Model fitting and evaluation

ahipp@mortonarb.org

15 June 2023

Overview

The last two weeks have been focused on fitting regression models using the formally equivalent methods of phylogenetic independent contrasts (PIC) and generalized least squares (GLS). Near the end of the last lecture, we introduced the likelihood calculation for GLS, and today we'll return to that. GLS is in some ways a more flexible framework for analysis than PIC, in part because the likelihood for a GLS model can easily be calculated. In this session, we'll cover three topics:

* **Likelihood**: calculation and practical extraction from R objects
* **Information criteria**: history and theory, calculation of various IC and IC weights, and use of IC to select among models and average parameters over models
* **Effect size and model fit**: paying attention to the fact that even an unambiguously "best" model among a candidate suite of models may do a very poor job of explaining your data
* For now, we are leaving two important topics for class discussion:
* **Parametric bootstrapping**: Monte Carlo simulation under alternative models
* **Bayesian approaches**: MCMC / rjMCMC to integrate over models and parameters

—

Likelihood

The likelihood of a hypothesis ($\mathcal{L}(H|D)$) is proportional to $P(D|H)$. Colloquially, we generally say that the likelihood *is* the probability of the data under a particular model, which is fine given that the constant of proportionality is arbitrary.

For example, the likelihood of a normal distribution for any given single data point is a function of the value of the point and the mean (μ) and standard deviation (σ) of the distribution. Calculate the likelihood for a given distribution given a single point in R thus:

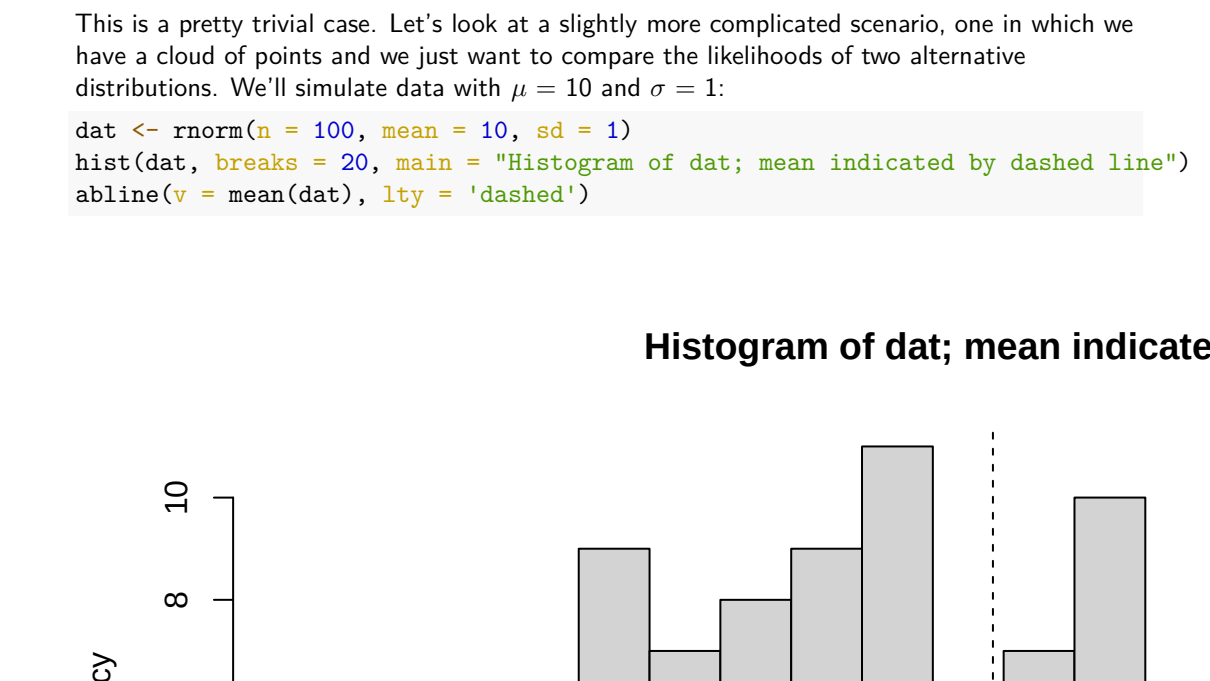
```
message(paste(
  'Likelihood of normal dist, mu = 10, sd = 1, at x = 10:', dnorm(x = 10, mean = 1
  '\nLikelihood of normal dist, mu = 10, sd = 1, at x = 8:', dnorm(x = 8, mean = 1
))
```

```
## Likelihood of normal dist, mu = 10, sd = 1, at x = 10: 0.398942280401433
## Likelihood of normal dist, mu = 10, sd = 1, at x = 8: 0.0539909665131881
```

Any distribution with $\mu = 10$ will have a higher likelihood for a single data point of 10 than for a single data point of 8.

This is a pretty trivial case. Let's look at a slightly more complicated scenario, one in which we have a cloud of points and we just want to compare the likelihoods of two alternative distributions. We'll simulate data with $\mu = 10$ and $\sigma = 1$:

```
dat <- rnorm(n = 100, mean = 10, sd = 1)
hist(dat, breaks = 20, main = "Histogram of dat; mean indicated by dashed line")
abline(v = mean(dat), lty = 'dashed')
```



Typically, we report not the likelihood but the log likelihood for a model. Let's compare this for two models. First, we'll compare three models:

```
model.fits = list(
  mean10.sd1 = dnorm(dat, mean = 10, sd = 1),
  mean10.sd2 = dnorm(dat, mean = 10, sd = 2),
  mean8.sd1 = dnorm(dat, mean = 8, sd = 1)
)
```

Now let's report the product of the likelihoods, understanding from our knowledge of probability that the probability of N independent events is the product of their individual probabilities.

```
sapply(model.fits, prod)
```

```
##    mean10.sd1    mean10.sd2     mean8.sd1
##  1.805554e-58  3.383826e-75 4.569668e-160
```

But the result appears to beyond the precision of the computer. Drat!

Let's instead use the fact that the product of two numbers can also be expressed as the sum of the logarithms:

```
model.fits.lnL <- sapply(model.fits, function(x) sum(log(x)))
model.fits.lnL
```

```
## mean10.sd1 mean10.sd2  mean8.sd1
##  -132.9591  -171.4749  -366.8942
```

That's better! So if we wanted to compare these models using maximum likelihood, we would say that the first model was the best fit. Given the rule of thumb of 2 log-likelihood units representing a significant difference among models with the same number of parameters, this is a highly significant difference.

Likelihood: two parameter example

But these too are trivial examples. Rarely are our models so highly specified. Usually, instead, we want to compare models that differ in parameters and estimate the parameter values by maximizing the likelihood for each model. Let's fit a fourth model, on in which we estimate the parameters σ and μ from the data and calculate the likelihood as we did above.

```
model.fits$k2 = dnorm(dat, mean = mean(dat), sd = sd(dat))
model.fits.lnL['k2'] <- sum(log(model.fits$k2))
model.fits.lnL
```

```
## mean10.sd1 mean10.sd2  mean8.sd1         k2
##  -132.9591  -171.4749  -366.8942  -130.2691
```

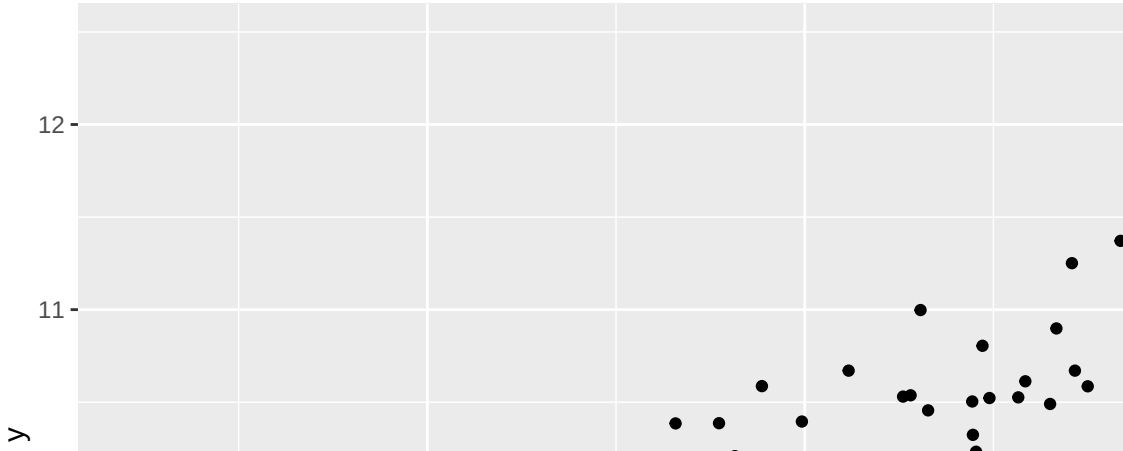
Not surprisingly, our fitted model has the highest likelihood of the models we have tested, as it is the only one that is not constrained. While the generated μ and σ were 10 and 1 respectively, the estimated values are 10.17 and 0.895 respectively.

When we use likelihood to compare models, likelihoods with all model parameters estimated from the data (e.g., our fourth model on the previous slide) are called unconstrained models. Models in which we set any parameters to a specific value (e.g., models 1$-$3 on the previous slide) are constrained, and they by necessity have lower likelihoods than an unconstrained model with the same parameters. The parameters we estimate in a model are referred to as the free parameters, and often abbreviated as K.

Likelihood: regression example

Let's look at the likelihood of a standard regression. First, simulate data with slope of 1:

```
library(ggplot2)
dat <- data.frame(y = dat, x = dat + rnorm(100, sd = 0.5))
qplot(x, y, data = dat)
```



What does this likelihood look like? We can get it from the distribution of the residuals.

```
library(magrittr)
dat.resid <- resid(lm(y ~ x, dat))
dat.lnL <- dnorm(dat.resid, mean = mean(dat.resid), sd = sd(dat.resid)) %>% log %>
dat.lnL
```

```
## [1] -49.74405
```

How much better does the model with a predictor do than a model with no predictor?

```
dat.resid.0 <- resid(lm(y ~ 0, dat))
dat.lnL.0 <-
  dnorm(dat.resid.0, mean = mean(dat.resid.0), sd = sd(dat.resid.0)) %>%
  log %>% sum
dat.lnL.0
```

```
## [1] -130.2691
```

The model $y = mx + b + \varepsilon$ fits considerably better than the no-predictor model ($y = b + \varepsilon$).
Note: the better-fit model has one extra parameter (m). We'll come back to this point later.

Likelihood: extracting from R objects

In R, you generally don't need to have your hand on all these distributions to get a likelihood. There is a generic function logLik in the stats package that comes with the R base distribution. The function has methods that apply to many object types. Take a look:

```
head(methods(logLik), 10)
```

```
##  [1] "logLik.ace"                "logLik.anc.ML"
##  [3] "logLik.anc.trend"          "logLik.Arima"
##  [5] "logLik.brownie.lite"       "logLik.corStruct"
##  [7] "logLik.fit.bd"             "logLik.fitdistr"
##  [9] "logLik.fitDiversityModel" "logLik.fitMk"
```

For the 15 packages currently loaded, there are no fewer than 41 logLik methods!

As a consequence, any properly formed objects of any of the classes following the dot in the list of logLik methods (e.g., ace, ML, trend, Arima, lite, corStruct, bd, fitdistr, fitDiversityModel, fitMk, ...) should yield a log-likelihood using the `logLik` function. Let's try on our last example:

```
dat.regModel <- lm(y ~ x, dat) # class = "lm"
logLik(dat.regModel)
```

```
## 'log Lik.' -49.74154 (df=3)
```

Compare this with our home-brewed log-likelihood:

```
dat.lnL
```

```
## [1] -49.74405
```

Not too bad! The difference of 0.0025168 is probably just rounding error, as the logLik.lm method calculates the log normal distribution directly rather than taking the log of the raw distribution. If you want to see this, you can use `stats:::logLik.lm` to inspect the function.

Information criteria

Okay. So we can get a likelihood for a set of models, but often we will want to compare models that differ in numbers of parameters, and we know that a more parameter-rich model will tend to convey a higher likelihood on the data. In fact, adding parameters to an existing model can only improve the model, or keep it the same if the parameters don't add any additional information. Look at this example: we'll check the likelihood for our initial model as well as a model with a *completely* random variable added in:

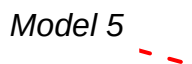
```
dat$z = rnorm(100)
logLik(dat.regModel) # original model
```

```
## 'log Lik.' -49.74154 (df=3)
```

```
dat.regModel2 <- lm(y ~ x + z, dat) # model with an additional random predictor
logLik(dat.regModel2)
```

```
## 'log Lik.' -49.66083 (df=4)
```

Adding a predictor known not to be correlated with the response increased model likelihood ever so slightly. If we simply compare likelihoods, we'll have a tendency to ratchet upward toward increasingly complex models irrespective of their explanatory value. We need a method that downweights models by complexity. Moreover, as scientists we generally want to think about multiple hypotheses and their relative evidential value. Wouldn't it be nice, we think to ourselves, if we could rank models / hypotheses by their proximity to reality? Can we find a tool to address both problems?



Enter Solomon Kullback, Richard Leibler, and Hirotugu Akaike

Information theory provides a solution to these problems. Originally introduced by Claude E. Shannon in 1948 for work in signal processing, the theory provided a framework for understanding information loss generally. >- Solomon Kullback and Richard Leibler derived Kullback-Leibler (K-L) divergence in 1951 to describe the distance between two probability distributions. Burnham and Anderson describe it as "the information lost when [a function g] is used to approximate [a function f]."

- 20 years later, Hirotugu Akaike realized this fundamental contribution to information theory could also be used to rank models in terms of predictive value. "On the morning of March 16, 1971, while taking a seat in a commuter train, I suddenly realized that the parameters of the factor analysis model were estimated by maximizing the likelihood and that the mean value of the logarithmus of the likelihood was connected with the Kullback-Leibler information number. This was the quantity that was to replace the squared error of prediction."

Calculating Akaike's information criterion (AIC) and ΔAIC

Assuming you already know the $log(\mathcal{L})$, Akaike's information criterion is calculated as

$$AIC = -2log(\mathcal{L}) + 2K$$

where K is the number of free parameters in the model. Recall that AIC estimates the K-L distance, the relative amount of information lost when approximating reality by any one of the models in the set of models you are evaluating. Thus:

1. Lower values of AIC indicate models that are estimated to predict full reality better.

Calculating Akaike's information criterion (AIC) and ΔAIC

Assuming you already know the $log(\mathcal{L})$, Akaike's information criterion is calculated as

$$AIC = -2log(\mathcal{L}) + 2K$$

where K is the number of free parameters in the model. Recall that AIC estimates the K-L distance, the relative amount of information lost when approximating reality by any one of the models in the set of models you are evaluating. Thus:

1. Lower values of AIC indicate models that are estimated to predict full reality better.
2. AIC values are only meaningful in the context of multiple models. For this reason, AIC is really only useful when we consider it in terms of the distance between each model in our set and the model the lowest AIC:

$$\Delta_i = AIC_i - AIC_{min}$$

where i indexes models.

Calculating AIC in R

Let's look back at the two models we compared earlier, one with a single predictor and one with two. We can get the number of free parameters using `attributes`

```
K <- c(
  regModel1 = attributes(logLik(dat.regModel))$df, # K for the single predictor m
  regModel2 = attributes(logLik(dat.regModel2))$df # K for the two-predictor mode
  )
K
```

```
## regModel1 regModel2
##         3         4
```

So let's use these now to get our AIC:

```
aic <- c(
  regModel1 = -2 * logLik(dat.regModel) + 2*K['regModel1'],
  regModel2 = -2 * logLik(dat.regModel2) + 2*K['regModel2']
  )
aic
```

```
## regModel1.regModel1 regModel2.regModel2
##            105.4831            107.3217
```

and $\Delta AIC = 1.8386\ldots$ model 1 is the better model from the K-L information theoretic standpoint. But how much better is good enough? Sometimes people use a threshold of 7-10 to decide that a model is unambiguously the best, but I don't care much for thresholds. Let's be more formal.

AIC weights, w_i

AIC weights (w_i, where i indexes the models you are considering) can be calculated for your models . AIC weights estimate the probability of your models [1], and thus they are useful for assessing the relative support for your models. AIC weights are calculated as:

$$\updownarrow_i = e^{-0.5\Delta AIC_i}$$

$$w_i = \frac{\updownarrow_i}{\sum_{j=1}^{n} \updownarrow_j}$$

where i and j index models, l estimates the model likelihood, and w estimates the model probability.

[1] The interested student may appreciate the equivalence between the BIC and AIC weights as derived in Burnham and Anderson 2002, p. 302-304, by modification of priors on the AIC weights, relative to the uniform priors assumed by BIC.

Let's calculate w_i in R!

In fact, let's create two functions, one to return AIC if we hand in a vector of $\log(\mathcal{L})$ and a vector of K, then one to calculate w_i. First, we'll define the function for AIC:

```
aic <- function(lnL, K) {
  if(length(lnL) != length(K)) stop('vectors lnL and K must be of same length')
  out <- -2 * lnL + 2 * K # don't you love how R handles vectors?
  if(!is.null(names(lnL))) names(out) <- names(lnL) # allows you to name models
  return(out)
}
```

Isn't that easy? Let's take our function for a test-drive:

```
dat.aic <- aic(lnL = c(reg1 = logLik(dat.regModel), reg2 = logLik(dat.regModel2))
dat.aic
```

```
##     reg1     reg2
## 105.4831 107.3217
```

Okay. Now let's make a function to generate w_i from an AIC vector.

```
aic.w <- function(aicVector) {
  delta.aic <- aicVector - min(aicVector)
  exp.aic <- exp(-0.5 * delta.aic)
  out <- exp.aic / sum(exp.aic)
  if(!is.null(names(aicVector))) names(out) <- names(aicVector)
  return(out)
}
```

Let's give it a whirl:

```
dat.aic.w <- aic.w(dat.aic)
dat.aic.w
```

```
##      reg1      reg2
## 0.7148975 0.2851025
```

This is kind of a dumb example, as the second model in our set was known *a priori* to be poor, but it's worth noting that when two models differ only in number of parameters, not in likelihood, that the simpler model will be favored.

Should we favor simpler models?

"Gaudí and Mies remind us that there is no disputing matters of taste when it comes to assessing the value of simplicity and complexity in works of art. Einstein and Newton say that science is different – simplicity, in science, is not a matter of taste. Reichenbach and Akaike provided some reasons for why this is so. The upshot is that there are three parsimony paradigms that explain how the simplicity of a theory can be relevant to saying what the world is like:

- Paradigm 1: sometimes simpler theories have higher probabilities.

from Elliot Sober's 2016 Essay, "Why is simpler better?" in the journal *Aeon*
https://aeon.co/essays/are-scientific-theories-really-better-when-they-are-simpler

Should we favor simpler models?

"Gaudí and Mies remind us that there is no disputing matters of taste when it comes to assessing the value of simplicity and complexity in works of art. Einstein and Newton say that science is different – simplicity, in science, is not a matter of taste. Reichenbach and Akaike provided some reasons for why this is so. The upshot is that there are three parsimony paradigms that explain how the simplicity of a theory can be relevant to saying what the world is like:

- Paradigm 1: sometimes simpler theories have higher probabilities.
- Paradigm 2: sometimes simpler theories are better supported by the observations.

from Elliot Sober's 2016 Essay, "Why is simpler better?" in the journal *Aeon*
https://aeon.co/essays/are-scientific-theories-really-better-when-they-are-simpler

Should we favor simpler models?

"Gaudí and Mies remind us that there is no disputing matters of taste when it comes to assessing the value of simplicity and complexity in works of art. Einstein and Newton say that science is different – simplicity, in science, is not a matter of taste. Reichenbach and Akaike provided some reasons for why this is so. The upshot is that there are three parsimony paradigms that explain how the simplicity of a theory can be relevant to saying what the world is like:

- Paradigm 1: sometimes simpler theories have higher probabilities.
- Paradigm 2: sometimes simpler theories are better supported by the observations.
- Paradigm 3: sometimes the simplicity of a model is relevant to estimating its predictive accuracy.

from Elliot Sober's 2016 Essay, "Why is simpler better?" in the journal *Aeon*
https://aeon.co/essays/are-scientific-theories-really-better-when-they-are-simpler

Small-sample AIC

It turns out that AIC is biased with small sample sizes, so Burnham and Anderson recommend using a small-sample correction, AICc:

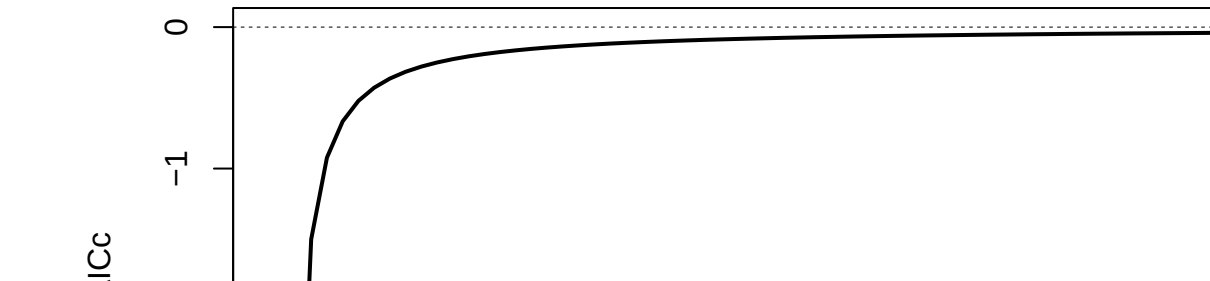
$$AIC_c = AIC + \frac{2K(K+1)}{n-K-1}$$

and we can make a function to do this as well:

```
aic.c <- function(l, k, n) aic(l,k) + 2*k*(k + 1) / (n-k-1)
```

- On the next page, let's demonstrate to ourselves that AICc approaches AIC as $N \to \infty$

```r
n = seq(from = 10, to = 1000, by = 10)
dat.lnL = sapply(n, function(x) logLik(lm(y ~ x, dat = dat[1:x, ])))
dat.aic.c <- aic.c(dat.lnL, n = n, k = rep(3, length(n)))
dat.aic <- aic(dat.lnL, K = rep(3, length(n)))
plot(n, dat.aic - dat.aic.c, type= 'l', lwd = 2, xlab = 'Sample size', ylab = 'AI
abline(h = 0, lty = 'dashed', lwd = 0.5)
```



Because AICc converges on AIC, it is generally favored for all applications; as your sample size gets larger, the second term in AICc goes to 0, so AICc can always be used.

There are other information criteria in use, the most common of which is the Bayesian information criterion (BIC) or Schwarz information criterion (SIC), calculated as

$$BIC = -2ln(\mathcal{L}) + ln(n)K$$

Sober and Forster point out that because AIC is an unbiased, minimum variance estimator of "predictive accuracy" (I assume here that they mean K-L distance, though they don't reference it in the article), BIC is biased and has higher variance.* However, BIC was derived to rank models by posterior probability, and it may be favored by some in the crowd on those grounds alone.

* Burnham and Anderson 2002 also critique BIC for an assumption that the true model is in the candidate set of models being evaluated. This argument is also articulated in a response article that I have only found online (https://sites.warnercnr.colostate.edu/kenburnham/wp-content/uploads/sites/25/2016/08/Response-to-Link-and-Barker.pdf), where they write: "BIC is clearly derived from asymptotics and approaches its target (the true model that is assumed to be in the model set) from below, often leading to the selection of under-fitting models when sample sizes are less than very large. Again, the dependence on the true model being in the model set must surely be viewed as a 'buried assumption.' "

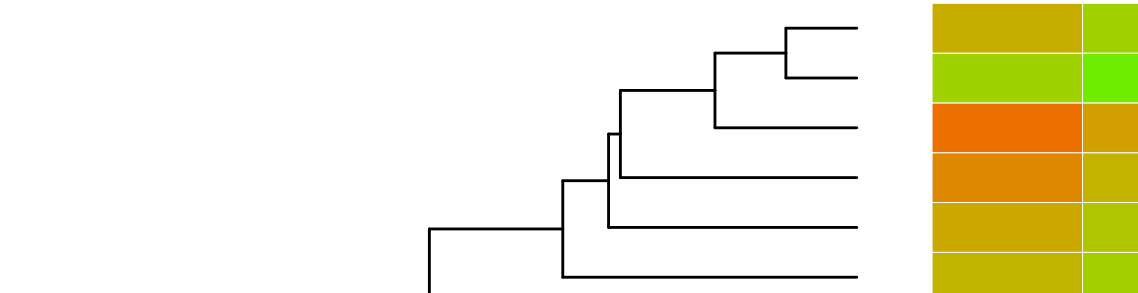
IC: Model averaging to estimate parameters

Model weights make explicit the relative probabilities of alternative models. Often, no single model has a posterior probability > 0.95. In such a case, conditioning on a single model for the remainder of your analyses has the effect of decreasing the variance about the parameters you are estimating. Using a single model to estimate parameters without taking into account uncertainty in model selection is analogous to reporting the point estimate of a parameter without taking into account the standard error on that estimate.

A model averaging example

Let's look at an example using example data from the geiger package. First, let's bring in the data and plot it.

```
require(ggtree)
require(geiger)
data(geospiza)
dat <- as.data.frame(scale(geospiza$dat))
tr <- drop.tip(geospiza$phy, which(!geospiza$phy$tip.label %in% row.names(dat)))
gheatmap(ggtree(tr), dat, colnames_angle = -45)
```



Let's set up five models that we think are plausible explanations of wing length:

- Wing length $\sim$ tarsus length

Are these reasonable models? I couldn't tell you, but working with one's own data, one should identify models that make sense biologically and each represent a reasonable expectation about how the world works.

Let's set up five models that we think are plausible explanations of wing length:

- Wing length $\sim$ tarsus length
- Wing length $\sim$ tarsus length $+$ culmen length

Are these reasonable models? I couldn't tell you, but working with one's own data, one should identify models that make sense biologically and each represent a reasonable expectation about how the world works.

Let's set up five models that we think are plausible explanations of wing length:

- Wing length $\sim$ tarsus length
- Wing length $\sim$ tarsus length + culmen length
- Wing length $\sim$ tarsus length + culmen length + gonys width

Are these reasonable models? I couldn't tell you, but working with one's own data, one should identify models that make sense biologically and each represent a reasonable expectation about how the world works.

Let's set up five models that we think are plausible explanations of wing length:

- Wing length $\sim$ tarsus length
- Wing length $\sim$ tarsus length + culmen length
- Wing length $\sim$ tarsus length + culmen length + gonys width
- Wing length $\sim$ tarsus length + culmen length + beak depth

Are these reasonable models? I couldn't tell you, but working with one's own data, one should identify models that make sense biologically and each represent a reasonable expectation about how the world works.

Let's set up five models that we think are plausible explanations of wing length:

- Wing length $\sim$ tarsus length
- Wing length $\sim$ tarsus length + culmen length
- Wing length $\sim$ tarsus length + culmen length + gonys width
- Wing length $\sim$ tarsus length + culmen length + beak depth
- Wing length is not predicted by any of these

Are these reasonable models? I couldn't tell you, but working with one's own data, one should identify models that make sense biologically and each represent a reasonable expectation about how the world works.

The corresponding models look like this in R (ignoring the phylogeny, since this is really just about the modeling):

```
geo.models <- list(
  w.t = lm(wingL ~ tarsusL, dat),
  w.tc = lm(wingL ~ tarsusL + culmenL, dat),
  w.tcg = lm(wingL ~ tarsusL + culmenL + gonysW, dat),
  w.tcb = lm(wingL ~ tarsusL + culmenL + beakD, dat),
  w.0 = lm(wingL ~ 1, dat)
  )
rbind(
  lnL = sapply(geo.models, logLik),
  k = sapply(geo.models, function(x) attributes(logLik(x))$df)
  ) %>% round(5)
```

```
##           w.t      w.tc     w.tcg     w.tcb       w.0
## lnL -15.10981 -8.04818 -1.54212 -0.33407 -17.92592
## k     3.00000  4.00000  5.00000  5.00000   2.00000
```

We have variation across these models in both $\mathcal{L}$ and K; can we trust the highest $\mathcal{L}$ to be the best model?

Let's look at the AICc scores for these.

```
geo.models.out <- sapply(geo.models, function(x) {
   out = c(lnL = logLik(x),
           aic.c = aic.c(logLik(x),
                         k = attributes(logLik(x))$df,
                         n = dim(dat)[1]),
           k = attributes(logLik(x))$df)
         })
geo.models.out <- rbind(geo.models.out,
                        w_i = aic.w(geo.models.out['aic.c', ])) %>% round(5)
geo.models.out
```

```
##             w.t      w.tc     w.tcg     w.tcb       w.0
## lnL   -15.10981 -8.04818 -1.54212 -0.33407 -17.92592
## aic.c  38.88628 29.09635 21.65567 19.23957  41.05185
## k       3.00000  4.00000  5.00000  5.00000   2.00000
## w_i     0.00004  0.00554  0.22876  0.76564   0.00001
```

In this case, the two most complex models carry $> 99\%$ of the evidentiary weight, but neither is conclusive.

So let's use model weights to average parameters over models. First, let's make a matrix with all the parameters for each model.

```
geo.params <- unique(unlist(lapply(geo.models, function(x) names(coef(x)))))
geo.mat <- geo.var.mat <- matrix(NA, length(geo.models), length(geo.params),
                  dimnames = list(names(geo.models), geo.params))
for(i in names(geo.models))
  geo.mat[i, names(coef(geo.models[[i]]))] <- coef(geo.models[[i]])
geo.mat
```

```
##         (Intercept)   tarsusL   culmenL    gonysW     beakD
## w.t   -2.224937e-15 0.5929594        NA        NA        NA
## w.tc  -1.128137e-15 0.4645332 0.6679108        NA        NA
## w.tcg -2.816696e-15 0.3347451 0.2443681 0.5964462        NA
## w.tcb -2.896886e-15 0.3690696 0.2778556        NA 0.5723542
## w.0   -4.187718e-15        NA        NA        NA        NA
```

What should we do about those blanks?

There are at least two schools of thoughts on what to do with missing parameters when you model average. One is that one only averages parameters over the models in which they occur. The rationale here I guess is that models without a given parameter must not be informative about that parameter. The second is that models lacking a parameter are actually best thought of as having a value of 0 (no effect) for that parameter. Let's try this 2nd option.

```
geo.mat[is.na(geo.mat)] <- 0
wi <- geo.models.out['w_i', ]
geo.mat <- rbind(geo.mat,
  modelAvg = apply(geo.mat, 2, weighted.mean, w = wi))
geo.mat <- cbind(geo.mat, 'w_i' = c(wi, sum(wi)))
round(geo.mat, 4)
```

```
##           (Intercept) tarsusL culmenL gonysW  beakD    w_i
## w.t                 0  0.5930  0.0000 0.0000 0.0000 0.0000
## w.tc                0  0.4645  0.6679 0.0000 0.0000 0.0055
## w.tcg               0  0.3347  0.2444 0.5964 0.0000 0.2288
## w.tcb               0  0.3691  0.2779 0.0000 0.5724 0.7656
## w.0                 0  0.0000  0.0000 0.0000 0.0000 0.0000
## modelAvg            0  0.3618  0.2723 0.1364 0.4382 1.0000
```

Quantifying uncertainty within and over models

You can stop there if all you need is a point estimate, but that's not very satisfying to a statistically savvy crew like this. Knowing that variances are additive, let's get the weighted average of the variances and use these to get a model average variance, equal to

$$var(\hat{\bar{Y}}) = \sum_{i=1}^{R} w_i \{var(\hat{Y}_i|g_i) + (\hat{Y}_i - \hat{\bar{Y}})^2\}$$

(note that this is slightly different than the formula in Burnham and Anderson, in which the model-averaging variance term is presented, incorrectly I believe, as $(\hat{Y}_i - \hat{\bar{Y}}_i)^2$).

Within model variance: $var(\hat{Y}_i|g_i)$

We can get the squared standard errors and variance using:

```
geo.var.list <- lapply(geo.models, function(i) summary(i)$cov.unscaled %>% diag) %
for(i in names(geo.var.list))
  geo.var.mat[i, names(geo.var.list[[i]])] <- geo.var.list[[i]]
geo.var.mat
```

```
##       (Intercept)    tarsusL   culmenL    gonysW     beakD
## w.t    0.07692308 0.08333333        NA        NA        NA
## w.tc   0.07692308 0.08653260 0.0865326        NA        NA
## w.tcg  0.07692308 0.09667709 0.1945652 0.2142413        NA
## w.tcb  0.07692308 0.09152848 0.1699370        NA 0.1795835
## w.0    0.07692308          NA        NA        NA        NA
```

To the within-model variances, let's add the among-model variance, $(\hat{Y}_i - \bar{\hat{Y}})^2$.

Among model variance: $(\hat{Y}_i - \hat{\bar{Y}})^2$

We'll get the among-model variance by subtracting each individual parameter from the model-averaged values, again treating missing parameters as 0 (discussion point... either now or after lecture):

```
geo.var.mat.amongModels <-
  (geo.mat[1:5,1:5] - # point estimates per model
    matrix(geo.mat['modelAvg', 1:5], 5, 5, byrow = T) # a matrix of model average
  ) ^ 2 # ... square the difference to get the variance
geo.var.mat.amongModels
```

```
##        (Intercept)      tarsusL      culmenL     gonysW      beakD
## w.t   4.144683e-31 5.345705e-02 7.417014e-02 0.01861707 0.19203824
## w.tc  3.029660e-30 1.056406e-02 1.564747e-01 0.01861707 0.19203824
## w.tcg 2.707392e-33 7.293494e-04 7.825349e-04 0.21160166 0.19203824
## w.tcb 7.928382e-34 5.355323e-05 3.040015e-05 0.01861707 0.01799154
## w.0   1.739733e-30 1.308642e-01 7.417014e-02 0.01861707 0.19203824
```

Note that the among-model variance is rather small in our case.

Summing variance components

Now we sum the variance components and take the weighted sum to get the $\sum_{i=1}^{R} w_i$ term at the beginning of the compound weighted average term. First the summing:

```
geo.var.mat[which(is.na(geo.var.mat))] <- 0 # substitutes in 0 for missing data
geo.var.mat <- geo.var.mat + geo.var.mat.amongModels
geo.var.mat
```

```
##       (Intercept)    tarsusL    culmenL     gonysW     beakD
## w.t    0.07692308 0.13679039 0.07417014 0.01861707 0.1920382
## w.tc   0.07692308 0.09709666 0.24300730 0.01861707 0.1920382
## w.tcg  0.07692308 0.09740644 0.19534778 0.42584299 0.1920382
## w.tcb  0.07692308 0.09158203 0.16996736 0.01861707 0.1975751
## w.0    0.07692308 0.13086419 0.07417014 0.01861707 0.1920382
```

and now the weighted average:

Summing variance components, pg 2

```
geo.var.total <- apply(geo.var.mat, 2, weighted.mean, w = wi)
geo.var.total
```

```
## (Intercept)     tarsusL     culmenL      gonysW       beakD
##  0.07692308  0.09294719  0.17617330  0.11177501  0.19627749
```

... which you can use to get standard errors by the standard formula, $\sigma_{\bar{x}} = \frac{\sigma}{\sqrt{n}}$.

```
sqrt(geo.var.total) / sqrt(13)
```

```
## (Intercept)     tarsusL     culmenL      gonysW       beakD
##  0.07692308  0.08455639  0.11641217  0.09272582  0.12287501
```

... and these, of course, are useful for estimating confidence intervals, which now integrate over both parameter and model uncertainty.

But how well does your model explain the data?

It's worth remembering that no matter how much time you spend considering models, you may not find any that do a good job of explaining the variance in your data in an absolute sense. Interpreting R^2 is a bit complicated for GLS models that underlie most of the continuous trait methods we use, in part due to the fact that the data are transformed. In the case of an OLS model, however, it's more easily interpreted as the proportion of variance explained.

```
sapply(geo.models, summary) %>%
  sapply(FUN = '[[', 'r.squared') %>%
  round(5)
```

```
##     w.t    w.tc   w.tcg   w.tcb     w.0
## 0.35160 0.78121 0.91959 0.93323 0.00000
```

But for many cases, an R^2 is not readily available. Pennell et al. (2015) illustrate one approach to dealing with such cases.

And what does each parameter tell you?

At the end of all this, don't forget to look at your parameter estimates. If you have rescaled your data to mean of 0 and unit variance, the multiple / partial regression coefficients tell you the relative effect of each predictor on the response, in sd units; and the effect of each if all the others are held at their mean value (0 on the rescaled data).

Effect size and direction is in the end generally what you care about and how your hypotheses were formulated: which predictors have the greatest effect on the response, and do they have a positive or negative effet on the response? Your analysis won't have amounted to much if you don't take time to interpret your results after the results are in.