Lab meeting Jan 26, 2016

We'll use the following R packages

```
## Loading required package: lme4

## Loading required package: Matrix

## Loading required package: Rcpp

## Loading required package: MuMIn

## Loading required package: arm

## Loading required package: MASS

##
## arm (Version 1.8-5, built: 2015-05-13)

## Working directory is /Users/Sean/Documents
```

We'll use data from the npk dataset describing some experiment where people measured plant yield following manipulations of N, P, and K in 6 treatment blocks.

```
npk
```

```
##    block N P K yield
## 1      1 0 1 1  49.5
## 2      1 1 1 0  62.8
## 3      1 0 0 0  46.8
## 4      1 1 0 1  57.0
## 5      2 1 0 0  59.8
## 6      2 1 1 1  58.5
## 7      2 0 0 1  55.5
## 8      2 0 1 0  56.0
## 9      3 0 1 0  62.8
## 10     3 1 1 1  55.8
## 11     3 1 0 0  69.5
## 12     3 0 0 1  55.0
## 13     4 1 0 0  62.0
## 14     4 1 1 1  48.8
## 15     4 0 0 1  45.5
## 16     4 0 1 0  44.2
## 17     5 1 1 0  52.0
## 18     5 0 0 0  51.5
## 19     5 1 0 1  49.8
## 20     5 0 1 1  48.8
## 21     6 1 0 1  57.2
## 22     6 1 1 0  59.0
## 23     6 0 1 1  53.2
## 24     6 0 0 0  56.0
```

```
attach(npk)
```

The simpliest method will be an additive linear model that we just analyze with ANOVA. OK if that's your thing but lets go a bit deeper...

```
global.lm <- lm(yield ~ N + K + P)
anova(global.lm)
```

```
## Analysis of Variance Table
##
## Response: yield
##           Df Sum Sq Mean Sq F value  Pr(>F)
## N          1 189.28 189.282  6.4880 0.01919 *
## K          1  95.20  95.202  3.2632 0.08592 .
## P          1   8.40   8.402  0.2880 0.59743
## Residuals 20 583.48  29.174
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
summary(global.lm)
```

```
##
## Call:
## lm(formula = yield ~ N + K + P)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.2667 -3.6542  0.7083  3.4792  9.3333
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   54.650      2.205  24.784   <2e-16 ***
## N1             5.617      2.205   2.547   0.0192 *
## K1            -3.983      2.205  -1.806   0.0859 .
## P1            -1.183      2.205  -0.537   0.5974
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.401 on 20 degrees of freedom
## Multiple R-squared:  0.3342, Adjusted R-squared:  0.2343
## F-statistic: 3.346 on 3 and 20 DF,  p-value: 0.0397
```

Lets take an IT approach. We can do this manually in base R if the number of models is small.

```
# Make a reduced model
reduced.lm <- lm(yield ~ N + K)
# Use AIC to compare with the global
AIC(global.lm, reduced.lm)
```

```
##            df      AIC
## global.lm   5 154.6920
## reduced.lm  4 153.0351
```

Reduced model has a lower AIC score, thus has more support.There are a bunch of other manual calcs that we can use to figure out the weights, delta AIC values etc. but there are easier shortcuts that do this for us and it can quickley get out of hand when there are more models. The MuMIn package can calculate AIC's of multiple models at the same time... Also, we should actually be using mixed effects models due to the experimental blocks.
The following code calculates AIC values (AICc for small sample sizes) of all possible subsets of the global model. Lets go through step by step.

```
## Global mixed effects model with all possible interaction terms using lme4
global <-lmer(yield ~ N*P*K + (1|block),npk,na.action = "na.fail")
```

Now we jump into the MuMIn package using the *dredge* function. For simplicity lets not consider models with 3-way interactions. We can suppress these with "m.lim" within the dredge function, which limits the number of terms allowed in a model

```
all.models <- dredge(global, rank ="AICc", m.max=3, REML = FALSE)
```

```
## Fixed term is "(Intercept)"
```

```
all.models
```

```
## Global model call: lmer(formula = yield ~ N * P * K + (1 | block), data = npk, na.action = "na.fail")
## ---
## Model selection table
##    (Int) K N P K:N K:P N:P df logLik  AICc  delta weight
## 4  54.06 + +               5  -65.697 153.0  0.00 0.445
## 12 52.88 + +   +           6  -62.520 154.0  1.02 0.267
## 3  52.07   +               4  -69.832 155.6  2.63 0.119
## 8  54.65 + + +             6  -64.029 156.0  2.98 0.100
## 7  52.66   + +             5  -68.093 158.4  5.39 0.030
## 2  56.87 +                 4  -71.888 160.0  6.99 0.014
## 1  54.88                   3  -75.292 160.7  7.66 0.010
## 39 51.72   + +         +   6  -65.348 160.8  7.81 0.009
## 6  57.46 +   +             5  -70.071 162.8  9.84 0.003
## 5  55.47     +             4  -73.426 163.2 10.24 0.003
## 22 57.60 +   +     +       6  -67.685 166.4 13.42 0.001
## Models ranked by AICc(x, REML = FALSE)
## Random terms (all models):
## '1 | block'
```

Now that we have our ranked candidate model set, we can make inferences into the relative importance among predictors by summing the weights across all models where a given predictor is included.

```
importance(all.models)
```

```
##                       N     K     K:N   P     N:P   K:P
## Importance:          0.97  0.83  0.27  0.15 <0.01 <0.01
## N containing models:    6     6     1     6     1     1
```

However, this still doesn't tell us about how well the models actually fit the data (will return to this later..) and if parameter slopes are actually meaningful. The next step is coming up with parameter estimates and associated 95% CIs by averaging across models. The following code uses the *model.avg* function to calculate model averaged 95% confidence intervals and slope estimates.

```
model.avg(all.models)
```

```
##
## Call:
## model.avg.model.selection(object = all.models)
##
## Component models:
## '12'     '124'    '2'      '123'    '23'     '1'      '(Null)' '236'
## '13'     '3'      '135'
##
## Coefficients:
##        (Intercept)        K1       N1     K1:N1         P1       N1:P1
## subset    53.55343 -3.227701 6.280082 -4.700000 -1.0685471 -3.76666667
## full      53.03640 -2.677067 6.094055 -1.253756 -0.1557582 -0.03377127
##               K1:P1
## subset 0.5666666667
## full   0.0003073354
```

If we want, we can subset only the models with a certain delta cut-off..Also embedding this within the *summary* command gives more info

```
summary(model.avg(all.models, subset = delta < 4))
```

```
##
## Call:
## model.avg.model.selection(object = all.models, subset = delta <
##     4)
##
## Component model call:
## lmer(formula = yield ~ <4 unique rhs>, data = npk, na.action =
##      na.fail)
##
## Component models:
##     df logLik   AICc delta weight
## 12   5 -65.70 152.99  0.00   0.48
## 124  6 -62.52 154.01  1.02   0.29
## 2    4 -69.83 155.62  2.63   0.13
## 123  6 -64.03 155.97  2.98   0.11
##
## Term codes:
##   K   N   P K:N
##   1   2   3   4
##
## Model-averaged coefficients:
##             Estimate Std. Error Adjusted SE z value Pr(>|z|)
## (Intercept)   53.530      2.232       2.366  22.625  < 2e-16 ***
## K1            -3.211      2.131       2.241   1.433  0.15186
## N1             6.290      2.111       2.221   2.831  0.00463 **
## K1:N1         -4.700      3.094       3.317   1.417  0.15650
## P1            -1.183      1.634       1.751   0.676  0.49919
##
## Full model-averaged coefficients (with shrinkage):
```

```
##             Estimate Std. Error Adjusted SE z value Pr(>|z|)
## (Intercept)  53.5305     2.2315      2.3660  22.625  < 2e-16 ***
## K1           -2.8001     2.2605      2.3517   1.191  0.23378
## N1            6.2897     2.1110      2.2214   2.831  0.00463 **
## K1:N1        -1.3461     2.6940      2.7688   0.486  0.62683
## P1           -0.1274     0.6496      0.6818   0.187  0.85171
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Relative variable importance:
##                      N    K    K:N  P
## Importance:          1.00 0.87 0.29 0.11
## N containing models:    4    3    1    1
```

Conditional vs. full model averaging Conditional averages parameters across only the models where they occur while full averages parameters across all models with zeros for models where a given parameter does not occur. Notice how the full model averaged parameters are closer to zero in the output

To extract model averaged 95% CIs The default is to use conditional averages but the command *full = TRUE* changes it to full averages. The same subset command from above also works here...

```
confint(model.avg(all.models), level = 0.95)
```

```
##                   2.5 %    97.5 %
## (Intercept)  48.812246 58.294616
## K1           -7.626191  1.170789
## N1            1.914577 10.645586
## K1:N1       -11.201201  1.801201
## P1           -4.934439  2.797345
## N1:P1       -11.616422  4.083088
## K1:P1        -8.606932  9.740265
```

Standardizing predictors

Before moving forward, we should go back and properly standardize predictors. This is important for interpretting effects when they are on different scales OR when interactions are present. For binary predictors, we re-scale standardizing to a mean of 0 and SD of 0.5. The command *binary.input = center* tells R we have binary predictors. Then we run dredge again to generate a new set of models and associated parameter estimates that are on the same scale.

```
global.std <- standardize(global, standardize.y = FALSE, binary.input = "center")
all.models.std <- dredge(global.std, m.max = 3, REML = FALSE)
```

```
## Fixed term is "(Intercept)"
```

Another useful command is *get.models*, which subsets all models that fall under a certain criteria and puts them in a list. You can select based on delta aic or the cumulative sum of weights

```
top.list <- get.models(all.models.std, subset = delta < 4)
top.list
```

```
## $`4`
## Linear mixed model fit by REML ['lmerMod']
## Formula: yield ~ c.K + c.N + (1 | block)
##    Data: npk
## REML criterion at convergence: 131.394
## Random effects:
##  Groups   Name        Std.Dev.
##  block    (Intercept) 3.644
##  Residual             3.942
## Number of obs: 24, groups:  block, 6
## Fixed Effects:
## (Intercept)          c.K          c.N
##      54.875       -3.983        5.617
##
## $`12`
## Linear mixed model fit by REML ['lmerMod']
## Formula: yield ~ c.K + c.N + (1 | block) + c.K:c.N
##    Data: npk
## REML criterion at convergence: 125.0406
## Random effects:
##  Groups   Name        Std.Dev.
##  block    (Intercept) 3.684
##  Residual             3.790
## Number of obs: 24, groups:  block, 6
## Fixed Effects:
## (Intercept)          c.K          c.N      c.K:c.N
##      54.875       -3.983        5.617       -4.700
##
## $`3`
## Linear mixed model fit by REML ['lmerMod']
## Formula: yield ~ c.N + (1 | block)
##    Data: npk
## REML criterion at convergence: 139.6647
## Random effects:
##  Groups   Name        Std.Dev.
##  block    (Intercept) 3.480
##  Residual             4.497
## Number of obs: 24, groups:  block, 6
## Fixed Effects:
## (Intercept)          c.N
##      54.875        5.617
##
## $`8`
## Linear mixed model fit by REML ['lmerMod']
## Formula: yield ~ c.K + c.N + c.P + (1 | block)
##    Data: npk
## REML criterion at convergence: 128.057
## Random effects:
##  Groups   Name        Std.Dev.
##  block    (Intercept) 3.628
##  Residual             4.002
## Number of obs: 24, groups:  block, 6
## Fixed Effects:
## (Intercept)          c.K          c.N          c.P
```

```
##      54.875       -3.983        5.617       -1.183
##
## attr(,"rank.call")
## AICc(x, REML = FALSE)
## attr(,"rank")
## function (x)
## do.call("rank", list(x, REML = FALSE))
## <environment: 0x10c46f068>
## attr(,"rank")attr(,"call")
## AICc(x, REML = FALSE)
## attr(,"rank")attr(,"class")
## [1] "function"     "rankFunction"
```

Marginal and Conditional R squared

We still haven't looked at how well our models actually fit the data. With random effects traditional R^2 doesn't work. Traditionally people use a psuedo R^2 which fits predicted against fitted values; however, there are apparently problems with this approach. Recently, people have developed a technique to partition the variance explained by fixed vs. random factors in the model. In our case, the marginal R^2 describes the proportion of variance explained by fixed effects only, and the conditional R^2 describes the proportion of variance explained by both fixed AND random effects.There is a function in MuMIn that can do this for you. We'll use *lapply* to run this function for each model in our top ranked list..

```
r.sqrd.list <- lapply(top.list, FUN=r.squaredGLMM)
r.sqrd.list
```

```
## $`4`
##       R2m       R2c
## 0.3003160 0.6227691
##
## $`12`
##       R2m       R2c
## 0.3307914 0.6559392
##
## $`3`
##       R2m       R2c
## 0.2028927 0.5014279
##
## $`8`
##       R2m       R2c
## 0.3038582 0.6179182
```

Finally, run the following code to spit out two clean data frames; one with slope estimates and 95% CI's, the second with top ranked models, AICc scores, and marginal and conditional R^2 values. Note, this is clunky but I haven't figured out a good method to automate.

```
## Slope coefficients and CI
top.sum <- summary(model.avg(all.models.std, subset = delta < 4))
top.coef <- as.data.frame(top.sum$coefTable)
top.coef
```

```
##              Estimate Std. Error Adjusted SE   Lower CI   Upper CI
```

```
## (Intercept) 54.875000   1.691387    1.808125  51.331139 58.4188607
## c.K         -3.983333   1.592170    1.703089  -7.321326 -0.6453404
## c.N          5.616667   1.625423    1.737233   2.211753  9.0215807
## c.K:c.N     -4.700000   3.094451    3.317000 -11.201201  1.8012012
## c.P         -1.183333   1.633622    1.751110  -4.615447  2.2487800
```

```
## AIC and model fits
#Extract R squareds
model.r2 <- as.data.frame(rbind(r.sqrd.list[[1]],r.sqrd.list[[2]],r.sqrd.list[[3]],r.sqrd.list[[4]]))
## Extract AIC table
aic.table <- top.sum$msTable
## Bind em' together
model.output <- cbind(aic.table, model.r2)
model.output
```

```
##     df    logLik     AICc    delta    weight       R2m       R2c
## 12   5 -65.69700 152.9902 0.000000 0.4778180 0.3003160 0.6227691
## 124  6 -62.52032 154.0138 1.023584 0.2864139 0.3307914 0.6559392
## 2    4 -69.83234 155.6236 2.633390 0.1280646 0.2028927 0.5014279
## 123  6 -64.02851 155.9699 2.979695 0.1077035 0.3038582 0.6179182
```