Lecture 15: Regularization and information criteria

** 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")
```

Last time we discussed over- and underfitting models and began to discuss information theory, divergence, and deviance.

As a reminder, we want to estimate the divergence of our model Q from the true model P (i.e., estimate the deviance). But this would seemingly require that we know the true model P.

- But if we know the true model why do statistical inference in the first place?

Luckily, we only need be concerned with comparing the divergences of different candidates, say Q and R.

- While we don't know P, we can estimate how far apart Q and R are by comparing the average log-probability from each model to get an estimate of the relative distances of each model from the target.
- The absolute magnitude of the values is not interpretable—neither $E \log q_i$ or $E \log r_i$ suggest a good or bad model by themselves.
- Only the difference $E \log q_i - E \log r_i$ informs us about the divergence of each model from the target P.

An approximation of the K-L divergence (and thus *relative* model fit) is the *deviance*:

$$D(Q) = -2 \sum_i \log q_i$$

where q_i is the likelihood of observation i.The -2 is for historical reasons because, in many cases, the distribution of two deviances has a χ^2-distribution.

Because we are in Bayes land, there is uncertainty about parameters; thus there will also be uncertainty about the deviance of a model.

- For any specific parameter values, there is a defined deviance. But a posterior distribution of parameter values begets a posterior distribution of deviances.

Regularization

So how can we prevent overfitting? One way is to prevent a model from getting too excited about the "training" sample.

When priors are flat or nearly flat, the likelihood (i.e., the data) run the show because the prior suggests every value (or nearly every value) is equally plausible.

- Any particular value therefore has very little prior weight.

We can avoid this by using more "skeptical" priors that slow the rate of learning from the sample.

- The most common way to do this is a *regularizing* or *shrinkage* prior on β coefficients.
- But too much skepticism is bad as well if we underfit the model. So we need to tune the prior so it is mildly skeptical.

Consider the following model:

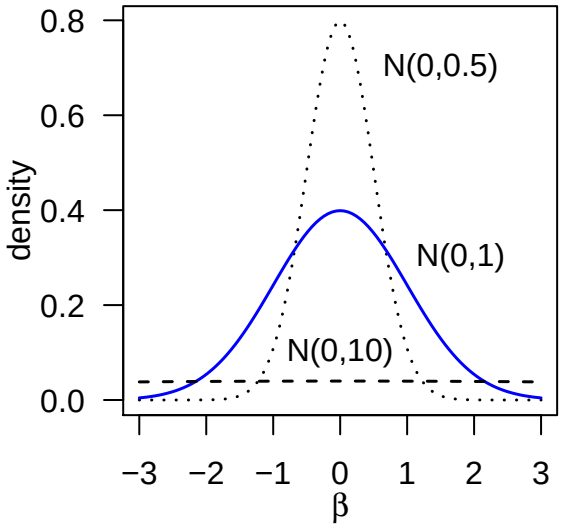
$$\begin{aligned} y_i &\sim \text{Normal}(\mu_i, \sigma) \\ \mu_i &= \alpha + \beta x_i \\ \alpha &\sim \text{Normal}(0, 20) \\ \beta &\sim \text{Normal}(0, 1) \\ \sigma &\sim \text{Cauchy}^{+}(0, 5) \end{aligned}$$

Assume that the predictor x has been standardized so that it's mean is 0 and SD is 1.

The prior on α will depend on the scale of y but should be given enough leeway to swing around both positively and negatively so that it has little impact on inference.

The prior on β, however, is narrower and is meant to shrink. The prior $\beta \sim \text{Normal}(0, 1)$ suggests that, before observing the data, our model should be skeptical of values below and above [-2, 2].

- This is because a normal distribution with a standard deviation of 1 assigns only 5% plausability to values above/below 2 SD.
- can interpret as saying that a change in 1 SD in x is unlikely to produce 2 units change in y.



Here is a figure of three different priors. The blue is $N(0, 1)$. An even more skeptical prior $N(0, 0.5)$ is also shown (dotted line), as is an essentially flat prior $(N(0, 10))$.

How strong to make the prior will depend on the data at hand and the sample size. With more data, even a very strong regularizing prior will be overcome by the likelihood, which is a good thing.

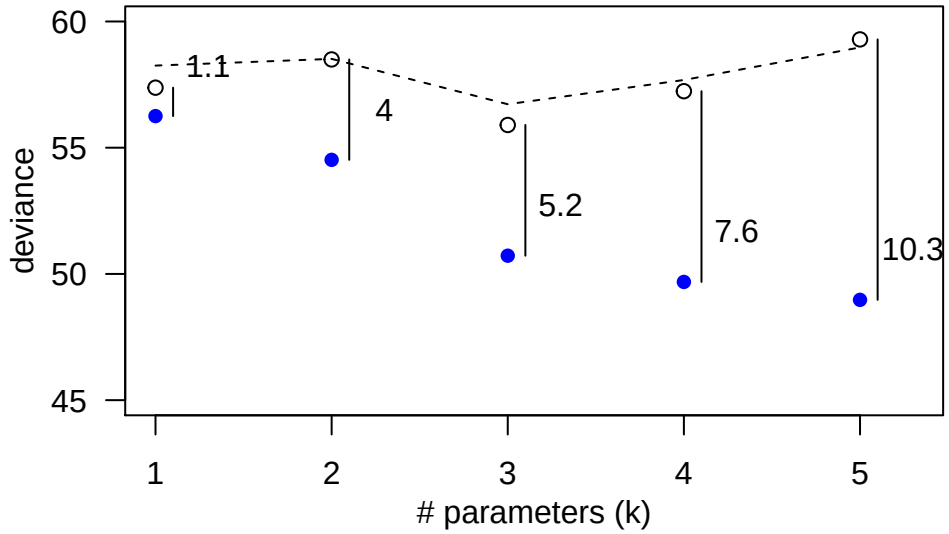
- As more information is added, we need to be less and less skeptical about modeling the noise vs. modeling the process.
- Could also repeatedly *cross-validate*, holding out part of the data, fitting the model, and then using that model on the saved data. But often we don't have enough data to do this.

What we might want as well is a way to use all of our data in a single model and forcast a model's out-of-sample deviance.

Information criteria

```
N <- 20
kseq <- 1:5
nSims <- 500

# If you have a mac set this
cores <- 4

dev <- sapply(kseq, function (k) {
#  print(k);
# If you have a mac use this
  r <- mcreplicate( nSims, sim.train.test(N=N, k=k), mc.cores=cores)
# If you don't have a mac use this
#  r <- replicate(nSims, sim.train.test( N=N, k=k ));
  c( mean(r[1,]), mean(r[2,]), sd(r[1,]), sd(r[2,]) )
} )
```

```r
par(mar=c(3,3.2,0.1,0.5))
plot(1:5, dev[1,], ylim=c(45,60),
  xlim=c(1,5.3), las=1, ann=FALSE, pch=16, col="blue")

mtext(text = "# parameters (k)", side=1, line = 2, cex=1)
mtext(text = "deviance", side=2, line = 2.2, cex=1)

points((1:5), dev[2,])
lines(kseq, dev[1,] + 2*kseq, lty=2)
for (i in kseq) {
  lines(c(i,i)+0.1, c(dev[1,i], dev[2,i]))
  text(i+0.3, dev[2,i] - (dev[2,i]-dev[1,i])+2,
    labels = eval(round(dev[2,i]-dev[1,i],1)))
}
```



Here is a plot of the in-sample (blue) vs. out-of-sample (black) deviances for the simulations we did last time.

- The lines measure the average distance between the training deviance and test deviance. Notice that each distance is approximately 2x the number of parameters labeled on the x-axis.
- The dashed line shows exactly the blue points + 2x the number of parameters.

This is the phenomenon behind *information criteria*. The most widely known information criterion is the *Akaike Information Criteria (AIC)*, which provides a simple measure of out-of-sample deviance:

$$\text{AIC} = D_{train} + 2k$$

where k is the number of free parameters estimated. In the above figure, the dashed lines trace out the AIC values of the models.

All information critera aim to provide an approximation of predictive accuracy measured by out-of-sample deviance. AIC is the oldest and most restrictive, being reliable only when:

1. The priors are flat or completely driven by the likelihood;

2. The posterior is approximately multivariate normal;
3. The sample size N is much greater than the number of parameters, k.

Flat priors are not usually the best, as discussed before, so we want something a bit better.

Deviance Information Criterion (DIC)

DIC is quite popular in Bayesian statistics. DIC is essentially a relaxed version of AIC that allows for informative priors.

- However, DIC requires a multivariate normal posterior, and so if the posterior is skewed, then DIC can give misleading inferences.

DIC is calculated from the posterior distribution of training deviance. Given that the parameters have a posterior, now so does the deviance.

We will define D now as the posterior distribution of deviance.

1. Compute deviance for each set of parameter values in the posterior distribution.
    - If we have 4,000 posterior iterations, we would have 4,000 deviance values.
2. Let $\bar{D}$ = the average of D.
3. Let $\hat{D}$ = the deviance calculated at the posterior mean.
    - i.e., calculate the mean of each parameter value then plug those averages into the deviance formula

DIC is calculated as

$$\text{DIC} = \bar{D} + (\bar{D} - \hat{D}) = \bar{D} + p_d.$$

$\bar{D} - \hat{D} = p_d$ is equivalent to the number of parameters used in calculating AIC.

- It is an "effective" number of parameters that measures how flexible the model is in fitting the training sample.
    - More flexible models are more prone to overfitting
    - p_d, often called the penalty term, is the expected distance between the in-sample deviance and out-of-sample deviance.

With flat priors, DIC reduces to AIC, but usually p_d is a fraction of the actual number of parameters because of regularizing priors that keep model overfitting in check.

WAIC

WAIC—the Widely Applicable or Watanabe Information Criterion—is another IC that we will use a lot. WAIC is more flexible than DIC because it doesn't require a multivariate Gaussian posterior, and it is often more accurate.

WAIC is *pointwise*. The prediction uncertainty is considered sample-by-sample or case-by-case.

- WAIC handles uncertainty where it matters, for each independent observation, then sums up across observations

- This is useful because often models do a better job estimating some data more than others.

We will define $\Pr(y_i)$ as the average likelihood of observation i.

- Do this by computing the likelihood of y_i for each set of parameters in each iteration of the posterior.
- Then we average the likelihoods for each observation i and sum over all observations. This gives us the *log-pointwise-predictive-density (lppd)*:

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

that is, *the log-pointwise-predictive-density is the total, across observations, of the logarithm of the average likelihood of each observation.*

- 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}. Call $var(y_i)$ the variance in log-likelihood for each observation i in the training sample,

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

Now WAIC is defined as

$$\text{WAIC} = -2(\text{lppd} - p_{WAIC})$$

Also note that because WAIC is pointwise, we can get a WAIC value for each observation. Not only do we get a WAIC value for each observation, we can get the standard error of the WAIC values; thus we get uncertainty in our model comparison estimates.

We are going to think about WAIC using 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 `15.multMod.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="15.multMod.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 5.72, 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)` = 1.36 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] 4.359904
```

and WAIC is

```
-2 * elpd
```

```
[1] -8.719808
```

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] 3.840982
```

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.

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="15.multMod.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="15.multMod.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="15.multMod.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         1.4 0.3
waic          -8.7 3.8
```

- 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      4.0 1.7
p_waic         1.7 0.3
waic          -8.0 3.5
```

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] 0.3804202
```

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

```
[1] 0.4699162
```

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

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

```
elpd_diff        se
     -0.4       0.5
```

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

```
detach("package:loo", unload=TRUE)
library(rethinking)
(waicTab <-  compare(mod1, mod2, mod3, mod4))
```

```
      WAIC pWAIC dWAIC weight   SE  dSE
mod4 -15.3   2.6   0.0   0.91 4.92   NA
mod3  -8.9   2.0   6.4   0.04 4.05 2.18
mod1  -8.7   1.4   6.6   0.03 3.73 3.89
mod2  -8.0   1.7   7.3   0.02 3.37 4.19
```

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.