Flexible modelling using basis expansions (Chapter 5)

- Linear regression: $y = X\beta + \epsilon, \qquad \epsilon \sim (0, \sigma^2)$
- 'Smooth' regression: $y = f(X) + \epsilon$: $f(X) = E(Y \mid X)$ to be specified
- Flexible linear modelling

$$f(X) = \Sigma_{m=1}^{M} \beta_m h_m(X)$$

- This is called a linear basis expansion, and h_m is the mth basis function
- For example if X is one-dimensional:
  $f(X) = \beta_0 + \beta_1 X + \beta_2 X^2$, or
  $f(X) = \beta_0 + \beta_1 \sin(X) + \beta_2 \cos(X)$, etc.
- Simple linear regression has $h_1(X) = 1$, $h_2(X) = X$.
  Several other examples on p.116

Regression splines

- Polynomial fits: $h_j(x) = x^j, j = 0, \ldots, m$
- Fit using linear regression with design matrix X, where $X_{ij} = h_j(x_i)$
- Justification is that any 'smooth' function can be approximated by a polynomial expansion (Taylor series)
- Can be difficult to fit numerically, as correlation between columns can be large
- May be useful locally, but less likely to work over the range of X
- (rafal.pdf) $f(x) = x^2 \sin(2x)$, $\epsilon \sim N(0, 2)$, $n = 200$
- It turns out to be easier to find a good set of basis functions if we only consider small segments in the X-space (local fits)
- Need to be careful not to overfit, since we are using only a fraction of the data

- Piecewise constant: $h_1(X) = I(X < \xi_1), h_2(X) = I(\xi_1 \leq X < \xi_2), h_3(x) = I(\xi_2 \leq X)$; corresponds to fitting by local averaging
- Similarly for piecewise linear fits (Figure 5.1), but constraints to make it continuous at the break points: $h_1(X) = 1, \quad h_2(X) = X, \quad h_3(X) = (X - \xi_1)_+, \quad h_4(X) = (X - \xi_2)_+$
- windows defined by *knots* $\xi_1, \xi_2, \ldots$
- To fit a cubic polynomial in each window: e.g. $a_1 + b_1X + c_1X^2 + d_1X^3$ in the first, $a_2 + b_2X + c_2X^2 + d_2X^3$ in second, etc.
- basis functions $\{1_{w_1}, X_{w_1}, X^2_{w_1}, X^3_{w_1}\}, \{1_{w_2}, X_{w_2}, X^2_{w_2}, X^3_{w_2}\}$, etc.
- Now require $f^{(k)}_{w_1}(\xi_1) = f^{(k)}_{w_2}(\xi_1)$ etc., puts constraints on the coefficients a, b, c, d
- Cubic splines require continuous function, first derivatives, second derivatives (Figure 5.2)

Regression splines

- Constraints on derivatives can be incorporated into the basis: $\{1, X, X^2, X^3, (X-\xi_1)^3_+, \ldots, (X-\xi_k)^3_+\}$ the truncated power basis
- procedure: choose number (K) and placement of knots $\xi_1, \ldots \xi_K$
- construct X matrix using truncated power basis set
- run linear regression with ?? degrees of freedom
- Example: heart failure data from Chapter 4 (462 observations, 10 covariates)

```
> bs.sbp <- bs(hr$sbp,df=4)    # the B-spline basis
> dim(bs.sbp)
[1] 462   4                  # this is the basis matrix
> bs.sbp[1:4,]
                1         2          3           4
[1,] 0.16968090 0.4511590 0.34950621 0.029653925
[2,] 0.35240669 0.4758851 0.17002102 0.001687183
[3,] 0.71090498 0.1642420 0.01087580 0.000000000
[4,] 0.09617733 0.3769808 0.44812470 0.078717201
```

- The *B*-spline basis is hard to described explicitly (see Appendix to Ch. 5), but can be shown to be equivalent to the truncated power basis:
- In R library(splines): `bs(x, df=NULL, knots=NULL, degree=3,intercept=FALSE, Boundary.knots=range(x))`
- Must specify either `df` or `knots`. For the *B*-spline basis, `# knots = df - degree` (degree is usually 3: see `?bs`).
- The knots are fixed, even if you use `df` (see R code)
- Natural cubic splines have better endpoint behaviour (linear) (p.120, 121)
- `ns(x, df=NULL, knots=NULL, degree=3,intercept=FALSE, Boundary.knots=range(x))`
- For natural cubic splines, `# knots = df - 1`

Regression splines (p.120) refers to using these basis matrices in a regression model.

```
> ns.hr.glm <- glm (chd ~ ns.sbp+ns.tobacco+ns.ldl+famhist+ns.obesity
+ + ns.alcohol + ns.age, family=binomial, data=hr)
> summary(ns.hr.glm)

Call:
glm(formula = chd ~ ns.sbp + ns.tobacco + ns.ldl + famhist +
    ns.obesity + ns.alcohol + ns.age, family = binomial, data = hr)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.7245  -0.8265  -0.3884   0.8870   2.9588

Coefficients:
               Estimate Std. Error z value Pr(>|z|)
(Intercept)     -2.1158     2.4067  -0.879 0.379324
ns.sbp1         -1.4794     0.8440  -1.753 0.079641 .
ns.sbp2         -1.3197     0.7632  -1.729 0.083769 .
ns.sbp3         -3.7537     2.0230  -1.856 0.063520 .
ns.sbp4          1.3973     1.0037   1.392 0.163884
ns.tobacco1      0.6495     0.4586   1.416 0.156691
ns.tobacco2      0.4181     0.9031   0.463 0.643397
ns.tobacco3      3.3626     1.1873   2.832 0.004625 **
ns.tobacco4      3.8534     2.3769   1.621 0.104976
ns.ldl1          1.8688     1.3266   1.409 0.158933
ns.ldl2          1.7217     1.0320   1.668 0.095248 .
ns.ldl3          4.5209     2.9986   1.508 0.131643
ns.ldl4          3.3454     1.4523   2.304 0.021249 *
famhistPresent   1.0787     0.2389   4.515 6.34e-06 ***
ns.obesity1     -3.1058     1.7187  -1.807 0.070748 .
ns.obesity2     -2.3753     1.2042  -1.972 0.048555 *
ns.obesity3     -5.0541     3.8205  -1.323 0.185871
```

Example: heart data

The individual coefficients don't mean anything, we need to evaluate groups of coefficients. We can do this with successive likelihood ratio tests, by hand, e.g.

```
> summary(glm(chd~ns.sbp+ns.ldl+famhist+ns.obesity+ns.alcohol+ns.age,
+ family=binomial, data=hr)) # I left out tobacco

 ... stuff omitted


Null deviance: 596.11  on 461  degrees of freedom
Residual deviance: 469.61  on 440  degrees of freedom
AIC: 513.61

Number of Fisher Scoring iterations: 5


> 469.61-457.63
[1] 11.98
> pchisq(11.98,4)
[1] 0.9824994
> 1-.Last.value
[1] 0.01750061   # doesn't agree exactly with the book, but close
```

See Figure 5.4

The function `stepAIC` does all this for you:

```
> ns.hr.step <- stepAIC(ns.hr.glm)
Start:  AIC= 509.63
 chd ~ ns.sbp + ns.tobacco + ns.ldl + famhist + ns.obesity + ns.alcohol +
    ns.age

             Df Deviance    AIC
- ns.alcohol  4   458.09 502.09
- ns.obesity  4   465.41 509.41
<none>            457.63 509.63
- ns.sbp      4   466.77 510.77
- ns.tobacco  4   469.61 513.61
- ns.ldl      4   470.90 514.90
- ns.age      4   480.37 524.37
- famhist     1   478.76 528.76

Step:  AIC= 502.09
 chd ~ ns.sbp + ns.tobacco + ns.ldl + famhist + ns.obesity + ns.age

             Df Deviance    AIC
<none>            458.09 502.09
- ns.obesity  4   466.24 502.24
- ns.sbp      4   467.16 503.16
- ns.tobacco  4   470.48 506.48
- ns.ldl      4   472.39 508.39
- ns.age      4   481.86 517.86
- famhist     1   479.44 521.44
> #
> # Here we are at Table 5.1; note that alcohol has been dropped from the
> # model
```

Example: heart data

The function `stepAIC` does all this for you:

```
> ns.hr.step <- stepAIC(ns.hr.glm)
Start:  AIC= 509.63
 chd ~ ns.sbp + ns.tobacco + ns.ldl + famhist + ns.obesity + ns.alcohol +
    ns.age

              Df Deviance    AIC
- ns.alcohol  4   458.09 502.09
- ns.obesity  4   465.41 509.41
<none>            457.63 509.63
- ns.sbp      4   466.77 510.77
- ns.tobacco  4   469.61 513.61
- ns.ldl      4   470.90 514.90
- ns.age      4   480.37 524.37
- famhist     1   478.76 528.76

Step:  AIC= 502.09
 chd ~ ns.sbp + ns.tobacco + ns.ldl + famhist + ns.obesity + ns.age

              Df Deviance    AIC
<none>            458.09 502.09
- ns.obesity  4   466.24 502.24
- ns.sbp      4   467.16 503.16
- ns.tobacco  4   470.48 506.48
- ns.ldl      4   472.39 508.39
- ns.age      4   481.86 517.86
- famhist     1   479.44 521.44
> #
> # Here we are at Table 5.1; note that alcohol has been dropped from the
> # model
```

The **degrees of freedom** fitted are the number of columns in the basis matrix (+ 1 for the intercept). This can also be computed as the trace of the **hat matrix**, which can be extracted from `lm`. There is something analogous for `glm`, because glm's are fitted using iteratively reweighted least squares.

Smoothing splines

- This is an approach closer to ridge regression. Put knots at each distinct x value, and then shrink the coefficients by penalizing the fit . (Figure 5.6)
- $$\text{argmin}_\beta \Sigma_i \left(y_i - \Sigma_j \beta_j h_j(x_i)\right)^2$$
  subject to:
  $$\beta^T \Omega \beta < c$$
- with no constraint we get usual least squares
- Ω controls the smoothness of the final fit:
  $$\Omega_{jk} = \int h_j''(x) h_k''(x) dx$$
- This solves the variational problem
  $$\text{argmin}_f \Sigma_i (y_i - f(x_i))^2 + \lambda \int_a^b \{f''(t)\}^2 dt$$
- the solution is a natural cubic spline with knots at each x_i

- How many parameters have been fit?
- It can be shown that the solution to the smoothing spline problem gives fitted values of the form

$$\hat{y} = S_\lambda y$$

- By analogy with ordinary regression, define the **effective degrees of freedom** (EDF) as

$$\text{trace } S_\lambda$$

- Reminder: ridge regression

$$\min_\beta \Sigma_{i=1}^N (y_i - \beta_0 - \beta_1 x_{i1} - \cdots - \beta_p x_{ip})^2 + \lambda \Sigma_{j=1}^p \beta_j^2$$

$$\iff \min_\beta \Sigma_{i=1}^N (y_i - \beta_0 - \beta_1 x_{i1} - \cdots - \beta_p x_{ip})^2 \quad \text{s.t. } \Sigma_{j=1}^p \beta_j^2 \leq s$$

has solution

$$\hat{y}_{ridge} = X(X^T X + \lambda I)^{-1} X^T y$$

- In the smoothing case it can be shown that

$$\hat{y}_{smooth} = H(H^T H + \lambda \Omega_H)^{-1} H^T y$$

  where H is the basis matrix. See p. 130, 132 for details on S_λ, the smoothing matrix.
- How to choose λ?
- a) Decide on df to be used up, e.g. `smooth.spline(x,y,df=6)`, note that increasing df means less 'bias' and more 'variance'.
- b) Automatic selection by cross-validation (Figure 5.9)

A smoothing spline version of logistic regression is outlined in §5.6, but we'll wait till we discuss generalized additive models. An example from the R help file for `smooth.spline`:

```
> data(cars)
> attach(cars)
> plot(speed, dist, main = "data(cars)  &  smoothing splines")
> cars.spl <- smooth.spline(speed, dist)
> (cars.spl)
Call:
smooth.spline(x = speed, y = dist)

Smoothing Parameter  spar= 0.7801305  lambda= 0.1112206 (11 iterations)
Equivalent Degrees of Freedom (Df): 2.635278
Penalized Criterion: 4337.638
GCV: 244.1044
> lines(cars.spl, col = "blue")
>      lines(smooth.spline(speed, dist, df=10), lty=2, col = "red")
>      legend(5,120,c(paste("default [C.V.] => df =",round(cars.spl$df,1)),
+                     "s( * , df = 10)"), col = c("blue","red"), lty = 1:2,
+             bg='bisque')
> detach()
```