model

$$\Pr(y_i > 250|x_i) = \frac{\exp(\beta_0 + \beta_1 C_1(x_i) + \cdots + \beta_K C_K(x_i))}{1 + \exp(\beta_0 + \beta_1 C_1(x_i) + \cdots + \beta_K C_K(x_i))} \tag{7.6}$$

in order to predict the probability that an individual is a high earner on the basis of `age`. The right-hand panel of Figure 7.2 displays the fitted posterior probabilities obtained using this approach.

Unfortunately, unless there are natural breakpoints in the predictors, piecewise-constant functions can miss the action. For example, in the left-hand panel of Figure 7.2, the first bin clearly misses the increasing trend of `wage` with `age`. Nevertheless, step function approaches are very popular in biostatistics and epidemiology, among other disciplines. For example, 5-year age groups are often used to define the bins.

7.3 Basis Functions

Polynomial and piecewise-constant regression models are in fact special cases of a *basis function* approach. The idea is to have at hand a family of functions or transformations that can be applied to a variable X: $b_1(X), b_2(X), \ldots, b_K(X)$. Instead of fitting a linear model in X, we fit the model

basis function

$$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \beta_3 b_3(x_i) + \cdots + \beta_K b_K(x_i) + \epsilon_i. \tag{7.7}$$

Note that the basis functions $b_1(\cdot), b_2(\cdot), \ldots, b_K(\cdot)$ are fixed and known. (In other words, we choose the functions ahead of time.) For polynomial regression, the basis functions are $b_j(x_i) = x_i^j$, and for piecewise constant functions they are $b_j(x_i) = I(c_j \leq x_i < c_{j+1})$. We can think of (7.7) as a standard linear model with predictors $b_1(x_i), b_2(x_i), \ldots, b_K(x_i)$. Hence, we can use least squares to estimate the unknown regression coefficients in (7.7). Importantly, this means that all of the inference tools for linear models that are discussed in Chapter 3, such as standard errors for the coefficient estimates and F-statistics for the model's overall significance, are available in this setting.

Thus far we have considered the use of polynomial functions and piecewise constant functions for our basis functions; however, many alternatives are possible. For instance, we can use wavelets or Fourier series to construct basis functions. In the next section, we investigate a very common choice for a basis function: *regression splines*.

regression spline

7.4 Regression Splines

Now we discuss a flexible class of basis functions that extends upon the polynomial regression and piecewise constant regression approaches that we have just seen.

7.4.1 Piecewise Polynomials

Instead of fitting a high-degree polynomial over the entire range of X, *piecewise polynomial regression* involves fitting separate low-degree polynomials over different regions of X. For example, a piecewise cubic polynomial works by fitting a cubic regression model of the form

piecewise polynomial regression

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \epsilon_i, \tag{7.8}$$

where the coefficients β_0, β_1, β_2, and β_3 differ in different parts of the range of X. The points where the coefficients change are called *knots*.

knot

For example, a piecewise cubic with no knots is just a standard cubic polynomial, as in (7.1) with $d = 3$. A piecewise cubic polynomial with a single knot at a point c takes the form

$$y_i = \begin{cases} \beta_{01} + \beta_{11} x_i + \beta_{21} x_i^2 + \beta_{31} x_i^3 + \epsilon_i & \text{if } x_i < c \\ \beta_{02} + \beta_{12} x_i + \beta_{22} x_i^2 + \beta_{32} x_i^3 + \epsilon_i & \text{if } x_i \geq c. \end{cases}$$

In other words, we fit two different polynomial functions to the data, one on the subset of the observations with $x_i < c$, and one on the subset of the observations with $x_i \geq c$. The first polynomial function has coefficients $\beta_{01}, \beta_{11}, \beta_{21}$, and β_{31}, and the second has coefficients $\beta_{02}, \beta_{12}, \beta_{22}$, and β_{32}. Each of these polynomial functions can be fit using least squares applied to simple functions of the original predictor.

Using more knots leads to a more flexible piecewise polynomial. In general, if we place K different knots throughout the range of X, then we will end up fitting $K + 1$ different cubic polynomials. Note that we do not need to use a cubic polynomial. For example, we can instead fit piecewise linear functions. In fact, our piecewise constant functions of Section 7.2 are piecewise polynomials of degree 0!

The top left panel of Figure 7.3 shows a piecewise cubic polynomial fit to a subset of the `Wage` data, with a single knot at `age=50`. We immediately see a problem: the function is discontinuous and looks ridiculous! Since each polynomial has four parameters, we are using a total of eight *degrees of freedom* in fitting this piecewise polynomial model.

degrees of freedom

7.4.2 Constraints and Splines

The top left panel of Figure 7.3 looks wrong because the fitted curve is just too flexible. To remedy this problem, we can fit a piecewise polynomial

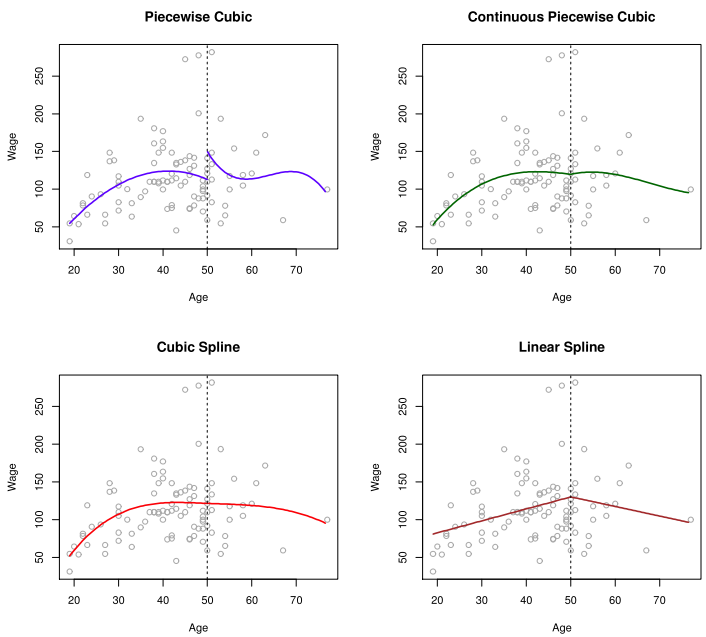

FIGURE 7.3. *Various piecewise polynomials are fit to a subset of the* `Wage` *data, with a knot at* `age=50`. Top Left: *The cubic polynomials are unconstrained.* Top Right: *The cubic polynomials are constrained to be continuous at* `age=50`. Bottom Left: *The cubic polynomials are constrained to be continuous, and to have continuous first and second derivatives.* Bottom Right: *A linear spline is shown, which is constrained to be continuous.*

under the *constraint* that the fitted curve must be continuous. In other words, there cannot be a jump when `age=50`. The top right plot in Figure 7.3 shows the resulting fit. This looks better than the top left plot, but the V-shaped join looks unnatural.

In the lower left plot, we have added two additional constraints: now both the first and second *derivatives* of the piecewise polynomials are continuous at `age=50`. In other words, we are requiring that the piecewise polynomial be not only continuous when `age=50`, but also very *smooth*. Each constraint that we impose on the piecewise cubic polynomials effectively frees up one degree of freedom, by reducing the complexity of the resulting piecewise polynomial fit. So in the top left plot, we are using eight degrees of freedom, but in the bottom left plot we imposed three constraints (continuity, continuity of the first derivative, and continuity of the second derivative)

derivative

and so are left with five degrees of freedom. The curve in the bottom left plot is called a *cubic spline*.[3] In general, a cubic spline with K knots uses a total of $4 + K$ degrees of freedom.

cubic spline

In Figure 7.3, the lower right plot is a *linear spline*, which is continuous at `age=50`. The general definition of a degree-d spline is that it is a piecewise degree-d polynomial, with continuity in derivatives up to degree $d - 1$ at each knot. Therefore, a linear spline is obtained by fitting a line in each region of the predictor space defined by the knots, requiring continuity at each knot.

linear spline

In Figure 7.3, there is a single knot at `age=50`. Of course, we could add more knots, and impose continuity at each.

7.4.3 The Spline Basis Representation

The regression splines that we just saw in the previous section may have seemed somewhat complex: how can we fit a piecewise degree-d polynomial under the constraint that it (and possibly its first $d - 1$ derivatives) be continuous? It turns out that we can use the basis model (7.7) to represent a regression spline. A cubic spline with K knots can be modeled as

$$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \cdots + \beta_{K+3} b_{K+3}(x_i) + \epsilon_i, \tag{7.9}$$

for an appropriate choice of basis functions $b_1, b_2, \ldots, b_{K+3}$. The model (7.9) can then be fit using least squares.

Just as there were several ways to represent polynomials, there are also many equivalent ways to represent cubic splines using different choices of basis functions in (7.9). The most direct way to represent a cubic spline using (7.9) is to start off with a basis for a cubic polynomial—namely, x, x^2, and x^3—and then add one *truncated power basis* function per knot. A truncated power basis function is defined as

truncated power basis

$$h(x, \xi) = (x - \xi)_+^3 = \begin{cases} (x - \xi)^3 & \text{if } x > \xi \\ 0 & \text{otherwise,} \end{cases} \tag{7.10}$$

where ξ is the knot. One can show that adding a term of the form $\beta_4 h(x, \xi)$ to the model (7.8) for a cubic polynomial will lead to a discontinuity in only the third derivative at ξ; the function will remain continuous, with continuous first and second derivatives, at each of the knots.

In other words, in order to fit a cubic spline to a data set with K knots, we perform least squares regression with an intercept and $3 + K$ predictors, of the form $X, X^2, X^3, h(X, \xi_1), h(X, \xi_2), \ldots, h(X, \xi_K)$, where $\xi_1, \ldots, \xi_K$ are the knots. This amounts to estimating a total of $K + 4$ regression coefficients; for this reason, fitting a cubic spline with K knots uses $K+4$ degrees of freedom.

[3] Cubic splines are popular because most human eyes cannot detect the discontinuity at the knots.

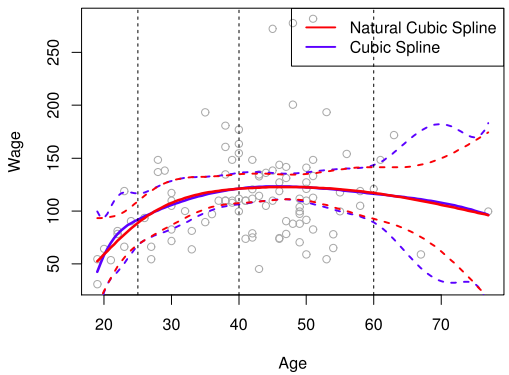

FIGURE 7.4. *A cubic spline and a natural cubic spline, with three knots, fit to a subset of the* `Wage` *data. The dashed lines denote the knot locations.*

Unfortunately, splines can have high variance at the outer range of the predictors—that is, when X takes on either a very small or very large value. Figure 7.4 shows a fit to the `Wage` data with three knots. We see that the confidence bands in the boundary region appear fairly wild. A *natural spline* is a regression spline with additional *boundary constraints*: the function is required to be linear at the boundary (in the region where X is smaller than the smallest knot, or larger than the largest knot). This additional constraint means that natural splines generally produce more stable estimates at the boundaries. In Figure 7.4, a natural cubic spline is also displayed as a red line. Note that the corresponding confidence intervals are narrower.

natural spline

7.4.4 Choosing the Number and Locations of the Knots

When we fit a spline, where should we place the knots? The regression spline is most flexible in regions that contain a lot of knots, because in those regions the polynomial coefficients can change rapidly. Hence, one option is to place more knots in places where we feel the function might vary most rapidly, and to place fewer knots where it seems more stable. While this option can work well, in practice it is common to place knots in a uniform fashion. One way to do this is to specify the desired degrees of freedom, and then have the software automatically place the corresponding number of knots at uniform quantiles of the data.

Figure 7.5 shows an example on the `Wage` data. As in Figure 7.4, we have fit a natural cubic spline with three knots, except this time the knot locations were chosen automatically as the 25th, 50th, and 75th percentiles of `age`. This was specified by requesting four degrees of freedom. The argument by which four degrees of freedom leads to three interior knots is somewhat technical.[4]

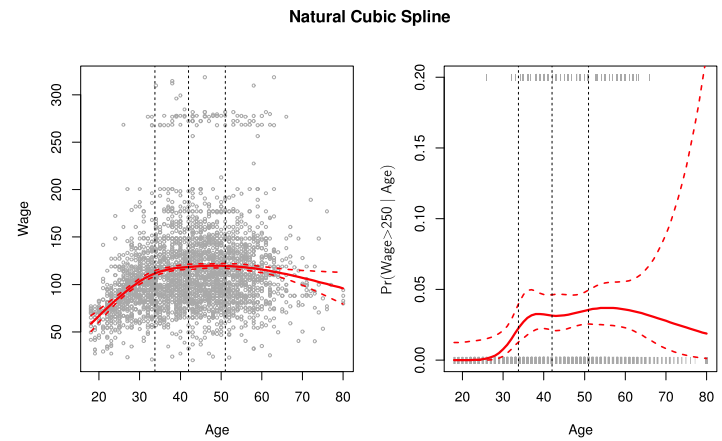

FIGURE 7.5. *A natural cubic spline function with four degrees of freedom is fit to the* `Wage` *data.* Left: *A spline is fit to* `wage` *(in thousands of dollars) as a function of* `age`. Right: *Logistic regression is used to model the binary event* `wage>250` *as a function of* `age`. *The fitted posterior probability of* `wage` *exceeding* $250,000 *is shown. The dashed lines denote the knot locations.*

How many knots should we use, or equivalently how many degrees of freedom should our spline contain? One option is to try out different numbers of knots and see which produces the best looking curve. A somewhat more objective approach is to use cross-validation, as discussed in Chapters 5 and 6. With this method, we remove a portion of the data (say 10 %), fit a spline with a certain number of knots to the remaining data, and then use the spline to make predictions for the held-out portion. We repeat this process multiple times until each observation has been left out once, and then compute the overall cross-validated RSS. This procedure can be repeated for different numbers of knots K. Then the value of K giving the smallest RSS is chosen.

Figure 7.6 shows ten-fold cross-validated mean squared errors for splines with various degrees of freedom fit to the `Wage` data. The left-hand panel

[4] There are actually five knots, including the two boundary knots. A cubic spline with five knots has nine degrees of freedom. But natural cubic splines have two additional *natural* constraints at each boundary to enforce linearity, resulting in $9 - 4 = 5$ degrees of freedom. Since this includes a constant, which is absorbed in the intercept, we count it as four degrees of freedom.

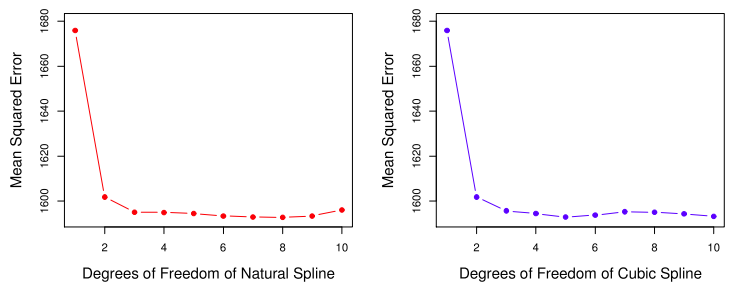

FIGURE 7.6. *Ten-fold cross-validated mean squared errors for selecting the degrees of freedom when fitting splines to the* `Wage` *data. The response is* `wage` *and the predictor* `age`. Left: *A natural cubic spline.* Right: *A cubic spline.*

corresponds to a natural cubic spline and the right-hand panel to a cubic spline. The two methods produce almost identical results, with clear evidence that a one-degree fit (a linear regression) is not adequate. Both curves flatten out quickly, and it seems that three degrees of freedom for the natural spline and four degrees of freedom for the cubic spline are quite adequate.

In Section 7.7 we fit additive spline models simultaneously on several variables at a time. This could potentially require the selection of degrees of freedom for each variable. In cases like this we typically adopt a more pragmatic approach and set the degrees of freedom to a fixed number, say four, for all terms.

7.4.5 Comparison to Polynomial Regression

Figure 7.7 compares a natural cubic spline with 15 degrees of freedom to a degree-15 polynomial on the `Wage` data set. The extra flexibility in the polynomial produces undesirable results at the boundaries, while the natural cubic spline still provides a reasonable fit to the data. Regression splines often give superior results to polynomial regression. This is because unlike polynomials, which must use a high degree (exponent in the highest monomial term, e.g. X^{15}) to produce flexible fits, splines introduce flexibility by increasing the number of knots but keeping the degree fixed. Generally, this approach produces more stable estimates. Splines also allow us to place more knots, and hence flexibility, over regions where the function f seems to be changing rapidly, and fewer knots where f appears more stable.

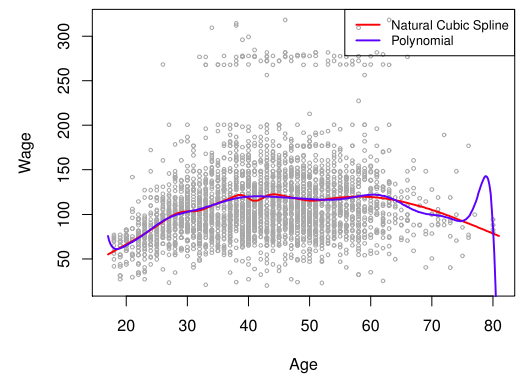

FIGURE 7.7. *On the* `Wage` *data set, a natural cubic spline with 15 degrees of freedom is compared to a degree-15 polynomial. Polynomials can show wild behavior, especially near the tails.*

7.5 Smoothing Splines

In the last section we discussed regression splines, which we create by specifying a set of knots, producing a sequence of basis functions, and then using least squares to estimate the spline coefficients. We now introduce a somewhat different approach that also produces a spline.

7.5.1 An Overview of Smoothing Splines

In fitting a smooth curve to a set of data, what we really want to do is find some function, say $g(x)$, that fits the observed data well: that is, we want RSS $= \sum_{i=1}^{n}(y_i - g(x_i))^2$ to be small. However, there is a problem with this approach. If we don't put any constraints on $g(x_i)$, then we can always make RSS zero simply by choosing g such that it *interpolates* all of the y_i. Such a function would woefully overfit the data—it would be far too flexible. What we really want is a function g that makes RSS small, but that is also *smooth*.

How might we ensure that g is smooth? There are a number of ways to do this. A natural approach is to find the function g that minimizes

$$\sum_{i=1}^{n}(y_i - g(x_i))^2 + \lambda \int g''(t)^2 dt \tag{7.11}$$

where λ is a nonnegative *tuning parameter*. The function g that minimizes (7.11) is known as a *smoothing spline*.

smoothing spline

What does (7.11) mean? Equation 7.11 takes the "Loss+Penalty" formulation that we encounter in the context of ridge regression and the lasso

in Chapter 6. The term $\sum_{i=1}^{n}(y_i - g(x_i))^2$ is a *loss function* that encourages g to fit the data well, and the term $\lambda \int g''(t)^2 dt$ is a *penalty term* that penalizes the variability in g. The notation $g''(t)$ indicates the second derivative of the function g. The first derivative $g'(t)$ measures the slope of a function at t, and the second derivative corresponds to the amount by which the slope is changing. Hence, broadly speaking, the second derivative of a function is a measure of its *roughness*: it is large in absolute value if $g(t)$ is very wiggly near t, and it is close to zero otherwise. (The second derivative of a straight line is zero; note that a line is perfectly smooth.) The $\int$ notation is an *integral*, which we can think of as a summation over the range of t. In other words, $\int g''(t)^2 dt$ is simply a measure of the total change in the function $g'(t)$, over its entire range. If g is very smooth, then $g'(t)$ will be close to constant and $\int g''(t)^2 dt$ will take on a small value. Conversely, if g is jumpy and variable then $g'(t)$ will vary significantly and $\int g''(t)^2 dt$ will take on a large value. Therefore, in (7.11), $\lambda \int g''(t)^2 dt$ encourages g to be smooth. The larger the value of λ, the smoother g will be.

loss function

When $\lambda = 0$, then the penalty term in (7.11) has no effect, and so the function g will be very jumpy and will exactly interpolate the training observations. When $\lambda \to \infty$, g will be perfectly smooth—it will just be a straight line that passes as closely as possible to the training points. In fact, in this case, g will be the linear least squares line, since the loss function in (7.11) amounts to minimizing the residual sum of squares. For an intermediate value of λ, g will approximate the training observations but will be somewhat smooth. We see that λ controls the bias-variance trade-off of the smoothing spline.

The function $g(x)$ that minimizes (7.11) can be shown to have some special properties: it is a piecewise cubic polynomial with knots at the unique values of $x_1, \ldots, x_n$, and continuous first and second derivatives at each knot. Furthermore, it is linear in the region outside of the extreme knots. In other words, *the function $g(x)$ that minimizes (7.11) is a natural cubic spline with knots at $x_1, \ldots, x_n$!* However, it is not the same natural cubic spline that one would get if one applied the basis function approach described in Section 7.4.3 with knots at $x_1, \ldots, x_n$—rather, it is a *shrunken* version of such a natural cubic spline, where the value of the tuning parameter λ in (7.11) controls the level of shrinkage.

7.5.2 Choosing the Smoothing Parameter λ

We have seen that a smoothing spline is simply a natural cubic spline with knots at every unique value of x_i. It might seem that a smoothing spline will have far too many degrees of freedom, since a knot at each data point allows a great deal of flexibility. But the tuning parameter λ controls the roughness of the smoothing spline, and hence the *effective degrees of freedom*. It is possible to show that as λ increases from 0 to ∞, the effective degrees of freedom, which we write df_λ, decrease from n to 2.

effective degrees of freedom

In the context of smoothing splines, why do we discuss *effective* degrees of freedom instead of degrees of freedom? Usually degrees of freedom refer to the number of free parameters, such as the number of coefficients fit in a polynomial or cubic spline. Although a smoothing spline has n parameters and hence n nominal degrees of freedom, these n parameters are heavily constrained or shrunk down. Hence df_λ is a measure of the flexibility of the smoothing spline—the higher it is, the more flexible (and the lower-bias but higher-variance) the smoothing spline. The definition of effective degrees of freedom is somewhat technical. We can write

$$\hat{\mathbf{g}}_\lambda = \mathbf{S}_\lambda \mathbf{y}, \tag{7.12}$$

where $\hat{\mathbf{g}}_\lambda$ is the solution to (7.11) for a particular choice of λ—that is, it is an n-vector containing the fitted values of the smoothing spline at the training points $x_1, \ldots, x_n$. Equation 7.12 indicates that the vector of fitted values when applying a smoothing spline to the data can be written as a $n \times n$ matrix $\mathbf{S}_\lambda$ (for which there is a formula) times the response vector $\mathbf{y}$. Then the effective degrees of freedom is defined to be

$$df_\lambda = \sum_{i=1}^{n} \{\mathbf{S}_\lambda\}_{ii}, \tag{7.13}$$

the sum of the diagonal elements of the matrix $\mathbf{S}_\lambda$.

In fitting a smoothing spline, we do not need to select the number or location of the knots—there will be a knot at each training observation, $x_1, \ldots, x_n$. Instead, we have another problem: we need to choose the value of λ. It should come as no surprise that one possible solution to this problem is cross-validation. In other words, we can find the value of λ that makes the cross-validated RSS as small as possible. It turns out that the *leave-one-out* cross-validation error (LOOCV) can be computed very efficiently for smoothing splines, with essentially the same cost as computing a single fit, using the following formula:

$$\mathrm{RSS}_{cv}(\lambda) = \sum_{i=1}^{n} (y_i - \hat{g}_\lambda^{(-i)}(x_i))^2 = \sum_{i=1}^{n} \left[\frac{y_i - \hat{g}_\lambda(x_i)}{1 - \{\mathbf{S}_\lambda\}_{ii}} \right]^2.$$

The notation $\hat{g}_\lambda^{(-i)}(x_i)$ indicates the fitted value for this smoothing spline evaluated at x_i, where the fit uses all of the training observations except for the ith observation (x_i, y_i). In contrast, $\hat{g}_\lambda(x_i)$ indicates the smoothing spline function fit to all of the training observations and evaluated at x_i. This remarkable formula says that we can compute each of these *leave-one-out* fits using only $\hat{g}_\lambda$, the original fit to *all* of the data![5] We have

[5] The exact formulas for computing $\hat{g}(x_i)$ and $\mathbf{S}_\lambda$ are very technical; however, efficient algorithms are available for computing these quantities.

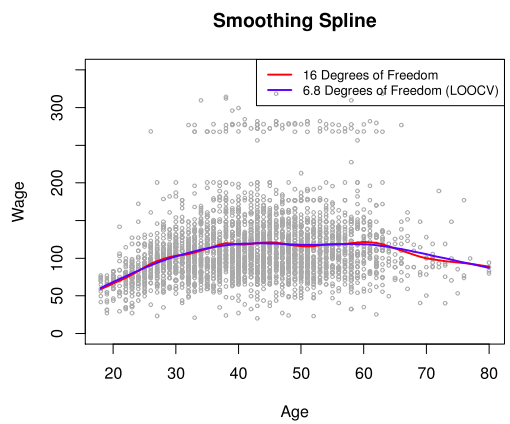

FIGURE 7.8. *Smoothing spline fits to the* `Wage` *data. The red curve results from specifying* 16 *effective degrees of freedom. For the blue curve,* λ *was found automatically by leave-one-out cross-validation, which resulted in* 6.8 *effective degrees of freedom.*

a very similar formula (5.2) on page 202 in Chapter 5 for least squares linear regression. Using (5.2), we can very quickly perform LOOCV for the regression splines discussed earlier in this chapter, as well as for least squares regression using arbitrary basis functions.

Figure 7.8 shows the results from fitting a smoothing spline to the `Wage` data. The red curve indicates the fit obtained from pre-specifying that we would like a smoothing spline with 16 effective degrees of freedom. The blue curve is the smoothing spline obtained when λ is chosen using LOOCV; in this case, the value of λ chosen results in 6.8 effective degrees of freedom (computed using (7.13)). For this data, there is little discernible difference between the two smoothing splines, beyond the fact that the one with 16 degrees of freedom seems slightly wigglier. Since there is little difference between the two fits, the smoothing spline fit with 6.8 degrees of freedom is preferable, since in general simpler models are better unless the data provides evidence in support of a more complex model.

7.6 Local Regression

Local regression is a different approach for fitting flexible non-linear functions, which involves computing the fit at a target point x_0 using only the nearby training observations. Figure 7.9 illustrates the idea on some simulated data, with one target point near 0.4, and another near the boundary

local regression

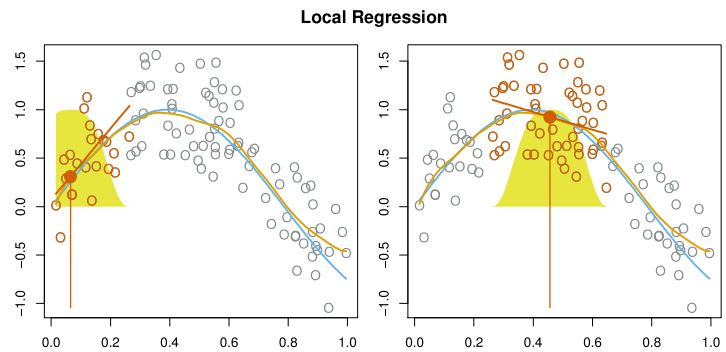

FIGURE 7.9. *Local regression illustrated on some simulated data, where the blue curve represents* $f(x)$ *from which the data were generated, and the light orange curve corresponds to the local regression estimate* $\hat{f}(x)$*. The orange colored points are local to the target point* x_0*, represented by the orange vertical line. The yellow bell-shape superimposed on the plot indicates weights assigned to each point, decreasing to zero with distance from the target point. The fit* $\hat{f}(x_0)$ *at* x_0 *is obtained by fitting a weighted linear regression (orange line segment), and using the fitted value at* x_0 *(orange solid dot) as the estimate* $\hat{f}(x_0)$*.*

at 0.05. In this figure the blue line represents the function $f(x)$ from which the data were generated, and the light orange line corresponds to the local regression estimate $\hat{f}(x)$. Local regression is described in Algorithm 7.1.

Note that in Step 3 of Algorithm 7.1, the weights K_{i0} will differ for each value of x_0. In other words, in order to obtain the local regression fit at a new point, we need to fit a new weighted least squares regression model by minimizing (7.14) for a new set of weights. Local regression is sometimes referred to as a *memory-based* procedure, because like nearest-neighbors, we need all the training data each time we wish to compute a prediction. We will avoid getting into the technical details of local regression here—there are books written on the topic.

In order to perform local regression, there are a number of choices to be made, such as how to define the weighting function K, and whether to fit a linear, constant, or quadratic regression in Step 3. (Equation 7.14 corresponds to a linear regression.) While all of these choices make some difference, the most important choice is the *span* s, which is the proportion of points used to compute the local regression at x_0, as defined in Step 1 above. The span plays a role like that of the tuning parameter λ in smoothing splines: it controls the flexibility of the non-linear fit. The smaller the value of s, the more *local* and wiggly will be our fit; alternatively, a very large value of s will lead to a global fit to the data using all of the training observations. We can again use cross-validation to choose s, or we can

Algorithm 7.1 *Local Regression At $X = x_0$*

1. Gather the fraction $s = k/n$ of training points whose x_i are closest to x_0.
2. Assign a weight $K_{i0} = K(x_i, x_0)$ to each point in this neighborhood, so that the point furthest from x_0 has weight zero, and the closest has the highest weight. All but these k nearest neighbors get weight zero.
3. Fit a *weighted least squares regression* of the y_i on the x_i using the aforementioned weights, by finding $\hat{\beta}_0$ and $\hat{\beta}_1$ that minimize
$$\sum_{i=1}^{n} K_{i0}(y_i - \beta_0 - \beta_1 x_i)^2. \tag{7.14}$$
4. The fitted value at x_0 is given by $\hat{f}(x_0) = \hat{\beta}_0 + \hat{\beta}_1 x_0$.

specify it directly. Figure 7.10 displays local linear regression fits on the `Wage` data, using two values of s: 0.7 and 0.2. As expected, the fit obtained using $s = 0.7$ is smoother than that obtained using $s = 0.2$.

The idea of local regression can be generalized in many different ways. In a setting with multiple features $X_1, X_2, \ldots, X_p$, one very useful generalization involves fitting a multiple linear regression model that is global in some variables, but local in another, such as time. Such *varying coefficient models* are a useful way of adapting a model to the most recently gathered data. Local regression also generalizes very naturally when we want to fit models that are local in a pair of variables X_1 and X_2, rather than one. We can simply use two-dimensional neighborhoods, and fit bivariate linear regression models using the observations that are near each target point in two-dimensional space. Theoretically the same approach can be implemented in higher dimensions, using linear regressions fit to p-dimensional neighborhoods. However, local regression can perform poorly if p is much larger than about 3 or 4 because there will generally be very few training observations close to x_0. Nearest-neighbors regression, discussed in Chapter 3, suffers from a similar problem in high dimensions.

varying coefficient model

7.7 Generalized Additive Models

In Sections 7.1–7.6, we present a number of approaches for flexibly predicting a response Y on the basis of a single predictor X. These approaches can be seen as extensions of simple linear regression. Here we explore the prob-

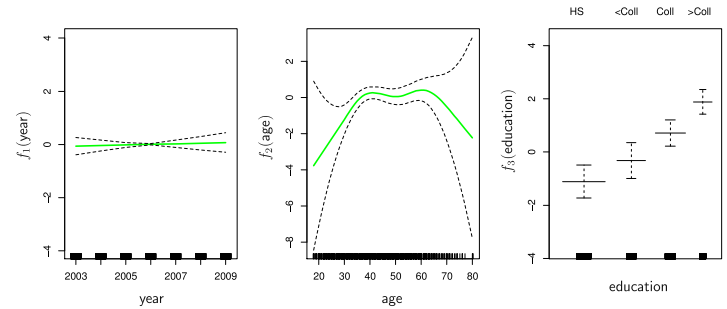

FIGURE 7.14. *The same model is fit as in Figure 7.13, this time excluding the observations for which* `education` *is* `<HS`*. Now we see that increased education tends to be associated with higher salaries.*

7.8.1 Polynomial Regression and Step Functions

We now examine how Figure 7.1 was produced. We first fit the model using the following command:

```
> fit <- lm(wage ~ poly(age, 4), data = Wage)
> coef(summary(fit))
              Estimate Std. Error t value Pr(>|t|)
(Intercept)    111.704      0.729  153.28   <2e-16
poly(age, 4)1  447.068     39.915   11.20   <2e-16
poly(age, 4)2 -478.316     39.915  -11.98   <2e-16
poly(age, 4)3  125.522     39.915    3.14   0.0017
poly(age, 4)4  -77.911     39.915   -1.95   0.0510
```

This syntax fits a linear model, using the `lm()` function, in order to predict `wage` using a fourth-degree polynomial in `age`: `poly(age, 4)`. The `poly()` command allows us to avoid having to write out a long formula with powers of `age`. The function returns a matrix whose columns are a basis of *orthogonal polynomials*, which essentially means that each column is a linear combination of the variables `age`, `age^2`, `age^3` and `age^4`.

orthogonal polynomial

However, we can also use `poly()` to obtain `age`, `age^2`, `age^3` and `age^4` directly, if we prefer. We can do this by using the `raw = TRUE` argument to the `poly()` function. Later we see that this does not affect the model in a meaningful way—though the choice of basis clearly affects the coefficient estimates, it does not affect the fitted values obtained.

```
> fit2 <- lm(wage ~ poly(age, 4, raw = T), data = Wage)
> coef(summary(fit2))
                        Estimate Std. Error t value Pr(>|t|)
(Intercept)             -1.84e+02   6.00e+01   -3.07 0.002180
poly(age, 4, raw = T)1   2.12e+01   5.89e+00    3.61 0.000312
poly(age, 4, raw = T)2  -5.64e-01   2.06e-01   -2.74 0.006261
poly(age, 4, raw = T)3   6.81e-03   3.07e-03    2.22 0.026398
poly(age, 4, raw = T)4  -3.20e-05   1.64e-05   -1.95 0.051039
```

There are several other equivalent ways of fitting this model, which showcase the flexibility of the formula language in R. For example

```
> fit2a <- lm(wage ~ age + I(age^2) + I(age^3) + I(age^4),
    data = Wage)
> coef(fit2a)
(Intercept)         age    I(age^2)    I(age^3)    I(age^4)
  -1.84e+02    2.12e+01   -5.64e-01    6.81e-03   -3.20e-05
```

This simply creates the polynomial basis functions on the fly, taking care to protect terms like `age^2` via the *wrapper* function `I()` (the ^ symbol has a special meaning in formulas).

wrapper

```
> fit2b <- lm(wage ~ cbind(age, age^2, age^3, age^4),
    data = Wage)
```

This does the same more compactly, using the `cbind()` function for building a matrix from a collection of vectors; any function call such as `cbind()` inside a formula also serves as a wrapper.

We now create a grid of values for `age` at which we want predictions, and then call the generic `predict()` function, specifying that we want standard errors as well.

```
> agelims <- range(age)
> age.grid <- seq(from = agelims[1], to = agelims[2])
> preds <- predict(fit, newdata = list(age = age.grid),
    se = TRUE)
> se.bands <- cbind(preds$fit + 2 * preds$se.fit,
    preds$fit - 2 * preds$se.fit)
```

Finally, we plot the data and add the fit from the degree-4 polynomial.

```
> par(mfrow = c(1, 2), mar = c(4.5, 4.5, 1, 1),
    oma = c(0, 0, 4, 0))
> plot(age, wage, xlim = agelims, cex = .5, col = "darkgrey")
> title("Degree-4 Polynomial", outer = T)
> lines(age.grid, preds$fit, lwd = 2, col = "blue")
> matlines(age.grid, se.bands, lwd = 1, col = "blue", lty = 3)
```

Here the `mar` and `oma` arguments to `par()` allow us to control the margins of the plot, and the `title()` function creates a figure title that spans both subplots.

`title()`

We mentioned earlier that whether or not an orthogonal set of basis functions is produced in the `poly()` function will not affect the model obtained in a meaningful way. What do we mean by this? The fitted values obtained in either case are identical:

```
> preds2 <- predict(fit2, newdata = list(age = age.grid),
    se = TRUE)
> max(abs(preds$fit - preds2$fit))
[1] 7.82e-11
```

In performing a polynomial regression we must decide on the degree of the polynomial to use. One way to do this is by using hypothesis tests. We

now fit models ranging from linear to a degree-5 polynomial and seek to determine the simplest model which is sufficient to explain the relationship between `wage` and `age`. We use the `anova()` function, which performs an *analysis of variance* (ANOVA, using an F-test) in order to test the null hypothesis that a model $\mathcal{M}_1$ is sufficient to explain the data against the alternative hypothesis that a more complex model $\mathcal{M}_2$ is required. In order to use the `anova()` function, $\mathcal{M}_1$ and $\mathcal{M}_2$ must be *nested* models: the predictors in $\mathcal{M}_1$ must be a subset of the predictors in $\mathcal{M}_2$. In this case, we fit five different models and sequentially compare the simpler model to the more complex model.

`anova()`

analysis of variance

```
> fit.1 <- lm(wage ~ age, data = Wage)
> fit.2 <- lm(wage ~ poly(age, 2), data = Wage)
> fit.3 <- lm(wage ~ poly(age, 3), data = Wage)
> fit.4 <- lm(wage ~ poly(age, 4), data = Wage)
> fit.5 <- lm(wage ~ poly(age, 5), data = Wage)
> anova(fit.1, fit.2, fit.3, fit.4, fit.5)
Analysis of Variance Table

Model 1: wage ~ age
Model 2: wage ~ poly(age, 2)
Model 3: wage ~ poly(age, 3)
Model 4: wage ~ poly(age, 4)
Model 5: wage ~ poly(age, 5)
  Res.Df     RSS Df Sum of Sq      F Pr(>F)
1   2998 5022216
2   2997 4793430  1    228786 143.59 <2e-16 ***
3   2996 4777674  1     15756   9.89 0.0017 **
4   2995 4771604  1      6070   3.81 0.0510 .
5   2994 4770322  1      1283   0.80 0.3697
---
Signif. codes:  0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1
```

The p-value comparing the linear `Model 1` to the quadratic `Model 2` is essentially zero ($<10^{-15}$), indicating that a linear fit is not sufficient. Similarly the p-value comparing the quadratic `Model 2` to the cubic `Model 3` is very low (0.0017), so the quadratic fit is also insufficient. The p-value comparing the cubic and degree-4 polynomials, `Model 3` and `Model 4`, is approximately 5 % while the degree-5 polynomial `Model 5` seems unnecessary because its p-value is 0.37. Hence, either a cubic or a quartic polynomial appear to provide a reasonable fit to the data, but lower- or higher-order models are not justified.

In this case, instead of using the `anova()` function, we could have obtained these p-values more succinctly by exploiting the fact that `poly()` creates orthogonal polynomials.

```
> coef(summary(fit.5))
              Estimate Std. Error  t value  Pr(>|t|)
(Intercept)     111.70     0.7288 153.2780 0.000e+00
poly(age, 5)1   447.07    39.9161  11.2002 1.491e-28
```

7.8.2 Splines

In order to fit regression splines in R, we use the splines library. In Section 7.4, we saw that regression splines can be fit by constructing an appropriate matrix of basis functions. The bs() function generates the entire matrix of basis functions for splines with the specified set of knots. By default, cubic splines are produced. Fitting wage to age using a regression spline is simple: bs()

```
> library(splines)
> fit <- lm(wage ~ bs(age, knots = c(25, 40, 60)), data = Wage)
> pred <- predict(fit, newdata = list(age = age.grid), se = T)
> plot(age, wage, col = "gray")
> lines(age.grid, pred$fit, lwd = 2)
> lines(age.grid, pred$fit + 2 * pred$se, lty = "dashed")
> lines(age.grid, pred$fit - 2 * pred$se, lty = "dashed")
```

Here we have prespecified knots at ages 25, 40, and 60. This produces a spline with six basis functions. (Recall that a cubic spline with three knots has seven degrees of freedom; these degrees of freedom are used up by an intercept, plus six basis functions.) We could also use the df option to produce a spline with knots at uniform quantiles of the data.

```
> dim(bs(age, knots = c(25, 40, 60)))
[1] 3000    6
> dim(bs(age, df = 6))
[1] 3000    6
> attr(bs(age, df = 6), "knots")
 25%  50%  75%
33.8 42.0 51.0
```

In this case R chooses knots at ages 33.8, 42.0, and 51.0, which correspond to the 25th, 50th, and 75th percentiles of age. The function bs() also has a degree argument, so we can fit splines of any degree, rather than the default degree of 3 (which yields a cubic spline).

In order to instead fit a natural spline, we use the ns() function. Here we fit a natural spline with four degrees of freedom. ns()

```
> fit2 <- lm(wage ~ ns(age, df = 4), data = Wage)
> pred2 <- predict(fit2, newdata = list(age = age.grid),
    se = T)
> lines(age.grid, pred2$fit, col = "red", lwd = 2)
```

As with the bs() function, we could instead specify the knots directly using the knots option.

In order to fit a smoothing spline, we use the smooth.spline() function. Figure 7.8 was produced with the following code: smooth.spline()

```
> plot(age, wage, xlim = agelims, cex = .5, col = "darkgrey")
> title("Smoothing Spline")
> fit <- smooth.spline(age, wage, df = 16)
> fit2 <- smooth.spline(age, wage, cv = TRUE)
> fit2$df
```



```
[1] 6.8
> lines(fit, col = "red", lwd = 2)
> lines(fit2, col = "blue", lwd = 2)
> legend("topright", legend = c("16 DF", "6.8 DF"),
    col = c("red", "blue"), lty = 1, lwd = 2, cex = .8)
```

Notice that in the first call to smooth.spline(), we specified df = 16. The function then determines which value of λ leads to 16 degrees of freedom. In the second call to smooth.spline(), we select the smoothness level by cross-validation; this results in a value of λ that yields 6.8 degrees of freedom.

In order to perform local regression, we use the loess() function. loess()

```
> plot(age, wage, xlim = agelims, cex = .5, col = "darkgrey")
> title("Local Regression")
> fit <- loess(wage ~ age, span = .2, data = Wage)
> fit2 <- loess(wage ~ age, span = .5, data = Wage)
> lines(age.grid, predict(fit, data.frame(age = age.grid)),
    col = "red", lwd = 2)
> lines(age.grid, predict(fit2, data.frame(age = age.grid)),
    col = "blue", lwd = 2)
> legend("topright", legend = c("Span = 0.2", "Span = 0.5"),
    col = c("red", "blue"), lty = 1, lwd = 2, cex = .8)
```

Here we have performed local linear regression using spans of 0.2 and 0.5: that is, each neighborhood consists of 20 % or 50 % of the observations. The larger the span, the smoother the fit. The locfit library can also be used for fitting local regression models in R.

7.8.3 GAMs

We now fit a GAM to predict wage using natural spline functions of year and age, treating education as a qualitative predictor, as in (7.16). Since this is just a big linear regression model using an appropriate choice of basis functions, we can simply do this using the lm() function.

```
> gam1 <- lm(wage ~ ns(year, 4) + ns(age, 5) + education,
    data = Wage)
```

We now fit the model (7.16) using smoothing splines rather than natural splines. In order to fit more general sorts of GAMs, using smoothing splines or other components that cannot be expressed in terms of basis functions and then fit using least squares regression, we will need to use the gam library in R.

The s() function, which is part of the gam library, is used to indicate that we would like to use a smoothing spline. We specify that the function of year should have 4 degrees of freedom, and that the function of age will have 5 degrees of freedom. Since education is qualitative, we leave it as is, and it is converted into four dummy variables. We use the gam() function in order to fit a GAM using these components. All of the terms in (7.16) are fit simultaneously, taking each other into account to explain the response. s() gam()