Statistics 360: Advanced R for Data Science

MARS, part III

Becky Lin

Topics

- Recap of forward stepwise MARS algorithm (Algorithm 2)
- Pruning basis functions in the backward stepwise algorithm (Algorithm 3)
  - LOF revisited
- Software features
- Data structures and functions

MARS forward algorithm (algorithm 2)

Algorithm 2 (MARS—forward stepwise)

$B_1(\mathbf{x}) \leftarrow 1$; $M \leftarrow 2$
Loop until $M > M_{\max}$: $\text{lof}^* \leftarrow \infty$
  For $m = 1$ to $M - 1$ do:
    For $v \notin \{v(k, m) | 1 \le k \le K_m\}$
      For $t \in \{x_{vj} | B_m(\mathbf{x}_j) > 0\}$
        $g \leftarrow \sum_{i=1}^{M-1} a_i B_i(\mathbf{x}) + a_M B_m(\mathbf{x})[+(x_v - t)]_+ + a_{M+1} B_m(\mathbf{x})[-(x_v - t)]_+$
        $\text{lof} \leftarrow \min_{a_1, \ldots, a_{M+1}} \text{LOF}(g)$
        if $\text{lof} < \text{lof}^*$, then $\text{lof}^* \leftarrow \text{lof}$; $m^* \leftarrow m$; $v^* \leftarrow v$; $t^* \leftarrow t$ end if
      end for
    end for
  end for
  $B_M(\mathbf{x}) \leftarrow B_{m^*}(\mathbf{x})[+(x_{v^*} - t^*)]_+$
  $B_{M+1}(\mathbf{x}) \leftarrow B_{m^*}(\mathbf{x})[-(x_{v^*} - t^*)]_+$
  $M \leftarrow M + 2$
end loop
end algorithm

Figure 1: Algorithm 2

Recap of forward algorithm

- The MARS forward stepwise algorithm (Algorithm 2) builds a linear prediction equation that is linear in basis functions $B_1(x), \ldots, B_{M_{max}}(x)$.
  - Coefficients of the basis functions are by least squares.
  - Have an intercept term $B_0(x) = 1$ (notation change from Alg 2)
  - Basis functions are products of hinge functions:

$$B_m(x) = \prod_{k=1}^{K_m} h(s_{km}(x_{v(k,m)} - t_{km})),$$

  where
  - K_m is the number of product terms,
  - $h(x) = \max(0, x)$,
  - s_{km} is $+1$ or -1 (recall mirror-image basis functions),
  - $v(k, m)$ is the kth variable used in B_m, and
  - t_{km} is the knot for the kth variable.

Forward algorithm testing result (1/3)

```
> head(testdata,5)
          y          x1         x2         x3         x4         x5         x6         x7         x8         x9         x10
1 -0.09957987 -0.56047565 -0.7104066  2.1988103 -0.7152422 -0.07355602 -0.6018928  1.07401226 -0.7282191  0.3562833 -1.0141142
2  0.70547203 -0.23017749  0.2568837  1.3124130 -0.7526890 -1.16865142 -0.9936986 -0.02734697 -1.5404424 -0.6580102 -0.7913139
3  7.48020063  1.55870831 -0.2466919 -0.2651451 -0.9385387 -0.63474826  1.0267851 -0.03333034 -0.6930946  0.8552022  0.2995937
4  1.69830766  0.07050839 -0.3475426  0.5431941 -1.0525133 -0.02884155  0.7510613 -1.51606762  0.1188494  1.1529362  1.6390519
5  1.63292893  0.12928774 -0.9516186 -0.4143399 -0.4371595  0.67069597 -1.5091665  0.79038534 -1.3647095  0.2762746  1.0846170
> dim(marstestdata)
[1] 100  11
> y <- marstestdata$y
> x <- marstestdata[,-1]
```

```
> fwd <- fwd_stepwise(y,x,control=mars.control(Mmax=4,d=3,trace=T))

> fwd
$y
[1] -0.099579872  0.705472027  7.480200632  1.698307661  1.632928928  7.022810365  2.907721192  0.241620737  0.068519824
...
[91]  4.380931695  3.119764429  2.214397181 -0.013217513 10.967745814  0.104057346  8.086971553 12.799833137  0.861418747
[100] -0.044695931

$B
    B0         B1         B2          B3         B4
1    1 0.42158428 0.00000000 0.000000000 0.00000000
2    1 0.09128613 0.00000000 0.000000000 0.00000000
3    1 0.00000000 1.69759968 0.360951494 0.00000000
4    1 0.00000000 0.20939975 0.065641660 0.00000000
5    1 0.00000000 0.26817910 0.246068083 0.00000000
     . . . . . .
100  1 0.88752954 0.00000000 0.000000000 0.00000000
```

Forward algorithm testing result (2/3)

```
$Bfuncs
$Bfuncs[[1]]
NULL

$Bfuncs[[2]]
      s v          t
[1,] -1 1 -0.1388914

$Bfuncs[[3]]
     s v          t
[1,] 1 1 -0.1388914

$Bfuncs[[4]]
      s v           t
[1,]  1 1 -0.13889136
[2,] -1 2 -0.03406725

$Bfuncs[[5]]
     s v           t
[1,] 1 1 -0.13889136
[2,] 1 2 -0.03406725
```

Figure 2: Algorithm 2-3

Forward algorithm testing result (2/3)

```
# the basis functions obtained from Bfunc
Bfuncs1 = rep(1,length(y))
Bfuncs2 = h(x[,1],-1,-0.1388914)
Bfuncs3 = h(x[,1],+1,-0.1388914)
Bfuncs4 = h(x[,1],+1,-0.13889136)*h(x[,2],-1,-0.03406725)
Bfuncs5 = h(x[,1],+1,-0.13889136)*h(x[,2],+1,-0.03406725)
# the model
mod_fwd = lm(y~Bfuncs2+Bfuncs3+Bfuncs4+Bfuncs5)
mod_fwd$coef

> mod_fwd$coef
(Intercept)     Bfuncs2     Bfuncs3     Bfuncs4     Bfuncs5
0.47231115 -0.41129999  3.49692286  2.75603224  0.06642471

# the basis function is already formed in B matrix
mod_fwd2 = lm(y~as.matrix(fwd$B)-1)
mod_fwd2$coef

> mod_fwd2$coef
as.matrix(fwd$B)B0 as.matrix(fwd$B)B1 as.matrix(fwd$B)B2 as.matrix(fwd$B)B3 as.matrix(fwd$B)B4
0.47231123         -0.41130005          3.49692290          2.75603224          0.06642471
```

Figure 3: Algorithm 2-4

Over-fitting

- During the forward algorithm, we added the basis function that improved LOF, the residual sum of squares (RSS)

$$\sum_{i=1}^{N}(y_i - \hat{f}_M(x_i))^2$$

  where $\hat{f}_M$ is a fitted model with M basis functions.
- RSS is OK for selecting among models with the same M, but not for comparing models with different M.
  - RSS decreases as we add predictors, even those not truly associated with the response, and so favours larger models.
- For model selection we need an unbiased measure of the "test error", which is the average squared error between observed and predicted values for data not used to fit the model.

Test error

- We call the RSS from an independent set of data not used to fit the model the validation error.
  - Split our data into "training" and "test" sets.
  - Estimate of test set error depends on split.
- Better: Use cross-validation (CV), which splits the data into "folds", fits the model on all but a hold-out, and averages the validation errors across folds.
- However, CV is time-consuming and so approximations are of interest.
- Generalized cross-validation, or GCV is one such approximation.

Generalized cross-validation (GCV)

- The LOF measure $LOF(\hat{f}(M)) = GCV(M)$ in Friedman's equations (30) and (32) is

$$\frac{1}{N}\frac{\sum_{i=1}^{N}(y_i - \hat{f}_M(x_i))^2}{(1-\tilde{C}(M)/N)^2} = RSS \times \frac{N}{(N-\tilde{C}(M))^2}$$

  where $\tilde{C}(M) = C(M) + dM$, $C(M)$ is the sum of the hat-values from the fitted model and d is a smoothing parameter.
  - $C(M) = M + 1$ if there are no linear dependencies between basis functions, but summing the hat-values is safest.
  - Friedman suggests that $d = 3$ works well.
  - Denominator decreases, so GCV increases as M increases.
- Notice that for fixed M, and assuming no linear dependencies between basis functions, the best model is the one with smallest RSS, so our forward stepwise algorithm is OK as-is.
- Use GCV to compare models with different M in the backward stepwise algorithm.

Alternatives to GCV

- Backward stepwise selection is implemented in the R function `step()`. However, `step()` uses Mallow's C_p instead of GCV, where

$$C_p = RSS/S^2 - N + 2(M+1)$$

  for an estimate S^2 of σ^2 from a low-bias model, usually the largest one fit.
  - C_p is very similiar to Akaike's Information Criterion (AIC)
- As with GCV we see that C_p penalizes RSS with a penalty that becomes larger as M increases.
  - The factor of 2 in $2(M+1)$ can be modified to apply more/less penalty.
  - Replacing 2 with $\log(N)$ gives a Bayesian Information Criterion (BIC)-like penalty.

Backwards stepwise algorithm (1/2)

Algorithm 3 (MARS—backwards stepwise)

$J^* = \{1, 2, \ldots, M_{\max}\}$; $K^* \leftarrow J^*$
$\text{lof}^* \leftarrow \min_{\{a_j | j \in J^*\}} \text{LOF}(\sum_{j \in J^*} a_j B_j(\mathbf{x}))$
For $M = M_{\max}$ to 2 do: $b \leftarrow \infty$; $L \leftarrow K^*$
  For $m = 2$ to M do: $K \leftarrow L - \{m\}$
    $\text{lof} \leftarrow \min_{\{a_k | k \in K\}} \text{LOF}(\sum_{k \in K} a_k B_k(\mathbf{x}))$
    if $\text{lof} < b$, then $b \leftarrow \text{lof}$; $K^* \leftarrow K$ end if
    if $\text{lof} < \text{lof}^*$, then $\text{lof}^* \leftarrow \text{lof}$; $J^* \leftarrow K$ end if
  end for
end for
end algorithm

Figure 4: Algorithm 3

Backwards stepwise algorithm (2/2)

- Initialize J^* to the set of all basis functions from the forward algorithm
- Outer loop over M in M_{max} to 2 (like `step()`):
  - Inner loop over M terms: Find the one that reduces GCV(M) the most (like `drop()`).
  - If GCV(M) best seen in outer loop, update J^*
- Algorithm terminates with best model J^*.

Software features

- Arguments/inputs
  - Formula interface to specify response and explanatory variables.
  - data argument for input data
  - list data structure of parameters that control algorithm, such as M_{max} and parameter d in GCV.
- Value/output
  - output of `lm()` from final fit
  - specification of basis functions from final fit that can be used for prediction
  - Return a list:
    `list(y=fwd$y,B=fwd$B[,Jstar],Bfuncs=fwd$Bfuncs[Jstar])`

Data structures and functions

- Input data structures
  - data in a data frame
  - list of parameters (see `glm.control`)
- Output data strctures
  - object of class `mars`
- Functions
  - `mars.control()` to set up input parameters, with default settings
  - `mars()`, the main MARS function that fits the model
  - `predict()`, `residuals()`, `fitted()`, `...` for objects of class `mars`
  - `plot()` for objects of class `mars` – what would this do?