Append.

Discriminant analysis aims at the classification of an object into one of K given classes based on information from a set of p predictor variables. A classic example is Fisher's Irish data where measurements of Petal Width , Petal Length , Sepal Width, Sepal Length are used to classify three iris species , Versicolor, Verginica and Setosa. The training data consist of 50 samples from each species. Discriminant analysis is also called classification or supervised learning in engineering applications such as character recognition, medical diagnosis, problems

1. Bayesian setting. Suppose there are K classes. The prior distribution on how likely a case will come from class i is denoted by π_i, $I = 1, \cdots, K$. The density function of the predictor $\mathbf{x}$ (typically multivariate) for cases from class i is denoted by $f_i(\mathbf{x})$. Then the Bayes classification rule is to assign class membership to a case with predictor $\mathbf{x}$ by maximizing the posterior probability (c.f. a diagram illustration), or equivalently,

$$\max_i \pi_i f_i(\mathbf{x})$$

Quite often, the true density $f_i(\mathbf{x})$ is unknown. Thus to obtain a Bayes rule, we need some density estimation methods for approximating $f_i(\mathbf{x})$. Kernel estimate is a popular method for nonparametric density estimation. The basic idea is quite simple.

$$\hat{f}_i(\mathbf{x}) = n_i^{-1} \sum_j \phi(\mathbf{x} - \mathbf{x}_j)$$

where the sum is over every case j in the training sample that comes from i's class. $\phi(\cdot)$ can be any density function (normal density, for example). Here I purposively absorb a scaling factor (called bandwidth or widow size) into ϕ. This scaling factor (standard deviation in the normal case) is unfortunately hard to specify and has been the focus of attention in most literature. All theoretical results have pointed out that as the sample size tends to infinity, this scaling factor has to converge to zero in such and such rates. This is because

$$E\hat{f}_i(\mathbf{x}) = \int \phi(\mathbf{x} - u) f_i(u) du = \int f_i(\mathbf{x} - v)\phi(v) dv$$

So unless the kernel function $\phi_(u)$ is concentrated at one point 0 (the so called delta function), there will be a bias. Delta function is usually approximated by rescaling : $\phi(t/h)h^{-1}$, with small h. The other balancing factor is the variance of the estimator (with the scaling parameter h),

$$var(\hat{f}_i(\mathbf{x})) = n_i^{-1} h^{-2} var(\phi((\mathbf{x} - \mathbf{x}_j)/h)$$

Using Taylor's expansion, the leading term of $var(\phi((\mathbf{x}_- \mathbf{x}_j)/h)$ can be shown to be equal to $h f_i(\mathbf{x}) \int \phi(v)^2 dv$. Thus the variance of the estimator is of order $1/nh$. This sets up a theoretical limit on how small h can be, if we want to maintain the property of "consistency". Yet, for a fixed sample size, it is hard to tell exactly how small the scaling factor should be. Still this asymptotic variance formula provide us a useful insight.

There are several automatic ways of choosing h : (1)cross-validation method, (2) plug-in method based on variance-bias calculation.

2. Nearest neighbor. To apply this method, when a new case arrives, we first find a case , called the nearest neighbor in the training sample that is closest to the arrival case. We then classify the new case according to the nearest neighbor's class membership.

It has been found that this method is quite unstable. An improved more stable version is to find k-nearest neighbor (where k is any fixed integer) and then classify by the majority rule.

Two problems : 1. how to choose k ; 2. How to define "distances"? Cross-validation can help.

3. Classification tree. This is a recursive partition method. All training samples are first assigned to one of several terminal nodes by answering a set of queries. The the final class assignment is by following the majority rule in each node. How to create the query set has been the focus of studies on tree construction. The most well-known package, called CART, uses univariate binary split. Cross-validation is used to trim down the size of a tree (i.e., avoid asking too many queries). A successful tree classifier should generate terminal nodes as pure (of one class) as possible, but without the artifact of overfitting.

4. Linear discriminant analysis.

4.1. Normal density; a little bit of geometry.

4.2. group separation

4.3. relationship with SIR.

5. Neuron network.

6. Support vector machine.

7. Bootstrapping

8. Others.