Lecture 9.3: Model comparison, selection, and averaging

** This lecture is based on chapter 6 of Statistical Rethinking by Richard McElreath. To reproduce some of it, you will need to install the 'rethinking' package from github. The code is below:*

```
install.packages(c('devtools','coda','mvtnorm','loo'))
library(devtools)
install_github("rmcelreath/rethinking")
```

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

WAIC revisited

When we stopped last time we had defined WAIC, the widely applicable (or Wantanabe) information criterion. WAIC is like AIC or DIC except that it is pointwise and fully Bayesian.

We are going to revisit WAIC with a worked example of exactly how to calculate it so that we take some of the mystery out. To do this, we will revisit the `milk.csv` dataset of energy content in primate milk.

- As last time, we will be interested in assessing how primate milk energetics ($kCal \cdot g^{-1}$) vary as a function of body mass (`mass`) and the proportion of the brain comprised of the neocortex (`cort`).

```
milk <- read.csv("milk.csv")
cort <- milk$neocortex.perc/100
obs <- milk$kcal.per.g
mass <- log(milk$mass)

nObs <- nrow(milk)
# priors for later
bMu <- 0; bSD <- 1; sigmaSD <- 2.5
```

For simplicity, we will set up a design matrix of response variables so that we only have to use one stan model:

```
xMat <- as.matrix(model.matrix(~cort + mass))
xMat[1:3,]
```

```
  (Intercept)   cort      mass
1           1 0.5516 0.6678294
2           1 0.6454 1.6582281
3           1 0.6454 1.6808279
```

Note that the first column is our intercept. In the models today, β_1 will refer to the intercept rather than calling it α.

We will use Stan model `mod9.3.stan`. This model is identical to the models that we have constructed thus far, except that our vector of $\beta's$ includes the intercept and so, again, there is no alpha.

The other difference is that we have added something (`log_lik`) to the generated quantities block to calculate the log-likelihood of each observation for each sample of the posterior:

```
generated quantities {
  vector[nObs] log_lik;

  for(i in 1:nObs)
    log_lik[n] = normal_lpdf(obs[n] | mu[n], sigma);
}
```

- The function using `normal_lpdf` calculates the log probability density for a Normal distribution given (`|`) an observation i, mean μ_i, and standard deviation σ.
- We could achieve the same exact thing for observation i by saving μ_i and σ from a given posterior sample iteration and plugging it into `dnorm`.
- If `mu` was a matrix of our posterior estimates that was `nObs` $\times$ `niterations`, and `sigma` was a 4000 length vector, we could calculate the log-likelihood for y_i and posterior sample j outside of Stan by:
  - `log_lik[i,j] <- dnorm(obs[i], mu=mu[i,j], sigma[j], log=TRUE)`.

Worked example

Let's begin by running a simple intercept-only model of the `milk` data, estimating 2 parameters; an overall mean β_1 and σ.

```
X1 <- as.matrix(xMat[,1]) # intercept-only design matrix
nVar <- ncol(X1)
# set up the data as a list:
d1 <- list(nObs=nObs, nVar=nVar, obs=obs, X=X1, bMu=bMu,
```

```
  bSD=bSD, sigmaSD=2.5)

mod1 <- stan(file="mod9.3.stan", data=d1, iter=2000,
  chains=4, seed=867.5309, pars=c("log_lik"))
```

First we need to extract the log-likelihood matrix. I am going to transpose it so that observations are rows and posterior samples are columns:

```
log_lik <- t(as.matrix(mod1, "log_lik"))
```

To compute the log pointwise predictive density (lppd)—the Bayesian deviance—for each observation, we average the samples in each row and then take the log.

- To do this with precision however, we need to do the averaging on a log scale by computing the log of a sum of exponentiated terms. Then we subtract the log number of iterations to get the log of the average.
  - I have a function (`logSumExp`) to do just that

```
logSumExp <- function(x, n.iter) {
  out <- log(sum(exp(x))) - log(n.iter)
  return(out)
}

lppd <- apply(log_lik, 1, logSumExp, n.iter=ncol(log_lik))
```

Typing `sum(lppd)`, which equals 5.68, will give us the lppd defined last time:

$$\hat{\text{lppd}} = \sum_{i=1}^{N} \log \Pr(y_i)$$

where $\Pr(y_i)$ is the average likelihood for obs_i.

The lppd is a pointwise analog of deviance averaged over the posterior. If we multiply it by -2, it would be on the same scale of deviance.

Next we need the effective number of parameters p_{WAIC}, which is the variance in log-likelihood for each observation i in the training sample:

```
pWAIC <- apply(log_lik, 1, var)
```

`sum(pWAIC)` = 1.31 gives us the effective number parameters as defined last time.

$$\hat{p_{WAIC}} = \sum_{i=1}^{N} var(y_i)$$

The *expected log-predictive density (elpd)* is just

$$\text{elpd} = (\hat{\text{lppd}} - \hat{p_{WAIC}})$$

```
(elpd <- sum(lppd) - sum(pWAIC))
```

```
[1] 4.37735
```

and WAIC is

```
-2 * elpd
```

```
[1] -8.7547
```

Because we have an lppd and p_{WAIC} value for each observation, we have a WAIC value for each observation as well.

- This means we have uncertainty in WAIC, which we can calculate as a standard error by:

$$SE_{WAIC} = \sqrt{N \times var(WAIC)}$$

```
waic_vec <- -2 * (lppd - pWAIC)
sqrt(nObs * var(waic_vec))
```

```
[1] 3.73864
```

Using information criteria

Now we know how to calculate AIC/DIC/WAIC for plausable models. But how do we use these metrics in practice?

- Often we talk about **Model Selection**, choosing the model with the lowest IC value and tossing the rest. But sometimes we are discarding information by doing this. Instead, lets talk about model comparison and model averaging (probably won't get here today).

Model comparison

Now let us go back to the milk data and compare our intercept-only model to models with combinations of the other two predictors: the proportion of brain that is neocortex (`cort`) and body mass (`mass`). We will run three additional models for a total of 4:

1. Intercept-only (`mod1`)
2. Intercept + neocortex (`mod2`)
3. Intercept + body mass (`mod3`)
4. Intercept + neocortex + body mass (`mod4`)

```
######### intercept + neocortex
X2 <- as.matrix(xMat[,1:2])
nVar <- ncol(X2)
d2 <- list(nObs=nObs, nVar=nVar, obs=obs, X=X2, bMu=bMu, bSD=bSD, sigmaSD=2.5)

mod2 <- stan(file="mod9.3.stan", data=d2, iter=2000,
  chains=4, seed=867.5309, pars=c("log_lik"))


######### intercept + mass
X3 <- as.matrix(xMat[,c(1,3)])
nVar <- ncol(X3)
d3 <- list(nObs=nObs, nVar=nVar, obs=obs, X=X3, bMu=bMu, bSD=bSD, sigmaSD=2.5)

mod3 <- stan(file="mod9.3.stan", data=d3, iter=2000,
  chains=4, seed=867.5309, pars=c("log_lik"))

######## intercept + neocortex + mass
X4 <- as.matrix(xMat)
nVar <- ncol(X4)
d4 <- list(nObs=nObs, nVar=nVar, obs=obs, X=X4, bMu=bMu, bSD=bSD, sigmaSD=2.5)

mod4 <- stan(file="mod9.3.stan", data=d4, iter=2000,
  chains=4, seed=867.5309, pars=c("log_lik"))
```

To compare models using WAIC, we first need to compute the criterion. Then we can rank models from lowest (best) to highest (worst) and calculate *weights*, which can help provide an interpretable measure of relative distances among models.

To calculate WAIC, we can use the `loo` package:

```
library(loo)
likM1 <- extract_log_lik(mod1) # extract log likelihood
(aicM1 <- waic(likM1))
```

```
Computed from 4000 by 17 log-likelihood matrix

          Estimate  SE
elpd_waic      4.4 1.9
p_waic         1.3 0.3
waic          -8.8 3.7
```

- Note that the function `extract_log_lik` is looking for an object in Stan called `log_lik` specifically. If you have named it something else you need to specify the name with the `parameter_name` call.

We should be gratified that these results are essentially the same results that we got calculating

by hand.

One WAIC value is not very useful. To compare models, we need at least two:

```
likM2 <- extract_log_lik(mod2) # extract log likelihood
(aicM2 <- waic(likM2))
```

```
Computed from 4000 by 17 log-likelihood matrix

          Estimate  SE
elpd_waic      4.1 1.7
p_waic         1.6 0.3
waic          -8.2 3.4
```

Unlike AIC or DIC, we cannot make an assessment of which model is better just by looking at whether the WAIC standard errors overlap.

- This is because—just as parameters are autocorrelated—so are WAIC values. Instead we need to take the difference between each pointwise WAIC value and calculate the SE of the difference.

```
dM1M2 <- aicM1$pointwise[,1] - aicM2$pointwise[,1]
(elpdDiff <- sum(dM1M2))
```

```
[1] 0.2857146
```

```
(seDiff <- sd(dM1M2) * sqrt(nObs))
```

```
[1] 0.4688352
```

Can also use the `compare` function of `loo`:

```
compare(aicM1, aicM2)
```

```
elpd_diff        se
     -0.3       0.5
```

Convienently, Richard McElreath provides a handy function for ranking models by WAIC also called compare:

```
detach("package:loo", unload=TRUE)
```

```
Warning: 'loo' namespace cannot be unloaded:
  namespace 'loo' is imported by 'rethinking' so cannot be unloaded
```

```
library(rethinking)
waicTab <-  compare(mod1, mod2, mod3, mod4)
```

The function returns the models ranked from best to worst, along with the following:

1. `WAIC`: smaller `WAIC` indicates better estimated out-of sample-deviance and so `mod4` is the best.

2. `pWAIC`: the effective number of parameters of each model (i.e., how flexible each model is)
3. `dWAIC` ($\Delta WAIC$): the difference between each WAIC and the best WAIC. For the best one, the difference is zero. Remember only relative differences matter
4. `weight`: the model weight for each model.
5. `SE`: The standard error of the WAIC estimate. Provided N is large enough, the uncertainty in WAIC is well approximated
6. `dSE`: the SE of the differences in WAIC in comparison to the best model.

The *Akaike weights*, `weights`, rescale the WAICs (so they sum to one) and also help for model averaging and assessing the relative predictive accuracy of each model. A weight is calculated as:

$$w_i = \frac{\exp(-\frac{1}{2}\Delta\text{WAIC}_\text{i})}{\sum_{j=1}^{m}\exp(-\frac{1}{2}\Delta\text{WAIC}_\text{j})}$$

We are putting WAIC on a probability scale by undoing the multiplication by -2 and exponentiating to reverse the log transformation.

- Then we standardize by dividing by the total.
- Larger numbers are better.

We can interpret these weights as an *estimate of the probability that the model will make the best predictions of the data,* ***conditional on the set of models considered****.*

In this example, the best model had more than 90% of the model weight. But we only have 17 observations, so the error is substantial.

Also notice that including either predictor variable by itself is expected to do worse at prediction than including neither (i.e., intercept only `mod1`). This is a great example of the masking effect that we talked about a few weeks ago.