Notes on the `earth` package

Stephen Milborrow

July 19, 2011

Contents

1 Introduction

This document is a set of notes to accompany the `earth` package [17]. It comes with the `earth` package, or can be downloaded from `www.milbo.org/doc/earth-notes.pdf`.

Most users will find it unnecessary to read this entire document. Just read the parts you need and skim the rest. Most of this text was originally written in response to email from users.

The R `earth` package builds regression models using the techniques in Friedman's papers "Multivariate Adaptive Regression Splines" [7] and "Fast MARS" [8]. The term "MARS" is copyrighted and thus not used in the name of the package. The package can be downloaded from `cran.r-project.org/web/packages/earth/index.html`.

2 Overview

Earth has numerous arguments, but many users will find that the following are all they need:

`formula, data`	Familiar from `lm`.
`x, y`	Alternative to the formula interface.
`degree`	The maximum degree of interaction. Default is `1`, use `2` for first-order interactions of the hinge functions.
`nk`	The maximum number of MARS terms. The default is determined semi-automatically from the number of predictors in `x`, but sometimes needs adjusting.
`trace`	Trace operation. Use `trace=1` to see why the forward pass terminated, and if necessary increase `nk`.

2.1 References

The Wikipedia article [21] is recommended for an elementary introduction to MARS `http://en.wikipedia.org/wiki/Multivariate_adaptive_regression_splines`.

The primary references are the Friedman MARS papers [7, 8]. Readers may find the MARS section in Hastie, Tibshirani, and Friedman [12] a more accessible introduction.

Faraway [5] takes a hands-on approach, using the ozone data to compare `mda::mars` with other techniques. (If you use Faraway's examples with `earth` instead of `mars`, use `$bx` instead of `$x`, and check out the book's errata.)

Friedman and Silverman [9] is recommended background reading for the MARS paper.

Earth's pruning pass uses the `leaps` [16] package which is based on techniques in Miller [18].

2.2 Other implementations

Given the same data, `earth` models are similar to but not identical to models built by other MARS implementations. The differences stem from the forward pass where small implementation differences (or perturbations of the input data) can cause somewhat different selection of terms and knots (although similar GRSq's). The backward passes give identical or near identical results, given the same forward pass results.

The source code of `earth` is derived from the function `mars` in the `mda` package written by Trevor Hastie and Robert Tibshirani [13]. See also the function `mars.to.earth` (in the `earth` package).

The term "MARS" is trademarked and licensed exclusively to Salford Systems `www.salfordsystems.com`. Their implementation uses an engine written by Friedman. It has a graphical user interface and includes some features not in `earth`. Salford Systems has a reputation for excellent customer support.

StatSoft also have an implementation which they call "MARSplines" `www.statsoft.com/textbook/stmars.html`.

2.3 Limitations

The following aspects of MARS are mentioned in Friedman's papers but not implemented in `earth`:

(i) Piecewise cubic models (to smooth out sharpness at the hinges).
(ii) Model slicing (the `plotmo` function goes part way).
(iii) Handling missing values.
(iv) Automatic grouping of categorical predictors into subsets.
(v) The h parameter of Fast MARS.

2.4 The forward pass

Understanding the details of the forward and pruning passes will help you understand `earth`'s return value and the admittedly large number of arguments. Figure 1 is an overview.

The result of the forward pass is the MARS basis matrix `bx` and the set of terms defined by `dirs` and `cuts` (these are all fields in `earth`'s return value, but the `bx` returned by the forward pass includes all terms before trimming back to `selected.terms`).

The `bx` matrix has a row for every observation (i.e. for every row in `x`). It has a column for each basis function (also referred to as a MARS term). An example `bx`:

```
     (Intercept)   h(x1-58)   h(x2-89)   h(89-x2)   h(56-x3)*h(x1-58)  ...
[1,]           1        3.2          0         56                   0  ...
[2,]           1        8.1          0         55                   0  ...
[3,]           1        3.7          0         54                   0  ...
....
```

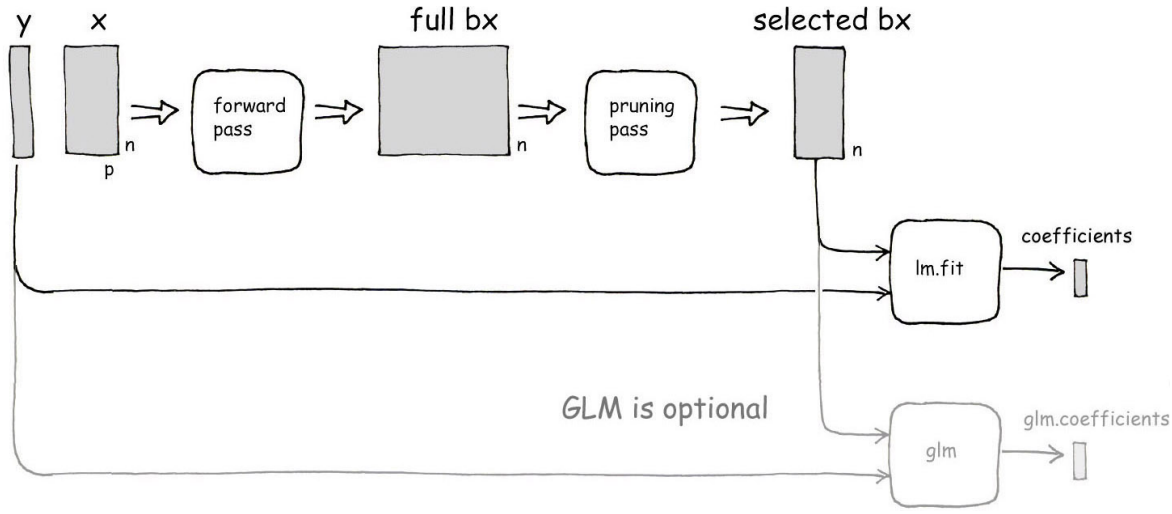

Figure 1: *Overview of* `earth`*'s internals*

2.4.1 Termination conditions for the forward pass

The forward pass adds terms in pairs until the first of the following conditions is met:

(i) Reached the maximum number of terms `nk`
(ii) Adding a term changes RSq by less than 0.001
(iii) Reached a RSq of 0.999 or more
(iv) GRSq is less than -10 (a pathologically bad GRSq, FAQs 12.10 and 12.11)
(v) Reached numerical accuracy limits (no new term increases RSq).

Set `trace >= 1` to see the stopping condition and `trace >= 2` to trace the forward pass.

The numbers 0.001 and 0.999 above can be changed by changing `earth`'s `thresh` argument. You can disable termination conditions ii, iii, and iv by setting `thresh=0` (FAQ 12.13). These conditions are not theoretically necessary, but they save time by terminating the forward pass when it is pointless to continue, and also usually terminate the search before numerical issues become thorny.

Note that GCVs (via GRSq) are used during the forward pass only as one of the (more unusual) stopping conditions and in `trace` prints. Changing the `penalty` argument does not change the knot positions.

The various stopping conditions mean that the actual number of terms created by the forward pass may be less than `nk`. There are other reasons why the actual number of terms may be less than `nk`: (i) the forward pass discards one side of a term pair if it adds nothing to the model — but the forward pass counts terms as if they were actually created in pairs, and, (ii) as a final step, the forward pass deletes linearly dependent terms, if any, so all terms in `dirs` and `cuts` are independent. And remember that the pruning pass will further discard terms.

2.5 The pruning pass

The pruning pass (also called the backward pass) is handed the set of terms `bx` created by the forward pass. Its job is to find the subset of those terms that gives the lowest GCV. The following description of the pruning pass explains how various fields in `earth`'s returned value are generated.

The pruning pass works like this: it determines the subset of terms in `bx` (using `pmethod`) with the lowest RSS (residual sum-of-squares) for each model size in `1:nprune`. It saves the RSS and term numbers for each such subset in `rss.per.subset` and `prune.terms`. It then calculates the GCV with `penalty` for each entry of `rss.per.subset` to yield `gcv.per.subset`. Finally it chooses the model with the lowest value in `gcv.per.subset`, puts its term numbers into `selected.terms`, and updates `bx` by keeping only the `selected.terms`.

After the pruning pass, `earth` runs `lm.fit` to determine the `fitted.values`, `residuals`, and `coefficients`, by regressing the response `y` on `bx`. This is an ordinary least-squares regression of the response `y` on the basis matrix `bx` (see Figure 1 and `example (model.matrix.earth)` for an example). If `y` has multiple columns then `lm.fit` is called for each column.

If a `glm` argument is passed to `earth` (Chapter 3), `earth` runs `glm` on (each column of) `y` in addition to the above call to `lm.fit`.

Set `trace >= 3` to trace the pruning pass.

2.6 Execution time

For a given set of input data, the following can increase the speed of the forward pass:

(i) decreasing `degree` (because there are fewer combinations of terms to consider),

(ii) decreasing `nk` (because there are fewer forward pass terms),

(iii) decreasing `fast.k` (because there are fewer potential parents to consider at each forward step),

(iv) increasing `thresh` (faster if there are fewer forward pass terms),

(v) increasing `min.span` (because fewer knots need to be considered).

The backward pass is normally much faster than the forward pass, unless `pmethod = "exhaustive"`. Reducing `nprune` reduces exhaustive search time. One strategy is to first build a large model and then adjust pruning parameters such as `nprune` using `update.earth`.

The following very rough rules of thumb apply for large models. Using `minspan=1` instead of the default `0` will increase times by 20 to 50%. Using `fast.k=5` instead of the default `20` can give substantial speed gains but will sometimes give a much smaller GRSq.

2.7 Memory use

Earth does not impose specific limits on the model size. Model size is limited only by the amount of memory on your system, the maximum memory addressable by R, and your patience.

To reduce total memory usage, it sometimes helps to remove variables (use R's `remove` function) and to call `gc` before invoking `earth`. Note that increasing the `degree` does not change the memory requirements (but increases the running time).

Memory use will be minimized if ALL the following requirements are met (because `earth` does not have to make its own copy of the `x` matrix):

(i) use `earth.default`, not `earth.formula` (i.e. invoke `earth` with `x,y` not with a formula),

(ii) `x` must be a `matrix` of `double`'s (not a `data.frame`),

(iii) `x` must have column names,

(iv) arguments like `subset` must be at their default value of `NULL`,

(v) `trace` must be less than 2.

The special value `trace=1.5` will make `earth`'s C routines print memory allocations.

Memory requirements will be reduced if `Use.beta.cache=FALSE` (use `trace=1.5` to see by how much).

2.8 Standard model functions

Standard model functions such as `case.names` are provided for `earth` objects and are not explicitly documented. Many of these give warnings when the results are not what you may expect. Pass `warn=FALSE` to these functions to turn of just these warnings. The full list of `earth` methods is:

`anova.earth`,
`case.names.earth`,
`deviance.earth`,
`effects.earth`,
`extractAIC.earth`,
`family.earth`,
`model.matrix.earth`,
`plot.earth`,
`print.earth`,
`print.summary.earth`,
`resid.earth`,
`residuals.earth`,
`summary.earth`,
`update.earth`,
`variable.names.earth`.

2.9 Multiple response models

If the response `y` has `k` columns then `earth` builds `k` simultaneous models. (Note: this will be the case if a factor in `y` is expanded by `earth`; Chapter 4 "Factors".) Each model has the same set of basis functions (the same `bx`, `selected.terms`, `dirs` and `cuts`) but different coefficients (the returned `coefficients` will have `k` columns). The models are built and pruned as usual but with the GCVs and RSSs summed across all `k` responses. Earth minimizes the overall GCV (the sum of the GCVs).

Once you have built your model, you can use `plotmo` and its `nresponse` argument to see how each response varies with the predictors.

Here are a couple of examples to show some of the ways multiple responses can be specified. Note that `data.frame`s can't be used on the left side of a formula, so `cbind` is used in the first example. The first example uses the standard technique of specifying a tag `ly2=` to name a column.

```
earth(cbind(y1, ly2=log(y2)) ~ ., data = my.data)
attach(my.data)
earth(data.frame(y1, y2), data.frame(x1, x2, log.x3=log(x3)))
```

Since `earth` attempts to optimize for all models simultaneously, the results will not be as "good" as building the models independently, i.e., the GRSq of the combined model will usually not be as good as the GRSq's for independently built models. However, the combined model may be a better model in other senses, depending on what you are trying to achieve. For example, it could be useful for `earth` to select the set of MARS terms that is best across *all* responses. This would typically be the case in a multiple response logistic model if some responses have a very small number of successes.

Note that automatic scaling of `y` (via the `scale.y` argument) does not take place if `y` has multiple columns. You may want to scale your `y` columns before calling `earth` so each `y` column gets the appropriate weight during model building (a `y` column with a big variance will influence the model more than a column with a small variance). You could do this by calling `scale` before invoking `earth`, or by setting the `scale.y` argument, or by using the `wp` argument.

Don't use a plus sign on the left side of the tilde. You might think that specifies a multiple response, but instead it arithmetically adds the columns.

For more details on using residual errors averaged over multiple responses see for example Section 4.1 of the FDA paper (Hastie, Tibshirani, and Buja [11]).

2.10 Migrating from mda::mars

Changing code from `mda::mars` to `earth` is usually just a matter of changing the call from `mars` to `earth`. But there are a few argument differences and `earth` will issue a warning if you give it a `mars`-only argument.

The resulting model will be similar but not identical because of small implementation differences. See also the documentation of the function `mars.to.earth` (in the `earth` package).

If you are further processing the output of `earth` you will need to consider differences in the returned value. The header of the source file `mars.to.earth.R` describes these. Perhaps the most important is that `mars` returns the MARS basis matrix in a field called "`x`" whereas `earth` returns "`bx`". Also, `earth` returns "`dirs`" rather than "`factors`".

A note on `wp` argument. Earth's internal normalization of `wp` is different from `mars`. Earth uses `wp <- sqrt(wp/mean(wp))` and `mars` uses `wp <- sqrt(wp/sum(wp))`. Thus in `earth`, a `wp` with all elements equal is equivalent to no `wp`. For models built with `wp`, multiply the GCV calculated by `mars` by `length(wp)` to compare it to `earth`'s GCV.

3 Generalized linear models

Earth builds a GLM model if the `glm` argument is specified. Earth builds the model as usual and then invokes `glm` on the MARS basis matrix `bx`.

In more detail, the model is built as follows. Earth first builds a standard MARS model, including the internal call to `lm.fit` on `bx` after the pruning pass. (See Figure 1 and Section 2.5 "The pruning pass".) Thus knot positions and terms are determined as usual and all the standard fields in `earth`'s return value will be present. Earth then invokes `glm` for the response on `bx` with the parameters specified in the `glm` argument to `earth`. For multiple response models (when `y` has multiple columns), the call to `glm` is repeated independently for each response. The results go into three extra fields in `earth`'s return value: `glm.list`, `glm.coefficients`, and `glm.bpairs`.

Earth's internal call to `glm` is made with the `glm` arguments `x`, `y`, and `model` set TRUE (see the documentation for `glm` for more information about those arguments).

Use `summary(my.model)` as usual to see the model. Use `summary(my.model, details=T)` to see more details, but note that the printed t-values for the GLM coefficients are meaningless. This is because of the amount of preprocessing by `earth` — the mantra is "variable selection overstates significance of the selected variables". And anyway, we already know that the MARS terms are significant — the forward and backward passes just did a lot of work carefully choosing those terms.

Use `plot(my.model$glm.list[[1]])` to plot the (first) `glm` model.

The approach used for GLMs in `earth` was motivated by work done by Jane Elith and John Leathwick ([15] is a representative paper).

3.1 GLM examples

The examples below show how to specify earth-glm models. The examples are only to illustrate the syntax and not necessarily useful models. In some of the examples, `pmethod="none"`, otherwise with these artificial models `earth` tends to prune away everything except the intercept term. You wouldn't normally use `pmethod="none"`. Also, `trace=1`, so if you run these examples you can see how `earth` expands the input matrices (as explained in Chapter 4 "Factors" and Section 3.2 "Binomial pairs").

(i) **Two-level factor or logical response.** The response is converted to a single column of 1s and 0s.

```
a1 <- earth(survived ~ ., data=etitanic,
            degree=2, trace=1, glm=list(family=binomial))

# equivalent but using earth.default
a1a <- earth(etitanic[,-2], etitanic[,2],
            degree=2, trace=1, glm=list(family=binomial))
```

(ii) **Factor response.** This example is for a factor with more than two levels. (For factors with just two levels, see the previous example.) The factor `pclass` is expanded

to three indicator columns (whereas in a direct call to `glm`, `pclass` would be treated as logical: the first level versus all other levels). Because of the masking problem, we mention you might consider FDA for factor responses with more than two levels (Chapter 7).

```
a2 <- earth(pclass ~ ., data=etitanic, trace=1,
            glm=list(family=binomial))
```

(iii) **Binomial model specified with a column pair.** This is a single response model but specified with a pair of columns (Section 3.2 "Binomial pairs"). For variety, this example uses a `probit` link and (unnecessarily) increases `maxit`.

```
ldose <- rep(0:5, 2) - 2 # V&R 4th ed. p. 191
sex <- factor(rep(c("male", "female"), times=c(6,6)))
numdead <- c(1,4,9,13,18,20,0,2,6,10,12,16)
pair <- cbind(numdead, numalive=20 - numdead)
a3 <- earth(pair ~ sex + ldose, trace=1, pmethod="none",
            glm=list(family=binomial(link=probit), maxit=100))
```

(iv) **Double binomial response** (i.e., a multiple response model) specified with two column pairs.

```
numdead2 <- c(2,8,11,12,20,23,0,4,6,16,12,14) # bogus data
doublepair <- cbind(numdead, numalive=20-numdead,
                    numdead2=numdead2, numalive2=30-numdead2)
a4 <- earth(doublepair ~ sex + ldose, trace=1, pmethod="none",
            glm=list(family="binomial"))
```

(v) **Poisson model.**

```
counts <- c(18,17,15,20,10,20,25,13,12) # Dobson 1990 p. 93
outcome <- gl(3,1,9)
treatment <- gl(3,3)
a5 <- earth(counts ~ outcome + treatment, trace=1, pmethod="none",
            glm=list(family=poisson))
```

(vi) **Standard earth model**, the long way.

```
a6 <- earth(numdead ~ sex + ldose, trace=1, pmethod="none",
            glm=list(family=gaussian(link=identity)))
print(a6$coefficients == a6$glm.coefficients)  # all TRUE
```

3.2 Binomial pairs

This section is only relevant if you use `earth`'s `glm` argument with a binomial or quasibinomial `family`.

Users of the `glm` function will be familiar with the technique of specifying a binomial response as a two-column matrix, with a column for the number of successes and a column for the failures. When given the argument `glm=list(family=binomial)`, `earth`

automatically detects when such columns are present in `y` (by looking for adjacent columns which both have entries greater than 1). The first column only is used to build the standard `earth` model. Both columns are then passed to `earth`'s internal call to `glm`. As always, use `trace=1` to see how the columns of `x` and `y` are expanded.

You can override this automatic detection by including a `bpairs` parameter. This is usually (always?) unnecessary. For example

```
glm=list(family=binomial, bpairs=c(TRUE, FALSE))
```

specifies that there are two columns in the response with the second paired with the first. These examples

```
glm=list(family=binomial, bpairs=c(TRUE, FALSE, TRUE, FALSE))
glm=list(family=binomial, bpairs=c(1,3)) # equivalent
```

specify that the 1st and 2nd columns are a binomial pair and the 3rd and 4th columns another binomial pair.

4 Factors

This chapter explains how factors in the `x` and `y` matrices get "expanded" before the matrices get passed to the MARS engine.

Use `trace=1` or higher to see the column names of the `x` and `y` matrices after factor expansion. Use `trace=4` to see the first few rows of `x` and `y` after factor expansion.

4.1 Factors in x

Earth treats factors in `x` in the same way as standard R models such as `lm`. Thus factors are expanded using the current setting of `contrasts`.

4.2 Factors in y

Earth treats factors in the response in a non-standard way that makes use of `earth`'s ability to handle multiple responses.

A *two level factor* (or logical) is converted to a single indicator column of 1s and 0s.

A *factor with three or more levels* is converted into `k` indicator columns of 1s and 0s, where `k` is the number of levels (the `contrasts` matrix is an identity matrix, see the help page for `contr.earth.response`). This happens regardless of the `options("contrasts")` setting and regardless of whether the factors are ordered or unordered. For example, if a column in `y` is a factor with levels `A`, `B`, and `C`, the column will be expanded to three columns like this (the actual data will vary but each row will have a single 1):

```
A B C  # one column for each factor level
0 1 0  # each row has a single 1
1 0 0
0 0 1
0 0 1
0 1 0
...
```

In distinction, a standard treatment contrast on the right hand size of a model with an intercept would have no first "A" column (to prevent linear dependencies on the right side of the model formula). See the help page for `contrasts` for details.

This expansion to multiple columns (which only happens for factors with more than two levels) means that `earth` will build a multiple response model as described in Section 2.9 "Multiple responses".

Paired binomial response columns in `y` are treated specially (Section 3.2 "Binomial pairs").

4.3 Factor example

Here is an example which uses the `etitanic` data to predict the passenger class (not necessarily a sensible thing to do):

```
> data(etitanic)
> head(etitanic) # pclass and sex are unordered factors

  pclass survived    sex    age sibsp parch
1    1st        1 female 29.000     0     0
2    1st        1   male  0.917     1     2
3    1st        0 female  2.000     1     2

> earth(pclass ~ ., data=etitanic, trace=1) # note col names in x and y below

x is a 1046 by 5 matrix: 1=survived, 2=sexmale, 3=age, 4=sibsp, 5=parch
y is a 1046 by 3 matrix: 1=1st, 2=2nd, 3=3rd
rest not shown here...
```

5 The `linpreds` argument

With `linpreds` we can specify which predictors should enter linearly, instead of in hinge functions. The `linpreds` argument does not stipulate that a predictor *must* enter the model, only that if it enters it should enter linearly. Starting with

```
fit1 <- earth(Volume ~ ., data = trees)
plotmo(fit1)
```

we see in the `plotmo` graphs (Figure 2, top row) or by running `evimp` that `Height` isn't as important as `Girth`. For collaborative evidence that `Girth` is a more reliable indicator of `Volume` we can use `pairs`:

```
pairs(trees, panel = panel.smooth)
```

Since we want the simplest model that describes the data, we may decide that `Height` should enter linearly, not in a hinge function (Figure 2, bottom row):

```
fit2 <- earth(Volume ~ ., data = trees, linpreds = 2)  # 2 is Height column
summary(fit2)
```

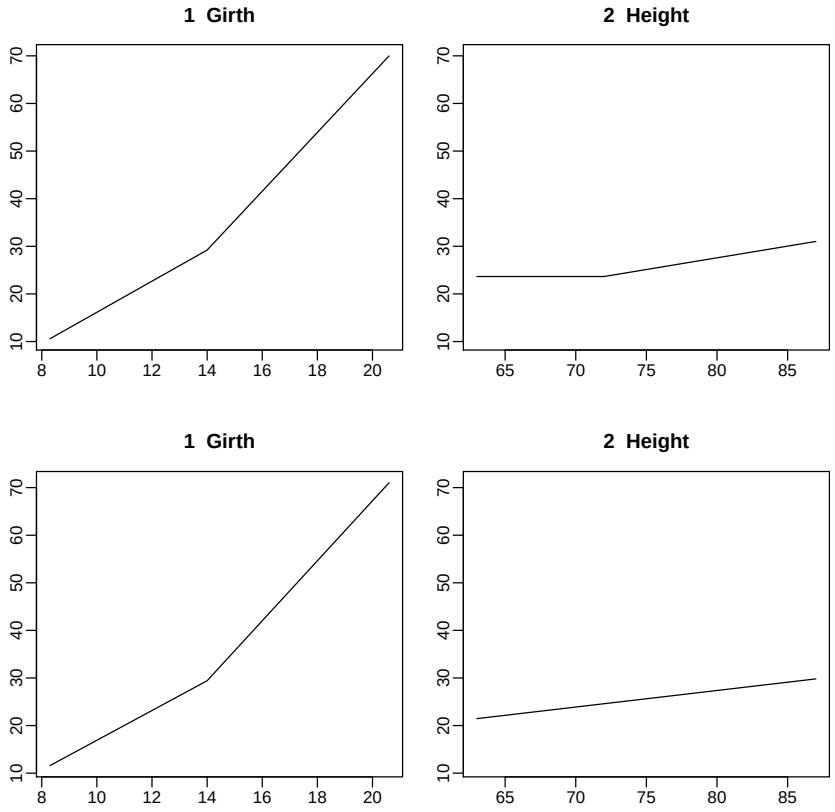

Figure 2: *The* `linpreds` *argument.*
Top row: Standard `earth` *model of the trees data.*
Bottom row: Same, but with Height entering linearly (`linpreds=2`*).*

which yields

```
             coefficients
(Intercept)         2.981
Height              0.348
h(Girth-14)         6.302
h(14-Girth)        -3.128
```

In this example, the second simpler model has almost the same RSS as the first model. We can make both `Girth` and `Height` enter linearly with

```
a3 <- earth(Volume ~ ., data = trees, linpreds = c(1,2))
```

or with (the single TRUE is recycled to the length of `linpreds`)

```
a4 <- earth(Volume ~ ., data = trees, linpreds = TRUE)
```

But specifying that all predictors should enter linearly is not really a useful thing to do. In our simple example, the all-linear MARS model is the same as a standard linear model

```
a5 <- lm(Volume ~ ., data = trees)
```

(compare the `summary` for each) but in general that will not be true. Earth will not include a linear predictor if that predictor does not improve the model (in the GCV sense).

Note: In the current implementation, the GCV penalty for predictors that enter linearly is the same as that for predictors with knots. That is not quite correct; linear terms should be penalized less.

6 The `allowed` argument

You can specify how variables are allowed to enter MARS terms with the `allowed` argument. Within each step of the forward pass, `earth` calls the allowed function after discovering the best knot for a variable. The potential term is considered for inclusion only if the `allowed` function returns `TRUE`. The default function always returns `TRUE`.

Your `allowed` function should have the following prototype

```
function(degree, pred, parents, namesx, first)
```

where

`degree` is the interaction degree of the candidate term. Will be 1 for additive terms.

`pred` is the index of the candidate predictor. A predictor's index in `pred` is the column number in the input matrix `x` after factors have been expanded. Use `trace=1` to see the column names after expansion.

`parents` is the candidate parent term's row in `dirs`.

`namesx` is optional and if present is the column names of `x` after factors have been expanded.

`first` is optional and if present is TRUE the first time your `allowed` function is invoked for the current model, and thereafter FALSE, i.e. it is TRUE once per invocation of `earth`.

6.1 Examples

The interface is flexible but requires a bit of programming. We start with a simple example, which completely excludes one predictor from the model:

```
example1  <- function(degree, pred, parents)   # returns TRUE if allowed
{
    pred != 2  # disallow predictor 2, which is "Height"
}
a1 <- earth(Volume ~ ., data = trees, allowed = example1)
print(summary(a1))
```

But that's not much use, because it's simpler to exclude the predictor from the input matrix when invoking `earth`:

```
a2 <- earth(Volume ~ . - Height, data = trees)
```

The example below is more useful. It prevents the specified predictor from being used in interaction terms. (The example is artificial because it's unlikely we would want to single out humidity from interactions in the ozone data.)

The `parents` argument is the candidate parent's row in the `dirs` matrix (`dirs` is described in the `Value` section of the `earth` help page). Each entry of `parents` is 0, 1, -1, or 2, and we index `parents` on the predictor index. Thus `parents[pred]` is non-zero if `pred` is in the parent term.

```
example2 <- function(degree, pred, parents)
{
    # disallow humidity in terms of degree > 1
    # 3 is the "humidity" column in the input matrix
    if (degree > 1 && (pred == 3 || parents[3]))
        return(FALSE)
    TRUE
}
a3 <- earth(O3 ~ ., data = ozone1, degree = 2, allowed = example2)
print(summary(a3))
```

The following example allows only the specified predictors in interaction terms. Interactions are allowed only for predictors in `allowed.set`, which you can change to suit your needs.

```
example3 <- function(degree, pred, parents)
{
    # allow only humidity and temp in terms of degree > 1
    # 3 and 4 are the "humidity" and "temp" columns
    allowed.set = c(3,4)
    if (degree > 1 &&
           (all(allowed.set != pred) || any(parents[-allowed.set])))
        return(FALSE)
    TRUE
}
a4 <- earth(O3 ~ ., data = ozone1, degree = 2, allowed = example3)
print(summary(a4))
```

6.2 Further notes on the `allowed` argument

The basic MARS model building strategy is always applied even when there is an `allowed` function. For example, `earth` considers a term for addition only if all factors of that term except the new one are already in a model term. This means that an `allowed` function that inhibits, say, all degree 2 terms will also effectively inhibit higher degrees too, because there will be no degree 2 terms for `earth` to extend to degree 3.

You can expect model building to be about 10% slower with an `allowed` function because of the time taken to invoke the `allowed` function. On the other hand, execution time can be faster if using `allowed` requires us to evaluate fewer potential MARS terms.

6.3 Using predictor names instead of indices in the `allowed` function.

You can use predictor names instead of indices using the optional `namesx` argument. If present, `namesx` is the column names of `x` after factors have been expanded. The first example above (the one that disallows `Height`) can be rewritten as

```
example1a  <- function(degree, pred, parents, namesx)
```

```
{
    namesx[pred] != "Height"
}
```

Comparing strings is inefficient [no longer true in modern versions of R] and the above example can be rewritten a little more efficiently using the optional `first` argument. If present, this is TRUE the first time your allowed function is called for the current model and thereafter FALSE.

```
iheight <- 0    # column index of "Height"

example1b  <- function(degree, pred, parents, namesx, first)
{
    if (first) {
        # first time this function is invoked, so
        # stash column index of "Height" in iheight
        iheight <<- which(namesx == "Height")   # note use of <<- not <-
        if (length(iheight) != 1) # sanity check
            stop("no Height in ", paste(namesx, collapse=" "))
    }
    pred != iheight
}
```

7 Using `earth` with `fda` and `mda`

Earth can be used with `fda` and `mda` in the `mda` package. Earth will generate a multiple response model when called by these functions. You can pass arguments such as `degree=2` to `earth` by including them in the call to `fda`. Use the `earth` argument `keepxy=TRUE` if you want to call `plotmo` later. Use the `fda/mda` argument `keep.fitted=TRUE` if you want to call `plot.earth` later (actually only necessary for large data sets, see the description of `keep.fitted` in `fda`'s help page).

Example (this gives the right side of Figure 3):

```
library(mda)
(fda <- fda(Species~., data=iris, keep.fitted=TRUE, method=earth, keepxy=TRUE))
summary(fda$fit)  # examine earth model embedded in fda model
plot(fda)         # right side of Figure 3
```

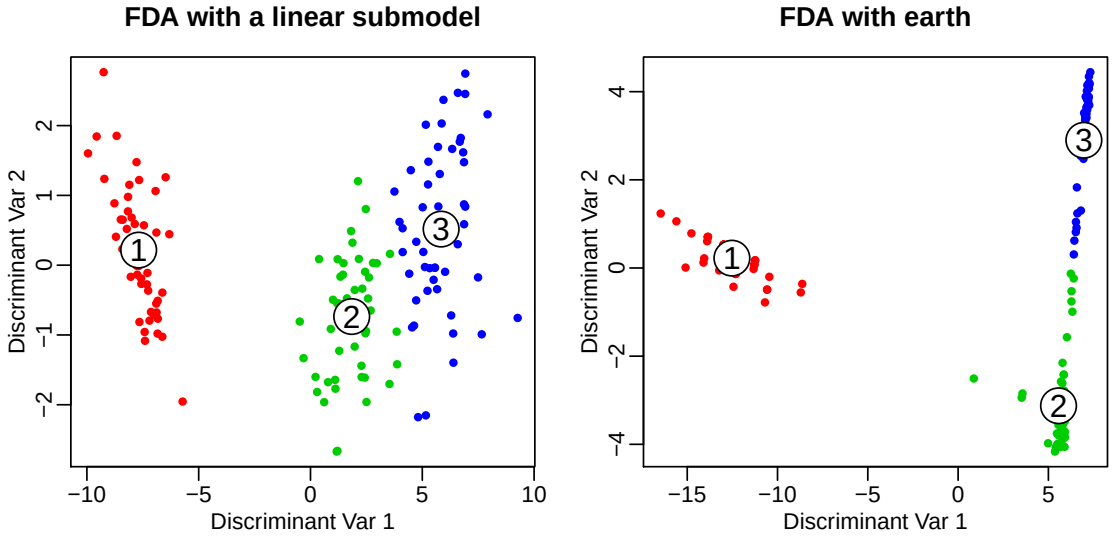

Figure 3: *Left: FDA of the iris data, built on a linear model.*
Right: FDA built with an `earth` *model using the code in the text. Note the better grouping of classes.*

The graphs show the training observations transformed into the discriminant space. This transformation is done by the regression function plugged into `fda` *and by optimal scoring (Section 7.1). There are three classes in this example so we have two discriminant variables. A new observation is classified by* `predict.fda` *as the class of the nearest centroid in discriminant space (the centroids are at the ringed numbers 1, 2, and 3).*

Using `plotmo` we can plot the per-predictor dependence of the `fda` variates like this:

```
plotmo(fda, type="variates", nresponse=1, clip=F) # 1st disc var (Figure 5)
plotmo(fda, type="variates", nresponse=2, clip=F) # 2nd disc var (not shown)
```

We can also look at the `earth` model embedded in the FDA model:

```
plotmo(fda$fit, nresponse=1, clip=F) # earth in FDA, 1st disc var (Figure 6)
plotmo(fda$fit, nresponse=2, clip=F) # earth in FDA, 2nd disc var (not shown)
```

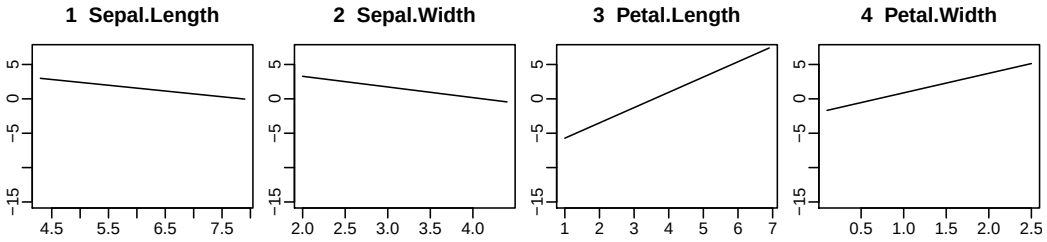

Figure 4: `plotmo` *graphs of the FDA model with a linear submodel (the left of Figure 3). The graphs show the contribution of each predictor to the first discriminant variable. The second discriminant variable is not shown. (The code to generate this plot is not in the text.)*

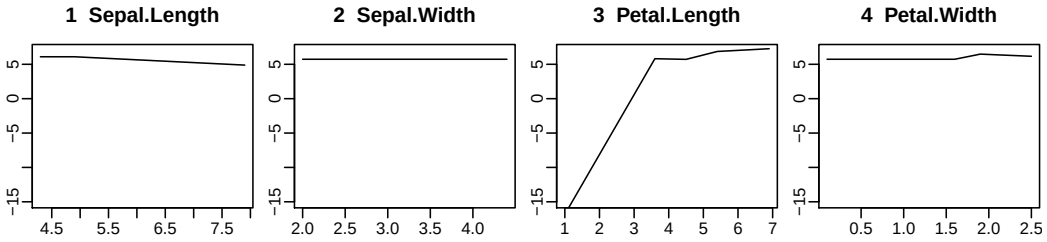

Figure 5: `plotmo` *graphs of the FDA model with an* `earth` *submodel (the right of Figure 3). The graphs show the contribution of each predictor to the first discriminant variable.*

Figure 6: `plotmo` *graphs of the* `earth` *model embedded in the FDA model (first discriminant variable before scoring). Earth did not include* `Sepal.Width` *in the model.*

7.1 A short introduction to Flexible Discriminant Analysis

Flexible Discriminant Analysis (FDA) is Linear Discriminant Analysis (LDA) on steroids. LDA uses a hyperplane to separate the classes. FDA replaces this hyperplane with a curved or bent surface to better separate the classes. The trick FDA uses to achieve this is to convert the classification problem into a regression problem. This allows us to plug in "any" regression function to generate the discriminant surface. If we plug in a linear regression function, FDA will generate a hyperplane, just like LDA. If we plug in `earth`, FDA will generate a surface defined by MARS hinge functions.

FDA converts a classification problem into a regression problem via *optimal scoring* (Figure 7). Essentially, this creates a new response variable by assigning new numbers (scores) to the factor levels in the original response. So for example `setosa=1`, `versicolor=2`, and `virginica=3` may become `setosa=1.2`, `versicolor=-1.2`, and `virginica=0`.

Actually, FDA creates several response variables like this, each with its own set of scores. If there are K response classes, FDA creates $K-1$ variables. So for the Iris data set, which has three classes (or "levels" in R parlance), we have two discriminant variables (Figure 3). For a binary response FDA creates one discriminant variable, and the discriminant space is one dimensional. Note that the dimension of the discriminant space depends on the number of classes, not on the number of predictors — a nice example of dimensionality reduction. Sometimes the best prediction results on independent data are obtained if we use only some of the discriminant variables, and thus a further reduction in dimensionality is possible.

Further details may be found in Hastie et al. [12] Section 12.5 and the FDA paper (Hastie, Tibshirani, and Buja [11]).

Using FDA is usually recommended for a response that is a factor with more than two levels, rather than using a regression function like `lm` or `earth` directly on an indicator matrix. This is because of the "masking problem" (e.g. Hastie et al. [12] Section 4.2

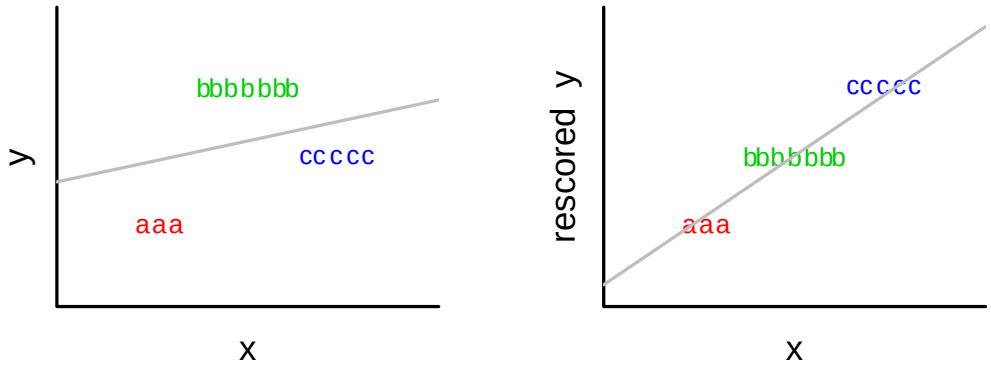

Figure 7: *Left: Some toy data with three response categories.*
Right: Rescoring assigns a new number to each category so the data are in a better form for linear separation.

"Linear Regression of an Indicator Matrix"). In practice the best advice is try it and see if you get better results on your data.

Two advantages of FDA are (i) FDA will often perform better than LDA (or QDA) because it generates a flexible surface to separate the classes, and (ii) for responses which have more than two levels, FDA will often perform better than regression on an indicator matrix because it does not suffer from the masking problem.

We mention that the acronym FDA for "Flexible Discriminant Analysis" is not to be confused with the same acronym for "Functional Data Analysis" [20].

TODO When is FDA exactly equivalent to LDA?

8 The `plot.earth` function

This chapter describes the graphs produced by `plot.earth`.

8.1 Short version of this chapter

For readers who do not wish to read this entire chapter, here is the least you need to know.

The `plot.earth` function produces four graphs (Figure 8).

Use the Model Selection plot to see how the fit depends on the number of predictors, how the final model was selected at the maximum GCV, and so on.

Use the Residuals vs Fitted graph to look for outliers and for any obviously strange behavior of the fitted function.

You can usually ignore the other two graphs.

8.2 Interpreting the `plot.earth` graphs

The graphs plotted by `plot.earth`, apart from the Model Selection plot, are standard tools used in residual analysis and more information can be found in most linear regression textbooks.

Heteroscedasity of the residuals isn't as important with `earth` models as it is with linear models, where homoscedasity of the (studentized) residuals is used a check that a linear model is appropriate. Also, in linear models homoscedasity of the residuals is required for the usual linear model inferences (such as calculation of p-values), which is not done with `earth` models.

Remember that the residuals are measured on the training data rather than on new data. In linear models that is usually not an issue, but for flexible models like MARS the residuals measured on the training data give an optimistic view of the model's predictive ability.

8.2.1 Nomenclature

The *residuals* are the differences between the values predicted by the model and the corresponding response values. The *residual sum of squares* (RSS) is the sum of the squared values of the residuals.

R-Squared (`RSq`, also called the *coefficient of determination*) is a normalized form of the RSS, and, depending on the model, varies from 0 (a model that always predicts the same value i.e. the mean observed response value) to 1 (a model that perfectly predicts the responses in the training data).[1]

[1]Not quite true, see FAQ 12.9 "Can R-Squared be negative?"

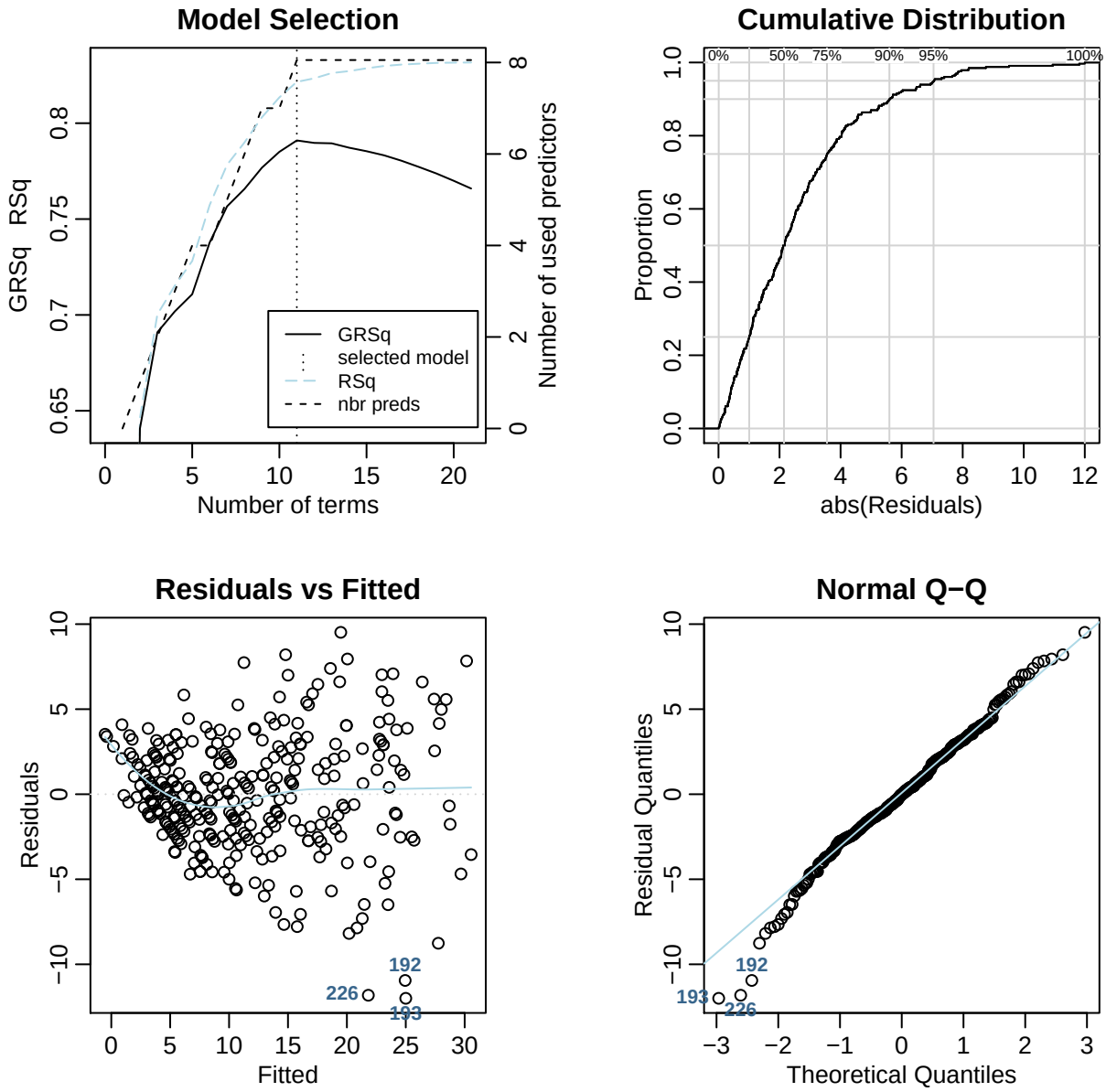

Figure 8: *Graphs produced by* `example(plot.earth)`.

The *Generalized Cross Validation* (GCV) is a form of the RSS penalized by the effective number of model parameters (and divided by the number of observations). More details can be found in FAQs 12.7 and 12.8. The *GRSq* normalizes the GCV in the same way that the RSq normalizes the RSS (see FAQ 12.10 and the definition of GRSq in the `Value` section of `earth`'s help page).

The GCV and GRSq are measures of the generalization ability of the model, i.e., how well the model would predict using data not in the training set. There is some arbitrariness in their values since the effective number of model parameters is a just an estimate in MARS models.

8.2.2 The Model Selection graph

For concreteness, the description of the graphs here is based on the plot produced by `example(plot.earth)` (Figure 8).

In the example Model Selection graph (top left of Figure 8) the RSq and GRSq lines run together at first, but diverge as the number of terms increases. This is typical behavior, and what we are seeing is an increased penalty being applied to the GCV as the number of model parameters increases.

The vertical dotted line is positioned at the selected model (at the maximum GRSq unless `pmethod="none"` was used) and indicates that the best model has 11 terms and uses all 8 predictors (the number of predictors is shown by the black dashed line).

We can also see the number of predictors and terms we would need if we were prepared to accept a lower GRSq (you can use the `earth` parameter `nprune` to trim the model).

To reduce clutter and the right-hand axis, use `col.npreds=0`.

8.2.3 The Residuals vs Fitted graph

The Residuals vs Fitted graph (bottom left of Figure 8) shows the residual for each value of the predicted response. By comparing the scales of the axes one can get an immediate idea of the size of the residuals relative to the predicted values.

The pale blue line is a `loess` fit. (Readers not familiar with `loess` fits can think of them as fancy moving averages.) In this instance it shows that the mean residual is more or less constant except at low fitted values. The end effect is possibly due to failure of the model in that region because of smaller residuals. The model fitting algorithm will not try hard to improve the fit on the left because there is little reduction in RSS to be gained in that area.

Ideally the residuals should show constant variance i.e. the residuals should remain evenly spread out, or homoscedastic, as the fitted values increase. (However, in flexible models like `earth`, constant variance of the residuals isn't as important as it is in linear models.) In the example graph we see heteroscedasity — the residuals spread out in a "<" shape. There is a decrease in the accuracy of the predictions as the predicted value increases.

To reduce the heteroscedasity, we can refit the model after performing a transform on the response. A cube root transform, for instance, evens out the residuals (Figure 9, middle plot):

```
fit <- earth(O3^.33333 ~ ., data = ozone1, degree = 2)
plot(fit)
```

Transforming the data may cause other problems, such as mismatches to a known underlying physical model or difficulties in interpretation, so it's best to consult (or become) an expert on the type of data being modeled (in this case, ozone pollution data — an expert may say that taking the cube root is meaningless, or conversely may say that it is essential).

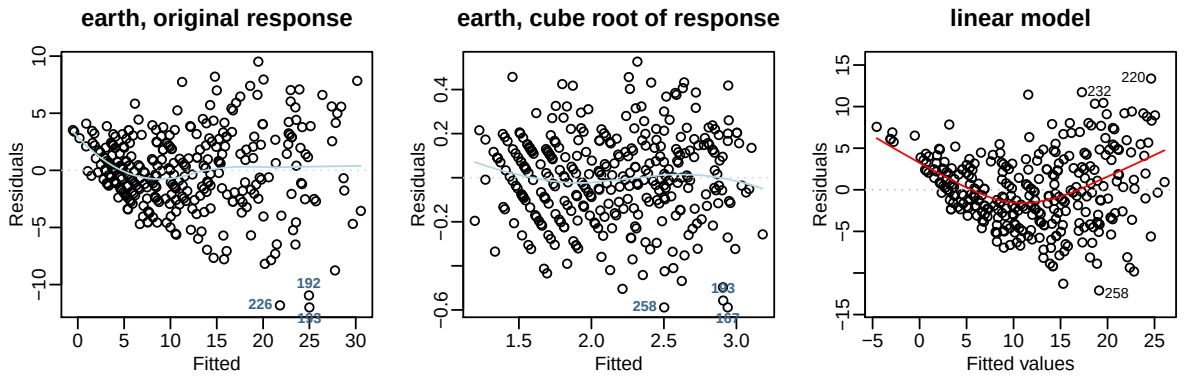

Figure 9: *Three models built from the ozone data.*
Left: The residuals of an `earth` *model (same as bottom left of Figure 8).*
Middle: The residuals of an `earth` *model built on the cube root of the response.*
Right: The residuals of a linear model.

Compare the residuals of the `earth` model to the linear model (Figure 9, right plot), and notice how the smooth line shows that the `earth` model is more successful at modeling non-linearities in the data. Also, the `earth` residuals are smaller — look at the left hand axis labels. The code for the linear model plot is:

```
fit.lm <- lm(O3 ~ ., data = ozone1)
plot(fit.lm, which=1)
```

One should always look at the residuals themselves as well as looking at the `loess` fit, which is itself an approximation. However, in the example plot the `loess` line appears reliable.

Cases 192, 193, and 226 have the largest residuals and fall suspiciously into a separate cluster. (If overplotting makes the labels hard to read, reduce the number of labels with the `id.n` argument of `plot.earth`.) As a general rule, it is worthwhile investigating cases with large residuals. Perhaps they should be excluded when building the model. Conversely, it is possible that they reveal something important about the data that could warrant changes to the model. In our example it is also worthwhile looking at cases with *small* residuals because of non-linearity in that region. To see the example input matrix ordered on the magnitude of the residuals, use `ozone1[order(abs(fit$residuals)),]`.

Sometimes groups of residuals appear in a series of parallel lines. These lines usually do not indicate a problem. They are formed when a set of plotted points has the same observed value, commonly due to discretization in the measurement of the observed response (e.g. by rounding to the nearest inch).

8.2.4 The Cumulative Distribution graph

The Cumulative Distribution graph (top right of Figure 8) shows the cumulative distribution of the absolute values of residuals. What we would ideally like to see is a graph that starts at 0 and shoots up quickly to 1. In the example graph, the median absolute residual is about 2.2 (look at the vertical gray line for 50%). We see that 95% of the

absolute values of residuals are less than about 7.1 (look at the vertical gray line for 95%). So in the training data, 95% of the time the predicted value is within 7.1 units of the observed value.

8.2.5 The QQ graph

The QQ (quantile-quantile) plot (bottom right of Figure 8) compares the distribution of the residuals to a normal distribution. If the residuals are distributed normally they will lie on the line. (Normality of the residuals isn't too important for `earth` models, but the graph is useful for discovering outlying residuals and other anomalies.) Following R convention, the abscissa is the normal axis and the ordinate is the residual axis; some popular books have it the other way round. In the example, we see divergence from normality in the left tail — the left tail of the distribution is fatter than that of a normal distribution. Once again, we see that cases 192, 193, and 226 have the largest residuals.

8.3 Earth-glm models and `plot.earth`

The `plot.earth` function ignores the `glm` part of the model, if any. ("Earth-glm" models are models created with `earth`'s `glm` argument, Chapter 3.) The plotted residuals are residuals from `earth`'s call to `lm.fit` after the pruning pass, not `glm` residuals.

For earth-glm models, `plotd` (in the `earth` package) can be convenient. Example (Figure 10):

```
fit <- earth(survived ~ ., data=etitanic, degree=2, glm=list(family=binomial))
plotd(fit, hist=TRUE)
```

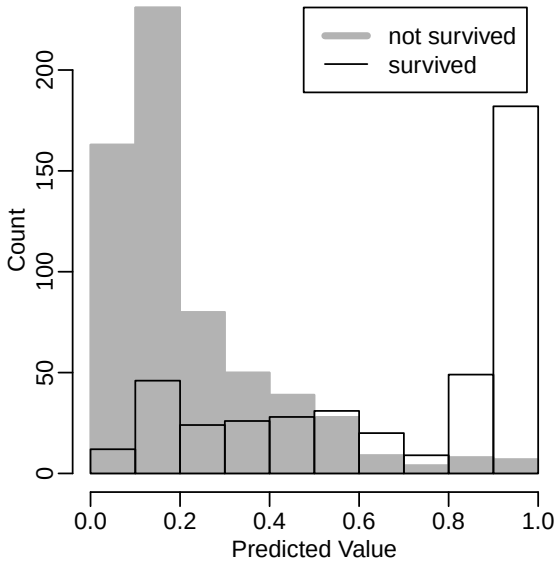

Figure 10: *Example of the* `plotd` *function.*

We can plot the `glm` model inside `earth` like this (not shown):

```
data(etitanic)
fit <- earth(survived~., data=etitanic, glm=list(family=binomial))
par(mfrow=c(2,2))
plot(fit$glm.list[[1]])
```

8.4 Cross-validated models and `plot.earth`

Earth builds cross-validated models with the `nfold` argument (Chapter 10 "Cross-Validation"). The Model Selection plot will show cross-validation statistics, but only if `keepxy=TRUE` was also used when building the model. (The cross-validation statistics are ignored in the other plots generated by `plot.earth`.) Here is an example (Figure 11):

```
fit <- earth(survived ~ ., data = etitanic, degree=2, nfold=5, keepxy=T)
plot(fit, which=1, col.rsq=0)    # which=1 for Model Selection plot only
```

In Figure 11, as usual the vertical black dotted line shows the optimum number of terms determined as usual by the peak GCV.

The pale pink lines show the out-of-fold RSq's for each fold model. The red line is the mean out-of-fold RSq for each model size.

The vertical red dotted line is at the maximum of the red line, i.e., the vertical line shows the optimum number of terms determined by cross-validation. This is *CV-with-averaging*; another approach (not supported by `plot.earth`) is *CV-with-voting* which uses the modal number-of-terms, i.e., the number-of-terms that is most often selected at a fold.

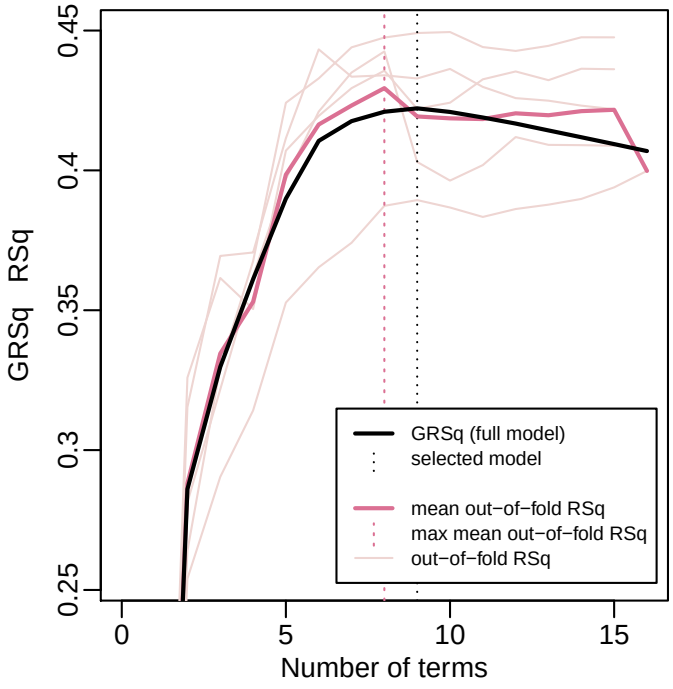

Figure 11: *A cross-validated* `earth` *model*

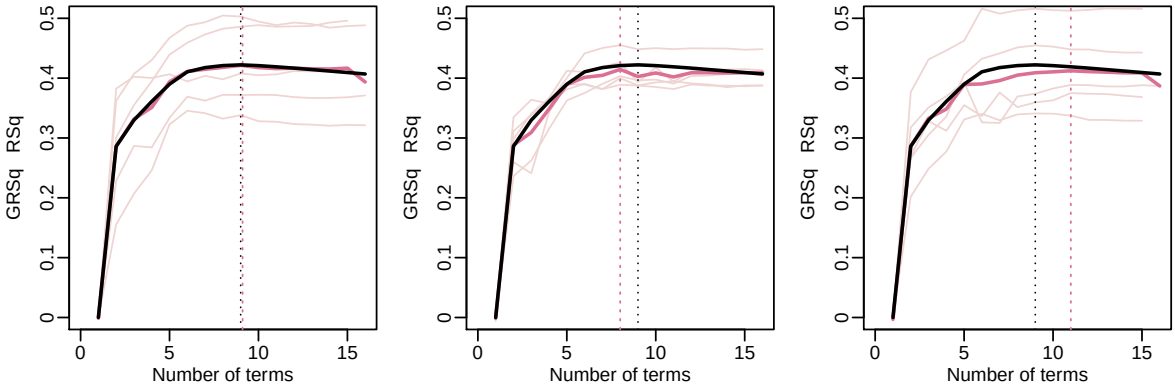

Figure 12: *The same model, cross-validated three times. Note the random variation in the cross-validation* `RSq`*'s as* `earth` *partitions the data into folds differently for each run.*

Ideally the number of terms selected using GRSq would always match the number of terms determined by cross-validation, and the two vertical lines coincide (although `plot.earth` slightly jitters the lines very slightly to prevent overplotting). In reality, the vertical lines are usually close but not identical. In the following example, note how the graph varies as the cross-validation folds vary in each invocation of `earth` (Figure 12):

```
plot1 <- function()
{
    fit <- earth(survived~., data = etitanic, degree=2,
                 nfold=5, keepxy=TRUE)
    plot(fit, which=1, ylim=c(0, .5), col.rsq=0)
}
plot1()
plot1()
plot1()
```

The above code keeps the vertical axis range constant across all three graphs with `ylim=c(0, .5)`, and reduces clutter with `col.rsq=0`.

We mention that in Figures 11 and 12 the cross-validation results are consistent with the results obtained in the standard way using the GCV. The solid black and red lines are very close. The vertical dotted red line dances around, but mostly because of the flatness of the curve after about 6 terms. If we used the one-standard-error rule (not yet supported by `earth`), the position of the vertical line would be much more stable.

9 Estimating variable importance

This chapter discusses how to estimate the relative importance of variables in an `earth` model. It was written in response to email about `evimp`.

9.1 Introduction to variable importance

What exactly is variable importance? A working definition is that a variable's importance is a measure of the effect that observed changes to the variable have on the observed response (or better, the expectation of that effect over the population). It is this measure of importance that `evimp` tries to estimate.

You might say that we can measure a variable's importance by changing the variable's value and measuring how the response changes. Indeed, the fact that an `earth` model is represented by an equation seems to imply that this is the way to go. However, except in special situations, there are problems with this way of thinking because:

(i) It assumes we can change the variable, which is usually not the case. For example, in the `trees` data, we cannot simply generate a new tree of arbitrary height.

(ii) It assumes that changes to a variable can occur in isolation. In practice, a variable is usually tied to other variables, and a change to the variable would never occur in the population without simultaneous changes to other variables. For example, in the `trees` data, a change in height is associated with a change in the girth.

(iii) It implies a causal relationship, which often is not the case. Changing the amount of mud does not tell us anything about the amount of rain.

Thus is is better to think in terms of the effect of the variable on the response averaged over the entire population. That is to say, the *expected* effect. In practice, we have to figure out how to use the model and the sample as a surrogate for the population, which isn't trivial.

Note that variable importance in the *equation that MARS derives from the data* is not really what we have in mind here. For example, if two variables are highly correlated, MARS will usually drop one when building the model. Both variables have the same importance in the data but not in the MARS equation (one variable does not even appear in the equation). Section 9.5 has a few words on how to use `plotmo` to estimate variable importance in the MARS equation.

9.2 Estimating variable importance

Estimating predictor importance is in general a tricky and even controversial problem. There is usually no completely reliable way to estimate the importance of the variables in a standard MARS model. The `evimp` function just makes an educated (and in practice useful) estimate as described below.

9.3 Three criteria for estimating variable importance

The `evimp` functions uses three criteria for estimating variable importance in a MARS model.

(i) The `nsubsets` criterion counts the number of model subsets that include the variable. Variables that are included in more subsets are considered more important.

By "subsets" we mean the subsets of terms generated by the pruning pass. There is one subset for each model size (from 1 to the size of the selected model) and the subset is the best set of terms for that model size. (These subsets are specified in `$prune.terms` in `earth`'s return value.) Only subsets that are smaller than or equal in size to the final model are used for estimating variable importance.

(ii) The `rss` criterion first calculates the decrease in the RSS for each subset relative to the previous subset. (For multiple response models, RSS's are calculated over all responses.) Then for each variable it sums these decreases over all subsets that include the variable. Finally, for ease of interpretation the summed decreases are scaled so the largest summed decrease is 100. Variables which cause larger net decreases in the RSS are considered more important.

(iii) The `gcv` criterion is the same, but uses the GCV instead of the RSS. Adding a variable can *increase* the GCV, i.e., adding the variable has a deleterious effect on the model. When this happens, the variable could even have a negative total importance, and thus appear less important than unused variables.

Note that using RSq's and GRSq's instead of RSS's and GCV's would give identical estimates of variable importance, because `evimp` calculates *relative* importances.

9.4 Example

This code

```
fit <- earth(O3 ~ ., data=ozone1, degree=2)
evimp(fit, trim=FALSE)  # trim=FALSE to show unused variables
```

prints the following:

```
           nsubsets   gcv    rss
temp             10 100.0  100.0
humidity          8  35.6   38.4
ibt               8  35.6   38.4
doy               7  33.6   36.0
dpg               5  25.9   28.0
ibh               4  30.9>  32.3>
vis               4  20.8   22.9
wind              1   8.7    9.9
vh-unused         0   0.0    0.0
```

The rows are sorted on `nsubsets`. We see that `temp` is considered the most important variable, followed by `humidity`, and so on. We see that `vh` is unused in the final model,

and thus is given an `unused` suffix. (Unused variable are printed here because we passed `trim=FALSE` to `evimp`. Normally they are omitted from the print.)

The `nsubsets` column is the number of subsets that included the corresponding variable. For example, `temp` appears in 10 subsets and `humidity` in 8.

The `gcv` and `rss` columns are scaled so the largest net decrease is 100.

A "`>`" is printed after `gcv` and `rss` entries that increase instead of decreasing (i.e., the ranking disagrees with the `nsubsets` ranking). We see that `ibh` is considered less important than `dpg` using the `nsubsets` criterion, but not with the `gcv` and `rss` criteria.

9.5 Estimating variable importance in the MARS equation

Running `plotmo` with `ylim=NULL` (the default) gives an idea of which predictors in the MARS equation make the largest changes to the predicted value (but only with all other predictors at their median values).

Note that there is only a loose relationship between variable importance in the MARS equation and variable importance in the data (Section 9.1).

9.6 Using `drop1` to estimate variable importance

As an alternative to `evimp`, we can use the `drop1` function (assuming we are using the formula interface to `earth`). Calling `drop1(my.earth.model)` will delete each predictor in turn from the model, rebuild the model from scratch each time, and calculate the GCV each time. We will get warnings that the `earth` library function `extractAIC.earth` is returning GCVs instead of AICs — but that is what we want so we can ignore the warnings. (Turn off just those warnings by passing `warn=FALSE` to `drop1`.) The column labeled `AIC` in the printed response from `drop1` will actually be a column of GCVs not AICs. The `Df` column isn't much use in this context.

Remember that this technique only tells us how important a variable is with the other variables already in the model. It does not tell us the effect of a variable in isolation.

We will get lots of output from `drop1` if we built the original `earth` model with `trace>0`. We can set `trace=0` by updating the model before calling `drop1`. Do it like this: `my.model <- update(my.model, trace=0)`.

9.7 Estimating variable importance by building many models

The variance of the variable importances estimated from an `earth` model can be high (meaning that the estimates of variable importance in a model built with a different realization of the data would be different).

This variance can be partially averaged out by building a bagged `earth` model and take the mean of the variable importances in the many `earth` models that make up the

bagged model. You can do this easily using the functions `bagEarth` and `varImp` in Max Kuhn's `caret` package [14].

Measuring variable importance using Random Forests is another way to go, independently of `earth`. See the functions `randomForest` and `importance` in the `randomForest` package.

TODO Enhance `evimp` to use cross-validation statistics when available.

9.8 Remarks on `evimp`

The `evimp` function is useful in practice but the following issues can make it misleading.

Collinear (or otherwise related) variables can mask each other's importance, just as in linear models. This means that if two predictors are closely related, the forward pass will somewhat arbitrarily choose one over the other. The chosen predictor will incorrectly appear more important.

For interaction terms, each variable gets credit for the entire term — thus interaction terms are counted more than once and get a total higher weighting than additive terms (questionably). Each variable gets equal credit in interaction terms even though one variable in that term may be far more important than the other.

MARS models can sometimes have a high variance — if the data change a little, the set of basis terms created by the forward pass can change a lot. So estimates of predictor importance can be unreliable because they can vary with different training data.

For factor predictors, importances are estimated on a per-level basis (because `earth` splits factors into indicator columns, essentially treating each level as a separate variable). The `evimp` function should have an option to aggregate the importances over all levels, but that has not yet been implemented.

10 Cross-validating earth models

Use cross-validation to get an estimate of R-Squared on independent data.
Example (note the `nfold` parameter):

```
> fit <- earth(survived ~ ., data=etitanic, degree=2, nfold=10)
> summary(fit)

Call: earth(formula=survived~., data=etitanic, nfold=10, degree=2)

    ... usual stuff not shown ...

GCV 0.14  RSS 140  GRSq 0.422  RSq 0.444  cv.rsq 0.42

Note: the cross-validation sd's below are standard deviations across folds

Cross validation:   nterms 8.50 sd 0.53    nvars 5.40 sd 0.52

    cv.rsq    sd     ClassRate    sd     MaxErr  sd
      0.42 0.082          0.80 0.028       -1.3 1.1
```

Cross-validation is done if `nfold` is greater than 1 (typically 5 or 10). Earth first builds a standard model with all the data as usual. This means that the standard fields in `earth`'s return value appear as usual, and will be displayed as usual by `summary.earth`. Earth then builds `nfold` cross-validated models. For each fold it builds an `earth` model with the in-fold data (typically nine tenths of the complete data) and using this model measures the R-Squared from predictions made on the out-of-fold data (typically one tenth of the complete data). The final mean `cv.rsq` printed by `summary.earth` is the mean of these out-of-fold R-Squared's.

The cross-validation results go into extra fields in `earth`'s return value. All of these have a `cv` prefix — see the `Value` section of the `earth` help page for details. For reproducibility, call `set.seed` before calling `earth` with `nfold`.

10.1 What is the best value for nfold?

The question of choosing the number of cross-validation folds remains in general an open research question. We can only suggest that you try 5- or 10-fold cross-validation, unless you have a small data set.

With a small data set some experimentation may be needed to get results without too much variance. A smaller `nfold` like 2 may be appropriate, otherwise the out-of-fold test sets are so small that variance becomes too large. (This assumes that the model is stable enough to be built on a small training set, Section 11.3.) We can average out the variance with `ncross=10` (say) but even that is unstable if the out-of-fold sets are too small.

10.2 Two ways of collecting R-Squared

Earth uses two ways of collecting the R-Squareds generated during cross-validation (Figure 13). The names used for these, `cv.rsq` and `oof.rsq`, are somewhat arbitrary, but used consistently in `earth`'s code and documentation.

(i) The `cv.rsq` is primarily for model assessment, i.e., estimating generalization ability. The `cv.rsq` is printed by `summary.earth`, but by default not plotted by `plot.earth`.

A `cv.rsq` is first calculated for each fold. A folds's `cv.rsq` is calculated from predictions made on the out-of-fold observations using the model built from the in-fold data. The model is selected as usual using the GCV of the training (in-fold) data. The `cv.rsq` printed by `summary.earth` is the mean of these per-fold `cv.rsq`'s.

(ii) The `oof.rsq`'s are primarily for model selection, i.e., for selecting the best number of terms. They aren't printed by `summary.earth`, but by default are plotted by `plot.earth` (Figure 14 on page 39).

The `oof.rsq` is calculated for every model size in each fold. (The model size is the number of terms in the model.) For each fold, it is calculated from predictions made on the out-of-fold observations using a model built with the in-fold data, after the model is pruned to the desired size. (At each fold the `oof.rsq` of the model with the highest GCV on the training data is that fold's `cv.rsq`.)

Note 1: Earth does not actually use cross-validation for automatic model selection (unlike, say, `rpart`). Earth's backward pass with GCVs is always used for that (in the current implementation at least; there is not yet a `pmethod="cv"` option). You can however use `plot.earth` to see the number of terms that would be estimated by cross-validation.

Note 2: The `oof.rsq`'s are calculated only when `keepxy=TRUE`, because calculating them is slow — `earth` has to call `update.earth` and `predict` for every model size in every fold.

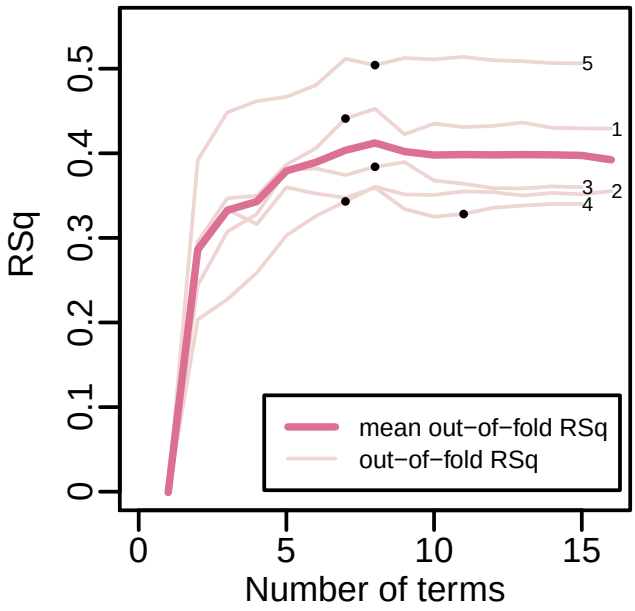

Figure 13: *Five-fold cross-validation of an* `earth` model.

The pink lines are the `oof.rsq`*'s for each fold. The thick red line is the mean of the pink lines.*

The black dots show the `cv.rsq`*'s for each fold. The* `cv.rsq` *printed by* `summary.earth` *is the mean of the vertical position of these dots.*

10.3 Cross-validation statistics returned by `earth`

The previous section described the R-Squareds collected during cross-validation. This section describes various additional cross-validation statistics.

Each of these is measured on the test (out-of-fold) set for each fold, and summarized by averaging across all folds (except that `MaxErr` is "summarized" by taking the worst error across all folds). Use `summary.earth` to see the summary statistics and their standard deviation across folds.[1] See the `Value` section of the `earth` page for more details of these statistics.

The statistics are:

- `cv.rsq` See Section 10.2 Part (i).
- `oof.rsq` See Section 10.2 Part (ii).
- `MaxErr` Signed max absolute difference between the predicted and observed response. This is the maximum of the absolute differences, multiplied by `-1` if the sign of the difference is negative. The "summary" `MaxErr` is the worst `MaxErr` across folds.
- `ClassRate` (discrete responses only) Fraction of out-of-fold observations correctly classified.

If we cross-validate a binomial or poisson model (specified using `earth`'s `glm` argument), `earth` returns the following additional statistics:

- `MeanDev` Deviation divided by the number of observations.
- `CalibInt`, `CalibSlope` Calibration intercept and slope (from regressing the observed response on the predicted response).
- `AUC` (Binomial models only) Area under the ROC curve.
- `cor` (Poisson models only) Correlation between the predicted and observed response.

For multiple response models, at each fold `earth` calculates these statistics for each response independently, and combines them by taking their mean, or weighted mean if the `wp` argument is used. Taking the mean is a rather dubious way of combining results from what are essentially quite different models, but can nevertheless be useful.

Explanations of the above GLM statistics can be found in the following (and many other) references: Pearce and Ferrier [19], Fawcett [6], and Harrell [10]. See the source code in `earth.cv.lib.R` for details of how the statistics are calculated, based on code kindly made available by Jane Elith and John Leathwick.

[1] We emphasize that `summary.earth` prints the deviation *across folds*, not the unknown deviation across samples. See Section 11.4 "Variance of cross-validation estimates". It is easy to confuse the two.

10.4 Tracing cross-validation

With `trace=.5` or higher, `earth` prints progress information as cross-validation proceeds. For example

```
CV fold 3: cv.rsq 0.622  n.oof 86 12%  n.fold.nz 384 41%  n.oof.nz 43 39%
```

shows that in cross-validation fold 3, the `cv.rsq` for the fold model was 0.622, measured on the 86 observations in the out-of-fold set.

The print also shows the number and percentage of non-zero values in the observed response in the in-fold and out-of-fold sets. This is useful if we have a binary or factor response and want to check that we have enough examples of each factor level in each fold. With the `stratify` argument (which is enabled by default), `earth` attempts to keep the numbers of occurrences of any given level in the response constant across folds.

10.5 Plotting cross-validation results

If you want `plot.earth` to show cross-validation statistics, use `keepxy=TRUE` so `earth` calculates the `oof.rsq` for every model size in every fold. Example (Figure 14, further discussion in Section 8.4):

```
fit <- earth(survived ~ ., data=etitanic, degree=2,
             nfold=5, keepxy=TRUE, trace=.5)
plot(fit, which=1) # which=1 selects just the Model Selection plot
```

In the figure, the pink lines for some of the fold models are truncated at the right. This is because the maximum number of terms for those models happens to be less than the 16 terms in the full model.

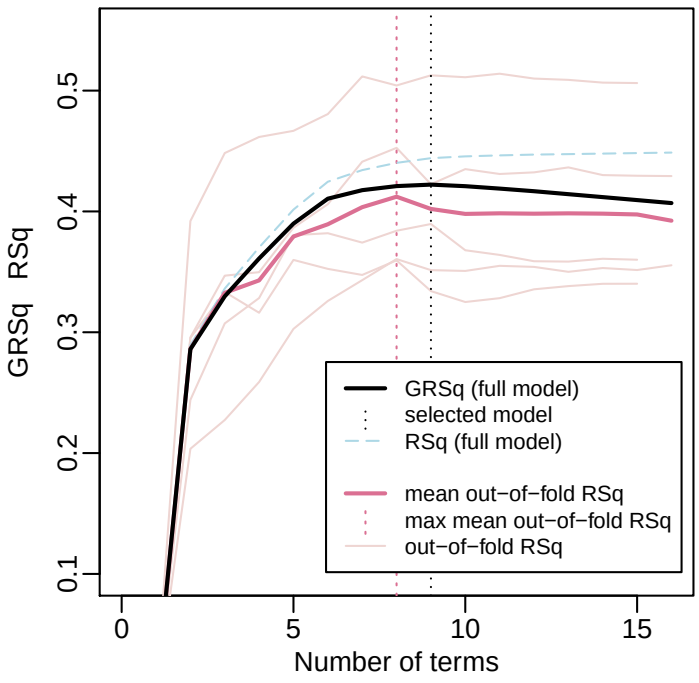

Figure 14: *Five-fold cross-validation of an* ***earth*** *model.*

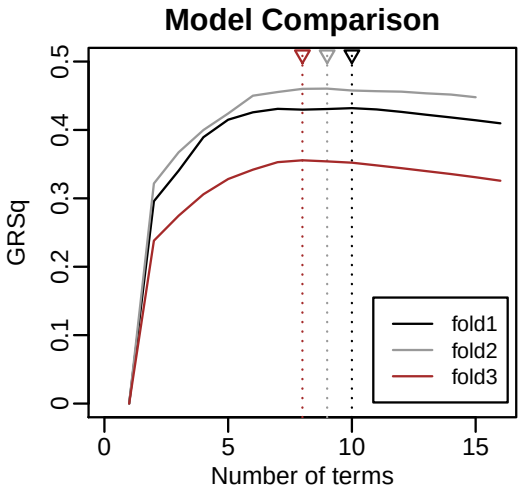

Figure 15: *Comparing the models at three cross-validation folds.*

For the curious, `plot.earth.models` can be used to compare the models built at each fold. Example (Figure 15):

```
fit <- earth(survived ~ ., data=etitanic, degree=2,
             nfold=3, keepxy=TRUE)
plot.earth.models(fit$cv.list, which=1, ylim=c(0, .5))
```

10.6 The `ncross` argument

If we run `earth` twice with the same `nfold` argument we will get different cross-validation results, because `earth` randomly splits the data into folds differently each time. To average out this variation for more stable results, use the `ncross` argument to repeat the whole process of taking `nfold` folds multiple times. Example (Figure 16):

```
fit <- earth(survived ~ ., data=etitanic, degree=2,
             ncross=3, nfold=5, keepxy=T, trace=.5)
plot(fit, which=1, col.rsq=0)
```

TODO What are the statistical properties of `ncross`?

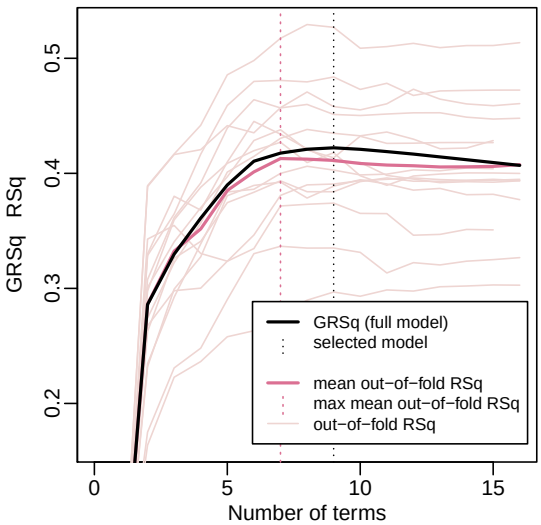

Figure 16: *Cross-validating with* `ncross=3, nfold=5`.
There are 15 folds in all.

10.7 An example: training versus generalization error

Figure 17 is reproduced from Figure 7.1 in Hastie et al. [12]. The figure was created from models built with 100 training sets generated synthetically.

Figure 18 is an example along the same lines. Instead of using new data, we use cross-validation. (Also, we use `earth` and the `mtcars` data.) The figure was created with the following code:

```
fit <- earth(mpg~., data=mtcars, ncross=10, nfold=2, keepxy=TRUE)
plot(fit, which=1,
     col.mean.infold.rsq="blue", col.infold.rsq="lightblue",
     col.grsq=0, col.rsq=0, col.vline=0, col.oof.vline=0)
```

Most of the pale lines are truncated because the maximum number of terms generated by `earth` for those fold models is less than the 9 terms in the full model. We mention also that many of the arguments in the call to `plot.earth` above simply remove display

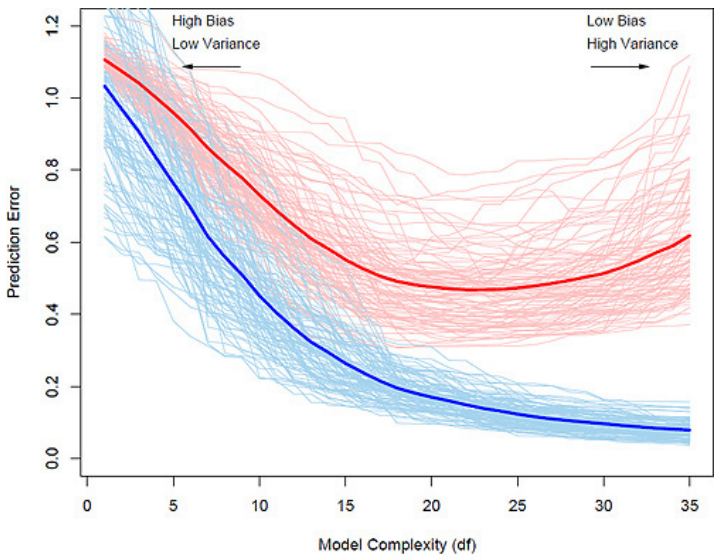

Figure 17: *Reproduced from Figure 7.1 of Hastie et al. [12]*

For models built from a 100 training sets, the pale blue lines show the prediction error measured on the training set. The pink lines show the error measured on a very large independent test set.

The thick lines show the expected error for the corresponding set of pale lines.

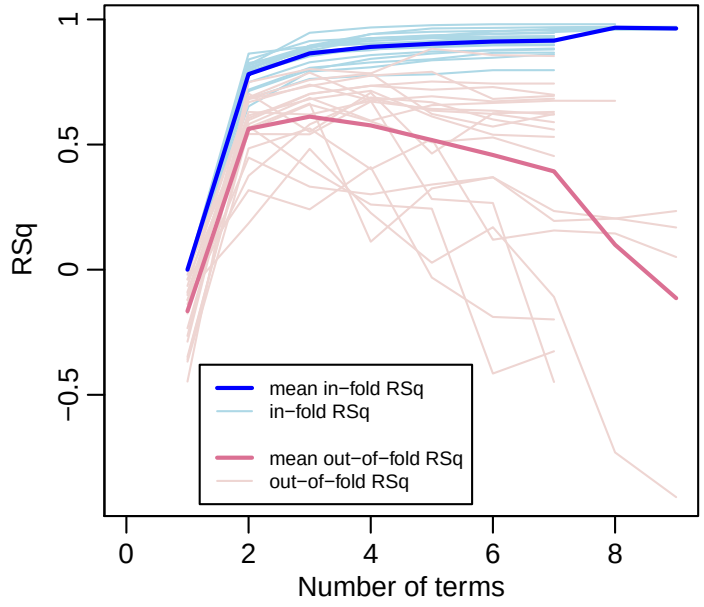

Figure 18: *Cross-validation of an* ***earth*** *model on the* ***mtcars*** *data.*

The Model Complexity along the horizontal axis in the figure above becomes the Number of Terms in this figure.

Note the negative R-Squareds in the pink curves at the far left of the graph (FAQ 12.9).

elements. The defaults for these arguments are inappropriate for this somewhat unusual plot (we are not usually interested in the in-fold R-Squareds).

Figure 18 is "upside down" with respect to Figure 17 because it plots model performance, not lack-of-performance. But we still see the same basic structure: the performance measured on the training data increases as we increase model complexity; on independent data the performance peaks and then decreases.

Much of the variation of the pink curves is due to the relatively small size of the out-of-fold data sets. If we measured R-Squared on a very large test set (instead of the out-of-fold data) we would still see variation, but much less. But note how variation of the pink lines increases with the number of terms. The more flexible the model, the more it overfits to randomness in the training set, and thus more randomness enters its estimation of R-Squared on independent data.

The `mtcars` data set is small (32 observations). Only two folds were used above (but repeated 10 times with `ncross`) to keep the out-of-fold sets large enough for somewhat stable results. With this small `nfold`, cross-validation bias may be an issue, because of the small size of the in-fold sets relative to the full data set. So the R-Squared on the out-of-fold data will tend to be smaller than it would be across full-sized independent samples. See Section 11.3 "Bias of cross-validation estimates".

The maximum mean out-of-fold RSq is at 3 terms, which in this example coincides with the number of terms selected by the GCV of the full model (set `col.grsq` to see this, not shown). A larger number of terms would have been selected with `degree` equal to 2, which is actually more appropriate for the `mtcars` data.

11 Understanding cross-validation

This chapter tries to clarify some aspects of cross-validation. It was written in response to email about cross-validating `earth` models, and your feedback would be appreciated. The chapter is mostly a general discussion, not limited to `earth` models. The exposition takes a frequentist approach, using arguments based on hypothetical situations where we have access to extra data.

We assume that you already know the basics of cross-validation (i.e., partition the data into `nfold` subsets, repeatedly build a model on all but one of those subsets, measure performance on the left-out data).

For a description of cross-validation, see for example Hastie et al. [12], Section 7.10, Duda et al. [4] Section 9.6, or even Wikipedia
`http://en.wikipedia.org/wiki/Cross-validation_(statistics)`.
An in-depth reference is Arlot and Celisse [1].

11.1 Data sets for measuring performance

In the next section we will discuss what it is that cross-validation actually measures. But first in this section we review some aspects of measuring a model's performance, and the data sets needed to do that. Understanding the role of these data sets is important for applying cross-validation correctly (Section 11.5 "Common cross-validation mistakes").

Typically we want to measure our model's *generalization* performance, and so want to measure prediction error on new data, i.e., not on the training data. (If that isn't immediately obvious, please see FAQ 12.8.) We typically want to measure performance in two scenarios:

(i) For **parameter selection**, i.e., to choose certain key model parameters during the model building process. For example, for `earth` models we need to select the best number of terms (so the parameter here is the number of terms). And for `rpart` trees we need to choose the optimum tree size. The new data are used as *parameter-selection data*, also commonly called *model-selection* or *validation* data.

(ii) For **model assessment**, i.e., to measure the performance of the final model. Here the new data are used as *test data*.

We thus require three independent data sets:

(i) the **training** data,
(ii) the **parameter-selection** data for (i) above,
(iii) the **test** data for (ii) above.

The ideal way to meet these requirements is to actually have large amounts of new data drawn from the same population. Usually we don't actually have access to such data, and so must resort to other techniques.

One such technique is the GCV used when building MARS models, which bypasses the need for model-selection data by using a formula to approximate the RSS that would be measured on new data.

Another technique, more universal, is cross-validation. Depending on how it is used, cross-validation can emulate either the parameter-selection or test data.

Some readers may wonder why we don't bother with the above data sets for linear models. When building a linear model, there is no separate model-selection step, so we don't need a model-selection set. And with these simple models the difference between the residual sum-of-squares measured on the training data and on independent test data is inconsequential, provided certain assumptions are met. So we don't need a separate test set. Things change with more flexible models (for example, if we are doing automatic variable selection for a linear model). The difference between the residual sum-of-squares on the training data and on independent test data becomes consequential, and formulas to estimate prediction error get complicated, if they exist at all.

11.2 What does cross-validation measure?

The mechanics of cross-validation are easy to understand. Understanding what cross-validation actually estimates involves some subtleties.

Cross-validation estimates "expected" not "conditional" error

Cross-validation does not really estimate the generalization performance of our model, built on a single set of data (which is usually what we want to know when applying a modeling technique like `earth`). Instead, cross-validation estimates the performance of our *model building algorithm* on a *range of training sets*. It approximates the average performance we would see in a hypothetical scenario where we build many models, each on a fresh sample of training data of approximately the same size as the original sample, and measure the performance of each of those models on independent test data (all data being i.i.d. from the same population).

In other words, cross-validation is better at estimating the expected prediction error across training sets, not the prediction error conditional on the training data we have at hand. In Figure 17 on page 41, our model is one pink line but cross-validation approximates the solid red line. See also Hastie et al. [12] Section 7.12 "Conditional or Expected Test Error?"

Some details. Cross-validation differs from the hypothetical scenario above because in cross-validation the training (in-fold) sets share observations and are not as varied as they would be in the hypothetical scenario. Also, the training set of a fold incorporates test sets from other folds. This induces a relationship between the residual errors (or whatever is used to measure performance) across folds, particularly in the presence of outliers.

Expected value of R-Squared across models

Consider the hypothetical scenario above, and for concreteness let us measure performance as R-Squared on the independent test data. If we built many models (with training sets of constant size) and took the average R-Squared over all the models, we would eventually close in on a stable average R-Squared value. This average R-Squared would be the same regardless of the size of the test sets, assuming we repeated the

experiment enough times (all data being i.i.d. from the same population). In other words, the *expected value* of R-Squared across models does not depend on the size of the test sets — but the *variance* of the R-Squared's certainly does, which leads to the next section.

Variance of R-Squared across models

Once we have an estimate of the generalization performance of the model (such as R-Squared on independent data), we typically want to know the *stability* of that estimate, usually expressed as the variance of the estimate.

Typically we want to know how our estimated R-Squared would be expected to change if we had a different training set — the sampling variance of R-Squared. This is the variance we would measure across models in the hypothetical scenario above if the test sets were extremely large (so all variation is due to the training sets, not the test sets).

On the other hand, if the test sets are small, the variance of R-Squared will include extra variation due to the small size of the test sets. And this is the scenario emulated by cross-validation. In cross-validation, it isn't possible in general to disentangle the variability due to the in-fold training sets and the variability due to the small out-of-fold test sets.

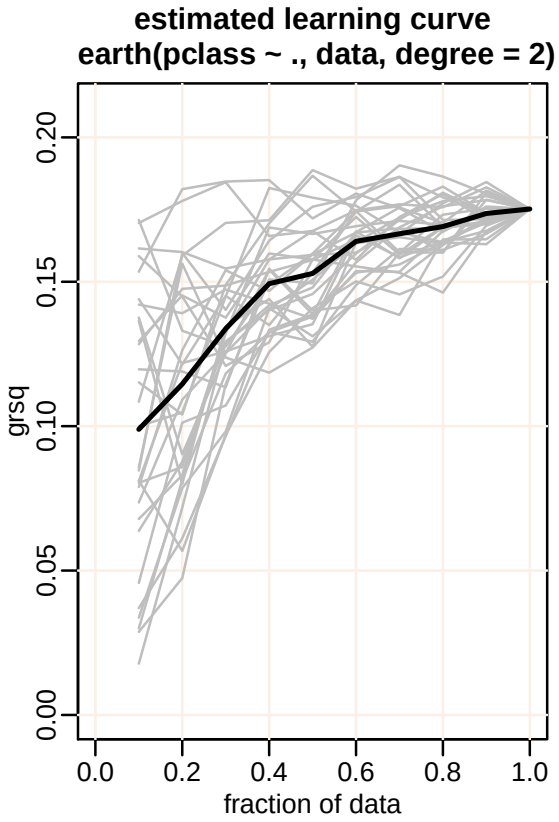

Figure 19: *The estimated learning curve of an* `earth` *model.*

The dark line is the mean of the gray lines.

Each gray line shows the GRSq for one set of subsets of the training data.

11.3 Bias of cross-validation estimates

Cross-validation tries to establish the quality of a model built on the full sample by using models built on *smaller* subsets of the sample. Generally a model built with a subset will be "worse" than a model built with the full sample. Thus the cross-validation R-Squared[1] will tend to be lower than it should be. That is, the cross-validation R-Squared is conservatively *biased*.

To see where the model sits on the learning curve (Figure 7.8 in Hastie et al. [12]), one technique for `earth` models is plot the GRSq of models built with different sized subsets of the sample, and average out variation by repeating several times. We make the assumption that the behavior of the model's GRSq in the face of a smaller sample is an adequate indication of that behavior for the full model's R-Squared (measured on independent data). Figure 19 gives an example, produced by the following R code. (You can ignore the code for now and just look at the figure.)

```
learning.curve <- function(data, func, field="grsq", ncurves=30)
{
    # set up the plot (call func on full data to establish ylim)
    body <- body(func) # needed only for the plot title
    plot(0, xlim=c(0,1), ylim=c(0, 1.2 * func(data)[[field]]), type="n",
         xlab="fraction of data", ylab=field, cex.main=1.1, xpd=NA,
         main=paste("estimated learning curve\n",
           substr(paste(deparse(substitute(body)), collapse=""), 1, 40)))
    grid(col="linen", lty=1)
    all.results <- rep(0, 10)
    for(curve in 1:ncurves) {                       # for each gray line
        sample <- sample.int(nrow(data))
        results <- double(10)
        for(fold in 1:10) {
            sub.data <- data[sample[1:(fold * nrow(data) / 10)],]
            results[fold] <- func(sub.data)[[field]]
        }
        lines((1:10)/10, results, col="gray")      # one gray line
        all.results <- all.results + results
    }
    lines((1:10)/10, all.results / ncurves, lwd=2) # the mean line
}

learning.curve(etitanic,
               function(data) earth(pclass~., data, degree=2))
```

From the figure, the bias is small enough with 10 folds (the curve is flat enough with 90% of the data).

Remember that this is just an *estimated* learning curve because it is created from a single training sample. Ideally we would like to estimate the learning curve using many fresh training sets. If we did that, we would see variation in the curves on the far right of graph which we do not see in Figure 19. So even with `nfold=5` the bias is acceptably small, relative to the (unknown) variance.

[1]The *Cross-validation R-Squared* here is the mean of the R-Squareds of each fold, each measured on the out-of-fold data. It is the `cv.rsq` printed by `summary.earth`.

11.4 Variance of cross-validation estimates

Cross-validation bias seems to be much discussed, but usually a more serious problem with cross-validation is the *variance* of cross-validation estimates across samples, and our inability to estimate this variance. How much would we expect the cross-validation R-Squared (averaged across folds) to change if we used a different training sample? (The new sample would be of the same size and drawn from the same population as the original, and, to make things more confusing, perhaps even generate the same model, but usually not.) It is not much comfort to know that the *expected* value of a statistic is correct up to small bias, if the single statistic we have at hand could be far from that expected value.

Quantifying the variance of cross-validation statistics in general is an ongoing research problem (see Bengio and Grandvalet [2] for a clear explanation of some of the difficulties involved). Unfortunately it is not really possible to estimate the variance of the cross-validation R-Squared of `earth` models.

An indication is given by the variance of the CV R-Squared across folds (printed by `summary.earth` as a standard deviation). However, this variance includes extra variability because we are looking at the R-Squared per fold instead of the mean R-Squared across folds. On the other hand, it incorporates less variability due to training sets than if we actually used fresh training data at each fold. Also it is unstable because of the small size of the out-of-fold test sets (the variance of the variance is high).

For `earth` models, another indication is the variance of GRSq in the estimated learning curve (Figure 19). Assuming GRSq is an acceptable surrogate for R-Squared on independent data, the variance across the gray curves gives an approximate lower bound of R-Squared variance across models for variously sized training sets. We say "lower bound" because the estimated learning curve is created from only a single training set (if we used fresh data for each model the variance at the right of the curve would not taper to zero).

11.5 Common cross-validation mistakes

In this section we list some mistakes that are easy to make when cross-validating. All of these make the model's performance seem better than it is. Given how easy it is to make these mistakes, a certain amount of skepticism is warranted when papers present final model assessment statistics based on cross-validation.

The central point is that *any data that have been used to select model parameters cannot be used as independent test data*. This rule can be violated in subtle ways, as discussed below.

Independence of observations

The out-of-fold data must play the role of new data. It is thus important that it isn't "contaminated" by the in-fold training data. This implies that the observations must be independent. Lack of independence means that the in-fold data used for training are partially included in some sense in the out-of-fold data used for testing. Thus cross-

validation will tend to give an optimistic R-Squared and select an overfitted model. At a minimum, we should avoid "twinned" observations.

Pre-tuning

Cross-validation must be applied to the entire model building process. Any parameter that is tuned to the training data must not be tuned before cross-validation begins. Instead, it must be included in the cross-validation process. (This does not apply to decisions made independently of the training data.) For example, it is a mistake to use the training data to choose which subset of the variables to include in the model (before calling `earth`), then cross-validate the `earth` model (with `nfold`) using only that subset of variables.[1] The convenience of `earth`'s `nfold` argument makes it perhaps a little too easy to make that kind of mistake.

A word of explanation for the above paragraph. Let's say we do in fact optimize a parameter to the full data set before cross-validation begins. By optimizing to the full data, we are also to some extent optimizing to the out-of-fold data used during cross-validation (because the out-of-fold data are, after all, drawn from the full data). The out-of-fold data are thus contaminated and can no longer legitimately play the role of independent test data, and the R-Squared's measured on the out-of-fold data will tend to be better than they should be.

There are however a few contexts where it is acceptable to select variables before cross-validation. See the comments in Hastie et al.. [12] Section 7.10.2.

Conflating the validation and test data

If the cross-validation R-Squared is used to select a model,[2] then the test R-Squared quoted for the selected model must be *recalculated* for that model using independent data. The cross-validation R-Squared used to select a model cannot be quoted as the R-Squared of that model — that would be conflating the validation and test data.

[1] The "parameter" being tuned here is which predictors are good.

[2] However the `earth` package does not provide facilities to do that (Section 10.2 Part 2).

12 FAQ

12.1 What are your plans for earth?

We would like to add support of case weights (to allow boosting), but that won't happen anytime soon.

12.2 How do I cite the earth package?

Thank you for asking that question :)

For earth the following BibTex entry seems to do the trick. The extra curly braces in the author field are necessary to get BibTex to order the entry correctly on the last name of the first author.

```
@Manual{earthpackage,
  title  = {earth: Multivariate Adaptive Regression Spline Models},
  author = {Stephen {Milborrow. Derived from mda:mars by
            Trevor Hastie and Rob Tibshirani.}},
  year   = {2011},
  note   = {R package \url{http://CRAN.R-project.org/package=earth}}
}
```

From within R you can use (you will have to massage the results to get BibTex to order the entry correctly):

```
> library(earth)
> citation("earth")
```

12.3 How can I establish variable importance?

Use the evimp function. See its help page and Chapter 9 for more details.

The summary.earth function lists the predictors in order of estimated importance using the nsubsets criterion of evimp.

12.4 Which predictors are used in the model?

The following function will give a list of predictors in the model:

```
get.used.pred.names <- function(obj) # obj is an earth object
{
  any1 <- function(x) any(x != 0)    # like any but no warning if x is double
  names(which(apply(obj$dirs[obj$selected.terms, , drop=FALSE], 2, any1)))
}
```

12.5 Which predictors were added to the model first?

You can see the forward pass adding terms with `trace=2` or higher. But remember, pruning will usually remove some of the terms. You can also use

```
summary(my.model, decomp="none")
```

which will list the terms remaining after pruning, in the order they were added by the forward pass. But it should be remarked that the order in which terms or predictors are added is not necessarily indicative of their importance.

12.6 How can I train on one set of data and test on another?

The example below demonstrates one way to train on 80% of the data and test on the remaining 20%.

```
train.subset <- sample(1:nrow(trees), .8 * nrow(trees))
test.subset <- (1:nrow(trees))[-train.subset]
fit <- earth(Volume ~ ., data = trees[train.subset, ])
yhat <- predict(fit, newdata = trees[test.subset, ])
y <- trees$Volume[test.subset]
print(1 - sum((y - yhat)^2) / sum((y - mean(y))^2)) # print R-Squared
```

In practice a data set larger than the one in the example should be used for splitting. The model variance is too high with this small set — run the example a few times to see how the model changes as `sample` splits the data set differently on each run. Also, remember that the test set should not be used for parameter tuning because you will be optimizing for the test set — instead use GCVs, separate parameter-selection sets, or techniques such as cross-validation with `earth`'s `nfold` parameter.

12.7 What is a GCV, in simple terms?

GCVs are important for MARS because the pruning pass uses GCVs to evaluate model subsets.

In general terms, when testing a model (not necessarily a MARS model) we want to test *generalization* performance, and so want to measure error on independent data, i.e., not on the training data. Often a decent set of independent data is unavailable and so we resort to cross-validation or leave-one-out methods. But that introduces other complications and can be painfully slow. As an alternative, for certain forms of model we can use a formula to approximate the error that would be determined by leave-one-out validation — that approximation is the GCV. The formula adjusts (i.e., increases) the training RSS to take into account the flexibility of the model. Summarizing, the GCV approximates the RSS (divided by the number of cases) that would be measured on independent data. Even when the approximation is not that good, it is usually good enough for comparing models during pruning.

GCVs were introduced by Craven and Wahba [3], and extended by Friedman and Silverman [7, 9]. See Hastie et al. [12], Section 7.10 "Cross-Validation", and the Friedman

MARS paper [7]. GCV stands for "Generalized Cross Validation", a perhaps misleading term. because no cross-validation is actually performed.

The GRSq measure used in the `earth` package standardizes the raw GCV, in the same way that R-Squared standardizes the RSS (FAQ 12.10).

12.8 If GCVs are so important, why don't linear models use them?

First a few words about overfitting. An overfit model fits the training data well but will not give good predictions on new data. The idea is that the training data capture the underlying structure in the system being modeled, plus noise. We want to model the underlying structure and ignore the noise. An overfit model models the specific realization of noise in the training data and is thus too specific to that training data.

The more flexible a model, the more its propensity to overfit the training data. Linear models are constrained, with usually only a few parameters (viz. the intercept and regression coefficients) and don't have the tendency to overfit like more flexible models such as MARS. This means that for linear models, the RSS on the data used to build the model is usually an adequate measure of generalization ability, and we don't need GCVs.

This is no longer true if we do automatic variable selection on linear models, because the process of selecting variables increases the flexibility of the model. Hence the AIC — as used in, say, `drop1`. The GCV, AIC, and friends are means to the same end. Depending on what information is available during model building. we use one of these statistics to estimate model generalization performance for the purpose of selecting a model.

12.9 Can R-Squared be negative?

Yes, R-Squared (`rsq`) can be negative if

(i) the test set is not the training set (for example, in cross-validation), and,

(ii) we use the general definition of R-Squared

```
rsq = 1 - rss / tss,
```

where `rss = sum((y - yhat)^2)` is the residual sum-of-squares and `tss = sum((y - mean(y)^2)` is the total sum-of-squares. This is the definition used in the `earth` code.

The simplest example is an intercept-only model. This will give a negative `rsq` on test data, unless the training data and test data happen to have the same mean. When we calculate `rsq` on the test data, the intercept-only model predicts the mean of the *training* data. Thus on the test data the residuals will be greater on the whole than if we predicted the mean of the test data.[1] That is another way of saying that on the test

[1]Recall that $\sum_i (x_i - \mu)^2$ is minimimized when μ is the mean of the x_i's.

data the residual sum-of-squares will be greater than the total sum-of-squares, and `rsq = 1 - rss / tss` will be negative. Examples can be seen in the left of Figure 18.

There is actually more than way of defining `rsq`. You may be more familiar with the definition

```
rsq = regression.sum.of.squares / tss,
```

which is indeed always non-negative. With `rsq` measured on the training data, the two definitions are equivalent for linear regression with an intercept and for `earth` models. The Wikipedia page on RSq has a clear explanation (accessed May 2011) `http://en.wikipedia.org/wiki/R-squared`.

The "squared" in R-Squared is misleading. Perhaps we should use the alternative term "coefficient of determination", but "R-Squared" is common.

12.10 Can GRSq be negative?

The statistic `GRSq` is `earth`'s estimate of the generalization performance of the model. It is defined, analogously to R-Squared (FAQ 12.9), as

```
GRSq = 1 - GCV / GCV.null,
```

where `GCV.null` is the GCV of an intercept-only model.

A negative `GRSq` indicates a severely over parameterized model — a model that would not generalize well even though it may be a good fit to the training data. During `earth` model building, `GRSq` can become negative. However, after pruning the model will end up with a non-negative `GRSq`.

Adding a term to the model will always increase the R-Squared on the training data (up to the limits of numerical accuracy). But adding that term could reduce the predictive power of the model on new data, and would thus *decrease* `GRSq`. (We see that happening in almost any `earth` Model Selection graph.) Decrease `GRSq` often enough

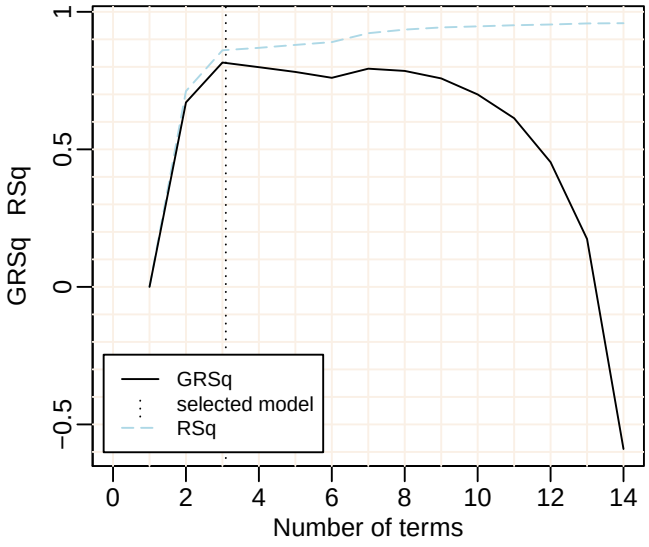

Figure 20: *Negative GRSq's.*

After term 3, adding a term reduces the estimated predictive power of the model, apart from term 7. By term 14 the GRSq is negative.

The R-Squared on the training data always increases as ***earth*** *adds terms.*

and it will eventually become negative. Watch the GRSq take a nose dive in this example (Figure 20):

```
fit <- earth(mpg~., data=mtcars, trace=4)
plot(fit, which=1, col.npreds=0,
     col.sel.grid="linen", legend.pos="bottomleft")
```

In severe cases, GRSq might even be set to -Inf, which brings us to the following FAQ.

12.11 Why am I seeing a GRSq of -Inf (with trace enabled)?

During the forward pass, if "too many" terms are generated relative to the number of observations, earth will set the model's GCV to Inf and consequentially the GRSq to -Inf. However, after pruning the final model's GCV and GRSq will be non-negative

Infinite GCVs were introduced in earth Version 3.1-2, to replace the warning issued in previous versions (which is no longer needed):

```
effective number of GCV parameters >= number of cases.
```

Some details. Earth sets the GCV to Inf during model building if the effective number of parameters for a term is greater than the number of observations. The GCV no longer approximates the leave-one-out RSS. From the formula for the GCV

```
GCV = RSS / (nobs * (1 - nparams / nobs)^2)),
```

we see that the GCV increases and then decreases if nparams / nobs approaches and then exceeds 1 as terms are added to the model. To prevent this undesirable behavior, if nparams / nobs >= 1 then earth does not use the formula but instead directly sets the GCV to Inf.

12.12 How is the default number of terms nk calculated?

If we don't explicitly specify nk, the default is used:

```
nk = min(200, max(20, 2 * ncol(x))) + 1
```

This doubles the number of predictors, forces that into the range of 20 to 200, and finally adds 1 for the intercept.

The numbers 20 and 200 are fairly arbitrary. The lower limit of 20 seems reasonable for situations where we have just a few predictors. The upper limit of 200 prevents excessive memory use in the forward pass. Typically we will reach one of termination conditions long before we reach 200 terms. (The termination conditions are described in Section 2.4.1 "Termination conditions for the forward pass".)

12.13 Why do I get fewer terms than nk, even with pmethod="none"?

There are several conditions that can terminate the forward pass, and reaching `nk` is just one of them. See Section 2.4.1 "Termination conditions for the forward pass".

Setting `earth`'s argument `thresh` to zero is treated as a special case: `thresh=0` disables all termination conditions except `nk` and conditions involving numerical limits. By disabling `thresh` we are allowing `earth` to continue processing even if numerical issues can cause instability. This opens up the possibility of nonsensical RSq's and GRSq's.

12.14 Why do I get fewer terms than nprune?

The pruning pass selects a model with the lowest GCV that has `nprune` or fewer terms. Thus the `nprune` argument specifies the *maximum* number of permissible terms in the final pruned model.

You can work around this because you will get exactly `nprune` if you specify `penalty=-1`. This special value of `penalty` causes `earth` to set the GCV to `RSS/nrow(x)`. Since the RSS on the training set always decreases with more terms, the pruning pass will choose the maximum allowable number of terms. An example:

```
earth(Volume ~ ., data=trees, trace=3, nprune=3, penalty=-1)
```

12.15 Is it best to hold down model size with nk or nprune?

If you want the best possible small model, build a big set of basis functions in the forward pass (by specifying a big `nk`) and prune this set back (by specifying a small `nprune`). This is better than directly building a small model by specifying a small `nk`. You will get a better set of terms because the pruning pass can look at all the terms whereas the forward pass can only see one term ahead. However, it is faster building a small model by specifying a small `nk`.

12.16 What about bagging MARS?

The `caret` package [14] provides functions for bagging `earth` (and for parameter selection). Our personal experience has been that bagging MARS does not give models with better predictive ability (probably because the MARS algorithm is fairly stable in the presence of perturbations of the data, and bagging works best for "unstable" models). Your mileage may vary (we would be interested if it does). We tested just a couple of data sets, but did try a few different approaches, including using a modified version of `earth` that randomized the set of variables available at each forward step to increase variability (similar to random forests).

12.17 Why do I get `Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred`?

You will only see this warning when using `earth`'s `glm` argument. You can safely ignore the warning in an `earth` context. The GLM coefficients for the model terms may be very large, but it doesn't matter — the predictive ability of the model is unimpaired.

The warning is issued when `glm.fit` generates a model that perfectly separates the classes. A perfect fit is usually considered a good thing, not something that should cause a warning. However, the warning is issued because certain model statistics (such as the t-values) generated by the mathematics inside `glm.fit` will be unreliable for subsequent inference on the model. That doesn't matter in an `earth` context, because `earth` doesn't use those statistics. (And, anyway, the t-values are meaningless even when the warning is not issued, because of the amount of processing done by `earth` to generate the terms before it calls `glm.fit`.)

The warning message is more likely to occur during cross-validation (using `earth`'s `nfold` parameter). With cross-validation we are looking at more, and smaller, data sets, so the chance of a perfectly separable set is more likely. If the warning is issued, the coefficients for the terms of the fold model may be very large, but they aren't used in the final model anyway. The cross-validation statistics calculated by `earth` (such as `cv.rsq`) remain valid.

12.18 Why do I get `Error: XHAUST returned error code -999`?

The short answer: you should never see the above message (fixed in `earth` version 2.6-0). If you do, please let us know.

One work-around is to change `pmethod` from `"exhaustive"` to `"backward"`.

You can also try the following. These instructions work on the assumption that the default value of `Exhaustive.tol` is too big for your data set. First please read the description of the `Exhaustive.tol` argument in the `Arguments` section of the `earth` help page. Then run `earth` with `trace=1`, so `earth` prints the reciprocal of the condition number of the `earth` basis matrix `bx`. (The condition number here is the ratio of largest to the smallest singular value of `bx`.) Then set `Exhaustive.tol` to greater than the printed value (something like `Exhaustive.tol=1e-8`), and run `earth` again. Now `earth` will automatically change `pmethod` from `"exhaustive"` to `"backward"` when necessary to avoid the above error message.

It must be said that it is hard to believe under these conditions that the resulting model will be much good. The data do not allow a decent predictive model to be built.

Some details. Certain data cause collinearity in the `earth` basis matrix `bx` which slips by the usual checks. This causes the `leaps` routine to fail. The usual checks are:

(i) while building the basis matrix, the C code does a check to drop collinear terms (`BX_TOL` and `QR_TOL` in the C code)

(ii) after building the basis matrix, the C code drops any remaining collinear terms

(RegressAndFix in the C code)

(iii) the leaps Fortran routine sing checks for collinearity.

Some data get through all these tests, probably because we are near the numerical noise floor and numerical rounding is essentially changing the data randomly. When pmethod="exhaustive", earth performs an SVD of bx, and as a last resort if the condition number is out-of-range forces pmethod from "exhaustive" to "backward".

12.19 How does summary.earth order terms?

With decomp="none", the terms are ordered as created by the forward pass.

With the default decomp="anova", the terms are ordered in increasing order of interaction. In detail:

(i) terms are sorted first on degree of interaction
(ii) then terms with a linpreds linear factor before standard terms
(iii) then on the predictors (in the order of the columns in the input matrix)
(iv) and finally on increasing knot values.

It's actually earth:::reorder.earth that does the ordering.

12.20 Why is plot.earth not showing the cross-validation data?

Use keepxy=TRUE in the call to earth (as well as nfold). See Section 10.5.

12.21 How do I add a plot to an existing page with plot.earth or plotmo?

Use do.par = FALSE, otherwise these plotting functions start a new page.

12.22 summary.earth lists predictors with weird names that aren't in x. What gives?

You probably have factors in your x matrix, and earth is applying contrasts. See Chapter 4 "Factors".

12.23 What happened to get.nterms.per.degree, get.nused.preds.per.subset, and reorder.earth?

From release 1.3.0, some earth functions are no longer public, to help simplify the user interface. The functions are still available (and stable) if you need them — use for example earth:::reorder.earth().

References

[1] S. Arlot and A. Celisse. *A Survey of Cross-Validation Procedures for Model Selection*. Statistic Surveys, 2010. Cited on page 43.

[2] Y. Bengio and Y. Grandvalet. *No Unbiased Estimator of the Variance of K-Fold Cross-Validation*. J. Mach. Learn. Res., 5:1089Ű1105, 2004. `http://jmlr.csail.mit.edu/papers/v5/grandvalet04a.html`. Cited on page 47.

[3] P. Craven and G. Wahba. *Smoothing Noisy Data with Spline Functions*. Numer. Math 31, 377-403, 1979. Cited on page 50.

[4] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley-Interscience Publication, 2000. `www.crc.ricoh.com/~stork/DHS.html`. Cited on page 43.

[5] Julian Faraway. *Extending the Linear Model with R*. CRC, 2005. `http://www.maths.bath.ac.uk/~jjf23`. Cited on page 4.

[6] Tom Fawcett. *ROC Graphs: Notes and Practical Considerations for Researchers. Revised version of Technical report HP Laboratories*. HP Labs, 2004. `http://biostat.mc.vanderbilt.edu/twiki/bin/view/Main/RmS`. Cited on page 38.

[7] Jerome H. Friedman. *Multivariate Adaptive Regression Splines (with discussion)*. Annals of Statistics 19/1, 1Ű141, 1991. `http://www.salfordsystems.com/doc/MARS.pdf`. Cited on pages 4, 50, and 51.

[8] Jerome H. Friedman. *Fast MARS*. Stanford University Department of Statistics, Technical Report 110, 1993. `http://www.milbo.users.sonic.net/earth/Friedman-FastMars.pdf`, `http://www-stat.stanford.edu/research/index.html`. Cited on page 4.

[9] Jerome H. Friedman and Bernard W. Silverman. *Flexible Parsimonious Smoothing and Additive Modeling*. Technometrics, Vol. 31, No. 1., 1989. Cited on pages 4 and 50.

[10] F. Harrell. *Regression Modeling Strategies with Applications to Linear Models, Logistic Regression, and Survival Analysis*. Springer, 2001. `http://biostat.mc.vanderbilt.edu/twiki/bin/view/Main/RmS`. Cited on page 38.

[11] Hastie, Tibshirani, and Buja. *Flexible Discriminant Analysis by Optimal Scoring*. JASA, 1994. `www-stat.stanford.edu/~hastie/Papers/fda.pdf`. Cited on pages 9 and 23.

[12] Hastie, Tibshirani, and Friedman. *The Elements of Statistical Learning (2nd ed.)*. Springer, 2009. `http://www-stat.stanford.edu/~hastie/pub.htm`. Cited on pages 4, 23, 41, 43, 44, 46, 48, and 50.

[13] Trevor Hastie and Robert Tibshirani. *mda: Mixture and flexible discriminant analysis*. R package, 2011. `http://CRAN.R-project.org/package=mda`. Cited on page 5.

[14] Max Kuhn. Contributions from Jed Wing, Steve Weston, Andre Williams, Chris Keefer, and Allan Engelhardt. *caret: Classification and Regression Training*, 2011. R package `http://CRAN.R-project.org/package=caret`. Cited on pages 35 and 54.

[15] J.R. Leathwick, D. Rowe, J. Richardson, J. Elith, and T. Hastie. *Using Multivariate Adaptive Regression Splines to Predict the Distributions of New Zealand's Freshwater Diadromous Fish.* Freshwater Biology, 50, 2034-2052, 2005. http://www-stat.stanford.edu/~hastie/pub.htm,http://www.botany.unimelb.edu.au/envisci/about/staff/elith.html. Cited on page 11.

[16] Thomas Lumley using Fortran code by Alan Miller. *leaps: regression subset selection.* R package, 2009. http://CRAN.R-project.org/package=leaps. Cited on page 4.

[17] Stephen Milborrow. Derived from mda:mars by Trevor Hastie and Rob Tibshirani. *earth: Multivariate Adaptive Regression Spline Models*, 2011. R package http://CRAN.R-project.org/package=earth. Cited on page 4.

[18] Alan Miller. *Subset Selection in Regression (2nd ed.).* CRC, 2002. http://www.cmis.csiro.au/Alan_Miller/index.html. Cited on page 4.

[19] J. Pearce and S. Ferrier. *Evaluating the Predictive Performance of Habitat Models developed using Logistic Regression.* Ecological modelling, 2000. Cited on page 38.

[20] J. O. Ramsay, Hadley Wickham, Spencer Graves, and Giles Hooker. *fda: Functional Data Analysis*, 2011. R package http://CRAN.R-project.org/package=fda. Cited on page 24.

[21] Wikipedia. *Multivariate Adaptive Regression Splines.* Wikipedia, Accessed May 2011. http://en.wikipedia.org/wiki/Multivariate_adaptive_regression_splines. Cited on page 4.