Ch. 7. Moving Beyond Linearity

Linear models are relatively simple to describe and implement, and have advantages over other approaches in terms of **interpretation** and **inference**.
However, standard linear regression can have significant limitations in terms of **predictive power**. This is because the linearity assumption is almost always an approximation, and **sometimes a poor one**.

In chapter 6 we saw an improvement of the linear model by reducing the complexity → reducing the variance of the estimates.

In this chapter we relax the linearity assumption while still attempting to maintain as much interpretability as possible.

In the following sections we present a number of approaches for modeling the relationship between the response Y and a single predictor X in a flexible way. In the last section (GAMs) we show that these approaches can be seamlessly integrated in order to model a response Y as a function of several predictors X_1, …, X_p.

Polynomial Regression

Historically, the standard way to extend linear regression to settings in which the relationship between the predictors and the response is nonlinear has been to replace the standard linear model with a polynomial function:

$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \ldots + \beta_d x_i^d + \epsilon_i$
This approach is known as *polynomial regression*. For large enough degree *d*, a polynomial regression allows us to produce an extremely non-linear curve. The coefficients can be easily estimated using least squares because it's just a linear model with predictors.
Generally speaking it's unusual to use d greater than 3 or 4 because for large values of d, the polynomial curve can become overly flexible, especially near the boundary of the X variable.
Even though this is a linear regression model like any other, the individual coefficients are not of particular interest. Instead we look at the entire fitted function across a grid of X values.

Step Functions

Using polynomial functions of the features as predictors in a linear model imposes a *global* structure on the non-linear function of X.

We can instead use *step functions* in order to avoid imposing such a global structure.

Here we break the range of X into bins, and fit a different constant in each bin. This amounts to converting a continuous variable into an *ordered categorical variable*.

In greater detail, we create cutpoints c_1, c_2, …, c_K in the range of X and then construct the K + 1 new variables:

$C_0(X) = I(X < c_1)$,
$C_1(X) = I(c_1 \leq X < c_2)$,
$C_2(X) = I(c_2 \leq X < c_3)$,
…
$C_{K-1}(X) = I(c_{K-1} \leq X < c_K)$,
$C_K(X) = I(c_K \leq X)$

Where I(.) is an *indicator function* that returns a 1 if the condition is true, and returns 0 otherwise. These are sometimes called *dummy* variables.
Notice that for any value of X, the sum of all these indicators ($C_0(X) + \ldots + C_K(X)$) is always equal to 1, since X must be in exactly one of the K + 1 intervals.

We then use least squares to fit a linear model using $C_1(X) + \ldots + C_K(X)$ as predictors. We exclude $C_0(X)$ as a predictor because it is redundant with the intercept; we could choose to include it and remove the intercept.

$$y_i = \beta_0 + \beta_1 C_1(x_i) + \beta_2 C_2(x_i) + \ldots + \beta_K C_K(x_i) + \epsilon_i$$

For a given value of X, at most one $C_1,...,C_k$ can be non-zero.

Note that when $X < c_1$, all of the predictors are zero, so β_0 can be interpreted as the mean value of Y for $X < c_1$. By comparison, the above equation predicts a response of $\beta_0+\beta_j$ for $c_j \leq X < c_{j+1}$ so β_j represents the average increase in the response for X in relative to $c_j \leq X < c_{j+1}$ relative to $X < c_1$.

Unfortunately, unless there are natural breakpoints in the predictors, piecewise-constant functions can miss the action.
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.

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: $b_1(X)$, $b_2(X)$, …, $b_K(X)$. Instead of fitting a linear model in X, we fit the model:

$$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \ldots + \beta_K b_K(x_i) + \epsilon_i$$

Note that the basis functions $b_1(.),..., b_K(.)$ are fixed and known (In other words, we choose these 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 $b_j(x_i) = I(c_j \leq x_i < c_{j+1})$.

As usual, we can think of the model as a standard linear model with predictors $b_1(x_i),..., b_K(x_i)$. Hence, we can use least squares to estimate the unknown regression coefficients. 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.

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

$$y_i = \square_0 + \square_1 x_i + \square_2 x_i^2 + \square_3 x_i^3 + \epsilon_i$$

where the coefficients differ in different parts of the range of X. The points where the coefficients change are called *knots*.
For example, a piecewise cubic with no knots is just a standard cubic polynomial with d = 3.
A piecewise cubic polynomial with a single knot at a point c takes the form:
$y_i = \square_{01} + \square_{11} x_i + \square_{21} x_i^2 + \square_{31} x_i^3 + \epsilon_i$ if $x_i < c$;
$y_i = \square_{02} + \square_{12} x_i + \square_{22} x_i^2 + \square_{32} x_i^3 + \epsilon_i$ if $x_i \geq c$.
In other words, we fit two different polynomial functions to the data, one on the subset of the obs with $x_i < c$ and one on the subset of the obs with $x_i \geq c$. Each of these polynomial functions can be fit using least squares applied to simple functions of the original predictor.

In general, if we place K different knots throughout the range of X, then we will end up fitting K + 1 different cubic polynomials.

Constraints and Splines

To reduce the flexibility of a piecewise polynomial model, we can fit it under the constraint that the fitted curve must be continuous(at the knots).

To reduce it further, we can add two additional constraints: both the first and second derivatives of the piecewise polynomials have to be continuous at the knots. In this way we are requiring that the piecewise polynomial has to be very *smooth* at the knots.

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.

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.

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

The Spline Basis Representation

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 to represent a regression spline.

A cubic spline with K knots can be modeled as

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

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

There are several ways to represent cubic splines using different choices of basis functions. The most direct way to represent a cubic spline is to start off with a basis for a cubic polynomial -- namely, x, x^2, x^3-- and then add one *truncated power basis* function per knot. A truncated power basis function is defined as

$h(x, \xi) = (x - \xi)_+^3 =$ if $(x - \xi)^3$ if $x > \xi$, 0 otherwise where ξ is the knot.

One can show that adding a term of the form $\beta_4 h(x,\xi)$ to the model 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 for X, X^2, X^3,$h(x,\xi_1)$, …, $h(X,\xi_K)$. 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.

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.

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.

Choosing the Number and Location 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.
One option would be 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.

In practice, i 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.

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. This procedure can be repeated for different numbers of knots K. Then the value of K giving the smallest RSS is chosen.

Comparison to Polynomial Regression

Regression splines often give superior results to polynomial regression. This is because unlike polynomials, which must use a high degree to produce flexible fits, splines introduce flexibility by increasing the number of knots (more coefficients) but keeping the degree fixed. Generally, this approach produces more stable estimates.

Smoothing Splines

We now introduce a somewhat different approach that also produces a spline.

In general, when we fit a (smooth) curve to a set datak what we really want to do is to find some function, say g(x) that fits the observed data well: that is we want RSS to be small. However, if we don't put any constraints in $g(x_i)$, 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:

$\sum_{i=1}^{n} (y_i - g(x_i))^2 + \lambda \int g''(t)^2 dt$ where λ is a *nonnegative tuning parameter*.

The function g is known as a *smoothing spline*.

What does this equation mean? It takes the **"Loss + Penalty"** formulation that we encounter in ridge and lasso.
The first term is a *loss function* that encourages g to fit the data well, and the second term is a *penalty term* that penalizes the variability in g.

The notation $g''(t)$ indicates the second derivative of the function g. Broadly speaking, the second derivative of a function is a measure of its *roughness*(concavity, curvature): if it's large in absolute value then g(t) is very wiggly near t, it is close to zero otherwise (the second derivative of a straight line is zero; note that a line is perfectly smooth).

The integral $\int g''(t)^2 dt$ is a measure of the total change in the function $g'(t)$ over its entire range.

If g is very smooth (think ~ a line), then $g'(t)$ would 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, $\lambda \int g''(t)^2 dt$ encourages g to be smooth.

We see that λ controls the bias-variance trade-off of the smoothing spline.

The function g(x) that minimizes the equation can be shown to have some special properties: **it is a piecewise cubic polynomial with knots at the unique values of $x_1,...,x_n$**.
However, it is not the same natural cubic spline that one would get if one applied the basis function approach previously described with knots at the same $x_1,...,x_n$, rather it is a *shrunken* version of such a natural cubic spline, where λ controls the level of shrinkage.

Choosing the Smoothing Parameter lambda

It might seem that a smoothing spline will have far too many degrees of freedom, since a knot at each(unique) 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.

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 better measure of the flexibility of the smoothing spline.

In fitting a smoothing spline, we do not need to select the number or location of the knots.
Instead we need to choose the value of λ.
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.

In general, if there is little difference between different (with different degrees of freedom) fits, we choose the simpler model (less flexible, smaller df_λ).

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.

Algo: 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 $\sum_{i=1}^{n} K_{i0}(y_i - \beta_0 - \beta_1 x_i)^2$
4. The fitted value at x_0 is given by $\hat{f}(x_0) = \hat{\beta}_0 + \hat{\beta}_1 x_0$.

Note that in Step 3 of the algorithm, 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 $\sum_{i=1}^{n} K_{i0}(y_i - \beta_0 - \beta_1 x_i)^2$ for a new set of weights.
Local regression is sometimes referred to as a *memory-based* procedure, because like knn, we need all the training data each time we wish to compute a prediction.

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 above.

While all these choices make some difference, the most important choice is the *span s*, defined in Step 1 above.
The span plays a role like that of the tuning parameter 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 specify it directly,

The idea of local regression can be generalized in many different ways. In a setting with multiple features $X_1,...,X_p$, 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 can perform very 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-neighbor regression suffers from a similar problem in high dimensions.

Generalized Additive Models

Generalized additive models (GAMs) provide a general framework for extending a standard linear model by allowing non-linear functions of each of the variables, while maintaining additivity. Just like linear models, GAMs can be applied with both quantitative and qualitative responses.

A natural way to extend the multiple linear regression model

$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip} + \epsilon_i$

in order to allow for non-linear relationships between each feature and the response is to replace each linear component with a (smooth) non-linear function $f_j(x_{ij})$. We would then write the model as

$$y_i = \beta_0 + \sum_{j=1}^{p} f_j(x_{ij}) + \varepsilon_i = \beta_0 + f_1(x_{i1}) + f_2(x_{i2}) + \ldots + f_p(x_{ip}) + \varepsilon_i$$

This is an example of a GAM. It is called an *additive* model because we calculate a separate f_j for each X_j, and then add together all of their contributions

In the previous sections we discussed many methods for fitting functions to a single variable. The beauty of GAMs is that we can use these methods as building blocks for fitting an additive model.

This method fits a model involving multiple predictors by repeatedly updating the fit for each predictor in turn, holding the others fixed. The beauty of this approach is that each time we update a function, we simply apply the fitting method for that variable to a *partial residual*.

A partial residual for X_3, for example, has the form $r_i = y_i - f_1(x_{i1}) - f_2(x_{i2})$. If we know f_1 and f_2, then we can fit f_3 by treating this residual as a response in a non-linear regression on X_3.

Pros and Cons of GAMs

- ⇧ GAMs allow us to fit a non-linear f_j to each X_j, so that we can automatically model non-linear relationships that standard linear regression will miss. This means that we do not need to manually try out many different transformations on each variable individually.
- ⇧ The non-linear fits can potentially make more accurate predictions for the response Y
- ⇧ Because the model is additive, we can still examine the effect of each X_j on Y individually while holding all of the other variables fixed. Hence if we are interested in inference, GAMs provide a useful representation.
- ⇧ The smoothness of the function f_j for the variable X_j can be summarized via degrees of freedom.
- ⇩ The main limitation of GAMs is that the model is restricted to be additive. With many variables, important interactions can be missed. However, as with linear regression, we can manually add interaction terms to the GAM model by including additional predictors of the form $X_j \times X_k$. In addition we can add low-dimensional interaction functions of the form $f_{jk}(X_j, X_k)$ into the model; such terms can be fit using two-dimensional smoothers such as local regression or two-dimensonal splines.

For fully general models, we have to look for even more flexible approaches such as random forests and boosting. GAMs provide a useful compromise between linear and fully nonparametric models.

GAMs for Classification Problems → p.286 (log odds = GAM equation → logistic regression GAM). Same pros and cons discussed for quantitative responses.