Rationale for MASSExtra

Bill Venables

2020-12-06

Preamble

This extension package to the classical `MASS` package (Venables & Ripley, of ancient lineage), whose origins go back to nearly 30 years, comes about for a number of reasons.

Firstly, in my teaching I found I was using some of the old functions in the package with consistently different argument settings to the defaults. I was also interested in supplying various convenience extensions that simplified teaching and including various tweaks to improve the interface. Examples follow below.

Secondly, I wanted to provide a few functions that were mainly useful as programming examples. For example, the function `zs` and its allies `zu`, `zq` and `zr` are mainly alternatives to `base::scale`, but they can be used to show how to write functions that can be used in fitting models in such a way that they work as they should when the fitted model object is used for prediction with new data.

Masking `select` from other packages

Finally, there is the perennial `select` problem. When `MASS` is used with other packages, such as `dplyr` the `select` function can easly be masked, causing confusion with users. `MASS::select` is rarely used, but `dplyr::select` is fundamental. There are standard ways of managing this kind of masking, but what we have done in `MASSExtra` is to export the more common functions used from `MASS` along with the extensions, in such a way that users will not need to have `MASS` attached to the search path at all, and hence masking is unlikely.

The remainder of this document will do a walk-through of some of the new functions provided by the package. We begin by setting the computational context:

```
suppressPackageStartupMessages({
  library(ggwebthemes) ## https://gitlab.com/peterbar/ggwebthemes/
  library(visreg)
  library(knitr)
  library(tidyverse)
  library(patchwork)
  library(MASSExtra)
})
options(knitr.kable.NA = "")
theme_set(theme_web_bw() + theme(title = element_text(hjust = 0.5)))
```

Amble

We now consider some of the extensions that the package offers to the originals. Most of the extensions will have a name that includes an underscore of two somewhere to distinguish it from the V&R original. Note that the original version is *also* exported so that scripts that use it may do so without change, via the new package.

The `box_cox` extensions

This original version, `boxcox` has a fairly rigid display for the plotted output which has been changed to give a more easily appreciated result. The $y-$axis has been changed to give the likelihood-ratio statistic rather

than the log-likelihood, and for the $x-$axis some attempt has been made to focus on the crucial region for the transformation parameter, λ,

The following example shows the old and new plot versions for a simple example.

```
par(mfrow = c(1, 2))
mod0 <- lm(MPG.city ~ Weight, Cars93)
boxcox(mod0)  ## MASS
box_cox(mod0) ## MASSExtra tweak
```

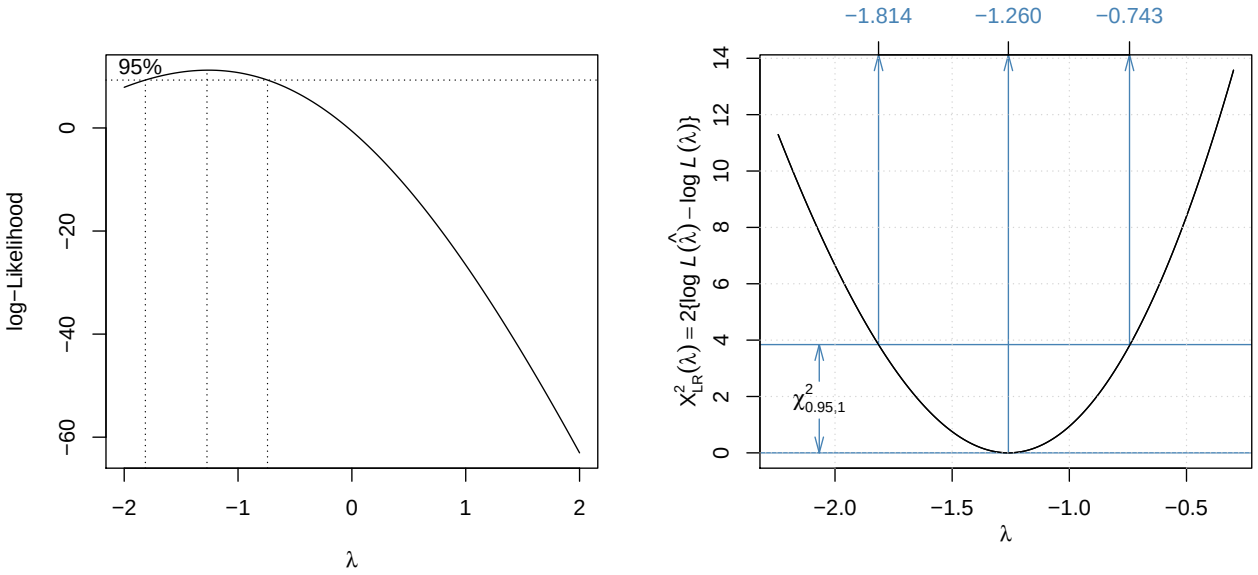

Figure 1: Box-cox, old and new displays

In addition, there are functions `bc` to evaluate the transformation for a given exponent, and a function `lambda` which finds the optimum exponent (not that a precise exponent will usually be needed).

It is interesting to see how in this instance the transformation can both straighten the relationship and provide a scale in which the variance is more homogeneous. See Figure 2.

```
p0 <- ggplot(Cars93) + aes(x = Weight) + geom_point(colour = "#2297E6")  + xlab("Weight (lbs)") +
  geom_smooth(se = FALSE, method = "loess", formula = y ~ x, size=0.7, colour = "black")
p1 <- p0 + aes(y = MPG.city) + ylab("Miles per gallon (MPG)") + ggtitle("Untransformed response")
p2 <- p0 + aes(y = bc(MPG.city, lambda(mod0))) + ggtitle("Transformed response") +
  ylab(bquote(bc(MPG, .(round(lambda(mod0), 2)))))
p1 + p2
```

A more natural scale to use, consistent with the Box-Cox suggestion, would be the reciprocal. For example we could use GPM = 100/MPG the "gallons per 100 miles" scale, which would have the added benefit of being more-or-less what the rest of the world uses to gauge fuel efficiency outside the USA. Readers should try this for themselves.

Stepwise model building extensions

The primary `MASS` functions for refining linear models and their allies are `dropterm` and `stepAIC`. The package provides a few extensions to these, but mainly a change of defaults in the argument settings.

1. `drop_term` is a front-end to `MASS::dropterm` with a few tweaks. By default the result is arranged in sorted order, i.e. with `sorted = TRUE`, and also by default with `test = TRUE` (somewhat in defiance of much advice to the contrary given by experienced practitioners: *caveat emptor!*).

   The user may specify the test to use in the normal way, but the default test is decided by an ancillary

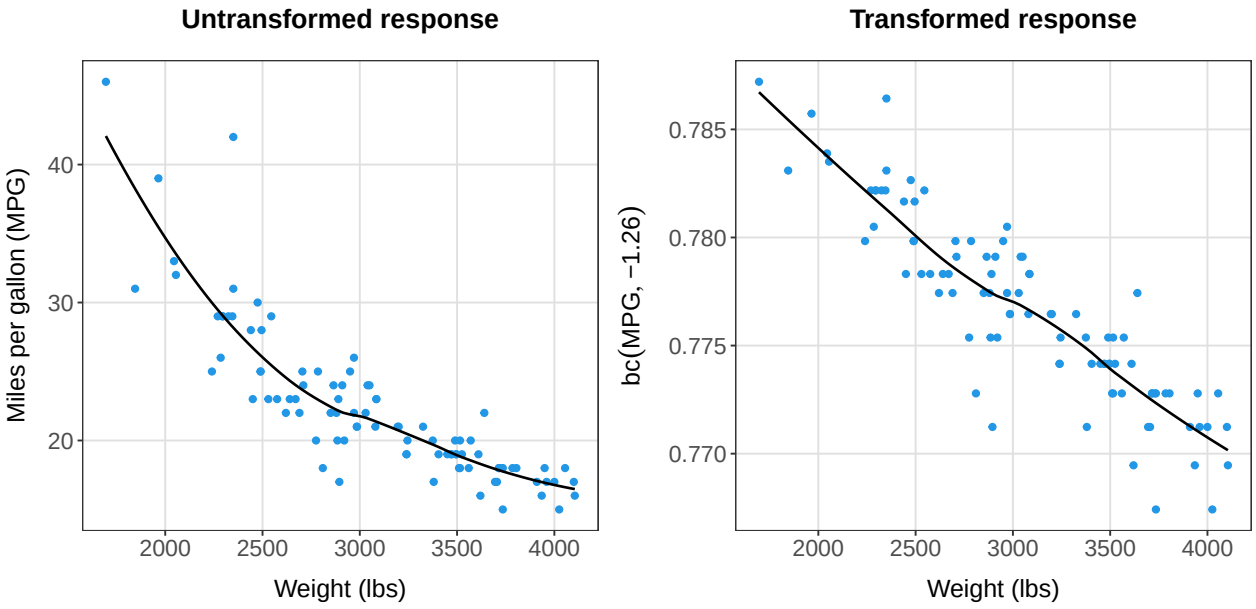

Figure 2: The Box-Cox transformation effect

generic function, `default_test`, which guesses the appropriate test from the object itself. This is an S3 generic and further methods can be supplied for new fitted model objects.

In addition `drop_term` returns and object which retains information on the criterion used, `AIC`, `BIC`, `GIC` (see below) or some specific penalty value `k`. The object also has a class `"drop_term"` for which a `plot` method is provided. Both the `plot` and `print` methods display the criterion. See the example below for how this is done.

2. `step_AIC` is a front-end to `MASS::stepAIC` with the default argument `trace = FALSE` set. This may of course be over-ruled, but it seems the most frequent choice by users, anyway. In addition the actual criterion used, by dafault `k = 2`, i.e. AIC, is retained with the result and passed on to methods in much the same say as for `drop_term` above.

   Since the (default) criterion name is encoded in the function name, two further versions are supplied, namely `step_BIC` and `step_GIC` (again, see below), which use a different, and obvious, default criterion.

   In any of `step_AIC`, `step_BIC` or `step_GIC` a different value of `k` may be specified in which case that value of `k` is retained with the object and displayed as appropriate in further methods.

   Finally in any of these functions `k` may be specified either as a numeric penalty, such as `k = 4` for example, or by character string `k = "AIC"` or `k = "BIC"` with an obvious meaning in either case.

3. Criteria. The **Akaike Information Criterion**, AIC, corresponds to a penalty `k = 2` and the **Bayesian Information Criterion**, BIC, corresponds to `k = log(n)` where `n` is the sample size. In addition to these two the present functions offer an intermediate default penalty `k = (2 + log(n))/2` which is "not too strong and not too weak", making it the **Goldilocks Information Criterion**, GIC. There is also a standalone function `GIC` to evaluate this `k` if need be.

   This suggestion appears to be original, but *no particular claim is made for it* other than with intermediate to largish data sets it has proved useful for exploratory purposes in our experience.

   Our strong advice is that these tools should ***only*** be used for exploratory purposes in any case, and should ***never*** be used in isolation. They have a well-deserved very negative reputation when misused, as they commonly are.

Examples

We consider the well-known and much maligned Boston house price data.