #mcmc_trace(stan_binomial_glm_reff_s, pars = vars(starts_with("beta")))
93
94 #####
95 #sans snow fortnights
96
97 X_f_nsf <- model.matrix(death_prop ~ Season + Snow_meters - 1, data = avalanches_prop)
98

```

```

99 stan_binomial_glm_reff_nsf_data <-
100   list(
101     success = success,
102     trials = trials,
103     X_f = X_f_nsf,
104     X_r = X_randomeff,
105     N = length(success),
106     P_f = ncol(X_f_nsf),
107     P_r = ncol(X_randomeff),
108     n_params = c(0, sqrt(10))
109   )
110
111 stan_binomial_glm_reff_nsf_s <-
112   sampling(
113     stan_binomial_glm_reff,
114     data = stan_binomial_glm_reff_nsf_data,
115     chains = 4,
116     control = list(adapt_delta = 0.9),
117     iter = 10000#,
118     #init_r = 0.1
119   )
120
121 #####
122 #hieratchical on station, sans snow forinights
123 X_r_station <- model.matrix(death_prop ~ Rec.station - 1, data = avalanches_prop)
124
125 stan_binomial_glm_reff_station_data <-
126   list(
127     success = success,
128     trials = trials,
129     X_f = X_f_nsf,
130     X_r = X_r_station,
131     N = length(success),
132     P_f = ncol(X_f_nsf),
133     P_r = ncol(X_r_station),
134     n_params = c(0, sqrt(10))
135   )
136
137 stan_binomial_glm_reff_station_s <-
138   sampling(
139     stan_binomial_glm_reff,
140     data = stan_binomial_glm_reff_station_data,
141     chains = 4,
142     control = list(adapt_delta = 0.9),
143     iter = 10000#,
144     #init_r = 0.1
145   )

```

../stan/binomial\_glm.stan

```

1  data {
2    int<lower=0> N;
3    int<lower=0> P;
4
5    int<lower=0> y[N];
6
7    matrix[N, P] X;
8
9    vector[2] n_params;
10 }
11
12 parameters {
13   vector[P] beta;
14 }
15
16 transformed parameters{
17   vector[N] lg_p = X * beta;
18 }
19
20 model {
21   beta ~ normal(n_params[1], n_params[2]);
22   y ~ binomial(1, inv_logit(lg_p));
23 }
24 generated quantities{
25   int data_ppred[N] = binomial_rng(1, inv_logit(lg_p));
26 }

```

../stan/binomial\_glm\_ranomeffects.stan

```
1 data {
2   int<lower=0> N;
3   int<lower=0> P_f;
4   int<lower=0> P_r;
5
6   int<lower=0> success[N];
7   int<lower=1> trials[N];
8
9   matrix[N, P_f] X_f;
10  matrix[N, P_r] X_r;
11
12  vector[2] n_params;
13 }
14
15 parameters {
16   vector[P_f] beta_f;
17   vector[P_r] sn_vec;
18   real<lower=0,upper=10> reff_sdv;
19 }
20
21 transformed parameters{
22   vector[P_r] beta_r = reff_sdv * sn_vec;
23   vector[N] lg_p = X_f * beta_f + X_r * beta_r;
24 }
25
26 model {
27   reff_sdv ~ uniform(0, 10);
28   sn_vec ~ std_normal(); //hence beta_r ~ normal(0, reff_sdv)
29   beta_f ~ normal(n_params[1], n_params[2]);
30   success ~ binomial(trials, inv_logit(lg_p));
31 }
32 generated quantities{
33   int data_ppred[N] = binomial_rng(trials, inv_logit(lg_p));
34   vector[N] data_prop = inv_logit(lg_p);
35 }
```