Solutions to seminar Week 3: Pre-Processing

Histograms and density estimation

A few of you asked why we were doing this. In Lecture I argued that the distinction between pre-processing and model training can be quite fuzzy. Think of it as a spectrum. On one hand of the spectrum, pre-processing can be as simple as scaling/normalising the data in which case all of the effort goes into model building. On the other end of the spectrum, most of the effort goes into pre-processing. Probability density estimation is an example of the latter. The idea is that if you can use the data to infer the class conditional probabilities $P(C_k|\mathbf{x})$ (i.e., the probability that data point $\mathbf{x}$ is of class C_k) of the data for all classes C_k, then you can use these class conditional probabilities as *features* (instead of the original data) in which case a simple model is to return the class with the highest class conditional probability. There is no model building/training to speak of. OK, moving on with the exercise itself.

The point of the first question (Create a histogram) was for you to explore the relationship between (a) the parameters of the distribution from which random values were taken, (b) the number of bins and (c) the total number of points. See code attached: `sem3_q1a.{m,py}`.

- The first thing to be clear about is that if you draw data from a distribution, the smaller the number of samples, the less likely you are to be able to accurately estimate the distribution. A simple example is as follows. Say I have a biased coin (e.g., a coin that returns heads a bit more often than tails). I know it's biased but you don't. If I only let you throw it 5 times, how likely are you to be able to infer that it is biased? What if I let you throw it 100 times? A million times? The more you can sample, the more your estimated probability density converges to the true probability density.
- The next question is to ask whether, for a given number of samples N, there is an optimal bin size (i.e., in how many bins you split the data). Here it's useful to think (and try it with the code!) of the extreme cases. If I use only 1 very large bin (so that all the data fall into that bin, then I am getting a single count value of N. **Pro**: my density estimation is insensitive to noise and sampling (i.e., if I try with a different set of random data, I will get the same 'histogram'). **Cons**: I have very little resolution. Let's now consider having an extremely large number of bins (so that, e.g., I have no more than 1 data point in each bin). **Pro**: Resolution is now very high. **Cons**: My estimation is extremely sensitive to noise and sampling. In other words, if I try with a different set of random data, I may see a totally different histogram. So clearly we need to find a trade-off. The best value will depend on the nature of the underlying distribution. If you have a very spiky distribution (a distribution with lots of peaks) then your best bin size is likely to be smaller than if you have a very smoother distribution (e.g., a distribution with just one peak). Anyhow, this means bin size is an *hyper-parameter* for your model. How do we select it? Well, in the absence of a model (as in this exercise) we do it qualitatively, by comparing it with the expected distribution. If we have a model, then, we do it by maximising the predictive power of the model.

The point of the second question (using kernel density estimation) is for you to appreciate the benefit of a smarter way to estimate density. In an histogram, we 'bin' the data. The resolution of the density estimation therefore depends on the size of the bin. But as we have seen, making resolution higher (reducing bin size) has big drawbacks if we have limited number of points. Kernel density estimation solves this problem (partly) by (a) estimating densities over overlapping bins and (b) applying a kernel to smooth the estimates (i.e., giving more weight to the count of nearby points, less to the count of far away points). Please see code attached: `sem3_q1b.{m,py}`.

Pre-Processing

See code attached: `sem3_q2.{m,py}`. Python users may want to look at the preprocessing package of sklearn as most operations can be done as part of a pipeline. I leave it to you to experiment with it.