# 10 General linear model for a normal response with both continuous and categorical explanatory variables

# ---------------------------------------------------------

The inferential situation considered in this chapter is the relationship between body mass and body length of the asp viper (a small snake) in three populations; Pyrenees, Massif Central, and the Jura mountains. We are particularly interested in population-specific differences of the mass-length relationship, that is, in the interactions between length and population.

# 10.2 Data generation

set.seed(10)

nPops <- 3

nSample <- 10

n <- nPops \* nSample # Total number of data points

x <- rep(1:nPops, rep(nSample, nPops)) # Indicator for population

pop <- factor(x, labels = c("Pyrenees", "Massif Central", "Jura"))

length <- runif(n, 45, 70) # Obs. body length (cm) is rarely less than 45

lengthC <- length-mean(length) # Use centered length

# Build the design matrix of an interactive combination of length and population, inspect that and select the parameter values.

Xmat <- model.matrix(~ pop\*lengthC)

print(Xmat, dig = 2)

beta.vec <- c(80, -30, -20, 6, -3, -4)

# Next, build up the body mass measurements y by adding the residual to the value of the linear predictor, with residuals drawn from a zero-mean normal distribution with a standard deviation of our choice (we take a value of 10 here). The value of the linear predictor is obtained by matrix multiplication of the design matrix (Xmat) and the parameter vector (beta.vec). Our vipers are probably all too fat, but that doesn’t really matter for our purposes.

sigma <- 10 # Choose residual SD

lin.pred <- Xmat[,] %\*% beta.vec # Value of lin.predictor

eps <- rnorm(n = n, mean = 0, sd = sigma) # residuals

mass <- as.numeric(lin.pred + eps) # response = lin.pred + residual

hist(mass) # (not shown) Inspect what we’ve created

par(mar = c(5,5,4,2), cex.lab = 1.5, cex.axis = 1.5)

matplot(cbind(length[1:10], length[11:20], length[21:30]), cbind(mass[1:10], mass[11:20], mass[21:30]), ylim = c(0, max(mass)), ylab = "Body mass (g)", xlab = "Body length (cm)", col = c("Red","Green","Blue"), pch = c("P", "M", "J"), las = 1, cex = 1.6, cex.lab = 1.5, frame = FALSE)

**# Save true values for later comparisons**

truth <- c(beta.vec, sigma)

# 10.3 Likelihood analysis with canned functions in R

summary(out10.3 <- lm(mass ~ pop \* lengthC))

**# Compare estimates with truth**

lm\_est <- c(coef(out10.3), sigma=summary(out10.3)$sigma)

cbind(truth=truth, lm=lm\_est)

# 10.4 Bayesian analysis with JAGS

# In JAGS we find it easier to fit the means parameterization of the model, i.e.,   
# to specify three separate linear regressions for each mountain range.

# The effects, i.e., the differences of the intercepts or slopes with reference to the intercept or   
# the slope in the Pyrenees, are trivially easy to recover as derived parameters in just a few lines

# of JAGS code. This allows for better comparison between input and output values.

**# Bundle and summarize data**

str(dataList <- list(mass = as.numeric(mass), pop = as.numeric(pop), lengthC = lengthC, nPops = nPops, n = n) )

**# Write JAGS model file**

cat(file="model10.4.txt", "

model {

**# Priors**

for (i in 1:nPops){

alpha[i] ~ dnorm(0, 0.0001) # Intercepts

beta[i] ~ dnorm(0, 0.0001) # Slopes

}

sigma ~ dunif(0, 100) # Residual standard deviation

tau <- pow(sigma, -2)

**# Likelihood**

for (i in 1:n) {

mass[i] ~ dnorm(mu[i], tau)

mu[i] <- alpha[pop[i]] + beta[pop[i]]\* lengthC[i]

}

# Define effects relative to baseline level

a.effe2 <- alpha[2] - alpha[1] # Intercept Massif Central vs. Pyr.

a.effe3 <- alpha[3] - alpha[1] # Intercept Jura vs. Pyr.

b.effe2 <- beta[2] - beta[1] # Slope Massif Central vs. Pyr.

b.effe3 <- beta[3] - beta[1] # Slope Jura vs. Pyr.

# Custom comparison

test1 <- beta[3] - beta[2] # Slope Jura vs. Massif Central

}

")

**# Function to generate starting values**

inits <- function(){ list(alpha = rnorm(nPops, 0, 2), beta = rnorm(nPops, 1, 1), sigma = runif(1))}

**# Parameters to estimate**

params <- c("alpha", "beta", "sigma", "a.effe2", "a.effe3", "b.effe2", "b.effe3", "test1")

**# MCMC settings**

na <- 1000 ; ni <- 3000 ; nb <- 1000 ; nc <- 3 ; nt <- 1

**# Call JAGS (ART <1 min), check convergence and summarize posteriors**

out10.4 <- jags(dataList, inits, params, "model10.4.txt", n.iter = ni, n.burnin = nb, n.chains = nc, n.thin = nt, n.adapt = na, parallel = TRUE)

par(mfrow=c(2, 2)); traceplot(out10.4) # not shown

print(out10.4, 3)

**# Compare likelihood with Bayesian estimates and with truth**

jags\_est <- out10.4$summary[c(1,8,9,4,10,11,7),1]

tmp <- cbind(truth=truth, lm=lm\_est, JAGS=jags\_est)

print(tmp, 4)

# 10.6 Bayesian analysis with Stan

# ----------------------------------------------------

library(rstan)

**# Bundle and summarize data**

str(dataList <- list(mass = as.numeric(mass), pop = as.numeric(pop), lengthC = lengthC, nPops = nPops, n = n) )

**# Write Stan model**

cat(file="model10\_6.stan", "

data {

int n; //sample size

int nPops; //number of populations

vector[n] mass; //response

vector[n] lengthC; //covariate

int pop[n]; //population of each observation

}

parameters {

vector[nPops] alpha; //intercepts

vector[nPops] beta; //slopes

real<lower=0> sigma; //residual standard deviation

}

model {

vector[n] mu; //expected value of observations

//Priors

alpha ~ normal(0, 1000);

beta ~ normal(0, 1000);

sigma ~ uniform(0, 100);

//Likelihood

for (i in 1:n){

mu[i] = alpha[pop[i]] + beta[pop[i]] \* lengthC[i];

mass[i] ~ normal(mu[i], sigma);

}

}

generated quantities {

real a\_effe2;

real a\_effe3;

real b\_effe2;

real b\_effe3;

real test1;

a\_effe2 = alpha[2] - alpha[1]; //Intercept Massif Central vs. Pyr.

a\_effe3 = alpha[3] - alpha[1]; //Intercept Jura vs. Pyr.

b\_effe2 = beta[2] - beta[1]; //Slope Massif Central vs. Pyr.

b\_effe3 = beta[3] - beta[1]; //Slope Jura vs. Pyr.

test1 = beta[3] - beta[2]; //Slope Jura vs. Massif Central

}

")

**# Parameters to estimate**

params <- c("alpha", "beta", "sigma", "a.effe2", "a.effe3", "b.effe2", "b.effe3", "test1")

**# HMC settings**

# Fewer iterations are needed than JAGS due to efficient sampler

ni <- 1000 ; nb <- 500 ; nc <- 3 ; nt <- 1

**# Call STAN (ART 34 sec/3 sec)**

system.time(

out10.6 <- stan(file = "model10\_6.stan", data = dataList,

warmup = nb, iter = ni, chains = nc, thin = nt) )

rstan::traceplot(out10.6) # not shown

print(out10.6, dig = 3) # not shown

**# Compare estimates with truth**

stan\_est <- summary(out10.6)$summary[c(1,8,9,4,10,11,7),1]

tmp <- cbind(truth=truth, lm=lm\_est, JAGS=jags\_est, Stan=stan\_est)

print(tmp, 4)

# 10.7 Do-it-yourself MLEs

# ----------------------------------------

The homegrown MLEs.

NLL <- function(param, y, Xmat) {

beta1 <- param[1]

beta2 <- param[2]

beta3 <- param[3]

beta4 <- param[4]

beta5 <- param[5]

beta6 <- param[6]

logsigma <- param[7]

sigma <- exp(logsigma)

mu <- Xmat %\*% c(beta1, beta2, beta3, beta4, beta5, beta6)

L <- dnorm(y, mu, sigma) # Likelihood contr. for 1 observation

LL <- log(L) # Loglikelihood contr. for 1 observation

NLL <- -sum(LL) # NLL for all observations (whole data set)

return(NLL)

}

**# Minimize that NLL to find MLEs and also get SEs**

# Crashes when initialized on all zeroes

inits <- c(mean(mass), rep(0, 5), 1)

names(inits) <- c(names(coef(out10.3)), 'log-sigma')

out10.7 <- optim(inits, NLL, y = mass, Xmat = Xmat, hessian=TRUE, method = "BFGS")

getMLE(out10.7, 4)

# Sweet !

**# Compare estimates with truth and previous estimates**

diy\_est <- c(out10.7$par[1:6], exp(out10.7$par[7]))

tmp <- cbind(truth=truth, lm=lm\_est, JAGS=jags\_est,   
 Stan=stan\_est, DIY=diy\_est)

print(tmp, 4)