Today I'll expand on a topic that is very important and very non-standardized in Bayesian statistics - model selection! I've already discussed DIC/WAIC in previous tutorials (specifically my "Linear Regression 2" tutorial) but today I'll discuss indicator variable selection. This is also referred to as "IV selection" or the "spike and slab method" depending on how formal you want to be. I'll be using it here in an occupancy model, but it can be applied to any scenario where you want to turn predictors on/off.

Before you use NIMBLE make sure R, and Rtools or Xcode are updated on your computer (otherwise a weird "shared library" error can come up). JAGS is downloadable here: https://sourceforge.net/projects/mcmc-jags/ and NIMBLE can be found here: https://r-nimble.org/download.

Please send any questions or suggestions to heather.e.gaya@gmail.com or find me on twitter @doofgradstudent

Contents

1 A Fake Scenario

We have 40 point count sites in our study area, each one visited 6 times throughout our study period. We recorded present/absent data for Canada Warblers (CAWA) as well as noise levels at the time of detection, and elevation and average rainfall at each site. We want to know if bird presence is best predicted by elevation, rainfall, both variables or neither. We will use a model selection techniques to try to estimate which model is the best predictor and get a model averaged estimate of total occupancy.

Here's a quick look at our site data:

	Site	Elevation	Rainfall
1	1	0.75	10.25
2	2	0.71	0.56
3	3	0.35	2.47

and at our bird data:

	Site	Obs1	Obs2	Obs3	Obs4	Obs5	Obs6	noise1	noise2	noise3	noise4	noise5	noise6
1	1	0	0	0	0	0	0	15.02	9.61	13.41	13.49	6.09	3.13
2	2	1	0	0	0	0	0	0.76	6.58	19.07	6.31	18.66	11.91
3	3	0	0	0	0	0	0	18.25	8.33	0.12	7.31	12.79	18.50

2 General Concept of IV selection

The basic idea of indicator variable selection is that we want to know which combination of predictor variables in our model produce the best model fit. We could just run the model in 4 different ways to test out 4 different model combinations and then look at some sort of goodness of fit test - but that's computationally expensive (aka it takes time) and there isn't always a "go to" goodness of fit test available.

Instead, we ask the model to do the work for us. At each iteration, JAGS is "trying out" possible values for all the parameters - sometimes that means elevation is used to model occupancy, sometimes rainfall, sometimes both (and sometimes neither!). Each time it runs an iteration, it looks at the likelihood that the data came from a distribution that relies on those predictors. *IF* one model really is a better model, we would expect the data to have a much higher likelihood under that model - a much better "fit". If a better model is "found", that combo of variables will be accepted more often than other combinations of our predictor variables. Then, after the model runs, we can ask "how many times was this model "accepted"?" In theory, the best model should be selected the most often. Even more cool, the posterior estimates of the parameters will be modeled averaged!

3 IV Selection in JAGS

First we need to think about what models we want to compare. In this case, we want to know if site occupancy (ψ) is best predicted by elevation, rainfall, both, or neither. If we

thought both were involved we would normally just write something like:

```
for (i in 1:n.sites) { #40 different sites
    logit(psi[i]) <- psi.b0 + psi.b1*elevation[i] + psi.b2*rainfall[i]
    occ[i] ~ dbern(psi[i])
   #and then the rest of the model
```

But in this case, we want to know - do we really need $psi.b1$? or $psi.b2$? So we add in switches that can force $psi.b1$ and $psi.b2$ to be 0.

```
for (i in 1:n.sites) { #40 different sites
logit(psi[i]) <- psi.b0 + switch[1]*psi.b1*elevation[i] + switch[2]*psi.b2
   *rainfall[i]
   occ[i] ~ dbern(psi[i])
#and then the rest of the model
```

Now we need to make sure that these switches are binary - they can only be 1 or 0. This means it's time for a Bernoulli draw! Of course, going in we have no idea what probability there is that one variable is on or off - if we did we wouldn't need to test it! So we'll make sure we draw from a Bernoulli with a uniform prior and ask JAGS to estimate how often the switches should be on/off.

So in our priors we need to write:

```
for (k in 1:2){
    switch[k] ~ dbern(p.w[k])  #is the switch on or off?
        p.w[k] ~ dunif(0,1) #probability the switch is on
    }
```

The other thing we have to worry about is what happens when the switch is off - for instance, what does $psi.b1$ do when switch 1 is off? If we just let it be any value it would mess up our model averaging because ANY value could be accepted when what we really want is for the beta to be 0. That's not what we want at all. So we need to make sure that when the switch is off that the beta is also 0. We can do this by drawing our priors from a normal distribution centered on 0 with a precision that changes depending on the value of the switch:

```
psi.b1 ~ dnorm(0, prec[1])
psi.b2 ~ dnorm(0, prec[2])

  for (k in 1:2){
    prec[k] <- (1-switch[k])*(1000) + switch[k]*(.37)
    }
```

When the switch is on (= 1), the precision of the prior switches to .37 - a vague prior on the logit scale. When the switch is off (= 1), the precision is now 1000, essentially ensuring the beta will be 0.

We'd also like to know which model was chosen at each iteration so that we can get an idea of model weights. We can create 4 models with a little bit of algebra:

```
1 + switch[1] + 2*switch[2]
```

In JAGS (and NIMBLE, and R), TRUE = 1 and FALSE = 0. So we can just ask JAGS, "is b = to this model number?" and if it is, it returns a 1 for that model. Otherwise it returns a 0.

```
for (b in 1:4){
    m[b] <- (b == 1 + switch[1] + 2*switch[2])
  }
```

Finally we need to deal with detection in our model, which is fairly straightforward:

```
for (i in 1:n.sites) { #40 different sites
for (t in 1:6) {
      logit(p[i,t]) <- p.b0 + p.b1*noise[i,t]
      obs[i,t] ~ dbern(p[i,t] * occ[i])
   }
  p.b0 ~ dnorm(0, 0.37)
  p.b1 ~ dnorm(0, 0.37)
```

We can put all this together into one big model.

```

modelstring.occ = "
model {
  ## Loop over sites
  for (i in 1:n.sites) { #40 different sites

  # Linear model for true occupancy; we'll call it psi to follow convention
    logit(psi[i]) <- psi.b0 + switch[1]*psi.b1*elevation[i] + switch[2]*psi.b2*rainfall[i] #add in switches
    occ[i] ~ dbern(psi[i])


#Now we need to deal with the detection process

    ## Loop over replicates within site
    for (t in 1:6) {
      logit(p[i,t]) <- p.b0 + p.b1*noise[i,t]

      #The actually observed data is obs[i,t]
      obs[i,t] ~ dbern(p[i,t] * occ[i])
   }
  }

  totalocc <- sum(occ[]) #this just says sum across all the sites in occ

  ## Priors
  psi.b0 ~ dnorm(0, 0.37)

  p.b0 ~ dnorm(0, 0.37)
  p.b1 ~ dnorm(0, 0.37)

  psi.b1 ~ dnorm(0, prec[1])
  psi.b2 ~ dnorm(0, prec[2])

  for (k in 1:2){ #one for each variable that controls psi
    # Create 'slab & spike' precision
    prec[k] <- (1-switch[k])*(1000) + switch[k]*(.37)

    # Sampling and prior for indicator variable (switch)
```

```
    switch[k] ~ dbern(p.w[k])  #is the switch on or off this iteration?
    p.w[k] ~ dunif(0,1) #probability the switch is on
  }

  # Converts the indicator variable pair into a model number
  for (b in 1:4){
    m[b] <- b == 1 + switch[1] + 2*switch[2]
  }
}
"
```

Time to run it through runjags as always:

```
jd.Foo <- list(n.sites = nrow(Birds), elevation = Sites$Elevation,
  rainfall = Sites$Rainfall,
  noise = as.matrix(Birds[,8:13]), obs = as.matrix(Birds[,2:7]))

ji.Foo <- function(){list(psi.b0 = rnorm(1, 0, 0.1),
    psi.b1 = rnorm(1, 0, 0.1),
    psi.b2 = rnorm(1, 0, 0.1), switch = rep(0,2),
    p.w = rnorm(2, 0.5, 0.1), p.b0 = runif(1,0,1),
      occ = rep(1,nrow(Birds)))}

# Parameters to estimate
params <- c("psi.b0", "psi.b1", "psi.b2", "p.b0",
  "p.b1", "switch", "m", "p.w", "totalocc")

#Run the model
Foo <- run.jags(model = modelstring.occ, monitor = params,
    data = jd.Foo, n.chains = 3, inits = ji.Foo,
    burnin = 4000, sample = 12000,
     adapt = 1000, method = "parallel")
summary(Foo)
```

	Lower95	Median	Upper95	Mean	psrf
psi.b0	-2.91	-0.83	1.66	-0.71	1.01
psi.b1	-0.08	2.35	4.43	2.31	1.01
psi.b2	-0.26	-0.01	0.09	-0.03	1.02
p.b0	-0.03	0.97	2.07	0.98	1.00
p.b1	-0.77	-0.51	-0.30	-0.52	1.00
m[1]	0.00	0.00	1.00	0.07	1.01
m[2]	0.00	1.00	1.00	0.76	1.01
m[3]	0.00	0.00	0.00	0.01	1.09
m[4]	0.00	0.00	1.00	0.17	1.01
p.w[1]	0.18	0.68	1.00	0.64	1.00
p.w[2]	0.00	0.35	0.88	0.39	1.00
totalocc	18.00	25.00	33.00	25.63	1.00

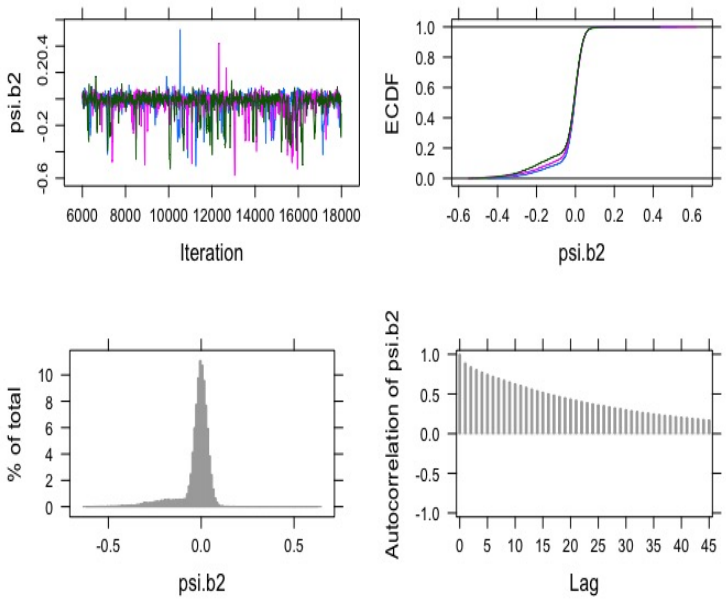
We can look at the mean values of the m parameters and the posterior distribution of the $p.w$ values to tell us our model "weights" In this case, mod 1 (no covariates for psi) was chosen about 7% of the time, mod2 (elevation) about 76%, mod 3 (rainfall) about 1% and

mod 4 (elevation + rainfall) about 17%. From this, we can see that model 2 is likely the best model! (And fun fact, it is also the model I simulated this data from.)

The parameter $p.w$ tells us how often that predictor was used in any model. In this case, elevation was used about 64% of the time, whereas rainfall was "on" about 39% of the time. These values aren't particularly useful for anything.

Finally, we can see that our total occupancy is estimated to fall between 18-33 sites with a point estimate of 26 sites. The true value is 25 sites, so this is a pretty decent estimate!

If we plot our mcmc chains (which we should always do!), they look a little strange compared to what we're used to:



You'll notice some of the plots look weird - almost like they didn't converge. This is why this method is sometimes called the "slab and spike" method. When the beta values ($psi.b1$, $psi.b2$) are turned off, they are forced to 0, causing strange jumps and horizontal lines in the MCMC chains. This is to be expected from this method - though when in doubt more iterations don't hurt.

4 IV Selection in NIMBLE

While writing this tutorial I learned there are a few key words to stay away from when naming parameters in NIMBLE. One of them is apparently the word "switch." Apparently this phrase is used in user-defined functions within NIMBLE so the program will throw a compilation error ("failed to create shared library") if you use the word switch in an unexpected way. For this reason, we'll change the variable to "sw" instead. We'll also need to be sure not to leave any empty brackets as NIMBLE won't like those. Otherwise the model is the same.

```
nimble.birds <- nimbleCode({

  ## Loop over sites
  for (i in 1:n.sites) { #40 different sites

  # Linear model for true occupancy; we'll call it psi to follow convention
    logit(psi[i]) <- psi.b0 + sw[1]*psi.b1*elevation[i] + sw[2]*psi.b2*rainfall[i] #add in switches
    occ[i] ~ dbern(psi[i])


#Now we need to deal with the detection process

    ## Loop over replicates within site
    for (t in 1:6) {
      logit(p[i,t]) <- p.b0 + p.b1*noise[i,t]

      obs[i,t] ~ dbern(p[i,t] * occ[i])
   }
  }

  totalocc <- sum(occ[1:n.sites]) #have to change indexing for NIMBLE

  ## Priors
  psi.b0 ~ dnorm(0, 0.37)

  p.b0 ~ dnorm(0, 0.37)
  p.b1 ~ dnorm(0, 0.37)

  psi.b1 ~ dnorm(0, prec[1])
  psi.b2 ~ dnorm(0, prec[2])

  for (k in 1:2){ #one for each variable that controls psi

    prec[k] <- (1-sw[k])*(1000) + sw[k]*(.37)

    sw[k] ~ dbern(p.w[k])  #is the switch on or off this iteration?
    p.w[k] ~ dunif(0,1) #probability the switch is on
  }

  for (b in 1:4){
    m[b] <- ((1 + sw[1] + 2*sw[2]) == b)
```

```
      }

})
```

We'll next need to give NIMBLE the data, constants, initial values and parameters. We need to make sure the word "switch" is changed to "sw" in our parameter and initial lists as well.

```
data <- list(obs = as.matrix(Birds[,2:7]))
```

```
constants <- list(n.sites = nrow(Birds), elevation = Sites$Elevation,
    rainfall = Sites$Rainfall,
                noise = as.matrix(Birds[,8:13]))
```

```
inits <- list(psi.b0 = rnorm(1, 0, 0.1), psi.b1 = rnorm(1, 0, 0.1),p.b1 =
    rnorm(1, 0, 0.1),
              psi.b2 = rnorm(1, 0, 0.1), sw = rep(0,2), #start with both
    off
              p.w = rnorm(2, 0.5, 0.1), p.b0 = runif(1,0,1), occ = rep(1,
    nrow(Birds)))

```

```
params <- c("psi.b0", "psi.b1", "psi.b2", "p.b0", "p.b1", "sw", "m", "p.w"
    , "totalocc")
```

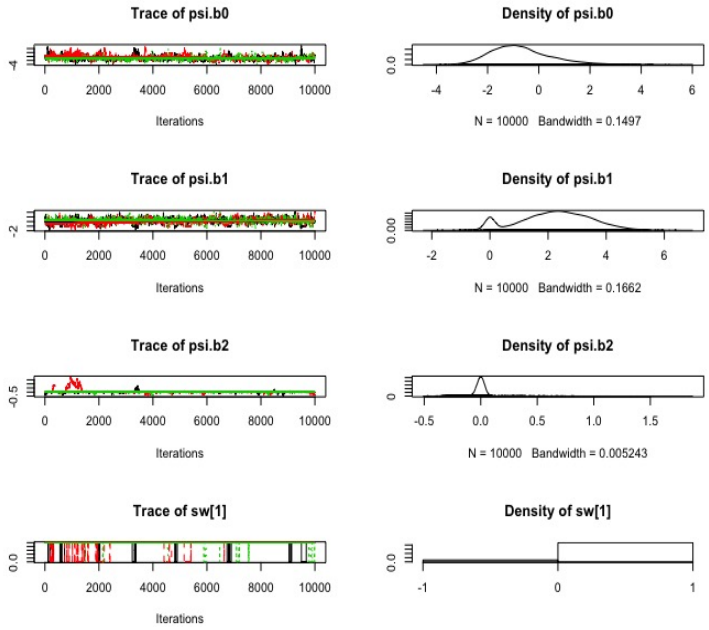
We'll run it the speedy way (one command) for ease:

```
b1 <- nimbleMCMC(code = nimble.birds, constants = constants,
                 data = data, inits = inits, monitors = params,
                 niter = 30000, thin = 1, nchains =3, nburnin = 20000,
                 samplesAsCodaMCMC = TRUE)
```

Our summary should be fairly similar to the answers we found from JAGS:

	Mean	SD
m[1]	0.05	0.23
m[2]	0.78	0.41
m[3]	0.01	0.11
m[4]	0.15	0.36
p.b0	0.96	0.53
p.b1	-0.52	0.12
p.w[1]	0.64	0.25
p.w[2]	0.39	0.27
psi.b0	-0.72	1.11
psi.b1	2.29	1.22
psi.b2	-0.01	0.18
sw[1]	0.93	0.25
sw[2]	0.17	0.37
totalocc	25.89	4.27

As with JAGS, our plots might look a liiiiiite funky, but they seem to be doing what we would expect!



5 Comparing Output JAGS and NIMBLE

Just for funsies, here's how the two program's results compare for this data and these models. A reminder that ψ refers to occupancy and p refers to detection. All our models allow for noise to change the detection for that point count, so detection changed for each time step. In this notation a "." means the null model - no variation with any covariate.

Model	Equation	JAGS Weight	NIMBLE Weight
Mod 1	$psi(.)p(t)$	0.09	0.07
Mod 2	$\psi(Elev)p(t)$	0.76	0.78
Mod 3	$\psi(Rain)p(t)$	0.01	0.02
Mod 4	$\psi(Elev, Rain)p(t)$	0.14	0.14