Advancing in R

Module 3: lm()

Introduction

In this practical exercise, we will introduce regression analysis and the main functions in R that will allow you to conduct it.

We will assume that you are familiar with the general syntax of coding in R, that you are adept at making basic plots, and that you have reviewed material covered in the first practical exercises. If you need to, review this material before the session starts.

Learning outcomes

Know how to explore bivariate relationships graphically

Know how to specify simple linear regression models

Know how to examine and interpret model diagnostic plots

Know how to extract values from model objects

Know how to interpret tables of coefficients

Know how to conduct null hypothesis tests on regression models

Know how to update models to examine the effect of high influence points

Know how to report regression statistics in written reports

Know how to graphically represent regression lines and their confidence intervals

Some useful commands

Below is a list of some (but not all) of the useful commands you may need to run at some stage of this practical exercise.

abline()
library()
lm()
matlines()
plot()
predict()
seq()
subset()
summary()
summary.aov()
update()
xlab() & ylab()

Data exploration

Begin by loading the RStudio client. Open a new project and a new Rscript, and save each of these using meaningful and descriptive filenames. Make sure you include the annotation at the start of your script that clears the workspace and identifies the purpose of the script and its author.

Find the data file called "tannin.csv" in the folder for this practical, and read it into a well-named data frame (*be careful not to give it the same name as one of the vectors within the dataframe!*). Verify that it has loaded properly, and examine the structure of the object.

This small dataset is from a study of how the concentration of dietary tannin (in g/kg) affects caterpillar growth (in grams of mass gained). Note that the scientist carefully controlled dietary tannin, so that this variable is likely to be measured with much less error than mass gain. Why is this important?

Use plotting commands to exercise data quality control. How would you check for outliers? Do so now. If necessary, clean the data and recode any variables that need to be read differently than they currently are (if, for example, any categorical variables are mistakenly being interpreted as numbers).

Examine the distribution of both variables. Have you got any concerns about the data? Note which of the variables is the response, and which is the predictor based on the information above. Also note that your sample is very small, and that deviations from a nice bell-shape in such small samples is common, and not necessarily a sign that the data are not parametric. (Remember that the residuals must be normally distributed, not the data themselves.)

Now use the *high level* plot command plot() to explore the relationship between the concentration of tannin and caterpillar growth. Do you think these data suggest a strong or weak relationship? Make a guess about the value of the slope and intercept for the line of best fit. Record your guesses in script annotations before we check to see if you are right.

Model construction

We will conduct a regression analysis using the *overloaded* function lm(). (Overloaded functions are those that produce different kinds of values depending on the context in which they are used. We will use lm() for many different purposes throughout this course!) The lm stands for "linear model". You may have already used it in to add a line of best fit to a plot in one of the previous modules. Adding this line actually required us to perform a regression! So you were working ahead of yourself without even knowing it.

The lm() command uses a series of arguments, typically beginning with a statement in the form of an equation, in which the response variable is on the left hand side of a tilde, and the predictors are listed on the right hand side:

```
> MOD.1<-lm(GROWTH~TANNIN,data=TANDAT)
```

Note that in the command above I stored the model in a new object, which I have unimaginatively called MOD.1. Storing models in named objects is almost always required, as you will want to do

much more than simply examine coefficients. I usually use numbered suffixes because it allows me to quickly navigate through alternatives when conducting model simplification (something we will start using soon).

Having stored the model, check the workspace and you'll see a new entry in the "Values" section of the workspace. Note that unlike the dataframe, this object is denoted as an lm object, and the [12] indicates that there are twelve different attributes stored in the model. To examine them, use the str() command:

```
> str(MOD.1)
List of 12
 $ coefficients : Named num [1:2] 11.76 -1.22
  ..- attr(*, "names")= chr [1:2] "(Intercept)" "TANNIN"
 $ residuals    : Named num [1:9] 0.244 -0.539 -1.322 2.894 -0.889 ...
  ..- attr(*, "names")= chr [1:9] "1" "2" "3" "4" ...
 $ effects      : Named num [1:9] -20.67 -9.42 -1.32 2.83 -1.01 ...
  ..- attr(*, "names")= chr [1:9] "(Intercept)" "TANNIN" "" "" ...
 $ rank         : int 2
 $ fitted.values: Named num [1:9] 11.76 10.54 9.32 8.11 6.89 ...
  ..- attr(*, "names")= chr [1:9] "1" "2" "3" "4" ...
 $ assign       : int [1:2] 0 1
 $ qr           :List of 5
  ..$ qr   : num [1:9, 1:2] -3 0.333 0.333 0.333 0.333 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:9] "1" "2" "3" "4" ...
  .. .. ..$ : chr [1:2] "(Intercept)" "TANNIN"
  .. ..- attr(*, "assign")= int [1:2] 0 1
  ..$ qraux: num [1:2] 1.33 1.26
  ..$ pivot: int [1:2] 1 2
  ..$ tol  : num 1e-07
  ..$ rank : int 2
  ..- attr(*, "class")= chr "qr"
 $ df.residual  : int 7
 $ xlevels      : Named list()
 $ call         : language lm(formula = GROWTH ~ TANNIN, data = TANDAT)
 $ terms        :Classes 'terms', 'formula' length 3 GROWTH ~ TANNIN
  .. ..- attr(*, "variables")= language list(GROWTH, TANNIN)
  .. ..- attr(*, "factors")= int [1:2, 1] 0 1
  .. .. ..- attr(*, "dimnames")=List of 2
  .. .. .. ..$ : chr [1:2] "GROWTH" "TANNIN"
  .. .. .. ..$ : chr "TANNIN"
  .. ..- attr(*, "term.labels")= chr "TANNIN"
  .. ..- attr(*, "order")= int 1
  .. ..- attr(*, "intercept")= int 1
  .. ..- attr(*, "response")= int 1
  .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
  .. ..- attr(*, "predvars")= language list(GROWTH, TANNIN)
  .. ..- attr(*, "dataClasses")= Named chr [1:2] "numeric" "numeric"
  .. .. ..- attr(*, "names")= chr [1:2] "GROWTH" "TANNIN"
 $ model        :'data.frame':   9 obs. of  2 variables:
  ..$ GROWTH: int [1:9] 12 10 8 11 6 7 2 3 3
```

```
  ..$ TANNIN: int [1:9] 0 1 2 3 4 5 6 7 8
  ..- attr(*, "terms")=Classes 'terms', 'formula' length 3 GROWTH ~
TANNIN
  .. .. ..- attr(*, "variables")= language list(GROWTH, TANNIN)
  .. .. ..- attr(*, "factors")= int [1:2, 1] 0 1
  .. .. .. ..- attr(*, "dimnames")=List of 2
  .. .. .. .. ..$ : chr [1:2] "GROWTH" "TANNIN"
  .. .. .. .. ..$ : chr "TANNIN"
  .. .. ..- attr(*, "term.labels")= chr "TANNIN"
  .. .. ..- attr(*, "order")= int 1
  .. .. ..- attr(*, "intercept")= int 1
  .. .. ..- attr(*, "response")= int 1
  .. .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
  .. .. ..- attr(*, "predvars")= language list(GROWTH, TANNIN)
  .. .. ..- attr(*, "dataClasses")= Named chr [1:2] "numeric" "numeric"
  .. .. .. ..- attr(*, "names")= chr [1:2] "GROWTH" "TANNIN"
 - attr(*, "class")= chr "lm"
```

Notice that doing regression involves many computations! We usually won't need to delve into the model contents ourselves (R has high level functions that allow us to work with these more conveniently), but sometimes we will want to examine some of these. Note that the second vector in the model is called $residuals. How many residuals are there? Why are there this many? What do these values represent?

Model diagnostics

Before we examine the model summary, we should run diagnostics to see if the model is any good. We'll use the overloaded plot() function to do this, but just for fun we'll also examine residuals directly. It will help if you first arrange the graphics device to display 6 panels before proceeding. If you don't do this, you will have to manually hit enter in the console window to roll through the four diagnostic plots produced by plot.

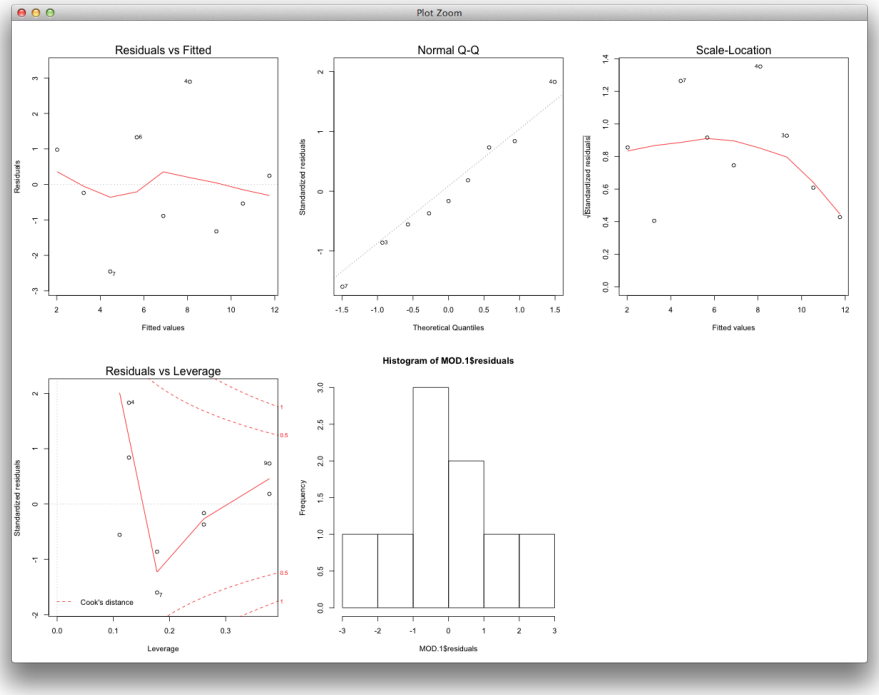
```
> par(mfrow=c(2,3))
> plot(MOD.1)
> hist(MOD.1$residuals)
> par(mfrow=c(1,1))
```

Note the syntax of the third line – I am asking to see a histogram of the residuals. Why is this interesting in the context of the assumptions of regression?

The plot() function generates the first four diagnostic plots (see next page), while the histogram of residuals is an extra one that some people find more useful for assessing whether your model adheres to parametric assumptions. (The histogram is actually somewhat redundant with the information in the Q-Q plot, so you do not normally need not produce it, but it's still interesting to explore this as a direct illustration of whether the model adheres to assumptions.) Work though each of the diagnostic plots in turn, trying to assess whether the current model is reasonable or not. You will need more practice to become good at this, so make a habit of carefully examining diagnostic plots every time you do regression, and always do this before you examine the model

summary. If the diagnostics are terrible, there's usually no point in examining the coefficients as they are likely to be wrong.

Although some of the plots are not quite ideal (maybe we would like to further examine points 7 and 9, which may have high leverage according to the fourth plot), with such a small sample of data, the fit is remarkably good (perhaps as expected based on the clear trend in the points that you observed in your exploratory plot).



Model coefficients

Now examine a summary of the model, using the summary() command:

```
> summary(MOD.1)

Call:
lm(formula = GROWTH ~ TANNIN, data = TANDAT)

Residuals:
    Min      1Q  Median      3Q     Max
-2.4556 -0.8889 -0.2389  0.9778  2.8944

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  11.7556     1.0408  11.295 9.54e-06 ***
TANNIN       -1.2167     0.2186  -5.565 0.000846 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 1.693 on 7 degrees of freedom
Multiple R-squared: 0.8157, Adjusted R-squared: 0.7893
F-statistic: 30.97 on 1 and 7 DF,  p-value: 0.0008461
```

The summary always produces a reminder of the structure of the model (the Call), a summary of residual quartiles (so you can quickly assess deviations from parametric assumptions), a table of coefficients (the heart of the model) and some summaries of model fit.

Examine the coefficients carefully. Do the estimates for the coefficients match your earlier predictions? Note that the y-intercept is listed as (Intercept), but the slope is listed as TANNIN. Why do you think it is done this way? The coefficients each have standard errors, a t-value which is a test statistic for the null hypothesis of a zero coefficient value, and a p-value associated with this null hypothesis. Which of the p-values is of most interest? Why? Based on this p-value, should you reject the null hypothesis of no association between dietary tanning and growth?

How many additional grams of growth are expected for every unit of TANNIN added to the diet? What is the confidence interval (coefficient ± 1.96 X SE) for the slope? Write a sentence that summarizes the effect of TANNIN on GROWTH. Your sentence should focus on the biology of the system rather than the stats details. The best way to do this is to include a statement of effect, the direction of effect, and a parenthetical reference to the coefficient, its standard error, the sample size, test statistic, and p-value. Your sentence may also refer to a figure illustrating the effect (we'll produce one of those soon).

Now examine the last line of the output, which lists an F-statistic. This is a test of the whole regression, and is associated with the null hypothesis that the model does not predict the response (note that in your model there is only one term: TANNIN). Compare the p-value associated with this F-statistic to the p-value for the slope term. Why are they the same?

Examine the R-squared value (usually listed as Multiple R-squared). It is known as the coefficient of determination, and roughly represents the fraction of variation in your response variable that is predicted by your regression model. In simple linear regression models that include an intercept, the coefficient of determination is the square of Pearson's Product Moment Correlation, r (a statistic obviously associated with correlation). What does the coefficient of determination for your regression tell you about how much of the variation in GROWTH is explained by TANNIN?

The adjusted R-squared is a value that helps assess model quality when more than one predictor variable is included, and it does not analogously represent the fraction of variation explained.

You can generate another kind of summary for your model using the following command:

```
> summary.aov(MOD.1)
            Df Sum Sq Mean Sq F value   Pr(>F)
TANNIN       1  88.82   88.82   30.97 0.000846 ***
Residuals    7  20.07    2.87
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Examine this output and try to work out the quantities represented. What are the Sum Sq and Mean Sq values, and how do they relate to the variances used to compute the regression? How does the p-value in this table compare to the p-values for the coefficients? Which of these summary commands, the summary() or the summary.aov(), provides more information? Which of these tables is most often reported in the recent literature, as opposed to older papers?

Let's try one last thing before we get to plotting. Recall that in the diagnostic plots, data points 7 and 9 may have high influence. You may consider computing the regression without them to see whether the coefficients are *sensitive to* the inclusion of points 7 or 9. (Do the coefficients change much when the seventh or ninth record is excluded?)

```
MOD.2<-update(MOD.1,data=TANDAT[-7,])
```

The update() command is an extremely useful function we'll rely heavily on in future modules. In the line of code above, we're asking R to use the same model structure as specified in model 1, but with only a subset of the data (which does not include row 7). You could of course alternatively use a new data= argument, and specify the subset using square brackets. Compare the summary() of MOD.2 to MOD.1. Are the coefficients substantially different in your judgment? Normally if they are the same one would retain the full model unless we had special reason to suspect that the high influence data point was measured or entered incorrectly. Try this procedure again, omitting record 9 instead.

Visualizing regression output

If you are happy that you understand the output of the regression, it's time to prepare a publication-quality figure illustrating your findings. There are of course several ways to do this.

First, you could use high level plotting functions plot() and the low level abline() to illustrate the scatterplot and line respectively. Alternatively, you may wish to illustrate not just the regression line, but the degree of confidence that you have in the slope. To do this using the {graphics} package, you will need to use the predict() command. (Some other graphics packages such as {ggplot2} will automatically add a confidence region to your figure unless you ask to suppress this feature.) Because we will rely heavily on predict() in future modules (and even spend most of one module focussing on its mechanics), it is important to be proficient in using the basic graphics package.

The idea behind predict() is that you use a model object to predict a response variable over a certain range. We'll use the seq() command to generate a new series of X variables that we'll run through the equation:

```
> NEWXVARS<-seq(0,8,length=100)
```

This command tells R to generate a new vector called NEWXVARS, which includes all numbers between 0 and 8, in intervals of 0.05. Why did I specify that particular range of x-values? Examine the vector, and you'll see it's just a series of increasing x-variables 100 rows long. In practice, the length doesn't matter too much, but you ideally want over 100 rows to make sure that the resulting lines are smooth. This will be particularly important when we start illustrating curved lines later on.

Now let's feed these new variables into a predict command, and store the results as a new object representing the predicted response variables:

```
> NEWYVARS<-predict(MOD.1,newdata=list(TANNIN=NEWXVARS),int="c")
```

The syntax here says three things: the model to use for prediction, the place to look for new predictor variables, and what type of interval to use around the prediction (here "c" stands for 95% confidence interval). Examine the newly created object – you'll see it has three rows, one called "fit", one called "lwr" and one called "upr", for the predicted y value, the lower 95% interval, and the upper 95% interval respectively. Neat!

Let's use these estimates to make a nice plot. Write a new line of code using the plot() command to illustrate your data. Make sure you alter the x- and y-axis labels so that they are descriptive and include units.

Now, instead of using abline to add the line of best fit, use another command called matlines(). Matlines is a command that plots a line for each vector in a matrix. In the current case, the first vector represents the fitted values, the second the upper 95% CI and the third the lower 95% CI.

```
> matlines(NEWXVARS,NEWYVARS,lty=c(1,2,2),col="black")
```

Here the syntax is specifying the predictor variables, the response variables, the types of lines for the fit, lwr, and upr vectors, and the colour of all three. (NB matlines is a low-level plotting function so only works if the plot function has already been called to create the plot on which the lines will be added.) Can you work out what the lty argument is doing, and why? Feel free to experiment with this code to make a visually appealing plot.

If you wish, you can practice doing more linear regressions by consulting the supplementary exercises for this module. These exercises will examine further what diagnostic plots look like when parametric assumptions are not met. Remember to thoroughly annotate your script, save your project and R script, and if you want, your figure as well.

~ End of Practical ~