Bayesian Hierarchical Weibull Regression

JQT paper with 200 drivers - Model 3

Miao Cai[*]

2018-10-25

1 Weibull Distribution

The probability density function of a Weibull distribution is

$$f(t) = \begin{cases} \frac{\kappa}{\theta}\left(\frac{t}{\theta}\right)^{\kappa-1} e^{(t/\theta)^{\kappa}}, & t > 8 \\ 0, & t \leq 0 \end{cases}$$

Then the survival function is:

$$S(t) = P(T > t) = 1 - P(T \leq t) = e^{-(t/\theta)^{\kappa}}$$

The hazard function is the first order derivative of the survival function:

$$h(t) = \frac{dS(t)}{dt} = \frac{\kappa}{\theta}\left(\frac{t}{\theta}\right)^{\kappa-1}$$

2 Weibull regression

We assume $Y_{i,d(i),s(i)}$ is the time until the first critical event from the start of a trip, then we have

$$\begin{aligned} Y_{i,d(i),s(i)} &\sim \text{WEIBULL}(\kappa, \theta) \\ \theta &= \exp(\beta_{0,d(i)} + \beta_{1,d(i)} \cdot \text{CT}_i + \xi \cdot \mathbf{W} + \nu \cdot \mathbf{D_i}) \end{aligned}$$

Where the θ is the scale parameter and κ is the shape parameter. When $\kappa > 1$, the hazard of having critical event is increasing as cumulative driving time gets longer, which indicates fatigued driving. When $\kappa = 1$, the Weibull distribution becomes an exponential distribution with constant hazard $\frac{1}{\theta}$, which indicates no fatigue. When $0 < \kappa < 1$, the hazard of having critical events is decreasing with longer cumulative driving time, which indicates anti-fatigue, or burn-in.

3 Examples

This is an example of Weibull regression provided by the STAN forum.

3.1 Example code 1

```
rm(list=ls(all=TRUE))
require(foreign)
require(rstan)
require(MASS)
require(VGAM)
```

[*]Department of Epidemiology and Biostatistics, Saint Louis University. Email address miao.cai@slu.edu

```r
require(loo)

time <- c(6, 6, 6, 6, 7, 9, 10, 10, 11, 13, 16, 17, 19, 20, 22, 23, 25, 32, 32, 34, 35,
          1, 1, 2, 2, 3, 4, 4, 5, 5, 8, 8, 8, 8, 11, 11, 12, 12, 15, 17, 22, 23)
drug <- c(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
status <- c(0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)

N_uncensored <- sum(status == 1)
N_censored <- sum(status == 0)

group_uncensored <- drug[status == 1] + 1
group_censored <- drug[status == 0] + 1

t_uncensored <- time[status == 1]
censor_time <- time[status == 0]
M = 2

#------------------------------------------------------------------------------
# THE MODEL.

modelString = "
data {
int<lower=0> N_uncensored;
int<lower=0> N_censored;
int<lower=0> M;
int<lower=1,upper=M> group_uncensored[N_uncensored];
int<lower=1,upper=M> group_censored[N_censored];
real<lower=0> censor_time[N_censored];
real<lower=0> t_uncensored[N_uncensored];
}

parameters {
real<lower=0> r;
real beta[M];
//  t_censored / censor_time
real<lower=1> t2_censored[N_censored];
}

model {
r ~ exponential(0.001);
beta ~ normal(0, 100);
for (n in 1:N_uncensored) {
t_uncensored[n] ~ weibull(r, exp(-beta[group_uncensored[n]] / r));
}
for (n in 1:N_censored) {
t2_censored[n] ~ weibull(r, exp(-beta[group_censored[n]] / r) / censor_time[n]);
}
}

generated quantities {
real median[M];
```

```
real drug;
// real veh_control;

for (m in 1:M)
median[m] = pow(log(2) * exp(-beta[m]), 1/r);
drug = beta[2] - beta[1];
}

" # close quote for modelstring
writeLines(modelString,con="model.txt")

#------------------------------------------------------------------------------
# THE DATA.

dataList = list(
  N_censored = N_censored ,
  N_uncensored = N_uncensored , # BUGS does not treat 1-column mat as vector
  M = M ,
  group_uncensored = group_uncensored ,
  group_censored = group_censored,
  censor_time = censor_time,
  t_uncensored = t_uncensored
)

#------------------------------------------------------------------------------
# INTIALIZE THE CHAINS.
# glmInfo = lm( dataList$y ~ dataList$x) # R func.
# show( glmInfo ) ; flush.console() # display in case glm() has troubles
# b0Init = glmInfo$coef[1]
# bInit = glmInfo$coef[-1]
# should use those from ordered logit to start with
if (FALSE) {
  threshInit = myologit$zeta
  bInit = myologit$coefficients
  initsList =  list(
    b = bInit,
    thresh = threshInit )
}

#------------------------------------------------------------------------------
# RUN THE CHAINS

parameters = c("beta", "drug")  # The parameter(s) to be monitored.
adaptSteps = 500              # Number of steps to "tune" the samplers.
burnInSteps = 500            # Number of steps to "burn-in" the samplers.
nChains = 3                   # Number of chains to run.
initsChains <- list()

if (FALSE) {
  for (i in 1:nChains) {
    initsChains[[i]] <- initsList
  }
}
```

```
numSavedSteps=6000           # Total number of steps in chains to save.
thinSteps=1                  # Number of steps to "thin" (1=keep every step).
nPerChain = ceiling( ( numSavedSteps * thinSteps ) / nChains ) # Steps per chain.

# Burn-in:
cat( "Burning in the MCMC chain...\n" )

# The saved MCMC chain:
cat( "Sampling final MCMC chain...\n" )

time.used <- proc.time()
mcmcSamples <- stan(model_code=modelString, data=dataList, seed = 47306,
                    pars=parameters, chains=nChains, # init=initsChains,
                    iter=nPerChain,
                    warmup=burnInSteps ) # init=initsChains
summary(mcmcSamples)
print(mcmcSamples)
# mcmcChain = as.matrix(mcmcSamples)
# Stop the clock
proc.time() - time.used
if (FALSE) {
  log_lik <- extract_log_lik(mcmcSamples)
  waic(log_lik)
  loo(log_lik)
  save.image("bayesGologitMod01aStan.Rdata")
  sink()
}
```

3.2 Example code 2

```
rm(list=ls(all=TRUE))
require(foreign)
require(rstan)
require(MASS)
require(VGAM)
require(loo)

time <- c(6, 6, 6, 6, 7, 9, 10, 10, 11, 13, 16, 17, 19, 20, 22, 23, 25, 32, 32, 34, 35,
          1, 1, 2, 2, 3, 4, 4, 5, 5, 8, 8, 8, 8, 11, 11, 12, 12, 15, 17, 22, 23)
drug <- c(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
status <- c(0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)

N_uncensored <- sum(status == 1)
N_censored <- sum(status == 0)

group_uncensored <- drug[status == 1]
group_censored <- drug[status == 0]

t_uncensored <- time[status == 1]
censor_time <- time[status == 0]
```

```
M = 1

#------------------------------------------------------------------------------
# THE MODEL.

modelString = "
data {
  int<lower=0> N_uncensored;
  int<lower=0> N_censored;
  int<lower=0> M;
  int<lower=0,upper=1> group_uncensored[N_uncensored];
  int<lower=0,upper=1> group_censored[N_censored];
  real<lower=0> censor_time[N_censored];
  real<lower=0> t_uncensored[N_uncensored];
}

parameters {
  real<lower=0> r;
  vector[M] beta;
  //  t_censored / censor_time
  real<lower=1> t2_censored[N_censored];
}

model {
  r ~ exponential(0.001);
  beta ~ normal(0, 100);
  for (n in 1:N_uncensored) {
    t_uncensored[n] ~ weibull(r, exp(-beta*group_uncensored[n] / r));
  }
  for (n in 1:N_censored) {
    t2_censored[n] ~ weibull(r, exp(-beta*group_censored[n] / r) / censor_time[n]);
  }
}

generated quantities {
  real median[M];
  real drug;
  // real veh_control;

  for (m in 1:M)
  median[m] = pow(log(2) * exp(-beta[m]), 1/r);
}

" # close quote for modelstring
writeLines(modelString,con="model.txt")

#------------------------------------------------------------------------------
# THE DATA.

dataList = list(
  N_censored = N_censored ,
  N_uncensored = N_uncensored , # BUGS does not treat 1-column mat as vector
  M = M ,
```

```
  group_uncensored = group_uncensored ,
  group_censored = group_censored,
  censor_time = censor_time,
  t_uncensored = t_uncensored
)

#------------------------------------------------------------------------------
# INTIALIZE THE CHAINS.
# glmInfo = lm( dataList$y ~ dataList$x) # R func.
# show( glmInfo ) ; flush.console() # display in case glm() has troubles
# b0Init = glmInfo$coef[1]
# bInit = glmInfo$coef[-1]
# should use those from ordered logit to start with
if (FALSE) {
  threshInit = myologit$zeta
  bInit = myologit$coefficients
  initsList =  list(
    b = bInit,
    thresh = threshInit )
}

#------------------------------------------------------------------------------
# RUN THE CHAINS

parameters = c("beta")  # The parameter(s) to be monitored.
adaptSteps = 500              # Number of steps to "tune" the samplers.
burnInSteps = 500            # Number of steps to "burn-in" the samplers.
nChains = 3                   # Number of chains to run.
initsChains <- list()

if (FALSE) {
for (i in 1:nChains) {
  initsChains[[i]] <- initsList
}
}

numSavedSteps=6000           # Total number of steps in chains to save.
thinSteps=1                   # Number of steps to "thin" (1=keep every step).
nPerChain = ceiling( ( numSavedSteps * thinSteps ) / nChains ) # Steps per chain.

# Burn-in:
cat( "Burning in the MCMC chain...\n" )

# The saved MCMC chain:
cat( "Sampling final MCMC chain...\n" )

time.used <- proc.time()
mcmcSamples <- stan(model_code=modelString, data=dataList, seed = 47306,
                    pars=parameters, chains=nChains, # init=initsChains,
                    iter=nPerChain,
                    warmup=burnInSteps ) # init=initsChains
summary(mcmcSamples)
print(mcmcSamples)
```

```r
# mcmcChain = as.matrix(mcmcSamples)
# Stop the clock
proc.time() - time.used
if (FALSE) {
  log_lik <- extract_log_lik(mcmcSamples)
  waic(log_lik)
  loo(log_lik)
  save.image("bayesGologitMod01aStan.Rdata")
  sink()
}
```