Functions and models in R

Overview

The goal of this lab is to visit the model functions created by R - to begin to learn how to build them, how to work with them, and how to compare different models. We will revisit the `Lily_sum` data from Thomson et al. (1996) used in earlier lectures. As a reminder, the goal of this study was to determine the the effects of factors such as water, soil quality, and predation by gophers influencing glacier lily propagation and survival. We'll restrict our attention to columns 3 to 8 today. In addition to the data in the `emdbook` package, we'll be using `ggplot2` for graphics, `MASS` for the `glm.nb` function, and `MuMIn` for the `model.sel` function.

```
library(ggplot2)
library(MASS)
library(MuMIn)
library(emdbook)
lilies <- Lily_sum[,3:8]
summary(lilies)
```

```
##     flowers         seedlings        vegetative        gopher
##  Min.   :  0.00   Min.   : 0.000   Min.   : 0.00   Min.   : 0.000
##  1st Qu.: 15.00   1st Qu.: 0.000   1st Qu.: 3.00   1st Qu.: 1.000
##  Median : 23.00   Median : 1.000   Median : 8.00   Median : 3.000
##  Mean   : 29.91   Mean   : 3.957   Mean   :11.82   Mean   : 3.684
##  3rd Qu.: 37.00   3rd Qu.: 6.000   3rd Qu.:17.25   3rd Qu.: 6.000
##  Max.   :139.00   Max.   :72.000   Max.   :60.00   Max.   :13.000
##    rockiness         moisture
##  Min.   :  0.00   Min.   :0.000
##  1st Qu.:  0.00   1st Qu.:2.500
##  Median : 14.50   Median :4.000
##  Mean   : 52.96   Mean   :3.854
##  3rd Qu.:101.00   3rd Qu.:5.000
##  Max.   :215.00   Max.   :8.000
```

Revisiting `lm`

The template for most of the model functions in R is the linear model, `lm`.

We'll start with a simple linear regression with two predictor variables.

```
lilymod1 <- lm(flowers~vegetative+gopher, data=lilies)
lilymod1
```

```
##
## Call:
## lm(formula = flowers ~ vegetative + gopher, data = lilies)
##
## Coefficients:
## (Intercept)   vegetative       gopher
##      25.429        1.068       -2.209
```

(1) What is the formula that `lilymod1` defines to predict the expected number of flowers as a function of vegetation and gopher activity. How many flowers are expected for a plot with a vegetative index of 20 and a gopher level of 5?

The object `lilymod1` contains quite a lot of information. To get a list of its components, we can use the `names` command.

```
names(lilymod1)
```

```
##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
##  [9] "xlevels"       "call"          "terms"         "model"
```

Many of these you probably won't ever use, but `coefficients`, `residuals` and `fitted.values` are quite useful. To access to coefficients of the model, we can use `lilymod1$coefficients` or `coefficients(lilymod1)`, with the latter generally being considered better practice. (It makes no difference in this situation, but using `$coefficients` instead of `coefficients()` when you are employing a link function might not give you what you expect.) You can type `coef()` instead of `coefficients()`. Similarly, get the residuals with `residuals()` or `resid()` and the fitted values (predictions) with `fitted.values()` or `fitted()`.

For some tasks, it is useful to add the residuals or the predictions to the data frame use to build the model. To add residuals as the variable `resid1`, enter

```
lilies$resid1 <- resid(lilymod1)
```

(2) It is important to evaluate a model against the actual data. With one variableAdd `pred1` to `lilies` using `fitted()` and make a plot showing the data and the predictions with the following code:

```
ggplot(lilies, aes(vegetative))+
  geom_point(aes(y=flowers, color="actual"))+
  geom_point(aes(y=pred1, color="predicted"))+
  scale_color_manual("", values=c("blue", "red"))
```

The blue points are the actual data points, and the red points are the predictions. Observe that the predictions have a linear ceiling. Using the coefficients of `vegetative` and `gopher` in `lilymod1` explain why this is the case - what line is that linear ceiling, and why do we get a ceiling at all?

It is important to note that **this procedure for adding model output to data only works if there are no missing values**. If you need to tell R to skip missing values when adding predictions, it's a little trickier and beyond what I want to go into in this lab.

More information about a model

Often, the first thing one does after creating any sort of model is take a look at the `summary`.

```
summary(lilymod1)
```

```
##
## Call:
## lm(formula = flowers ~ vegetative + gopher, data = lilies)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -52.796 -10.287  -2.686   7.358  79.407
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  25.4286     2.5659   9.910  < 2e-16 ***
## vegetative    1.0676     0.1061  10.061  < 2e-16 ***
```

```
## gopher       -2.2094     0.4107  -5.379  1.7e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 17.35 on 253 degrees of freedom
## Multiple R-squared:  0.4522, Adjusted R-squared:  0.4479
## F-statistic: 104.4 on 2 and 253 DF,  p-value: < 2.2e-16
```

This gives us some diagnostic information, such as R^2 and standard errors in the coefficients. The added information about the coefficients can be accessed using row and column notation. For example, to get the standard error in the estimate for coefficient of `gopher`:

```
coef(summary(lilymod1))[3,2]
```

```
## [1] 0.4107283
```

(3) Use the `coef()` command to calculate the upper and lower bounds of a 95% confidence interval for the `vegetative` coefficient. This task can also be accomplished using `confint(lilymod1)`. Use this to check your answer.

Note: building confidence intervals can be used to check if differences in the coefficients of indicator variables are statistically significant.

To access the statistics such as R^2, use the `$` notation, for example `summary(lilymod1)$fstatistic`.

Making predictions on new data

Sometimes we want to see what a model would do with new input data. The `predict()` function does this too. We need to give the new data to `predict()` as a data frame where the variable names are the input variables in the model. Try the following code to compute what `lilymod1` predicts for sites with `gopher`=5 and `vegetative` between 20 and 25. The code `20:25` creates the list `c(20, 21, 22, 23, 24, 25)`, and setting `gopher = 5` works because R will repeat the 5 to match the six entries given for `vegetative`.

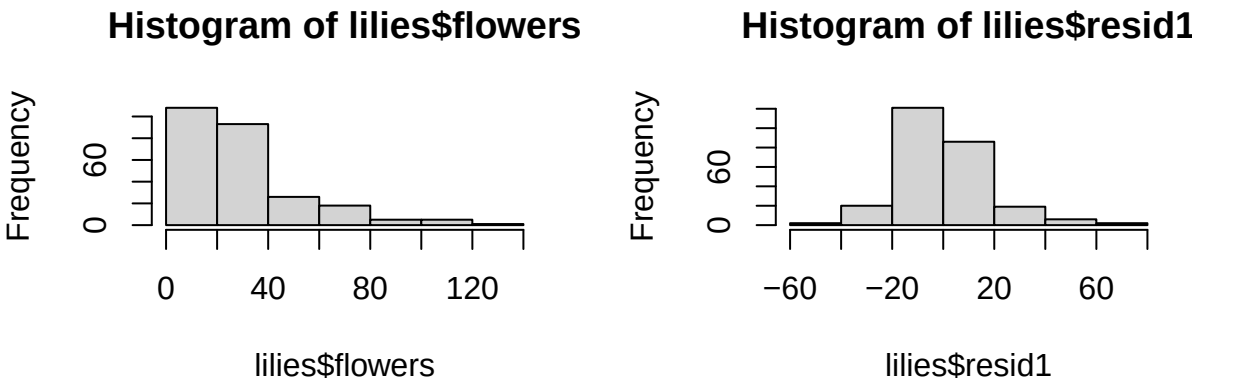
```
newlily <- data.frame(vegetative = 20:25, gopher = 5)
newlily$prediction <- predict(lilymod1, newdata = newlily)
newlily
```

(4) Compute the predictions from `lilymod1` for `vegetative`=25 and `gopher` between 5 and 10.

Transformations and link functions

A histogram of the response variable `flowers` shows significant right skew, some of which is still present in the residuals for `lilymod1`. A log transform might make sense.

```
hist(lilies$flowers)
hist(lilies$resid1)
```



You can use `lm()` to fit a model for the mean of log-transformed `flowers` or `glm` to fit the log of the mean of `flowers`. We'll continue to use `vegetative` and `gopher` as predictors so that we can compare with `lilymod1`.

Fitting to the log-transformed variable

A first attempt to make a model might reasonably use the following code. Try it and notice what goes wrong.

```
lilymod2 <- lm(log(flowers)~vegetative+gopher, data = lilies)
```

There are a number of solutions to this problem, which is caused by the fact that log(0) is not defined. Before we do anything, we should check how extensive the problem is. Use `sum(lilies$flowers==0)` to count how many times `flowers` is exactly 0. Since this only happens once, we might consider leaving out this observation. Another common fix to log transforming a variable with zeros is to add a small amount before transforming.

```
# Put any computations done in a formula in I() so that to they are
# properly interpreted as arithmetic.
lilymod2 <- lm(I(log(flowers+1))~vegetative+gopher, data = lilies)
```

(5) Compare the output of `summary(lilymod2)` to `summary(lilymod1)`. In what ways do they tell the same story?

Back-transforming to the original scale

To make predictions on the scale of the original data, we have to reverse the computation log(`flowers`+1) by exponentiating and then subtracting 1. Also, we first add half the variance in the residuals to correct for how the error transforms. (In practice, this step is often ignored.)

```
vr <- var(resid(lilymod2))
lilies$pred2 <- exp(fitted(lilymod2)+vr/2)-1
```

(6) Using the coefficients in `lilymod2`, the residual variance `vr` and a little algebra, write the formula that `lilymod2` uses to predict the mean number of flowers based on vegetation and gophers. Interpret the coefficients of `lilymod2` in the context of the study. (An increase of one unit for the `vegetative` index is expected to correspond to . . .)

$$flowers = 27.3636 \times \exp(0.02847veg - 0.08397gph)$$

Fitting using a log link

To fit with a link function, we use `glm` (Generalized Linear Model) instead of `lm`. A big difference between `glm` and `lm` is that the nice calculus tricks that make linear models easy to compute don't work in general, so to find the coefficients `glm` does a guess-check-adjust-repeat process. In many cases, the computer can come

up with a reasonable starting guess, but sometimes it needs to be given a hint. Since we have the coefficients of `lilymod2` we can use those.

```
lilymod3 <- glm(flowers~vegetative+gopher, data = lilies,
                family=gaussian(link="log"), start = coef(lilymod2))
```

We can compare the coefficients and their standard errors using `summary` on `lilymod2` and `lilymod3`. The bottom part of the `summary` output for `glm` is different though. Instead of *Residual standard error*, R^2, and F statistics, we have *Null deviance*, *Residual deviance* and *AIC*.

As before, we can add the predicted values to the data. The `fitted` command for `glm` does the back transform automatically. (But `lilymod3$fitted` can be used to get the untransformed predictions.)

```
lilies$pred3 <- fitted(lilymod3)
```

(7) Find the formula for `lilymod3` as you did for `lilymod1` and `lilymod2`. How different are `lilymod2` and `lilymod3`?

Model comparison

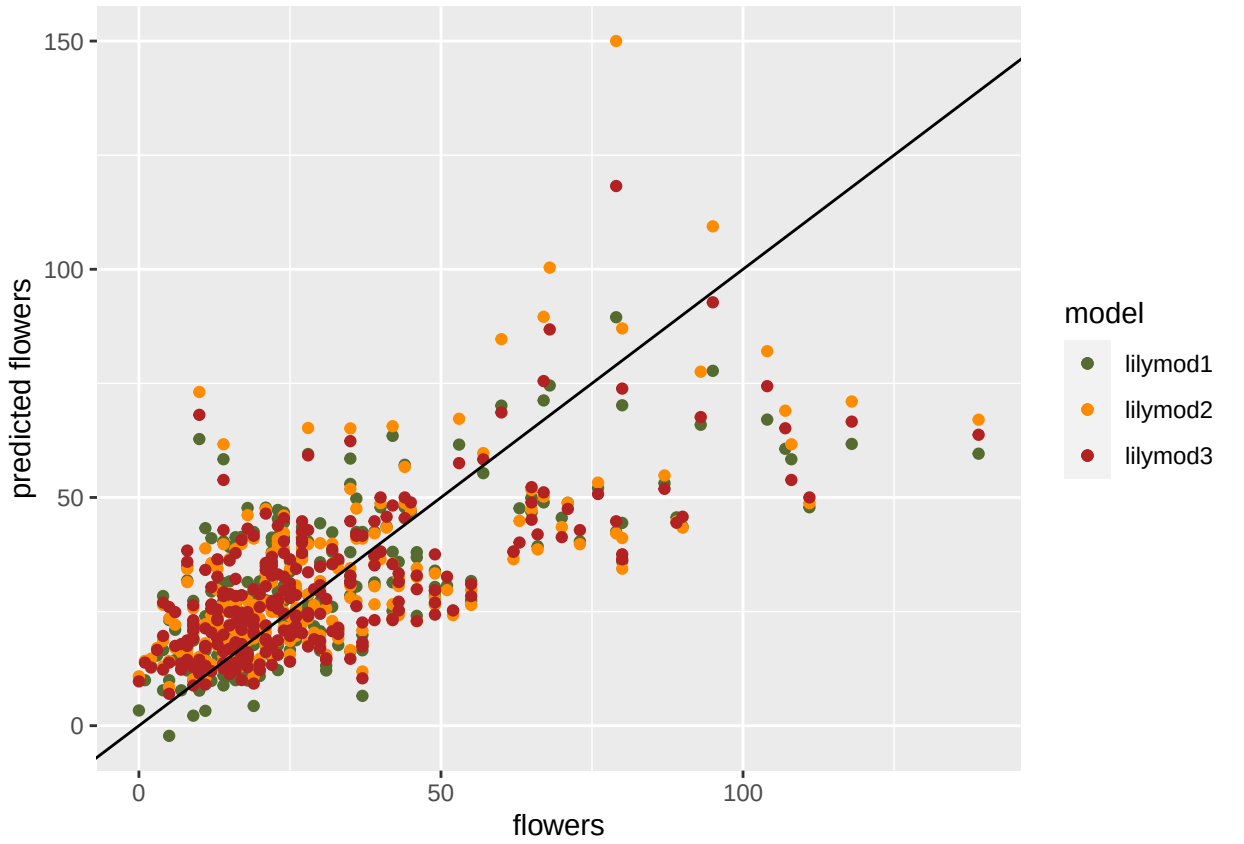
Now we have three different models describing flower counts in terms of indices of vegetation and gopher activity. How can we determine which we prefer? The situation is similar to when we use ANOVA to compare models that do or don't include certain terms, but since the models are not nested, ANOVA does not make sense.

The ideal situation is when there is a theoretical justification for the model you choose. But if you are just getting to know the data or experimenting with different designs, there are various ways to compare models. These sort of comparisons are also important to justify the validity of your choice when it is made based on theory.

Comparing the models graphically

One way that the models can be compared is by plotting the predictions against the actual data. We add a reference line of what would be accurate predictions.

```
ggplot(lilies, aes(flowers))+
  geom_point(aes(y=pred1, color="lilymod1"))+
  geom_point(aes(y=pred2, color="lilymod2"))+
  geom_point(aes(y=pred3, color="lilymod3"))+
  geom_abline(slope = 1, intercept = 0)+
  scale_color_manual("model", values = c("darkolivegreen", "darkorange", "firebrick"))+
  ylab("predicted flowers")
```



The predictions made by all three models appear rather similar. I do not discern any strong patterns distinguishing them.

Comparison using error metrics

Recall that regression is choosing a model that minimizes the mean squared error (MSE).

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y})^2$$

We can use this metric to compare our models. To do this, we'll write a custom command for computing the MSE from vectors of predictions and actual data.

```
mse <- function(prediction, actual){
  sum((actual - prediction)^2)/length(actual)
}
mse(lilies$pred1, lilies$flowers)
```

```
## [1] 297.5452
```

```
mse(lilies$pred2, lilies$flowers)
```

```
## [1] 305.7774
```

```
mse(lilies$pred3, lilies$flowers)
```

```
## [1] 287.965
```

This metric supports the observation that the three models are similar, with `lilymod3` having the lowest mean squared error and `lilymod2` the highest, but only slightly in both cases.

Mean squared error is only one of infinitely many possible error metrics we could have chosen. A very similar option would have been mean absolute error (MAE).

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}|$$

(8) Revise the function for mean squared error to compute mean absolute error. How do the models compare by this metric?

Akaike information criterion (AIC)

Another metric that we can use is the Akaike information criterion, or AIC. We'll discuss how AIC is defined in a few weeks. A fact about AIC is that only models built from exactly the same data can be compared, because AIC in part quantifies how likely it is that the observed data would be generated by the model under consideration. This creates a difficulty in this scenario because we log transformed the response variable in `lilymod2`, so it is built from different data. This would take some work to overcome and we'll leave that for another day.

```
AIC(lilymod1)
```

```
## [1] 2192.561
```

```
AIC(lilymod3)
```

```
## [1] 2184.183
```

Smaller values of AIC indicate a better fit, so this metric implies that `lilymod3` is preferred. We will revisit AIC later, but the essential computation to make is this: Given a set of models, compute their AICs. If AIC_{min} is the smallest of these values and the AIC for the i^{th} model is AIC_i then

$$\exp\left(\frac{AIC_{min} - AIC_i}{2}\right)$$

is roughly interpretable as the probability that the i^{th} model is actually better than the model with the lower AIC.

(9) What is the probability that `lilymod1` is actually better than `lilymod3`, based on AIC?

Poisson and negative binomial regression

Since the response variable in this study is a count, it is theoretically preferred to use `family=poisson` over `family=gaussian(link="log")`. The `family` argument to `glm` describes the expected distribution of the response for a particular set of values for the predictors. We'll talk in more detail about this difference next week. A log link is the default for the `poisson` family, so it doesn't need to be specified. A generalized linear model with a Poisson family is often called a *Poisson regression model*.

Create a Poisson regression model and call it `lilymod4`.

```
lilymod4 <- glm(flowers~vegetative+gopher, family=poisson, data=lilies)
```

A problem that is frequently encountered when using Poisson regression is that the data are not homogeneous as is assumed by the model, but instead clumped or *overdispersed*. Since lilies often grow in patches, this is a reasonable thing to expect here. Overdispersion can be checked using the ratio of the residual deviance to the degrees of freedom - if this ratio is much more than 1, then we conclude that the data are overdispersed.

(10) Use the `summary` command and find the ratio of the residual deviance to the degrees of freedom for `lilymod4`. Do these data appear overdispersed?

Two solutions for overdispersed data are to use a *quasi-Poisson model* or a *negative binomial model*. Both of these models add another parameter to model the overdispersion, the difference is in how that other parameter is estimated. Details and discussion can be found for example in Ver Hoef and Boveng (2007). For a quasi-Poisson model, use `glm(..., family=quasipoisson)` and for negative binomial use the `glm.nb` command from the `MASS` library that we loaded above. The default link for both of these models is the log.

Create a quasi-Poisson and a negative binomial model and call them `lilymod5` and `lilymod6`.

```
lilymod5 <- glm(flowers~vegetative+gopher, family=quasipoisson, data=lilies)
lilymod6 <- glm.nb(flowers~vegetative+gopher, data=lilies)
```

We can use MSE, MAE, AIC or other metrics to compare these models to each other and the ones we built earlier, with the exception that AIC is not defined for the quasi-Poisson model. AICc is a correction to AIC for small sample sizes, and is the default used by the `model.sel` function in the `MuMIn` package.

```
model.sel(lilymod1,lilymod3,lilymod4,lilymod5,lilymod6)
```

(11) Which of these models is best, according to AICc?

More Practice

(12) In this lab so far, we have not considered the other variables `seedlings`, `moisture` and `rockiness`, or any interactions between the predictors. Experiment with adding terms and come up with your preferred model. Describe your model. Support your preference by comparing it to other models that you considered.

(13) Is there an ecological story you can tell based on the model you selected in the question above?

References

Thomson, James D., George Weiblen, Barbara A. Thomson, Satie Alfaro, and Pierre Legendre. 1996. "Untangling Multiple Factors in Spatial Distributions: Lilies, Gophers, and Rocks." *Ecology* 77 (6): 1698–1715. http://www.jstor.org/stable/2265776.

Ver Hoef, Jay M, and Peter L Boveng. 2007. "Quasi-Poisson Vs. Negative Binomial Regression: How Should We Model Overdispersed Count Data?" *Ecology* 88 (11): 2766–72.