Projection Pursuit Regression

Suppose that

$$\vec{x} = (x_1, x_2, \ldots, x_p)^T$$

is a p-dimensional vector of inputs, and that $\vec{\omega}_1, \vec{\omega}_2, \ldots, \vec{\omega}_{M-1}$, and $\vec{\omega}_M$ are p-dimensional unit vectors (which can be thought of as directions in the p-dimensional input space). Letting Y be the response variable, and letting $f(\vec{x})$ denote $E(Y|\vec{x})$, the *projection pursuit regression* (PPR) model has the form

$$f(\vec{x}) = \sum_{i=1}^{M} g_m(\vec{\omega}_m^T \vec{x}).$$

This is an additive model, but in the derived features, $v_m = \vec{\omega}_m^T \vec{x}$ $(m = 1, \ldots, M)$ rather than the inputs, $x_1, x_2, \ldots, x_{p-1}$ and x_p. The g_m $(m = 1, \ldots, M)$ are unspecified functions which are estimated, along with the $\vec{\omega}_m$ (the directions), using some flexible smoothing method.

$g_m(v_m) = g_m(\vec{\omega}_m^T \vec{x})$ is called a *ridge function*, and it varies only in the direction defined by $\vec{\omega}_m$. (Although it's a function which maps $\mathbb{R}^p$ to $\mathbb{R}$, it only varies along a single direction. See Fig. 11.1 on p. 348 of HTF for some examples.)

The scalar variable $v_m = \vec{\omega}_m^T \vec{x}$ is the projection of $\vec{x}$ onto $\vec{\omega}_m$. Altogether, the PPR method tries to find directions, $\vec{\omega}_1, \ldots, \vec{\omega}_{M-1}$, and $\vec{\omega}_M$, and functions, $g_1, \ldots, g_{M-1}$, and g_M, which together constitute a good regression model. The name *projection pursuit* comes from the iterative search for good directions. (There are similarities with principle components regression. But with PCR, the directions used are based only on the input values — the directions are not determined by seeking ones that provide a good regression model for Y. Also, with PCR, once the PCs are determined, one does plain old regression modeling with them, whereas with PPR, nonlinearities are accounted for by adapting to the data in an iterative manner, as the right $\vec{\omega}_m$ and g_m are converged to.)

The PPR model is very general — it has great flexibility. Power terms such as x_2^3 can be included if one of the $\vec{\omega}_m$ becomes $(0, 1, 0, \dots, 0)^T$, and the corresponding g_m becomes a cubic function (although this typically will not occur). P. 348 of HTF indicates how product terms such as $x_1 x_2$ can be represented (but again, this typically will not occur). Also, each of the g_m can be a *nonlinear* function of its v_m.

If M is made large enough, the PPR model can, in principle, approximate any continuous function on $\mathbb{R}^p$ arbitrarily well — we have what is called a *universal approximator*. But this generality comes at a price: because each input can enter the model in a complex and multifaceted way, interpretation of the fitted model can be extremely difficult. (An exception is the case of $M=1$, which gives us what is known as the *single index model* in econometrics.)

Neural network models suffer the same problems when it comes to interpretation and gaining an understanding of the phenomenon being modeled. But both fitted PPR models and fitted neural network models can be great predictors! Of course, with such flexible models, one should be concerned about overfitting. With PPR, overfitting can be prevented by using cross-validation to determine an appropriate value to use for M.

To fit a PPR model, approximate minimizers are sought for

$$\sum_{i=1}^{n} [\, y_i - \sum_{m=1}^{M} g_m(\vec{\omega}_m^T \vec{x}) \,]^2.$$

Let's first consider using $M = 1$. In this case, a direction, $\vec{\omega}$, and a function, g, are sought to minimize

$$\sum_{i=1}^{n} [\, y_i - g(\vec{\omega}^T \vec{x})]^2.$$

Given an initial direction vector, $\vec{\omega}$, the derived variable values,

$$v_i = \vec{\omega}^T \vec{x}_i \qquad (i = 1, \ldots, n),$$

are determined. Then, a scatterplot smoother, such as a smoothing spline, is applied to the ordered pairs,

$$(v_i,\ y_i) \qquad (i = 1, \ldots, n),$$

to obtain an initial g. Now, given a g, we want to determine a better $\vec{\omega}$. Letting

$\vec{\omega}_c$ be the current direction, and $\vec{\omega}_n$ be the new direction, we can use a 1st-order Taylor series approximation:

$$g(\vec{\omega}_n^T\vec{x}_i) \approx g(\vec{\omega}_c^T\vec{x}_i) + g'(\vec{\omega}_c^T\vec{x}_i)\,(\vec{\omega}_n^T\vec{x}_i - \vec{\omega}_c^T\vec{x}_i).$$

This gives us that

$$\sum_{i=1}^n [\,y_i - g(\vec{\omega}_n^T\vec{x}_i)]^2$$

$$\approx \sum_{i=1}^n [\,y_i - g(\vec{\omega}_c^T\vec{x}_i) - g'(\vec{\omega}_c^T\vec{x}_i)\,\vec{\omega}_n^T\vec{x}_i + g'(\vec{\omega}_c^T x_i)\,\vec{\omega}_c^T\vec{x}_i\,]^2$$

$$= \sum_{i=1}^n \left[\,g'(\vec{\omega}_c^T\vec{x}_i)\left\{\vec{\omega}_c^T\vec{x}_i + \frac{[\,y_i - g(\vec{\omega}_c^T\vec{x}_i)]}{g'(\vec{\omega}_c^T\vec{x})} - \vec{\omega}_n^T\vec{x}_i\right\}\right]^2.$$

$$= \sum_{i=1}^n [\,g'(\vec{\omega}_c^T\vec{x}_i)]^2 \left[\left(\vec{\omega}_c^T\vec{x}_i + \frac{\{y_i - g(\vec{\omega}_c^T\vec{x}_i)\}}{g'(\vec{\omega}_c^T\vec{x})}\right) - \vec{\omega}_n^T\vec{x}_i\right]^2.$$

It follows, that to minimize this approximation of

$$\sum_{i=1}^n [\,y_i - g(\vec{\omega}_n^T\vec{x}_i)]^2,$$

we can do a weighted least squares regression using no intercept, the $[g'(\vec{\omega}_c^T\vec{x}_i)]^2$ as weights, and the

$$\vec{\omega}_c^T\vec{x}_i + \frac{\{\,y_i - g(\vec{\omega}_c^T\vec{x}_i)\}}{g'(\vec{\omega}_c^T\vec{x}_i)}$$

as the response variable values. Upon expanding $\vec{\omega}_n^T \vec{x}_i$ to be

$$\omega_{n,1}\, x_{i,1} + \omega_{n,2}\, x_{i,2} + \cdots + \omega_{n,p}\, x_{i,p},$$

we can clearly see the model being fit. The original inputs are the predictors in this regression model, and the components of $\vec{\omega}_n$ are the coefficients to be fit using weighted least squares. (Summarizing, we approximate $\sum_{i=1}^n [y_i - g(\vec{\omega}_n^T \vec{x}_i)]^2$ with something of the form

$$\sum_{i=1}^n w_i \left[y_i^* - (\omega_{n,1} x_{i,1} + \cdots + \omega_{n,p} x_{i,p})\right]^2,$$

which allows us to use weighted least squares to obtain the minimizing values of $\omega_{n,1}, \omega_{n,2}, \ldots, \omega_{n,p-1}$, and $\omega_{n,p}$. With the w_i, $\vec{x}_i$, and y_i^* known, the minimizing $\vec{\omega}_n$ can be obtained.)

Using the new $\vec{\omega}$, we get new v_i values, which are then used (with the unchanging y_i) to get a new g. From the new g, we obtain a new $\vec{\omega}$, and we iterate until convergence occurs.

For $M > 1$, the model is built in a forward stagewise manner, adding a new $(\vec{\omega}_n, g_m)$ pair at each stage. P. 349 of HTF gives some details.

- As a new direction is being determined, the previously obtained $\vec{\omega}_m$ are typically not readjusted.
- After each step, the g_ms from previous steps can be readjusted (using backfitting).

One wants to use a smoothing method (to determine the g_m) for which it is easy to obtain the necessary derivatives. R's ppr function offers a choice of smoothing methods.

Missing in HTF and the R documentation is information about what should be used as an initial value for $\vec{\omega}_1$. One source suggests that the first principal component be used. As an initial value for $\vec{\omega}_2$, one could use the second principal component, or it might work well to use the part of the second principal component which is orthogonal to the converged $\vec{\omega}_1$.

P. 350 of HTF states that PPR has not become widely used, and suggests that perhaps this is because that when it was introduced in 1981 its computational demands were too great. Now it is feasible to do PPR with a lot of data sets, and R's ppr function provides us with a convenient way to use this modern method of regression, which allows for great flexibility in fitted models. (But PPR is not included on the menus of some commonly used statistical software.)