

## Instance-Based Learning

**Nearest-neighbor (kNN)**

1NN—simple intuition: to classify a new data point, just return the class of the closest training point.

$k$ NN—instead of single closest point, average over the  $k$  nearest.

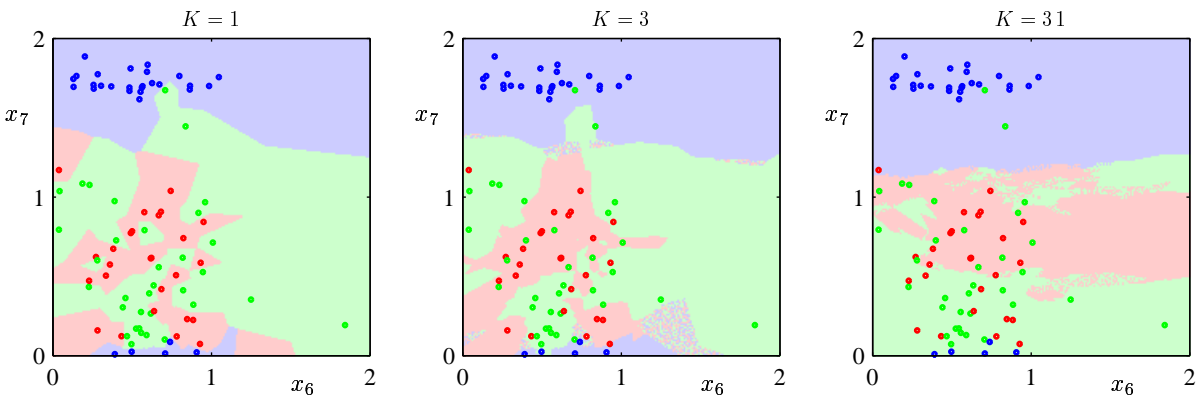
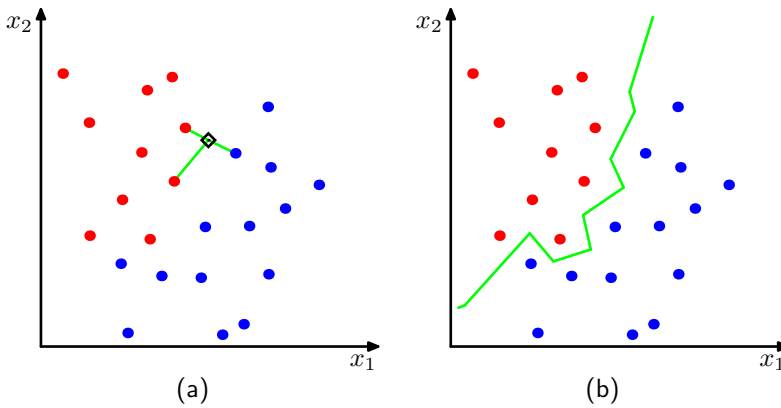
$\delta$ NN—instead of the  $k$  nearest, use as many as fit in a ball of radius  $\delta$ .

How does one train such an algorithm?

Despite its simplicity,  $k$ NN is a really really good classifier. (But very sensitive to irrelevant features.)

Can be used also for regression, by averaging responses.

(See figs 2.27a,b and 2.28a,b,c in PRML.)



$k$ NN algorithms are embarrassingly effective: they just work really really well for a large number of problem. Of course, if you have a huge number of irrelevant features, they don't do as well. One way around this is feature scaling, which attempts to change the weight of each dimension so as to down-weight irrelevant features.

**Locally Weighted Regression**

LWR can be seen as a hybrid between linear regression and nearest-neighbor algorithms. The idea is to do regression, but to define the error function only over “nearby” points (where nearby is specified by some kernel, typically Gaussian). The nice thing about LWR is that (if formulated right) prediction only depends on nearest neighbors, and not on the entire data set. But it still requires you to be able to compute nearest neighbors for test points efficiently, which is difficult.

---

### **RBF networks**

This is basically the same as LWR. The idea is to plop a bunch of Gaussians down in space and to LWR to them, rather than to nearest neighbors. This can be seen as a particular form of a neural network, where our sigmoid activation is replaced by a kernel (typically Gaussian). We still get non-linearity (because the kernel is non-linear), but they’re much easier to train once the kernel locations and widths are fixed.

A simple RBF network is when there is one hidden unit for every data point and the kernel is Gaussian. The big problem here is the computational complexity of prediction – it scales linearly in the number of training points.