Kernel Methods for Regression

Letting $N_k(\vec{x})$ denote the set of k points nearest to $\vec{x}$, the *k-nearest-neighbor average*,

$$\frac{\sum_{i\,:\,\vec{x}_i \in N_k(\vec{x})} y_i}{k},$$

is a simple way to estimate $E(Y|\vec{x})$.

Leave-one-out cross-validation (n-fold c-v) is a good way to select a suitable value to use for k.

A related method is to use the *Nadaraya-Watson kernel-weighted average*, which

estimates $E(Y|\vec{x})$ (or predicts a new value of Y at a given value of $\vec{x}$) with

$$\frac{\sum_{i=1}^{n} w_i y_i}{\sum_{i=1}^{n} w_i},$$

where the w_i are nonnegative weights which are typically largest for the points with $\vec{x}_i$ nearest $\vec{x}$, and decrease as the distance between $\vec{x}_i$ and $\vec{x}$ increases, possibly being equal to 0 for all points for which $\vec{x}_i$ is more than a certain set distance from $\vec{x}$. (Really, it might be better to think of $w_j / \sum_{i=1}^{n} w_i$ as the weight for y_j. (This way, the sum of the weights is equal to 1.))

A popular choice is to use the *Epanechnikov kernel* to obtain the weights, letting

$$w_i = \frac{3}{4}\left(1 - \left[\frac{d(\vec{x}_i, \vec{x})}{\lambda}\right]\right)^2$$

if $d(\vec{x}_i, \vec{x}) < \lambda$, and letting $w_i = 0$ o/w, where $d(\vec{x}_i, \vec{x})$ is the distance between $\vec{x}_i$ and $\vec{x}$. If the *smoothing parameter*, λ, which is the half-width of the local neighborhood is fixed, then instead of a fixed number of points contributing equally, as is the case for k-nearest-neighbors, all points within a certain distance of the target contribute, with the weight of the contribution decreasing as the distance from

$\vec{x}$ increases.

R's kknn function in a sense combines the kernel smoothing concept with k-nearest neighbors. Instead of using a fixed value of λ, which could result in 30 points contributing to some predictions and only 3 points contributing to others, a different value of λ can be used for each prediction, with things set up so that exactly k points have nonzero weights for each prediction, where k is specified, and where the weights decrease as points become farther from the target (unless the

rectangular kernel is chosen, which gives constant weight to each of the k points, which makes kknn do ordinary k-nearest-neighbors predictions). If the function train.kknn is used, one can have cross-validation not only determine the best value to use for k, but it can also determine the most effective kernel (from a list of choices).

My guess is that more sophisticated kernel methods are better for predicting numerical responses, but these types of kernel weighting methods can be good

for some classification settings. The weights are used to perform a weighted voting to arrive at predicted classes.

With a numerical response variable, *locally weighted regression* typically does a better job than nearest-neighbors and kernel-smoothing methods. The main idea behind local regression is that within a suitably small region, the expected value of the response may be close to being a linear fn of the explanatory variables, or, if there is appreciable local curvature, it can be well approxi-

mated using a 2nd-order polynomial fit. To get away from a global fit, a weighted regression is done, which allows the coefficients to differ from location to location.

To estimate $E(Y|\vec{x})$ for a given value of $\vec{x}$,

$$\sum_{i=1}^{n} w_i \left(y_i - [\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}] \right)^2$$

is minimized, and the weighted least squares estimates are used to give the prediction formula

$$\hat{y} = b_0 + b_1 x_1 + \cdots + b_p x_p.$$

Alternatively, if the local curvature is great enough so that the decrease in bias

will more than offset the associated increase in variance, a full 2nd-order model can replace $\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$. (A problem is that if there are more than two predictor variables, it can be difficult to assess the curvature. Of course, cross-validation or an independent validation set can be used to compare the 1st-order and 2nd-order local fits.) In either case, a kernel can be used to obtain weights (as described on p. 3), providing the greatest weights to the points closest to the target point.

Fig. 6.3 on p. 169 of HTF illustrates how a locally weighted 1st-order regression can outperform a locally weighted average for points near a "boundary," and Fig. 6.5 shows how a local quadratic fit can outperform a local linear fit for points near the top of a "hill" (local linear fits tend to "trim the hills" and "fill the valleys.") Fig. 6.6 (p. 172) shows that a local quadratic fit can have an appreciably greater variance than a local linear fit.

In a multiple regression setting, it is common practice to standardize all explanatory

variables by dividing by the sample standard deviations, so that, for example, price (in dollars) won't dominate weight (in tons) when determining distances when doing a study of automobile data. But this doesn't address the problem that some variables may be unrelated, or weakly related, to the response, and perhaps shouldn't even be used to determine the distances. One strategy is to see what variables appear to be the most powerful predictors when other regression methods are used, and only use them when applying local methods of regression. Of course, one could use cross-validation or an independent validation

set to compare the estimated MSPE values which result when different sets of explanatory variables are used.

R's loess fn is one way to do local regression. The lowess fn is an alternative choice if there is just one predictor. — it does things a bit differently than loess. Lowess and loess are not fundamentally different methods. Loess is just a particular implementation of local regression that has become popular.