ECON 950 — Winter 2020

Prof. James MacKinnon

9. Going Beyond Linear Models

Linear regression, linear discriminant analysis, and logistic regression all rely on models that are linear in the inputs.

We often need to go beyond such models. The easiest way to do so is to augment or replace the original inputs $\boldsymbol{x}$ with transformed inputs that allow for more flexible relationships with the output variable.

Let $h_m(\boldsymbol{x})$ denote a transformation of $\boldsymbol{x}$. There are M of these, and typically $M > p$, perhaps much greater.

We then use the **linear basis expansion**

$$f(\boldsymbol{x}) = \sum_{m=1}^{M} \beta_m h_m(\boldsymbol{x}). \tag{1}$$

Once we have chosen the basis expansion, we can fit all sorts of models just as we did before.

In particular, we can use ordinary linear regression or logistic regression, perhaps with L_1, L_2, elastic-net, or some fancier form of regularization.

- If $h_m(\boldsymbol{x}) = x_m, m = 1, \ldots, p$ with $M = p$, we get the original linear model.
- If $h_m(\boldsymbol{x}) = x_j^2$ and/or $x_j x_k$ for $m > p$, we can augment the original model with polynomial and/or cross-product terms. Unfortunately, a full polynomial model of degree d requires $O(p^d)$ terms.

  When $d = 2$, we have p linear terms, p quadratic terms, and $p * (p-1)/2$ cross-product terms.

  Global polynomials do not work very well. It is much better to use **piecewise polynomials** and **splines** which allow for local polynomial representations.
- We can use transformations of single inputs, such as $h_m(\boldsymbol{x}) = \log(x_j)$ or $\sqrt{x_j}$, either replacing x_j or as additional regressors.
- We can use functions of several inputs, such as $h(\boldsymbol{x}) = ||\boldsymbol{x}||$.

- We can create all sorts of indicator (dummy) variables, such as $h_m(\boldsymbol{x}) = \mathbb{I}(l_m \leq x_k < u_m)$. In such cases, we often create a lot of them.

  For example, we could divide the range of x_k into M_k nonoverlapping segments. The M_k dummies that result would allow a piecewise constant relationship between x_k and y.

  In such a case, it would often be a very good idea to use a form of L_2 regularization, not shrinking the coefficients towards zero, but instead shrinking their first differences, as in Shiller (1973) and Gersovitz and MacKinnon (1978).

To avoid estimating too many coefficients, we often have to impose restrictions. One popular one is **additivity**:

$$f(\boldsymbol{x}) = \sum_{j=1}^{p} f_j(x_j) = \sum_{j=1}^{p} \sum_{m=1}^{M} \beta_{jm} h_{jm}(x_j). \tag{2}$$

In many cases, it is important to employ selection methods, such as the (possibly grouped) lasso.

It is also often important to use regularization, including lasso, ridge, and other methods to be discussed.

9.1. Piecewise Polynomials and Splines

A **piecewise polynomial** function $f(x)$ simply divides the domain of x (assumed one-dimensional for now) into continguous intervals.

In the simplest case, with $M = 3$,

$$h_1(x) = \mathbb{I}(x \leq \xi_1), \quad h_2(x) = \mathbb{I}(\xi_1 < x \leq \xi_2), \quad h_3(x) = \mathbb{I}(\xi_2 < x). \tag{3}$$

Thus, for this **piecewise constant** function,

$$f(x) = \sum_{m=1}^{3} \beta_m h_m(x). \tag{4}$$

The least squares estimates are just

$$\hat{\beta}_m = \bar{y}_m = \frac{1}{\boldsymbol{\iota}^\top \boldsymbol{h}_m} \boldsymbol{y}^\top \boldsymbol{h}_m, \tag{5}$$

where $\boldsymbol{h}_m$ is a vector with typical element $h_m(x_i)$. It is a vector of 0s and 1s.

Of course, a piecewise constant function is pretty silly; see the upper left panel of ESL-fig5.01.pdf.

A less stupid approximation is **piecewise linear**. Now we have six basis functions when there are three regions. Numbers 4 to 6 are

$$h_1(x)x, \quad h_2(x)x, \quad \text{and} \quad h_3(x)x. \tag{6}$$

But $f(x)$ still has gaps in it; see the upper right panel of ESL-fig5.01.pdf.

The best way to avoid gaps is to constrain the basis functions to meet at ξ_1 and ξ_2, which are called **knots**. This is most easily accomplished by using **regression splines**. In the linear case with three regions, there are four basis functions:

$$h_1(x) = 1, \quad h_2(x) = x, \quad h_3(x) = (x - \xi_1)_+, \quad h_4(x) = (x - \xi_2)_+. \tag{7}$$

See the lower left panel of ESL-fig5.01.pdf. $h_3(x)$ is plotted the lower right panel. Here $h_3(x)$ and $h_4(x)$ are **truncated power basis functions**.

Linear functions are pretty restrictive.

It is very common to use a **cubic spline**, which is constrained to be continuous and to have continuous first and second derivatives.

With three regions (and therefore two knots) a cubic spline has six basis functions. These are

$$\begin{aligned} &h_1(x) = 1, \quad h_2(x) = x, \quad h_3(x) = x^2, \quad h_4(x) = x^3, \\ &h_5(x) = (x - \xi_1)_+^3, \quad h_6(x) = (x - \xi_2)_+^3. \end{aligned} \tag{8}$$

In general, a spline is of order M and has K knots. Thus (7) is an order-2 spline with 2 knots, and (8) is an order-4 spline that also has 2 knots.

The basis functions for an order-M spline with K knots ξ_k are:

$$h_j(x) = x^{j-1}, \quad j = 1, \ldots, M, \tag{9}$$

$$h_{M+k}(x) = (x - \xi_k)_+^{M-1}, \quad k = 1, \ldots, K. \tag{10}$$

In practice, $M = 1, 2, 4$. For some reason, cubic splines are much more popular than quadratic ones.

For computational reasons when N and K are large, it is desirable to use B-splines rather than ordinary ones. See the Appendix to Chapter 5 of ESL.

9.2. Natural Cubic Splines

Polynomials tend to behave badly near boundaries. Using them to extrapolate is extremely dangerous.

A **natural cubic spline** adds additional constraints, namely, that the function is linear beyond the boundary knots.

This saves four degrees of freedom (2 at each boundary knot) and reduces variance, at the expense of higher bias. The freed degrees of freedom can be used to add additional interior knots.

In general, a cubic spline with K knots can be represented as

$$f(x) = \sum_{j=0}^{3} \beta_j x^j + \sum_{k=1}^{K} \theta_k (x - \xi_k)_+^3; \tag{11}$$

compare (8).

The boundary conditions imply that

$$\beta_2 = \beta_3 = 0, \quad \sum_{k=1}^{K} \theta_k = 0, \quad \sum_{k=1}^{K} \xi_k \theta_k = 0. \tag{12}$$

Combining these with (11) gives us the basis functions for the natural cubic spline with K knots. Instead of $K+4$ basis functions, there will be just K.

The first two basis functions are just 1 and x. The remaining $K-2$ are

$$N_{k+2}(x) = \frac{(x-\xi_k)_+^3 - (x-\xi_K)_+^3}{\xi_K - \xi_k} - \frac{(x-\xi_{K-1})_+^3 - (x-\xi_K)_+^3}{\xi_K - \xi_{K-1}} \tag{13}$$

for $k = 1, \ldots, K-2$.

Since the number of basis functions is only K, we may be able to have quite a few knots.

When there are two or more inputs, they cannot both include a constant among the basis functions.

ESL discusses a heart-disease dataset. For the continuous regressors, they use natural cubic splines with 3 interior knots.

ESL fits a logistic regression model, without regularization, using backwards step deletion.

ESL drops terms, not individual regressors, using AIC. Only the `alcohol` term is actually dropped.

The final model has one binary variable and five continuous ones. Each of the latter uses a natural spline with 3 interior knots.

There are 22 coefficients (a constant, a family history dummy, and five splines with 4 each).

ESL-fig5.04.pdf plots the fitted natural-spline functions along with pointwise bands of width two standard errors.

Let $\boldsymbol{\theta}$ denote the entire vector of parameters, and $\hat{\boldsymbol{\theta}}$ its ML estimator. Then $\widehat{\mathrm{Var}}(\hat{\boldsymbol{\theta}}) = \hat{\boldsymbol{\Sigma}}$ is computed by the usual formula for logistic regression:

$$\widehat{\mathrm{Var}}(\hat{\boldsymbol{\theta}}) = \left(\boldsymbol{X}^\top \boldsymbol{\Upsilon}(\hat{\boldsymbol{\theta}}) \boldsymbol{X}\right)^{-1}, \tag{14}$$

where

$$\Upsilon_t(\boldsymbol{\theta}) \equiv \frac{\lambda^2(\boldsymbol{X}_t\boldsymbol{\theta})}{\Lambda(\boldsymbol{X}_t\boldsymbol{\theta})\big(1 - \Lambda(\boldsymbol{X}_t\boldsymbol{\theta})\big)}\,. \tag{15}$$

Here $\Lambda(\cdot)$ is the logistic function, and $\lambda(\cdot)$ is its derivative. That is,

$$\Lambda(x) \equiv \frac{1}{1+e^{-x}} = \frac{e^x}{1+e^x} \tag{16}$$

$$\lambda(x) \equiv \frac{e^x}{(1+e^x)^2} = \Lambda(x)\Lambda(-x). \tag{17}$$

Then the pointwise standard errors are the square roots of

$$\widehat{\mathrm{Var}}\big(\hat{f}(\boldsymbol{x}_{ji})\big) = \boldsymbol{h}_j(x_{ji})^\top \hat{\boldsymbol{\Sigma}}_{jj} \boldsymbol{h}_j(x_{ji}), \tag{18}$$

where x_{ji} is the value of the jth variable for observation i, and $\boldsymbol{h}_j(x_{ji})$ is the vector of values of the basis functions for x_{ji}. In this case, it is a 4–vector.

Notice how the confidence bands are wide where the data are sparse and narrow where they are dense.

Notice the unexpected shapes of the curves for `sbp` (systolic blood pressure) and `obesity` (BMI). These are retrospective data!

9.3. Smoothing Splines

Consider the minimization problem

$$\min_f \left(\sum_{i=1}^{N} \big((y_i - f(x_i, \lambda)\big)^2 + \lambda \int \big(f''(t)\big)^2 dt \right), \tag{19}$$

where λ is a fixed smoothing parameter.

When $\lambda = \infty$, minimizing (19) yields the OLS estimates, since the second derivative must be 0.

When $\lambda = 0$, minimizing (19) yields a perfect fit, provided every x_i is unique, because $f(x)$ can be any function that goes through every data point.

It can be shown (see Exercise 5.7, which is nontrivial) that the minimizer is a natural cubic spline with knots at all of the (unique) x_i.

The penalty term shrinks the spline coefficients towards the least squares fit.

We can write the solution as

$$f(x) = \sum_{j=1}^{N} \theta_j N_j(x), \tag{20}$$

where the $N_j(x)$ are an N-dimensional set of basis functions for this family of natural splines. Thus $N_1(x) = 1$, $N_2(x) = x$, and the remaining $N = 2$ are given by (13) with $K = N$.

Thus we are actually minimizing

$$(\boldsymbol{y} - \boldsymbol{N}\boldsymbol{\theta})^\top(\boldsymbol{y} - \boldsymbol{N}\boldsymbol{\theta}) + \lambda\boldsymbol{\theta}^\top\boldsymbol{\Omega}_{\boldsymbol{N}}\boldsymbol{\theta}, \tag{21}$$

where $\boldsymbol{N}$ is an $N \times N$ matrix with typical element $N_j(x_i)$, and

$$(\boldsymbol{\Omega}_{\boldsymbol{N}})_{jk} \equiv \int N_j''(t)N_k''(t)dt. \tag{22}$$

This involves the second derivatives of the basis functions defined in (13). But what does it actually mean?

The solution that minimizes (21) is

$$\hat{\boldsymbol{\theta}} = (\boldsymbol{N}^\top\boldsymbol{N} + \lambda\boldsymbol{\Omega}_{\boldsymbol{N}})^{-1}\boldsymbol{N}^\top\boldsymbol{y}, \tag{23}$$

which looks a lot like the solution to ridge regression, except that the identity matrix has been replaced by $\boldsymbol{\Omega}_{\boldsymbol{N}}$.

Evidently,

$$\hat{f}(x) = \sum_{j=1}^{N} \hat{\theta}_j N_j(x). \tag{24}$$

Efficient computational techniques use B-splines instead of natural splines. Thus we replace $\boldsymbol{N}$ by $\boldsymbol{B}$.

The matrix $\boldsymbol{B}$ has $N+4$ columns instead of N, but the big advantage is that it is lower 4-banded. In consequence, $\boldsymbol{B}^\top \boldsymbol{B} + \lambda \boldsymbol{\Omega}$ is 4-banded.

This makes computing these matrices (with sorted values of the x_i cheap) and Cholesky decomposition also cheap.

Use Cholesky to find $\boldsymbol{L}$ such that $\boldsymbol{L}\boldsymbol{L}^\top = \boldsymbol{B}^\top \boldsymbol{B} + \lambda \boldsymbol{\Omega}$. Then solve the equations

$$\boldsymbol{L}\boldsymbol{L}^\top \boldsymbol{\theta} = \boldsymbol{B}^\top \boldsymbol{y} \tag{25}$$

by back substitution to find $\boldsymbol{\theta}$ and hence $\hat{\boldsymbol{f}}$.

In practice, we do not really need all N interior knots. For large N, we can get away with a number proportional to (but much larger than) $\log N$; see the `smooth.spline` function in the `stats` package in R.

9.4. Smoother Matrices

Even though we do not compute it this way, we can write

$$\begin{aligned}\hat{\boldsymbol{f}} &= \boldsymbol{N}(\boldsymbol{N}^\top \boldsymbol{N} + \lambda \boldsymbol{\Omega}_{\boldsymbol{N}})^{-1} \boldsymbol{N}^\top \boldsymbol{y} \\ &= \boldsymbol{S}_\lambda \boldsymbol{y}\end{aligned} \tag{26}$$

This should look familiar! Here $\boldsymbol{S}_\lambda$ is a **smoother matrix**. It is a linear operator, and it is symmetric and positive semidefinite.

If $\lambda = 0$, $\boldsymbol{S}_\lambda$ would be a projection matrix.

If we were using natural or cubic splines, we could write

$$\hat{\boldsymbol{f}} = \boldsymbol{B}_{\boldsymbol{\xi}}(\boldsymbol{B}_{\boldsymbol{\xi}}^\top \boldsymbol{B}_{\boldsymbol{\xi}})^{-1} \boldsymbol{B}_{\boldsymbol{\xi}}^\top \boldsymbol{y} = \boldsymbol{H}_{\boldsymbol{\xi}} \boldsymbol{y}, \tag{27}$$

where $\boldsymbol{B}_{\boldsymbol{\xi}}$ is an $N \times M$ matrix of basis functions. Then $\boldsymbol{H}_{\boldsymbol{\xi}}$ would be a projection matrix with rank M.

In contrast, the rank of $\boldsymbol{S}_\lambda$ is N.

Unlike $\boldsymbol{H}_{\boldsymbol{\xi}}$, $\boldsymbol{S}_\lambda$ is not idempotent. Instead

$$\boldsymbol{S}_\lambda \boldsymbol{S}_\lambda \preceq \boldsymbol{S}_\lambda, \tag{28}$$

which means that the difference between $\boldsymbol{S}_\lambda$ and $\boldsymbol{S}_\lambda \boldsymbol{S}_\lambda$ is a positive semidefinite matrix.

Thus we can see that the smoother matrix shrinks whatever it multiplies.

The **effective degrees of freedom** of a smoothing spline is

$$\mathrm{df}_\lambda = \mathrm{Tr}(\boldsymbol{S}_\lambda). \tag{29}$$

Instead of specifying λ, we can specify df_λ, and solve for λ by solving (29).

We can rewrite $\boldsymbol{S}_\lambda$ in the **Reinsch form** as

$$\boldsymbol{S}_\lambda = (\mathbf{I} - \lambda \boldsymbol{K})^{-1}. \tag{30}$$

The argument goes as follows:

$$\begin{aligned} \hat{\boldsymbol{y}} &= \boldsymbol{N}(\boldsymbol{N}^\top \boldsymbol{N} + \lambda \boldsymbol{\Omega}_{\boldsymbol{N}})^{-1} \boldsymbol{N}^\top \boldsymbol{y} \\ &= \boldsymbol{N}\big(\boldsymbol{N}^\top (\mathbf{I} + \lambda (\boldsymbol{N}^\top)^{-1} \boldsymbol{\Omega}_{\boldsymbol{N}} \boldsymbol{N}^{-1}) \boldsymbol{N}\big)^{-1} \boldsymbol{N}^\top \boldsymbol{y} \end{aligned} \tag{31}$$

Since $\boldsymbol{N}$ is a square, invertible matrix, the second row here becomes

$$\begin{aligned}
&\boldsymbol{N}\boldsymbol{N}^{-1}\big(\mathbf{I}+\lambda(\boldsymbol{N}^\top)^{-1}\boldsymbol{\Omega}_{\boldsymbol{N}}\boldsymbol{N}^{-1}\big)^{-1}(\boldsymbol{N}^\top)^{-1}\boldsymbol{N}^\top\boldsymbol{y} \\
&\quad = \big(\mathbf{I}+\lambda(\boldsymbol{N}^\top)^{-1}\boldsymbol{\Omega}_{\boldsymbol{N}}\boldsymbol{N}^{-1}\big)^{-1}\boldsymbol{y} \\
&\quad = \big(\mathbf{I}+\lambda\boldsymbol{K}\big)^{-1}\boldsymbol{y}
\end{aligned} \tag{32}$$

Thus we see that equation (30) holds.

The matrix $\boldsymbol{K}$ is called the **penalty matrix**.

The eigen decomposition of $\boldsymbol{S}_\lambda$ is

$$\sum_{k=1}^{N}\rho_k(\lambda)\boldsymbol{u}_k\boldsymbol{u}_k^\top, \quad \rho_k(\lambda)=\frac{1}{1+\lambda d_k}, \tag{33}$$

where d_k is the kth eigenvalue of $\boldsymbol{K}$ and $\boldsymbol{u}_k$ is the corresponding eigenvector.

Observe that the eigenvectors do not depend on λ. It affects $\boldsymbol{S}_\lambda$ only through the eigenvalues $\rho_k(\lambda)$

The fitted values are

$$\boldsymbol{S}_\lambda \boldsymbol{y} = \sum_{k=1}^{N} \rho_k(\lambda) \boldsymbol{u}_k \boldsymbol{u}_k^\top \boldsymbol{y}. \tag{34}$$

If there were no shrinkage, this would just be $\sum_{k=1}^{N} \boldsymbol{u}_k \boldsymbol{u}_k^\top \boldsymbol{y}$. The contributions of the various eigenvectors are shrunk differently.

As the $\rho_k(\lambda)$ become smaller, the corresponding eigenvectors become more complex and get shrunk more.

The first two $\rho_k(\lambda)$ always equal 1. They correspond to the first two basis vectors, namely, $\boldsymbol{\iota}$ and $\boldsymbol{x}$. They are 1 because $d_1 = d_2 = 0$. Thus linear functions are not penalized.

The objective function (21) can be rewritten as

$$(\boldsymbol{y} - \boldsymbol{U}\boldsymbol{\theta})^\top (\boldsymbol{y} - \boldsymbol{U}\boldsymbol{\theta}) + \lambda \boldsymbol{\theta}^\top \boldsymbol{D} \boldsymbol{\theta}. \tag{35}$$

Here $\boldsymbol{U}$ has k^{th} column $\boldsymbol{u}_k$, and $\boldsymbol{D}$ is a diagonal matrix with k^{th} diagonal element d_k.

Since $\mathrm{df}_\lambda = \mathrm{Tr}(\boldsymbol{S}_\lambda)$, it must be the case from (33) that

$$\mathrm{df}_\lambda = \sum_{k=1}^{N} \rho_k(\lambda). \tag{36}$$

In contrast, for **projection smoothers** like natural and cubic splines, all the nonzero eigenvalues are 1, so that df is the dimension of the space spanned by the basis functions.

As $\lambda \to 0$, $\mathrm{df}_\lambda \to N$ and $\boldsymbol{S}_\lambda \to \mathbf{I}$.

As $\lambda \to \infty$, $\mathrm{df}_\lambda \to 2$ and $\boldsymbol{S}_\lambda \to \boldsymbol{H} = \boldsymbol{P}_{\boldsymbol{X}}$.

Although this may not be obvious, a smoothing spline is actually a local fitting method, like locally linear/quadratic kernel regression.

See ESL-fig5.08.pdf. The smoother matrix is nearly banded. Thus $\hat{f}(x_i)$ depends only on points "near" x_i.

9.5. Selection of Smoothing Parameters

For regression splines, we have to select the number of knots and their locations. This may involve picking a lot of parameters.

For smoothing splines, we just need to pick λ.

Since $\mathrm{df}(\lambda) = \mathrm{Tr}(\boldsymbol{S}_\lambda)$ is monotonic in λ, we can pick df and solve for λ numerically. This should not be hard even when N is large, because λ is a scalar and we only need the diagonals of $\boldsymbol{S}_\lambda$.

The variance of $\hat{\boldsymbol{f}}$ is

$$\boldsymbol{S}_\lambda \mathrm{Var}(\boldsymbol{y}) \boldsymbol{S}_\lambda^\top \propto \boldsymbol{S}_\lambda \boldsymbol{S}_\lambda. \tag{37}$$

If the disturbances are independent and homoskedastic, then $\mathrm{Var}(\boldsymbol{y}) = \sigma^2 \mathbf{I}$.

The bias of $\hat{\boldsymbol{f}}$ is

$$\mathrm{E}(\hat{\boldsymbol{f}}) - \boldsymbol{f} = \boldsymbol{S}_\lambda \boldsymbol{f} - \boldsymbol{f}. \tag{38}$$

Thus the mean squared error is

$$\sigma^2 \boldsymbol{S}_\lambda \boldsymbol{S}_\lambda + (\boldsymbol{S}_\lambda - \mathbf{I}) \boldsymbol{f} \boldsymbol{f}^\top (\boldsymbol{S}_\lambda - \mathbf{I})^\top. \tag{39}$$

Obviously, making λ too small (df too big) causes excessive variance, and making λ too big (df too small) causes excessive bias.

See ESL-fig5.09.pdf. When λ is too big (df $= 5$), the spline **underfits**. When it is too small (df $= 15$), the spline **overfits**.

The integrated squared prediction error, or **expected prediction error (EPE)**, is σ^2 plus the MSE (39) averaged over all points.

Unfortunately, because we don't know $\boldsymbol{f}$ (except in the context of simulation experiments where we generate the data), we cannot compute EPE.

One possibility is to use leave-one-out (N-fold) cross-validation. This turns out to be remarkably easy, because

$$\begin{aligned} \mathrm{CV}(\hat{\boldsymbol{f}}_\lambda) &= \sum_{i=1}^{N} \left(y_i - \hat{f}_\lambda^{(-i)}(x_i)\right)^2 \\ &= \sum_{i=1}^{N} \left(\frac{y_i - \hat{f}_\lambda(x_i)}{1 - S_\lambda(i,i)}\right)^2 . \end{aligned} \tag{40}$$

To compute this, we just need the original fitted values $\hat{f}_\lambda(x_i)$ and the diagonals $S_\lambda(i,i)$ of $\boldsymbol{S}_\lambda$.

This is based on essentially the same result as the one that lets us compute leave-one-out OLS estimates using just $\boldsymbol{y}$, $\hat{\boldsymbol{u}}$, and the diagonals of the hat matrix. See equation (2.62) in ETM.

In the second line of (40), the numerator is the residual for the smoothing spline based on the entire sample, and the denominator is the analog of $1 - h_i$.

This makes it very easy to compute the leave-one-out CV curve; see Figure 5.9.

9.6. Nonparametric Logistic Regression

We can use smoothing splines with logistic regression and other estimators.

Instead of $f(x)$ denoting the fitted value conditional on x, it now denotes the log of the odds:

$$\log \frac{\Pr(y = 1 \,|\, x)}{\Pr(y = 0 \,|\, x)} = f(x). \tag{41}$$

Therefore

$$p(x) \equiv \Pr(y = 1 \,|\, x) = \frac{\exp\big(f(x)\big)}{1 + \exp\big(f(x)\big)}. \tag{42}$$

The penalized criterion function is based on the loglikelihood function. It is

$$\sum_{i=1}^{N} \Big(y_i \log\big(p(x_i)\big) + (1 - y_i) \log\big(1 - p(x_i)\big) \Big) - \tfrac{1}{2}\lambda \int \big(f''(t)\big)^2 dt. \tag{43}$$

This is minimized by making f a natural spline with knots at every unique value of x. Therefore

$$f(x) = \sum_{j=1}^{N} \theta_j N_j(x), \tag{44}$$

which looks just like (20).

The gradient is

$$\boldsymbol{g}(\boldsymbol{\theta}) = \boldsymbol{N}^\top(\boldsymbol{y} - \boldsymbol{p}) - \lambda \boldsymbol{\Omega}_{\boldsymbol{N}} \boldsymbol{\theta}, \tag{45}$$

and the Hessian is

$$\boldsymbol{H}(\boldsymbol{\theta}) = -\boldsymbol{N}^\top \boldsymbol{W} \boldsymbol{N} - \lambda \boldsymbol{\Omega}_{\boldsymbol{N}}, \tag{46}$$

where $\boldsymbol{W}$ is an $N \times N$ diagonal matrix with typical diagonal element $p(x_i)\big(1-p(x_i)\big)$.

Of course, if we set $\lambda = 0$, (45) and (46) would correspond to ML estimation of logit model.

Finding $\hat{\boldsymbol{\theta}}$ such that the gradient (45) is $\mathbf{0}$ requires an iterative procedure. It could be based on

$$\boldsymbol{f}^{(h+1)} = \boldsymbol{N}(\boldsymbol{N}^\top \boldsymbol{W} \boldsymbol{N} + \lambda \boldsymbol{\Omega}_{\boldsymbol{N}})^{-1}\big(\boldsymbol{f}^{(h)} + \boldsymbol{W}^{-1}(\boldsymbol{y} - \boldsymbol{p}^{(h)}\big). \tag{47}$$

However, because $\boldsymbol{N}$ is $N \times N$, this will become prohibitively expensive when N is large. There must be tricks, such as using B-splines evaluated at far fewer than N points, to make it feasible.

9.7. Multidimensional Splines

When there are two or more regressors (in general, d of them), we can use splines in many ways.

The easiest approach is an **additive model**, where we simply use (say) a natural cubic spline for x_1 and another one for x_2. This will be discussed in the next section, on generalized additive models. It evidently makes strong assumptions.

A less restrictive approach is the **tensor product basis**.

Suppose we have M_1 basis functions to represent x_1 and M_2 basis functions to represent x_2. Then

$$g_{jk}(x) = h_{1j}(x_1)h_{2k}(x_2), \;\; j = 1, \ldots, M_1, \;\; k = 1, \ldots, M_2. \tag{48}$$

So each $g_{jk}(x)$ is a product of two basis functions. Then

$$g(x) = \sum_{j=1}^{M_1} \sum_{k=1}^{M_2} \theta_{jk}\, g_{jk}(x). \tag{49}$$

Even if M_1 and M_2 are quite small, $M_1 M_2$ can easily be large, and things rapidly get out of hand for $d >> 2$.

We can also use higher-dimensional smoothing splines. As before, we minimize

$$\sum_{i=1}^{N} \big(y_i - f(\boldsymbol{x}_i)\big)^2 + \lambda J[f], \tag{50}$$

where $J[f]$ is a penalty functional. A natural generalization of the one-dimensional penalty

$$\lambda \int \big(f''(t)\big)^2 dt \tag{51}$$

to the two-dimensional case is

$$\lambda \int\int \left(\left(\frac{\partial^2 f(\boldsymbol{x})}{\partial x_1^2} \right)^2 + 2 \left(\frac{\partial^2 f(\boldsymbol{x})}{\partial x_1 \partial x_2} \right)^2 + \left(\frac{\partial^2 f(\boldsymbol{x})}{\partial x_2^2} \right)^2 \right) dx_1 dx_2. \tag{52}$$

Minimizing (50) with this penalty leads to a **thin-plate spline**, which is similar to a cubic smoothing spline.

Thin-plate splines have the following properties:

- As $\lambda \to 0$, the solution approaches an interpolating function. It fits perfectly but is completely useless.
- As $\lambda \to \infty$, the solution approaches linear least squares.

For intermediate values of λ, the solution has the form

$$f(\boldsymbol{x}) = \beta_0 + \boldsymbol{\beta}^\top \boldsymbol{x} + \sum_{j=1}^{N} \alpha_j h_j(\boldsymbol{x}), \tag{53}$$

where

$$h_j(\boldsymbol{x}) = ||\boldsymbol{x} - \boldsymbol{x}_j||^2 \log ||\boldsymbol{x} - \boldsymbol{x}_j||. \tag{54}$$

These are called **radial basis functions**.

The solution is found by plugging (54) into (50) and minimizing it. Unfortunately, there is no sparse structure to exploit, so the solution is $O(N^3)$.

The good news is that we can get away with many fewer than N knots. Using K knots reduces the computational cost to $O(NK^2 + K^3)$.

We can also use additive smoothing splines, based on the model

$$f(\boldsymbol{x}) = \alpha + \sum_{j=1}^{d} f_j(x_j), \tag{55}$$

where each of the $f_j(\boldsymbol{x})$ is a univariate spline, and the penalty functional

$$J[f] = \sum_{j=1}^{d} \int f_j''(t_j)^2 dt_j. \tag{56}$$

We could also add cross-product terms to (56) if we wanted:

$$f(\boldsymbol{x}) = \alpha + \sum_{j=1}^{d} f_j(x_j) + \sum_{j=1}^{d} \sum_{k<j} f_{jk}(x_j, x_k). \tag{57}$$

Of course, there are potentially many ways to choose the f_j and the f_{jk}.

9.8. Generalized Additive Models

For regression, a **generalized additive model** has the form

$$\mathrm{E}(y \,|\, \boldsymbol{x}) = \alpha + f_1(x_1) + f_2(x_2) + \ldots + f_p(x_p), \tag{58}$$

where the f_j are smooth nonparametric functions.

Instead of using basis functions, which would allow estimation by OLS, we fit each function using a **scatterplot smoother**, such as a smoothing spline or a kernel smoother.

The trick is to estimate all p functions at once.

For logistic regression, we are interested in the mean of the binary response function, $\mu(\boldsymbol{x})$. This can be modeled as

$$\log\left(\frac{\mu(\boldsymbol{x})}{1-\mu(\boldsymbol{x})}\right) = \alpha + f_1(x_1) + f_2(x_2) + \ldots + f_p(x_p). \tag{59}$$

There is evidently a close relationship between (58) and (59). The difference is the **link function** $g(\mu)$.

- For regression models, $g(\mu) = \mu$.
- For logistic regression models, $g(\mu) = \log\big(\mu/(1-\mu)\big)$.
- For probit regression models, $g(\mu) = \Phi^{-1}(\mu)$.
- For loglinear models, $g(\mu) = \log(\mu)$.

Of course, not all the f_j need to be flexible. We could allow $f_j(x_j) = x_j$ for some predictors. This is necessary if a predictor is qualitative.

We could also interact dummy variables for the levels of one or more qualitative predictors with the values of other predictors.

Thus, for example, one of the f_j might be a function of income but only when age < 30, another might be a function of income when $30 \leq$ age < 50, and a third might be a function of income when age ≥ 50.

For smoothing splines, the objective function is

$$\sum_{i=1}^{N}\Big(y_i - \alpha - \sum_{j=1}^{p} f_j(x_{ij})\Big)^2 + \sum_{j=1}^{p} \lambda_j \int f_j''(t_j)^2 dt_j, \tag{60}$$

where the λ_j are tuning parameters.

As in the univariate case, the minimizer of (60) has each of the $f_j(x_{ij})$ a cubic spline with knots at every unique value of x_{ij}.

We need to impose restrictions to make α identifiable. It is normally assumed that $\sum_{j=1}^{p} f_j(x_{ij}) = 0$, which implies that $\hat{\alpha} = \bar{y}$.

In order to identify the linear parts of the cubic splines, the matrix of the x_{ij} must have full column rank.

There is a conceptually simple iterative procedure for finding a solution. It is called **backfitting**.

1. Set $\hat{\alpha} = \bar{y}$ and $\hat{f}_j(x_{ij}) = 0 \; \forall \; i, j$.
2. Cycle over $j = 1, \ldots, p$ repeatedly. At each j,

$$\hat{f}_j(x_{ij}) = S_j\left(\left\{y_i - \hat{\alpha} - \sum_{k \neq j} \hat{f}(x_{ik})\right\}_1^N\right), \tag{61}$$

   where the notation means that we apply a cubic smoothing spline, with tuning parameter λ_j, to the points inside the braces for all i.

3. After finding the $\hat{f}_j(x_{ij})$ for each j, renormalize them using

$$\hat{f}_j(x_{ij}) \leftarrow \hat{f}_j(x_{ij}) - \frac{1}{N}\sum_1^N \hat{f}_j(x_{ij}). \tag{62}$$

Mathematically, this step is unnecessary. Computationally, it is a good idea.

4. Stop when every $\hat{f}_j(x_{ij})$ is changing by less than some specified amount ε.

Essentially, this procedure involves estimating a great many univariate smoothing splines. It will be feasible if it uses a computationally efficient procedure for the smoothing splines. This means not actually inverting $N \times N$ matrices.

The same backfitting algorithm can be used with many other types of smoothers. These might include local polynomial regression (e.g. natural cubic splines instead of smoothing splines), kernel regression, or various types of series expansions.

If the smoothers are applied only at the training points, they can be represented by $N \times N$ matrices $\boldsymbol{S}_j$. Then the degrees of freedom for the jth are approximately $\text{Tr}(\boldsymbol{S}_j) - 1$. The -1 is there because we treated the constant separately.

Similar backfitting procedures can also be used for generalized additive logistic models. But the smoothing steps will involve nonlinear estimation.

1. Compute starting values $\hat{\alpha} = \log\big(\bar{y}/(1-\bar{y})\big)$ and $\hat{f}_j(x_{ij}) = 0 \;\forall\, i, j$.

2. Cycle over j repeatedly:

i. Compute $\hat{\eta}_i = \hat{\alpha} + \sum_j \hat{f}_j(x_{ij})$ and $\hat{p}_i = 1/\big(1 + \exp(-\hat{\eta}_i)\big)$.

ii. Construct the working target variable

$$z_i = \hat{\eta}_i + \frac{y_i - \hat{p}_i}{\hat{p}_i(1 - \hat{p}_i)}.$$

iii. Construct weights $w_i = \hat{p}_i(1 - \hat{p}_i)$.

iv. Fit an additive model to the targets z_i with weights w_i, using a weighted backfitting algorithm. This gives new estimates $\hat{\alpha}$ and $\hat{f}_j(x_{ij})$.

3. Stop when every $\hat{f}_j(x_{ij})$ is changing by less than some specified amount ε.

Why do we not omit $\hat{f}_j(x_{ij})$ from the sum when computing $\hat{\eta}_i$, by analogy with (61)?

This algorithm combines the iterations needed for weighted nonlinear least squares estimation of a logit function with the iterations needed for backfitting.

When the objective is classification, some classification errors may be more serious than others.

For example, if we are trying to classify incoming emails as spam or not-spam, we may care more about misclassifying not-spam.

Suppose that $y_i = 0$ if not-spam and $y_i = 1$ if spam. Then, instead of maximizing the usual loglikelihood (weighted NLS) function, we could give more weight to observations with $y_i = 0$ than to observations with $y_i = 1$.

Additive models are widely used and can work well when p is not too large.

They are easy to interpret, because we can graph the $\hat{f}_j$ against x_j.

Backfitting works for a wide variety of estimation and smoothing procedures. The key step is essentially using a one-dimensional smoother.

Estimating typically requires hand-holding as the investigator searches over various specifications.

They are not suitable for cases with p large, unless they are combined with lasso or some other procedure for selecting inputs.

9.9. Projection Pursuit Regression

This method uses an additive model, not in the original inputs but in features derived from them.

As usual, $\boldsymbol{x}$ is an input vector, and y is an output. The **projection pursuit regression** model, or PPR model, has the form

$$y = \sum_{m=1}^{M} g_m(\boldsymbol{x}^\top \boldsymbol{\omega}_m), \tag{63}$$

where the functions g_m are called **ridge functions** and are unspecified. The **direction vectors** $\boldsymbol{\omega}_m$ are unit vectors; that is, they have length unity.

If we knew the direction vectors, we could treat (63) like a generalized additive model, using smoothing splines, local polynomials, or kernel regression to estimate the ridge functions, together with backfitting methods.

When $M = 1$, (63) is called the **single index model** in econometrics.

If M is large enough, the PPR model (63) can approximate any continuous function in R^p arbitrarily well.

To estimate (63), we need to minimize

$$\sum_{i=1}^{N}\left(y_i - \sum_{m=1}^{M} g_m(\boldsymbol{x}_i^\top \boldsymbol{\omega}_m)\right)^2 \tag{64}$$

with respect to the direction vectors and the ridge functions.

If we set $M = 1$ for now, then, conditional on $\boldsymbol{\omega}$, we simply have a one-dimensional smoothing problem.

Given $\boldsymbol{g}(\cdot)$, we need to minimize the SSR over $\boldsymbol{\omega}$. ESL suggests using a quasi-Newton method very similar to the GNR.

If $\boldsymbol{\omega}_{(j)}$ denotes the value of $\boldsymbol{\omega}$ at step j of the iterative procedure, we have by a first-order Taylor expansion

$$g(\boldsymbol{x}_i^\top \boldsymbol{\omega}) \cong g(\boldsymbol{x}_i^\top \boldsymbol{\omega}_{(j)}) + g'(\boldsymbol{x}_i^\top \boldsymbol{\omega}_{(j)})\boldsymbol{x}_i^\top(\boldsymbol{\omega} - \boldsymbol{\omega}_{(j)}). \tag{65}$$

Therefore,

$$y_i - g(\boldsymbol{x}_i^\top \boldsymbol{\omega}) \cong y_i - g(\boldsymbol{x}_i^\top \boldsymbol{\omega}_{(j)}) + g'(\boldsymbol{x}_i^\top \boldsymbol{\omega}_{(j)})\boldsymbol{x}_i^\top \boldsymbol{\omega}_{(j)} - g'(\boldsymbol{x}_i^\top \boldsymbol{\omega}_{(j)})\boldsymbol{x}_i^\top \boldsymbol{\omega}. \tag{66}$$

This suggests that we want to run the regression

$$y_i - g(\boldsymbol{x}_i^\top \boldsymbol{\omega}_{(j)}) + g'(\boldsymbol{x}_i^\top \boldsymbol{\omega}_{(j)}) \boldsymbol{x}_i^\top \boldsymbol{\omega}_{(j)} = g'(\boldsymbol{x}_i^\top \boldsymbol{\omega}_{(j)}) \boldsymbol{x}_i^\top \boldsymbol{\omega} \tag{67}$$

to estimate the vector $\boldsymbol{\omega}$. This does not ensure that $||\boldsymbol{\omega}|| = 1$, but that can easily be imposed afterwards.

For reasons that are not entirely clear to me, ESL writes this in a much more complicated way. Maybe their method does ensure that $||\boldsymbol{\omega}|| = 1$.

We iterate between the regression for $\boldsymbol{\omega}$ and the smoother to estimate $g(\cdot)$ until convergence.

When there is more than one term, we add additional $(g_m, \boldsymbol{\omega}_m)$ pairs one at a time.

Backfitting can be used to re-estimate previously fitted g_m functions. In principle, it could also be used to re-estimate previously fitted $\boldsymbol{\omega}_m$ vectors, but ESL say this is rarely done.

The value of M is usually chosen by comparing model fit with M and $M-1$ terms. Cross-validation can also be used.

PPR is a very old idea, dating back to Friedman and Stuezle (1981). However, ESL say it is not widely used, maybe because it was computationally infeasible in the early 1980s.

How does it compare with simpler methods like kernel regression and smoothing splines?