161.326 Statistical Machine Learning
Part 9: Smoothing

Mark Bebbington

Summary

This part (the equivalent of 4 lecture hours) deals with nearest neighbour methods, which are the alternative to the structural models discussed in Part 1. The idea is that points close together in the input space should have similar outputs. Re-read slides 4-6 of Part 5. Most importantly, we are now in a nonlinear setting.

Chapter 4 of the text looks at splines, which are a related technique. We will instead concentrate on kernels, on which the book touches briefly in Section 8.2.

As usual, we will proceed with a mixture of theory and implementation in NumPy.

Once you have finished the lecture slides and attempted the exercises you should then be able to do question 2 of Assignment 3.

The remainder of this document is a commentary on the lecture slides, which should be read in conjunction with them.

Slides

3. Smoothing is a local method. Once the localization is done by, e.g., weighting inversely to the distance, one can use equations which can be described as regression on the selected points to obtain an estimate. The key point is that the *same formula* is used everywhere, only the inputs vary as we move around the space.
4. The result is bumpy because, as you move along, suddenly one point is replaced by another, perhaps with a very different output, which gives a sharp jump in the average. Note that the x_i need not be evenly distributed around x – they could all be on one side for example.
5. x0 is the set of points at which we will calculate the function. These should be selected at an even spacing, ideally using the minimum and maximum of the data. d is the distance from the calculation point to the data point, which is calculated for each data point. The distances are then glued on to the outputs, and the result sorted by distance (`rowsort') carrying the outputs along. The first N rows of the result then contain the outputs at the shortest distance from the calculation point.
6. Use 1000 points (no particular reason, I just like the number)
7. Results for 5 and 10 nearest neighbors. Note the 'jaggedness', especially for 5 nearest neighbors, and the long flat sections at the edges, where all the evaluation points have the same nearest neighbors, and hence the same value.
8. The Epanechnikov quadratic kernel has bounded support (is non-zero only on a finite range).
9. The idea here is to make the estimate `smoother' rather than bumpy. Hence we weight a point's contribution to the average according to the distance.
10. Remember, lambda is a reserved word. As the square of lam is used in every iteration, it saves computing time to calculate it once at the beginning and give it its own variable.

Arrays are used because we want element-by-element multiplication, not matrix multiplication.

11. We see a much smoother result. But is it too smooth, especially with $\lambda = 1$?
12. These are the three common kernels (there are many others). They have contrasting properties – the Gaussian has infinite support (every point has some weight, albeit possibly very small), while the other two have zero weight beyond the distance λ. The tri-cube kernel is differentiable at λ, which can be quite important. Basically, what the smoother does is to add noise, with a distribution equal to the kernel, to the location of each point. Thus the output for a given point is `smeared' along the x-axis in this case.
13. See 'ex9_1.py'
14. The earlier nearest-neighbor method can be seen as an adaptive neighborhood. Apart from this, adaptive neighborhoods are beyond the scope of this course.
15. The old question of variance versus bias is alive and well. The issue of what we do near the boundary creates yet another trade-off (TNSTAAFL).
16. Instead of just a weighted average of the outputs, why not do a linear regression across the kernel width, but with weights. Note that we have a different set of parameter estimates at every point we evaluate at. In other words, we are calculating the slope of the smoothed curve separately at each evaluation point. B is a 2-column matrix (a column of ones plus the column of inputs). We will be interested in the *equivalent kernel* later.
17. As we want matrix multiplication, we work with matrices. Note how W is constructed. The function returns fhat, the matrix of local regression fits.
18. Note use of the variable `fineness' in the program – it can be very useful to get code working using a small set of points, and then increase the density for the final result. Note that the λ's are larger than for the previous programs – a singular matrix results from small values as there are no data included. This does not stop the previous programs, but you may notice discontinuities in the plots. We will look at how to chose λ later.
19. Just as we can do quadratic regression etc., we can do so in a local setting. However, it needs optimization methods, so it will have to wait until later.
20. Linear methods don't fit curves well, unless there is a lot of data at the curves.
21. Here we see the use for the equivalent kernel.
22. Basically we use the locally linear regression function without output data ☺
23. We have seen how the effective degrees of freedom can be used for model selection in Part 6.
24. We are now moving away from regression type methods, as our `output' is now going to be the probability density at a given point. In other words, we now have 1-D data (just the inputs, no outputs). The natural estimate consists of running a window along and counting the number of observations in it at each point. The result is obviously a step function, which should be borne in mind if you want a correct estimate.
25. See 'ex9_2.py'
26. The analogy to the kernel-weighted average is to apply `noise' to each point, or in other words, to replace each observation by a normal pdf, and sum the contributions at each point. Note how this can be seen as a convolution. (No – you won't have to use this fact, but it looks nice).
27. The technique is generalizable to p dimensions if you have multivariate data.
28. Using the Gaussian kernel. This gives positive weight to all the data points, hence the use of N in the denominator.
29. As the data is 1-D, we generate a pseudo output which separates them for display purposes. Note use of min(x) and max(x) to set boundaries. We will now look just at the onset dates of eruptions of Mt Ruapehu. Is there any pattern? Note how increasing λ reduces the number of

peaks. Obviously there is an increase in density in the later record, which may simply be a greater probability of observation, but are there `cycles' in the data?

30. The answer depends on what value of λ.we choose.
31. Remember that the kernel is (usually) symmetric. Hence at the boundaries of the data, half of it will be empty. We can either a) keep away from the boundaries – unfortunately in a temporal setting, the need to forecast doesn't permit this, or b) create artificial data to feed to the smoother, or c), inversely weight the estimate by the area under the kernel within the boundaries.
32. Welcome to a new package scipy.stats which has a few basic distribution commands, such as cdf – the area under the normal density function. If all of the kernel is within the boundaries (impossible with the Gaussian kernel) then Kint will be 1.
33. Slight difference at the ends, with the density estimate elevated slightly. Try different λ's and boundaries (such as 1997, say).
34. The idea of selecting a `best' bandwidth λ is to see how each observed point is predicted if the density is computed using the other points only. We want a high density at each removed point. The density will vary with λ and so we select the value of λ that does best over all the points.
35. You have seen this before except for the bit where we split x into x0 and x1. Note that x0 is now the data, not an arbitrary set of points for calculating the density at, and we calculate at each data point x1 in turn using the remaining data.
36. We now have a new outer loop for λ. It includes a boundary correction, but the boundaries are not data dependent. The process here is recursive – try a large range of λ, and progressively localize in order to get the required precision.
37. See 'ex9_3.py'
38. See 'ex9_4.py'

You can now read "Altman NS (1992) An introduction to kernel and nearest-neighbor nonparametric regression. The American Statistician, 46, 175-185", available through the library electronic catalog.

39. The idea here is that we fit separate densities (over the same input range) to the observations associated with each output. Then, given a point in input space, it is classified as belong to the class with the highest density at that point. Note that it has to be normalized by the relative proportions of each class, as the kernel density estimate has total mass one, regardless of whether there are 10 or 1000 observations in the class.
40. The only new parts are the estimate of the prior probabilities pi1 and pi2, and the method of dividing x according to whether the observation is 1 (> 0.5) or 0 (< 0.5). We calculate two density estimates at each point, one for each class. The last line is the estimated probability of the point being in class 1 (= 1 minus the probability of being in class 2).
41. We saw this data in Part 1. Here we will use only one predictor variable (see `Naïve Bayes' later), the systolic blood pressure.
42. We see that both classes have a peak at about the same blood pressure. However, once normalized by the size of each class, we get the posterior probability in the lower graph, where a subject is classified as a heart attack risk if their blood pressure is between 170-185 and 195+ (approx.). Is this particularly sensible? The problem is that the region between 180 and 200 blood pressure is sparse in both classes, so the method is very variable in this region. Experiment with different values of λ.
43. The important thing to note is that this can work much better than the density estimation alone.

44. This is ugly and works, so it “ain’t ugly”. A histogram estimate replaces the smoothed (continuous) estimate with a discrete distribution on the points only.
45. Note that we now calculate a matrix of density estimates. Work out what the columns and rows of the estimates mean. The other new bit here is the last few lines where these matrices are collapsed to the two overall densities which are then compared.
46. Note that the error rate is higher than when using logistic regression (slide 8.38). However, Naïve Bayes is easily extended to more than 2 classes, not a statement one can make about logistic regression.

You can now read “Sheather SJ (2004) Density estimation. Statistical Science19, 588-597”, available through the library electronic catalog.