CSC 120: Computer Science for the Sciences (R section)

Radford M. Neal, University of Toronto, 2016

http://www.cs.utoronto.ca/∼radford/csc120/

Week 11

Statistical Facilities in R

In this course, we've mostly looked at R as a programming language, and at general programming concepts.

But R is most popular as a language for statistical applications. So it has many special facilities for doing statistics. I'll talk about some now.

Don't worry if you don't understand some of the statistical concepts — that's OK for this course. Though learning about R's statistical facilities is one good way to learn statistics in a hands-on way!

Standard Distributions and Associated Functions

R knows about quite a few probability distributions, for each of which it has an abbreviation. These include:

`unif`	Uniform distribution ranging from a given minimum to a given maximum.
`norm`	Normal (Gaussian) distribution with given mean and standard deviation.
`exp`	Exponential distribution with given rate.
`binom`	Binomial distribution with given n and p.
`pois`	Poisson distribution with given mean.

For each of these distributions, R provides four functions, whose names are the distribution's abbreviation preceded by `d`, `p`, `q`, or `r`. Here are their meanings, illustrated for normal distributions:

`dnorm`	Probability density function (PDF).
`pnorm`	Cumulative Distribution Function (CDF).
`qnorm`	Quantile (inverse CDF) function.
`rnorm`	Random number generator.

The density function is really the probability mass function when the distribution is discrete (eg, `dpois`).

Examples for Normal Distributions

The `dnorm`, `pnorm`, and `qnorm` functions all take a numeric vector as their first argument, followed by the mean (default 0) and standard deviation (default 1).

For example:

```
> dnorm (c(-2.5,0,2.5))     # Standard normal, default mean 0 and sd 1
[1] 0.0175283 0.3989423 0.0175283
> dnorm (c(-2.5,0,2.5),1,2) # Mean 1 and sd 2
[1] 0.04313866 0.17603266 0.15056872
> qnorm (0.025)             # Some familiar numbers
[1] -1.959964
> pnorm (c(-2,-1,0,1,2))
[1] 0.02275013 0.15865525 0.50000000 0.84134475 0.97724987
```

The `rnorm` function generates random numbers from a normal distribution:

```
> rnorm(6)
[1]  0.5855288  0.7094660 -0.1093033 -0.4534972  0.6058875 -1.8179560
> rnorm(6,100,50)
[1] 131.50493  86.19079  85.79201  54.03390  94.18761 190.86560
```

Statistical Modeling in R

One big part of statistics is fitting a *model* to data. R has many functions for doing this, but I'll mention only `lm`, which fits a linear model.

Models in R are often specified using *formulas*, that say how one thing is modelled in terms of other things.

For `lm`, we want to specify that some *response* variable is modelled as a linear combination (plus noise) of some *explanatory* variables. This is done using a formula such as

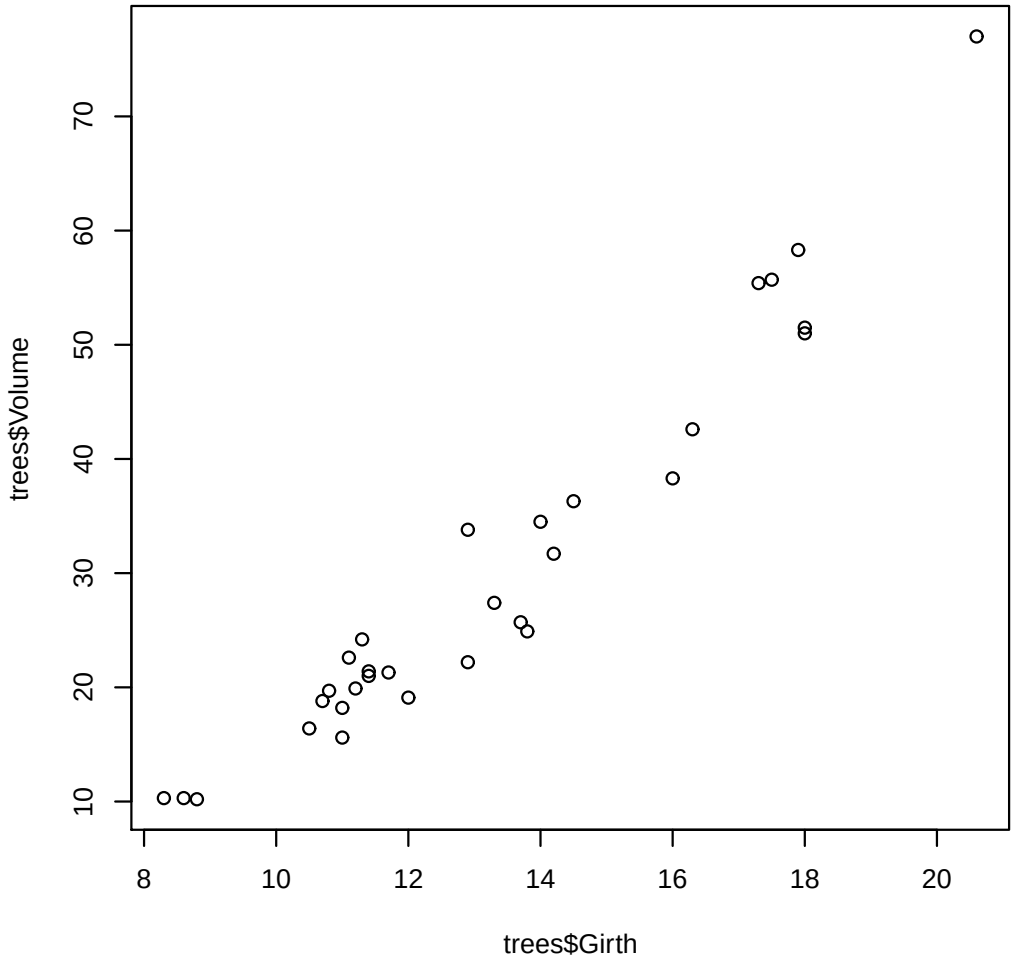
```
growth ~ ave_temp + fertilizer + variety
```

This might express that the amount by which some plant grows is linearly related to the average temperature, the amount of fertilizer used, and a set of indicator variables indicating the variety of the plant.

A Simple Example of a Linear Model

Here, I'll show the results of a very simple linear model, relating the volume of wood in a cherry tree to its girth (diameter of trunk). The data is in the data frame `trees` that comes with R.

Here's a plot of the data:



Fitting the Model with `lm`

We can fit a linear model for volume given girth as follows:

```
> lm (trees$Volume ~ trees$Girth)


Call:
lm(formula = trees$Volume ~ trees$Girth)


Coefficients:
(Intercept)  trees$Girth
    -36.943        5.066
```

The result says that best fit model for the volume is

$$\text{Volume} \;=\; -36.943 \;+\; 5.066\,\text{Girth} \;+\; \text{noise}$$

We can get the same result with an abbreviated formula by saying the data comes from the data frame `trees`:

```
lm (Volume ~ Girth, data=trees)
```

Using the Result of `lm`

The value returned by `lm` is an object of class `"lm"`, which has special methods for printing and other operations.

We can save the result, and then get the regression coefficients with `coef`.

```
> m <- lm (Volume ~ Girth, data=trees)
> coef(m)
(Intercept)       Girth
 -36.943459    5.065856
```

We could use these coefficients to predict the volume for a new tree, with girth of 11.6:

```
> coef(m) %*% c(1,11.6)   # %*% will compute the dot product
         [,1]
[1,] 21.82048
```

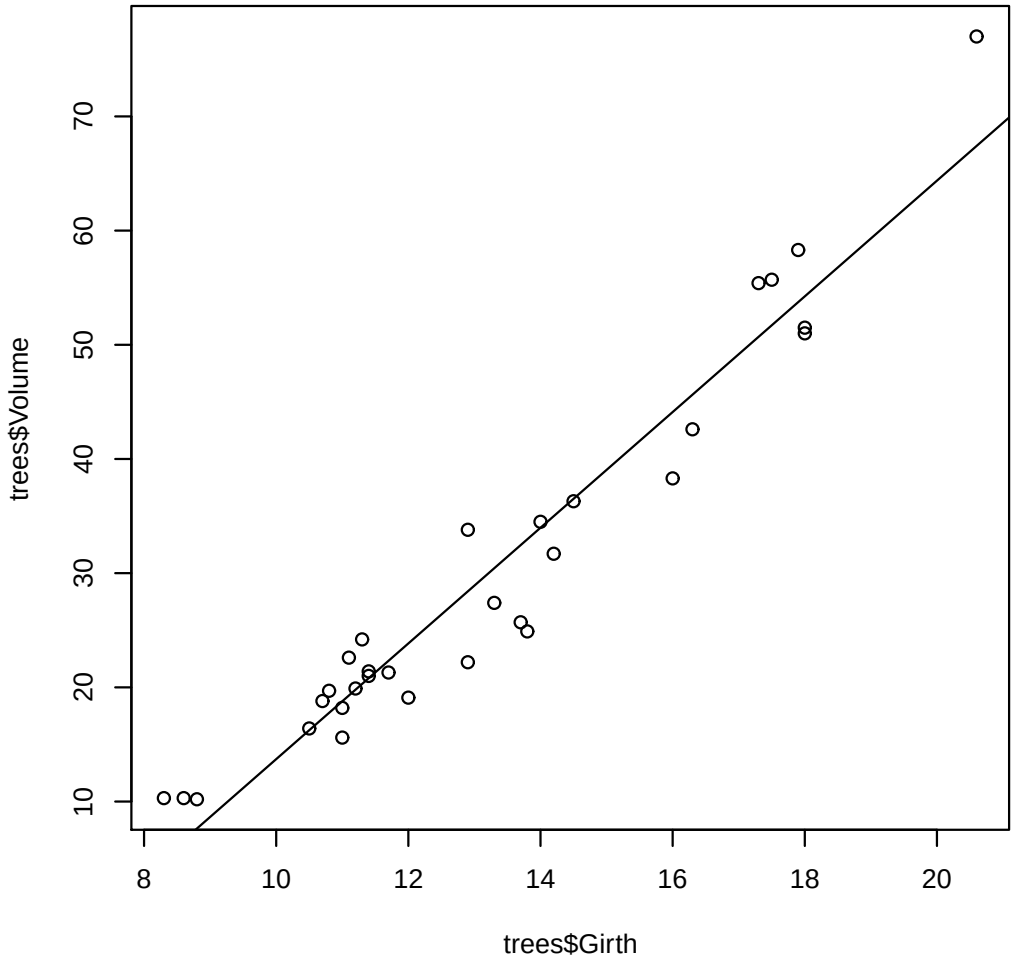
Getting More Details on the Model Fitted

We can also ask for more statistical details with `summary`:

```
> summary(m)

Call:
lm(formula = Volume ~ Girth, data = trees)

Residuals:
   Min     1Q Median     3Q    Max
-8.065 -3.107  0.152  3.495  9.587

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -36.9435     3.3651  -10.98 7.62e-12 ***
Girth         5.0659     0.2474   20.48  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.252 on 29 degrees of freedom
Multiple R-squared: 0.9353,    Adjusted R-squared: 0.9331
F-statistic: 419.4 on 1 and 29 DF,  p-value: < 2.2e-16
```

Plotting the Regression Line

We can also plot the regression line from the fitted model on top of a scatterplot of the data, using `abline(m)`:



The plot shows some indication that the relationship is actually curved.

Trying a Quadratic Model

Let's try fitting volume to both girth and the square of girth:

```
> Girth_squared <- trees$Girth^2
> summary (lm (trees$Volume ~ trees$Girth + Girth_squared))
Call:
lm(formula = trees$Volume ~ trees$Girth + Girth_squared)

Residuals:
    Min      1Q  Median      3Q     Max
-5.4889 -2.4293 -0.3718  2.0764  7.6447

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)   10.78627   11.22282   0.961 0.344728
trees$Girth   -2.09214    1.64734  -1.270 0.214534
Girth_squared  0.25454    0.05817   4.376 0.000152 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.335 on 28 degrees of freedom
Multiple R-squared: 0.9616,    Adjusted R-squared: 0.9588
F-statistic: 350.5 on 2 and 28 DF,  p-value: < 2.2e-16
```

R Packages

The basic features of R can be extended using "packages" of function definitions, data sets, etc.

These packages are available in several ways:

- A few packages come with R and are available for use by default. One such is the `stats` package that defines the `lm` function.
- Some more packages come with R, but have to be loaded manually before you can use them. An example is the `survival` package for analysing survival data. To use it, say `library(survival)`.
- Many other packages (thousands) are available for installation from package repositories, to which many people have contributed. The CRAN repository is the best-known, and is the default when you try to install such packages using the `install.packages` function.
- You can write your own packages.

Some Things Packages May Do

- Provide general extensions to the R language.
  Example: `magrittr` defines an operator `%>%` for "piping" data from one function to another.
- Provide convenient ways of interacting with R.
  Example: `knitr`, which we've been using to produce convenient output.
- Interface to other software.
  Example: `foreign` provides ways of reading data from other statistics packages (eg, SAS) into R.
- Provide more elaborate ways of producing graphical output.
  Example: `ggplot2` is a very popular way of producing graphs.
- Provide additional statistical methods.
  Example: `tgp` implements "Treed Gaussian Process" models, that can model how a response variable relates to explanatory variables more flexibly than a linear model.