Side-Note: Nonparametric Nonlinear Prediction

36-462, Spring 2009

22 January 2009, to accompany Lecture 4

Parametric prediction is, in principle, easy: (1) estimate the parameters of the model, (2) estimate the state, and then (3) plug in to the model to calculate what should happen in the future. (To make things easy on ourselves, let's suppose we're only trying for a point prediction, rather than to predict a distribution of outcomes.) Of course all three steps are easier said than done...

We are not blessed with a parametric model — or would like some kind of additional check on one we have. What are some non-parametric ways of making predictions?

There are three big ones, of which we will look at two.

1. *Massively-parametric regression.* With a high-enough order polynomial, you can match any smooth function arbitrarily well. So use a lot of parameters, by including higher-order nonlinear terms. What is usually meant by "non-parametric regression"; figure out the order you need (at least to within the noise level), then fit. Polynomials are usually a bad choice (they have this nasty habit of zooming off to infinity outside your sampling range, which is implausible), so better-behaved things are used likes splines [7], neural networks [6], support-vector machines [3], additive models [4], etc., but the principle is the same. We won't cover this because it needs, a whole course (which I teach in the fall).

2. *Nearest-neighbor methods.* Observe a whole bunch of states and their subsequent behavior. To predict what will happen at a new state, not previously observed, find the most similar state you have seen, and predict it will do likewise. Maybe, for safety, find a few of the closest states and predict the average of their behavior. (How this leads to safety is discussed in the slides.)

   Assumes: future behavior varies smoothly with state; that you have seen enough states that the new ones are reasonably close to the old ones; that you can figure out states from observations.

3. *Kernel prediction.* This lies somewhere between the other two. Instead of predicting the average of the k nearest neighbors, average *all* previously observed points, but with a weight that falls off the further the old point

is from the one where we're trying to predict. This weighting function is called the **kernel**. As more and more data comes in, give further-away points less and less weight.

You can think of this as giving us a regression function where the order grows with each data point.

The hand-out from Kantz and Schreiber [5] talks about nearest-neighbor methods; we will be using them in R, rather than their custom software. I want to say a little about the kernel methods. (For much more, see [1, 2].

1 Kernel Methods

Suppose U_t is our time-series of inputs and V_t is our time-series of outputs, both of length n. The **kernel predictor** is then

$$R_{n,h,K}(u) = \sum_{t=1}^{n} V_t \frac{K\left(\frac{u-U_t}{h}\right)}{\sum_{s=1}^{n} K\left(\frac{u-U_t}{h}\right)} \quad (1)$$

where K is the kernel function and h is called the **bandwidth**. That is, to predict the value of V at the input point u, we take the average of all the observed values of V_t, weighting them by how far their inputs were from u, with weights which are proportional to the kernel function.

A good kernel function, then, should be symmetric, non-negative and go to zero at infinity; in fact it's conventional and convenient to make a kernel a probability density function, so it integrates to 1. The bandwidth h re-scales the kernel function; it should go to zero as $n \rightarrow \infty$, so that when we have a lot of data, we really emphasize points which are close to u, and pay little attention to points which are far from u.

In our case, a natural choices for "input" and "output" are the time-delay vector of our observed time series, $U_t = (s_t, s_{t-1}, \ldots s_{t-k+1})$, and the next observation, $V_t = s_{t+1}$.

We also need to choose a kernel. Two common ones are the rectangular or uniform kernel ($= 1$ on the unit cube centered at zero, $= 0$ elsewhere), the ball kernel ($= 1$ on the unit ball, $= 0$ elsewhere)[1] and the standard Gaussian density, $= (2\pi)^{-k/2}e^{-\|x\|^2/2}$.

How then to choose k and h? Well, if we have succeeded in convincing ourselves that we've found a good embedding dimension, use that k. Otherwise, a good practice is to use **cross-validation**.

2 Cross-validation

Here's how it works. (There are many variations.) Pick 10% of the data points at random and hold them aside as the **test set**. Pick an arbitrary combination

[1]This doesn't integrate to 1, but rather to $\frac{4}{3}\pi$. But the normalization constant would just cancel out.

of k and h from a range of values you decide *a priori* are plausible. Take the remaining 90% of the data, the **training set**, and use it to estimate R by plugging in to Equation 1. Now use this $\widehat{R}$ to predict the test set. Calculate the out-of-sample error,

$$err(h,k) = \sum_{t \in \text{test set}} |V_t - \widehat{R}(U_t)|$$

(You might want to use the squared error, rather than the absolute error, if you have good reason to believe in Gaussian distributions.) Repeat this for a couple of different random splits into training and test sets, and average the out-of-sample errors. Finally, pick the combination of h and k which has the best out-of-sample performance, i.e., the smallest error.

This is obviously very computationally intensive. It is also a pretty standard method in statistical learning. Notice that nothing really depends on using the kernel-predictor here; we could use it to fix control settings for any other predictor, provided that we have a way of calculating its out-of-sample performance.

References

[1] Bosq, Denis (1998). *Nonparametric Statistics for Stochastic Processes: Estimation and Prediction*. Berlin: Springer-Verlag, 2nd edn.

[2] Caires, S. and J. A. Ferreira (2005). "On the Non-parametric Prediction of Conditionally Stationary Sequences." *Statistical Inference for Stochastic Processes*, **8**: 151–184. doi:10.1007/s11203-004-0383-2. Correction, vol. 9 (2006), pp. 109–110.

[3] Cristianini, Nello and John Shawe-Taylor (2000). *An Introduction to Support Vector Machines: And Other Kernel-Based Learning Methods*. Cambridge, England: Cambridge University Press.

[4] Hastie, Trevor, Robert Tibshirani and Jerome Friedman (2001). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. New York: Springer-Verlag.

[5] Kantz, Holger and Thomas Schreiber (2004). *Nonlinear Time Series Analysis*. Cambridge, England: Cambridge University Press, 2nd edn.

[6] Ripley, Brian D. (1996). *Pattern Recognition and Neural Networks*. Cambridge, England: Cambridge University Press.

[7] Wahba, Grace (1990). *Spline Models for Observational Data*. Philadelphia: Society for Industrial and Applied Mathematics.