

Statistics 360: Advanced R for Data Science

MARS, part IV

Brad McNeney

More details on the implementation

- ▶ Input
- ▶ Pre-processing
- ▶ Output
- ▶ Methods

Software arguments/inputs

- ▶ Formula interface to specify response and explanatory variables.
- ▶ data argument for input data
- ▶ object of class `mars.control` that is a list comprised of elements `Mmax`, `d` and `trace`. `Mmax` is for the forward, `d` is for the backward and `trace` is to print details of the fitting process.
 - ▶ Write a constructor, validator and helper function for this class. The helper should have defaults. If the user does not specify a `mars.control` object on input, the helper should create one with the defaults.

Pre-processing example: `lm()` vs `lm.fit()`

- ▶ `lm()` does a bit of pre-processing, including setup of the design matrix `x` and response variable `y`, and then calls `lm.fit()` to do the fitting.
- ▶ What we have so far in `fwd_selection()` and `bwd_selection()` amounts to an implementation of `lm.fit()`.
- ▶ We need code to transform the formula, data, and control parameters into the inputs to `fwd_selection()`.

lm()

- `lm()` is very flexible and can read data from a data argument or the calling environment.

```
function (formula, data, subset, weights, na.action, method = "qr",
  model = TRUE, x = FALSE, y = FALSE, qr = TRUE, singular.ok = TRUE,
  contrasts = NULL, offset, ...)
{
  ret.x <- x
  ret.y <- y
  cl <- match.call()
  mf <- match.call(expand.dots = FALSE)
  m <- match(c("formula", "data", "subset", "weights", "na.action",
    "offset"), names(mf), 0L)
  mf <- mf[c(1L, m)]
  mf$drop.unused.levels <- TRUE
  mf[[1L]] <- quote(stats::model.frame)
  mf <- eval(mf, parent.frame())
  if (method == "model.frame")
    return(mf)
  else if (method != "qr")
    warning(gettextf("method = '%s' is not supported. Using 'qr'",
      method), domain = NA)
  mt <- attr(mf, "terms")
  y <- model.response(mf, "numeric")
  w <- as.vector(model.weights(mf))
  if (!is.null(w) && !is.numeric(w))
    stop("'weights' must be a numeric vector")
  offset <- model.offset(mf)
  mlm <- is.matrix(y)
  ny <- if (mlm)
    nrow(y)
  else length(y)
  if (!is.null(offset)) {
    if (!mlm)
```

Pre-processing: model frames

- ▶ We'll simplify ...
- ▶ The data frame and formula are bundled into a “model frame”, which is the data frame plus a terms attribute.
- ▶ R has tools for extracting the response and design matrix from a model frame.

```
mars <- function(formula,data,control=NULL...) {  
  cc <- match.call() # save the call  
  mf <- model.frame(formula,data)  
  y <- model.response(mf)  
  mt <- attr(mf, "terms")  
  x <- model.matrix(mt, mf)  
  # if(is.null(control)) call the helper with default values  
  # Then you are ready to go ...  
}
```

Value/output

- ▶ object of S3 class `mars`.
- ▶ inherits from class `lm` and includes all of the components of the `lm()` from the final fit
 - ▶ Use `c()` to combine these with any of your own components.
- ▶ include `splits` data structure from final fit.
- ▶ write a constructor for this class – no need for a validator or helper since you are the only one who will call the constructor.

Methods

- ▶ Use `methods()` to find a list of methods implemented for the S3 class `lm`.
- ▶ Write more informative `print` and `summary` methods for `mars` objects
- ▶ Write a `plot` method.
 - ▶ The details are up to you, but you should consult Section 3.5 of the Friedman paper (ANOVA decomposition).
 - ▶ Two sources of inspiration are the `plot.earth` method for `earth` objects (see the `earth` package), and `plot.Gam` for plotting generalized additive model components (see the `gam` package).
- ▶ Write a `predict` method with the same interface as `predict.lm`.
- ▶ `residuals()`, `fitted()`, `hatvalues()` and others that depend only on the final `lm` can be used as-is