Lecture 6: Summarizing posterior distributions

Zachary Marion

2/12/2018

As before, we need to load some packages and set some options prior to running any models:

```
library(rstan)
library(shinystan)
rstan_options(auto_write = TRUE)
options(mc.cores = parallel::detectCores())
source("../utilityFunctions.R")
```

The last bit is to source in some functions that will be handy that I have written or cobbled together from other sources. Make sure that the path to the `.R` script is correct!

Summarizing posterior samples

As a motivating example, suppose 5 people tossed our inflatable model Earth in the air 1–4 times each. Their results were:

N	2	3	3	2	4
W	1	3	3	2	4

Depending on the question of interest, we may want to know things like:

- Which parameter value has the highest posterior probability?
- Which parameter value marks the lower 5% of the posterior probability?
- How much posterior probability lies between some values?
- How much posterior probability lies above some value?

These usually boil down to questions about:

1. point estimates
2. intervals of defined boundries
3. intervals of defined probability mass.

We can answer these questions with a Stan model using a binomial likelihood:

```
data {
  int<lower=0> nObs;  // Total number of observations
  int<lower=0> N[nObs];
  int<lower=0> obs[nObs];   // obs as scalar
  real<lower=0, upper=1> omega;  // mode as input data
  real<lower=2> kappa;  // concentration
}
```

```
parameters {
  real<lower=0, upper=1> p;     // prob. of water
}

transformed parameters {
  real<lower=0> alpha;
  real<lower=0> beta;

  alpha = omega * (kappa - 2) + 1;
  beta = (1 - omega) * (kappa - 2) + 1;
}

model {
  p ~ beta(alpha, beta);                // prior on theta
  obs ~ binomial(N, p);
}
```

Note that, as last time, I am specifying a flat prior on θ in terms of the mode ($\omega = 0.5$) and concentration ($\kappa = 2$)

```
nObs <- 5
N <- c(2, 3, 3, 2, 4)
obs <- c(1, 3, 3, 2, 4)
omega <- 0.5
kappa <- 2

dat <- list(nObs=nObs, N=N, obs=obs, omega=omega, kappa=kappa)

mod1 <- stan(file="06.binomialMode.stan", #path to .stan file
  data=dat,
  iter=2000, # number of MCMC iterations
  chains=4,  # number of independent MCMC chains
  seed=3,
  verbose = FALSE) # get rid of stupid messages and warnings

p <- as.matrix(mod1, "p") # Extract posterior of parameter p
```

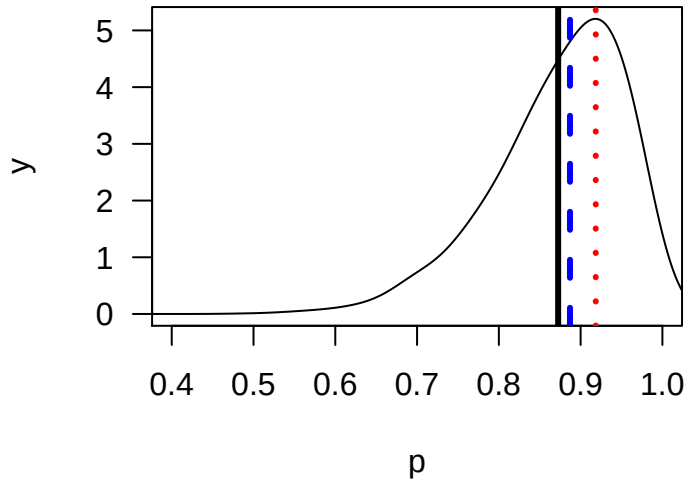
1) Point estimates

Because the posterior is a distribution, it usually makes more sense to think in terms of intervals. However, we like single numbers, so given the entire posterior distribution, which point estimate should we report?

We have already talked about the mean and median and how the median is often more appropriate for skewed samples. We can approximate the value with the highest posterior probability, the *maximum a posteriori* (MAP) estimate.

```r
pDens <- as.data.frame(density(p, from=0, to=1.2, n=1024, adjust=2)[1:2])

# Use which.max to identify the index for the highest density value
MAP <- pDens$x[which.max(pDens$y)]
plot(pDens, xlim = c(0.4,1), xlab="p", main="", las=1, type="l")
abline(v=c(mean(p), median(p), MAP), lty=1:3,
  col=c("black", "blue", "red"), lwd=3)
```



However, MAP estimates should be used with caution.

- Approximation is sample size (and density function algorithm) dependent
- They are not invariant under reparamaterization.

In contrast, the mean and median are optimal under squared-error and linear-error loss functions (we will talk in depth about these later).

Intervals of defined boundries

We may be interested in summarizing the posterior in terms of a frequency above, below, or in between certain values.

- For example, the probability the proportion of water on our globe is less than 0.75 or between 0.8–0.9:

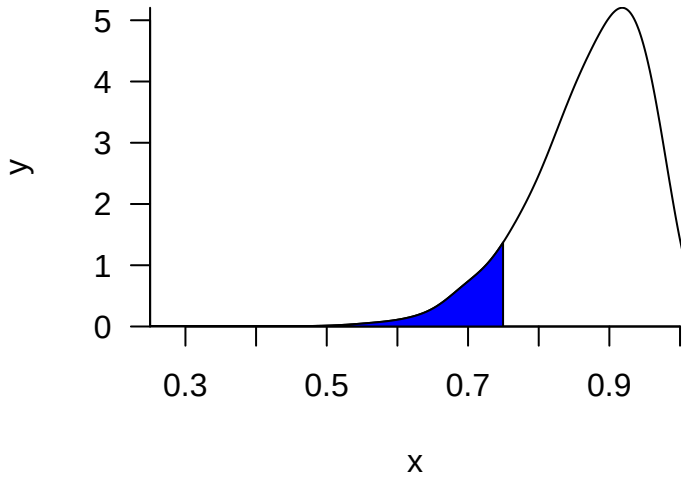
```r
sum(p < 0.75)/length(p) # less than 0.75
```

```
[1] 0.083
```

```r
int <- pDens[pDens$x < 0.75,]
sum(p > 0.8 & p < 0.9)/sum(p)
```

```
[1] 0.4412423
```

```r
plot(pDens, xlim = c(0.25,1), xaxs = "i", yaxs = "i", main="",
  las = 1, type = "l", bty="l")
polygon(x=c(int$x, rev(int$x)), y=c(rep(0,dim(int)[1]), rev(int$y)), col="blue")
```



Intervals of defined probability mass

Usually we are most interested in uncertainty intervals. For example, most people report the middle 95% *Uncertainty Interval* in papers, which is easy to calculate using `quantile`:

```
quantile(p, probs=c(0.025, 0.975))
```

```
     2.5%     97.5%
0.6876961 0.9826254
```

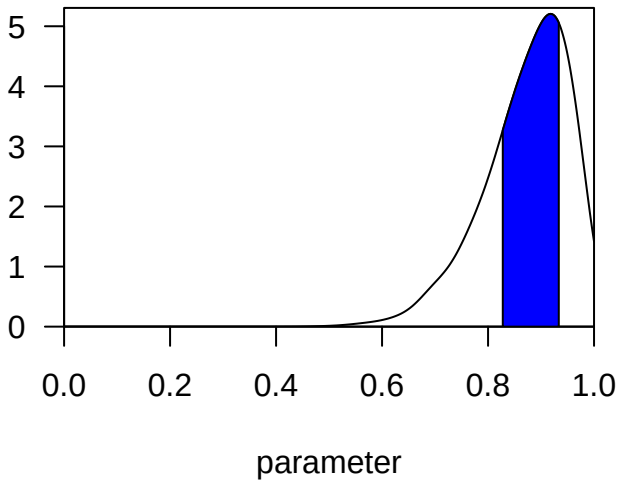
These are not difficult to calculate and do a good job at communicating the shape of the distribution unless the posterior is asymmetrical. However, they can be a bit misleading.

We can use the `plotInterval` function I wrote and plot the 50% CI to see this:

```
round(quantile(p, c(0.25, 0.75)), 2)
```

```
 25%  75%
0.83 0.93
```

```
plotInterval(p, probs=c(0.25,0.75), HDI=FALSE, col="blue")
```



Because equal probability mass is in the upper and lower tails, it excludes parameter values $> \approx 0.93$ despite their having a high probability.

In contrast, *the highest density interval* (HDI) is the narrowest interval containing the specified probability mass.

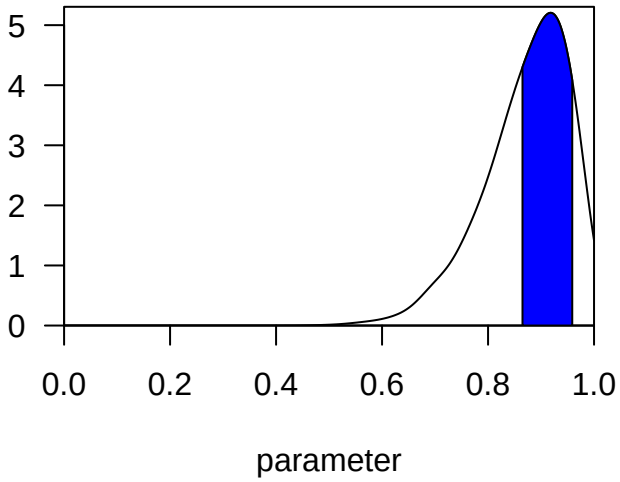
- i.e., the densest interval most consistent with the data

We can calculate the HDI using the utility function I wrote called (suprise) `HDI`.

```
HDI(p, credMass=0.5)
```

```
[1] 0.86 0.96
```

```
plotInterval(p, HDI=TRUE, interval=0.5, col="blue")
```



This interval captures the parameter values with the highest probability and is often narrower than the equal-tailed interval.

In most cases, the equal tailed and HD intervals are similar. It is only when distributions are strongly skewed that they differ.

The disadvantage of the HDI is that it can be computationally intensive and, more important, is more sensitive to simulation variance.

- For example, look at the difference in confidence at the tails when running our previous model for 200 iterations vs the previous 4000 iterations:

```
mod200 <- stan(file="06.binomialMode.stan", #path to .stan file
  data=dat,
  iter=100, # number of MCMC iterations
  chains=4,  # number of independent MCMC chains
  seed=3,
  verbose = FALSE) # get rid of stupid messages and warnings

p200 <- as.matrix(mod200, "p")
HDI(p200,0.5) # 200 iterations
```

```
[1] 0.8826388 0.9646006
```

```r
HDI(p,0.5) # 4000 iterations
```

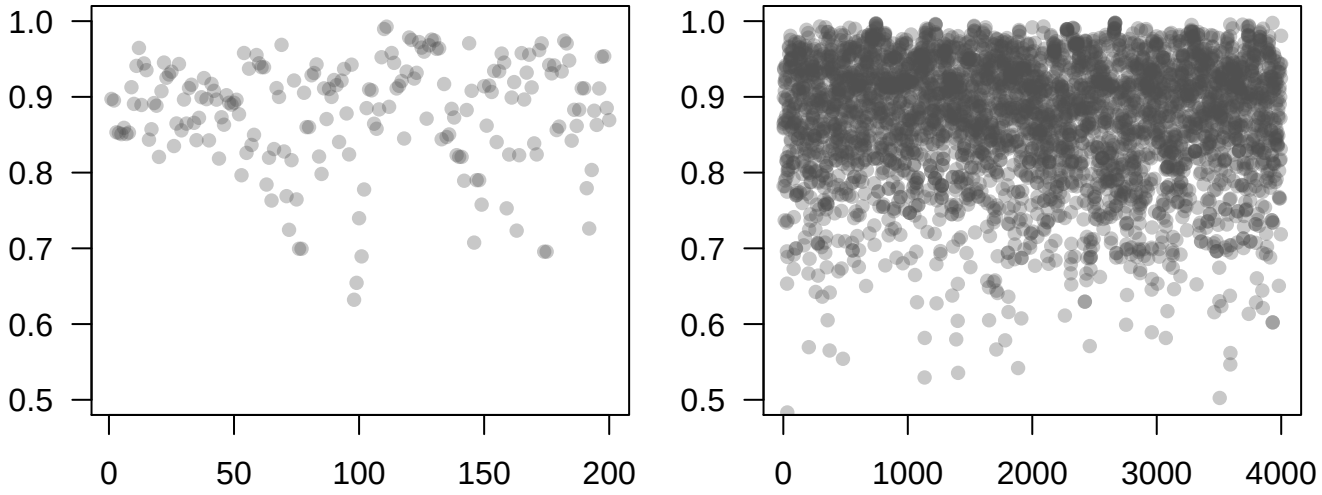
```
[1] 0.8638897 0.9607301
```

If we extract the estimates for p and plot the posterior as a "bird's eye view," the sampling in the tails of the distribution is more sparse, while the point estimates are often similar.

- This is especially a problem for complex hierarchical models

```r
par(mfrow=c(1,2))
par(mar=c(3,3,0.1,0.5))

# using a grey50 with 50% transparency
plot(p200, pch=16, col="#50505050",las=1, ylim=c(0.5,1))
plot(p, pch=16, col="#50505050",las=1, ylim=c(0.5,1))
```



Posterior predictive simulation

Bayesian models make it very easy to simulate new data. This is useful for several reasons, notably

1. Model validation
2. Prediction
3. Power analysis

For model validation, if the model is a good fit then we should be able to use it to generate data that looks like the data we observed. To generate this data, we use the *posterior predictive distribution*:

$$p(\tilde{y}|y) \sim \int p(\tilde{y}|p)p(p|y)dp \tag{1}$$

For each draw of p from the posterior $p(p|y)$ we simulate data $\tilde{y}$ from the posterior predictive distribution $p(\tilde{y}|y)$.

To do posterior predictive simulation, we need to use a new Stan model block: `generated quantities`. This comes after the `model` block.

```
generated quantities {
  int yNew[nObs];       // define new object

  for (n in 1:nObs) {
    yNew[n] = binomial_rng(N[n], p);
  }
}
```

Note that the random number generators that Stan uses are not vectorized, so you have to use loops.

To illustrate the utility of posterior predictive simulations, we are going to simulate some globe tosses. We will also use a very weakly informed `beta(2,2)` prior.

```
set.seed(2)
nObs <- 20
N <- round(runif(nObs, 4,10),digits=0)
obs <- rbinom(nObs, size = N, prob=0.73)
omega <- 0.5
kappa <- 4
dat1 <-  list(nObs=nObs, N=N, obs = obs, omega=omega, kappa=kappa)

modGQ <- stan(file="06.binomialGQ.stan", #path to .stan file
  data=dat1,
  iter=2000, # number of MCMC iterations
  chains=4,  # number of independent MCMC chains
  seed=3,
  verbose = FALSE) # get rid of stupid messages and warnings

yNew <- as.matrix(modGQ, "yNew")
```

The first diagnostic we can use is often called a Bayesian "p-value." For each iteration, we take the average number of times that $y > \tilde{y}$ and then average across iterations. The value should be between 0–1, and ideally, be close to 0.5.

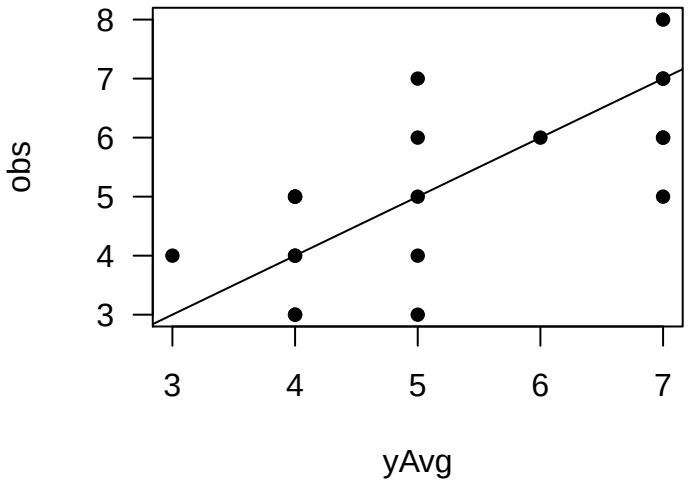
- Values close to 0 or 1 are indicative of a mis-specified model

```
obsMat <- t(replicate(obs,n = nrow(yNew), simplify=TRUE))
mean(apply(obsMat > yNew, 2, mean))
```

```
[1] 0.3912
```

We can also plot the mean or median values of $\tilde{y}$ against the original data. If our model is doing a good job, the data should fall on a a 45 degree line with no systematic over or underestimation.

```
yAvg <- apply(yNew,2, median)
plot(yAvg, obs, las=1, pch=16)
abline(a=0,b=1)
```



The `shinystan` package has a nice interface for posterior predictive simulation.