# Multivariate statistical methods and data mining in particle physics

## Lecture 3 (18 June, 2008)


Glen Cowan
RHUL Physics
**www.pp.rhul.ac.uk/~cowan**


Academic Training Lectures
CERN
16–19 June, 2008

# Outline

Statement of the problem

Brief review of statistical formalism

Some general considerations

Multivariate classifiers:

- Linear discriminant function
- Neural networks
- Naive Bayes classifier

Wednesday start

- Density estimation methods
- Decision trees
- Support Vector Machines

Other multivariate problems:

- Multivariate regression
- Unsupervised learning

# Lectures etc. on web

The lectures (slides + video) are on

**indico.cern.ch/conferenceDisplay.py?confId=24827**

The programs used to create the simple TMVA example shown yesterday are on

**www.pp.rhul.ac.uk/~cowan/stat/root/tmva/**

These are standalone C++ programs that use TMVA classes.

# Probability density estimation methods

If we could estimate the pdfs $p(\boldsymbol{x}|H_0)$, $p(\boldsymbol{x}|H_1)$ for the classes of events we want to separate, then we could form the optimal discriminating function from their ratio:

$$y(\vec{x}) = \frac{p(\vec{x}|H_0)}{p(\vec{x}|H_1)}$$

So the problem reduces to estimating the joint pdfs $p(\boldsymbol{x})$. We may choose different methods for numerator and denominator.

Methods for estimating pdfs can be

parametric, i.e., we have a function $p(\vec{x}; \theta_1, \ldots, \theta_m)$

non-parametric, i.e., model independent (e.g. histogram, ...); also contain parameters but they are "local"; not rigidly tied to any model.

# Beyond Naive Bayes

Recall that in the naive Bayes approach we approximated the $n$-dimensional joint pdf as the product of the marginal densities:

$$p(\vec{x}) \approx \prod_{i=1}^{n} p_i(x_i)$$

So the problem is reduced to estimating the one-dimensional marginal pdfs $p_i(x_i)$, usually straightforward.

But this does not capture the higher order nonlinearities of $p(\boldsymbol{x})$.

# Lancaster models

Lancaster models approximate an $n$-dimensional joint pdf $p(\boldsymbol{x}) = p(x_1,\ldots,x_n)$ in terms of the marginal distributions for up to a certain number $s$ of the $n$ components.

For $s = 1$ this was Naive Bayes:. For e.g. $s = 2$ we approximate $p(\boldsymbol{x})$ in terms of one- and two-dimensional marginal densities $p_i(x_i)$ and $p_{ij}(x_i,x_j)$ as

$$p(\vec{x}) \approx \left[ \sum_{i,j,i<j} \frac{p_{ij}(x_i, x_j)}{p_i(x_i)\, p_j(x_j)} - \left[ \binom{n}{2} - 1 \right] \right] \prod_{i=1}^{n} p_i(x_i)$$

This will not capture the full nonlinear structure of $p(\boldsymbol{x})$ but goes further in that direction than assuming independence. (cf. Webb Ch. 3)

# Parametric density estimation

If we have a parametric function for one or both of the densities,

$$p(\vec{x};\theta_1,\ldots,\theta_m)$$

then we can estimate the parameters using the training data with e.g. the method of maximum likelihood, i.e., choose the parameter estimates $\hat{\theta}_1,\ldots,\hat{\theta}_m$ to be the values that maximize the likelihood function:

$$L(\theta_1,\ldots,\theta_m)=\prod_{i=1}^{N} p(\vec{x}_i;\theta_1,\ldots,\theta_m)$$

Product over all training events (assumes events statistically independent, not strictly true if we have multiple candidate "events" per collision event).

Finally simply take

$$\hat{p}(\vec{x})=p(\vec{x};\hat{\theta}_1,\ldots,\hat{\theta}_m)$$

Function evaluation generally fast, storage requirements low.

# Parametric density estimation (2)

The number of parameters in a reasonable model is usually much smaller than the corresponding number of degrees of freedom in nonparametric methods, so a parametric estimate of the pdf will have higher statistical accuracy for a given amount of training data. Trade off:

few parameters: model not flexible and may not describe data, but parameters accurately determined.

many parameters: model flexible enough to describe the true pdf, but parameter estimates have large statistical errors.

Even if a full parametric model is not available, $p(\boldsymbol{x})$ may (approximately) factorize into a parametric part for a subset of the variables:

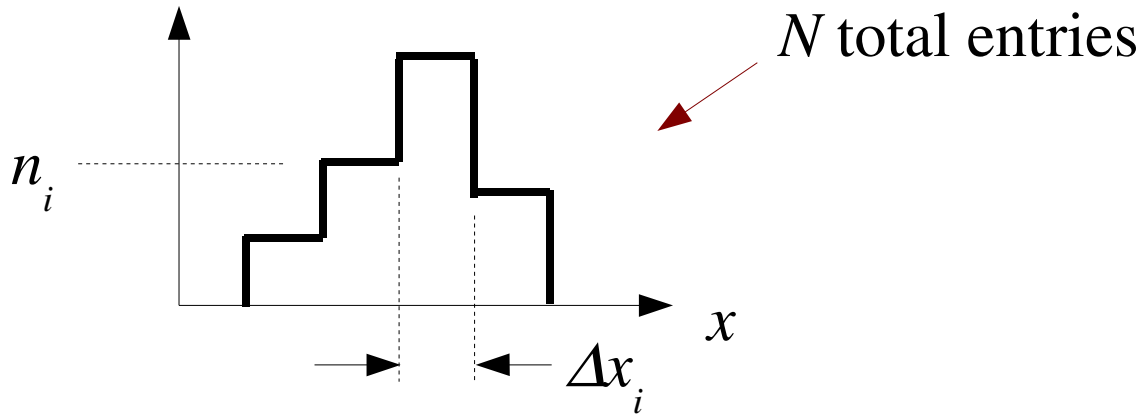
$$p(x_1,\ldots,x_n) = p(x_1,\ldots,x_s;\theta_1,\ldots,\theta_m)\,q(x_{s+1},\ldots,x_n)$$

so we can mix parametric and non-parametric methods.

# Histograms

Start by considering one-dimensional case, goal is to estimate pdf $p(x)$ of continuous r.v. $x$.

Simplest non-parametric estimate of $p(x)$ is a histogram:




$$\hat{p}(x) = \frac{n_i}{N \Delta x_i} \quad \text{for } x \text{ in bin } i$$

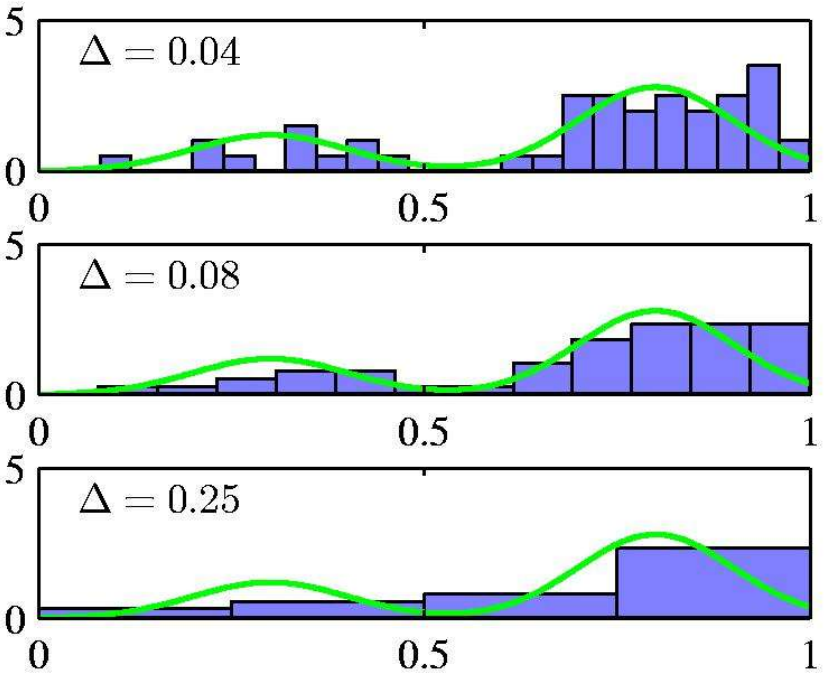
Bishop Section 2.5

# Histograms (2)

Small bin width: estimate is very spiky, structure not really part of underlying distribution.

Medium bin width: best

Large bin width: too smooth and thus fails to capture e.g. bimodal character of parent distribution




Bishop Section 2.5

# Histograms (3)

Advantages: once histogram computed, the data can be discarded.

Disadvantage: discontinuities at bin edges, scaling with dimensionality.

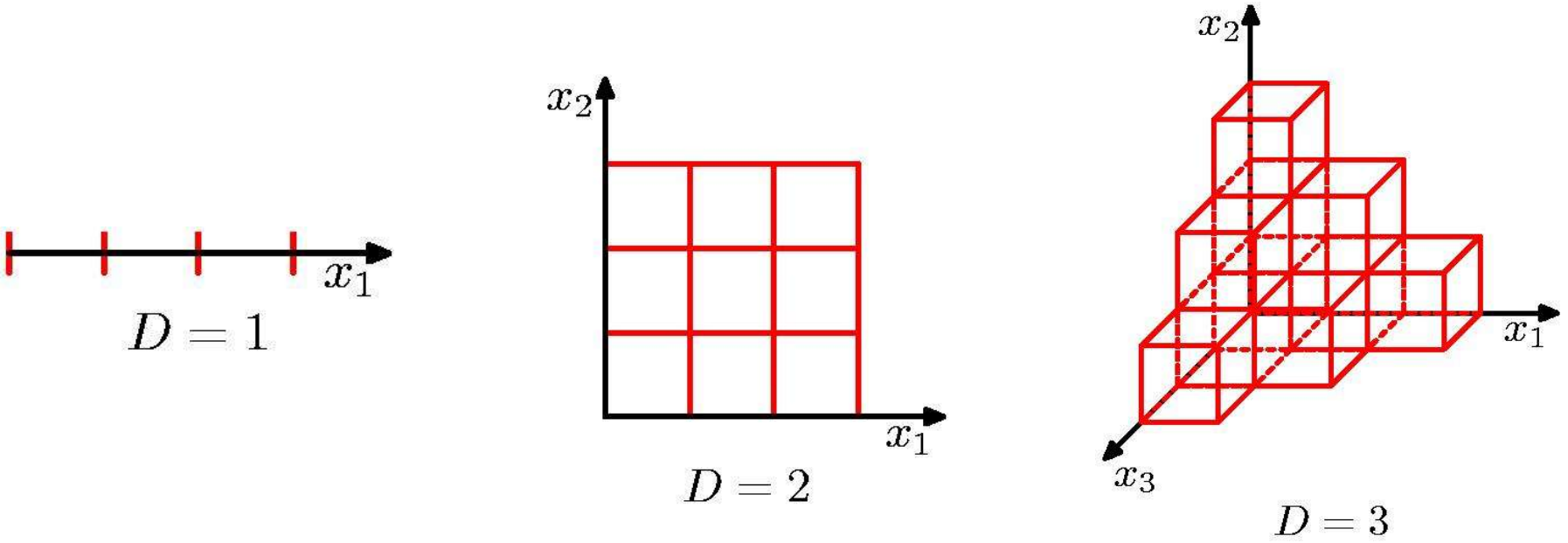
In general we can do much better than histograms, but they still show important features common to many methods:

To estimate pdf at $\boldsymbol{x}$, = $(x_1, ..., x_D)$ we should count the number of events in some local neighbourhood near $\boldsymbol{x}$ (requires definition of "local", i.e., a distance measure, e.g., Euclidian).

The bin width $\Delta x$ plays the role of a smoothing parameter defining the size of the local neighbourhood. If it is too large, local structure is washed out; too small, and the estimate is subject to statistical fluctuations.

# The curse of dimensionality

The difficulty in determining the density in a high-dimensional histogram is an example of the "curse of dimensionality" (Bellman, 1961).




The number of cells in a *D*-dimensional histogram grows exponentially.

# Counting events in a local volume

Consider a small volume $V$ centred about $\boldsymbol{x} = (x_1, \ldots, x_D)$.

This is in contrast to the histogram where the bin edges were fixed.

Suppose from $N$ total events we find $K$ in $V$.

Take as estimate for $p(\boldsymbol{x})$ $\quad \hat{p}(\boldsymbol{x}) = \dfrac{K}{N\,V}$

Two approaches:

Fix $V$ and determine $K$ from the data

Fix $K$ and determine $V$ from the data

# Kernels

E.g. take $V$ to be hypercube centered at the $\boldsymbol{x}$ where we want $p(\boldsymbol{x})$.

Define $k(\boldsymbol{u}) = 1 \text{ for } |u_i| \leqslant 1/2$ and 0 otherwise, $i = 1, \ldots, D$

i.e., the function is nonzero inside a unit hypercube centred about $\boldsymbol{x}$ and zero outside.

$k(\boldsymbol{u})$ is an example of a kernel function (here called a Parzen window).

# Kernel density estimators

$\boldsymbol{x}$ where we want estimate

$\boldsymbol{x}$ of $i$th training event

Estimate $p(\boldsymbol{x})$ using $\hat{p}(\boldsymbol{x}) = \frac{1}{N h^D} \sum_{i=1}^{N} k\left( \frac{\boldsymbol{x} - \boldsymbol{x}_i}{h} \right)$

side of hypercube

where we used $V = h^D$ for the volume of the hypercube.

Thus the estimate at $\boldsymbol{x}$ is the obtained from the sum of $N$ hypercubes, one centred about each of the data points $\boldsymbol{x}_i$.

This is an example of a kernel density estimator (KDE or Parzen estimator).

# Gaussian KDE

The Parzen window KDE has discontinuities at the edges of the hypercubes; we can avoid these with a smoother kernel function e.g., Gaussian:

$$\hat{p}(x) = \frac{1}{N}\sum_{i=1}^{N} \frac{1}{(2\pi h^2)^{1/2}} \exp\left(\frac{-\|\vec{x}-\vec{x}_i\|^2}{2h^2}\right)$$

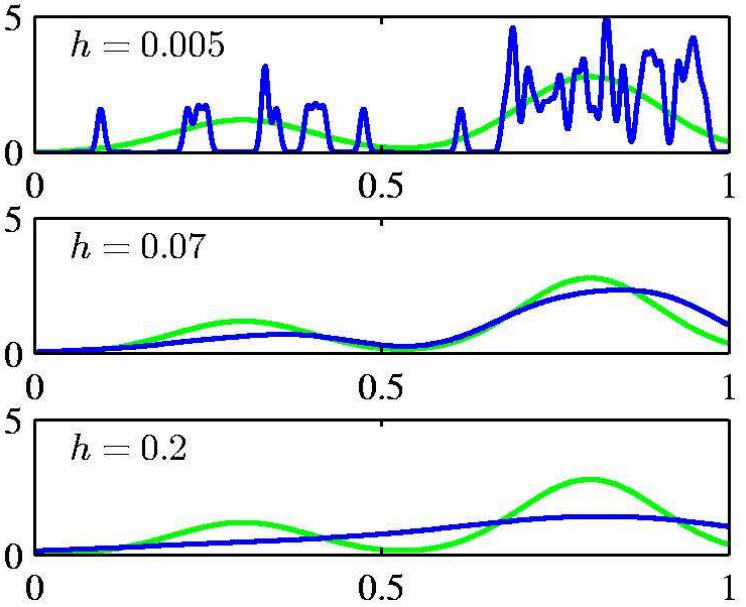
That is, to estimate $p(\boldsymbol{x})$:

Place a Gaussian of standard deviation $h$ centred about each training data point;

At a given $\boldsymbol{x}$, add up the contribution from all the Gaussians and divide by $N$.

# Gaussian KDE, choice of $h$

The Gaussian KDE shows the same basic issues as did the histogram:




Bishop Section 2.5

# KDE – general

We can choose any kernel function $k(\boldsymbol{u})$ as long as it satisfies

$$k(\boldsymbol{u}) \geqslant 0,$$

$$\int k(\boldsymbol{u})\, d\boldsymbol{u} = 1$$

Advantage of KDE: essentially no training!

To get $p(\boldsymbol{x})$ simply compute the required sum of terms.

Disadvantages: A single function evaluation of $p(\boldsymbol{x})$ requires carrying out the sum over all events, and the entire data set must be stored.

# Expectation value of $\hat{p}(\vec{x})$

To see role of kernel, compute expectation value of $\hat{p}(\vec{x})$

$$E[\hat{p}(\boldsymbol{x})] = \frac{1}{N h^D} \sum_{i=1}^{N} E\left[ k\left( \frac{\boldsymbol{x} - \boldsymbol{x}_i}{h} \right) \right] = \frac{1}{h^D} E\left[ k\left( \frac{\boldsymbol{x} - \boldsymbol{x}_i}{h} \right) \right]$$

$$= \frac{1}{h^D} \int k\left( \frac{\boldsymbol{x} - \boldsymbol{x}'}{h} \right) p(\boldsymbol{x}')\, d\boldsymbol{x}'$$

Expectation value of the estimator $\hat{p}(\boldsymbol{x})$ is the convolution of the true density $p(x)$ with the kernel function.

For $h \to 0$ the kernel becomes a delta function, and $E[\hat{p}(\boldsymbol{x})]$ approaches the true density (zero bias). But for finite $N$ the variance $V[\hat{p}(\boldsymbol{x})]$ becomes infinite.

# Choice of $h$ using mean squared error

Suppose we knew the true $p(\boldsymbol{x})$ (or had a reference standard). We can compute the Mean Squared Error (MSE) of our estimator:

$$\mathrm{MSE}[\hat{p}(\boldsymbol{x})] = E[(\hat{p}(\boldsymbol{x}) - p(\boldsymbol{x}))^2]$$

$$= \left(E[\hat{p}(\boldsymbol{x}) - p(\boldsymbol{x})]\right)^2 + E\left[(\hat{p}(\boldsymbol{x}) - p(\boldsymbol{x}))^2\right]$$

bias squared variance

We could e.g. choose $h$ so that it minimizes the integral of the MSE over $\boldsymbol{x}$ (or maybe in some region of interest):

$$\int \mathrm{MSE}\left[\hat{p}(\boldsymbol{x})\right] d\boldsymbol{x}$$

If both the kernel and $p(\boldsymbol{x})$ are a Gaussian, this gives $h = \left(\frac{4}{3}\right)^{1/5} \sigma N^{-1/5}$

# Adaptive KDE

In the simplest form of KDE the smoothing parameter $h$ is a constant.

In regions high density (lots of events) we want small $h$ so as to not wash out structure.

In regions of low density, small $h$ would lead to statistical fluctuations in the estimate (structure not present in parent distribution).

So we may want to allow the size of the local neighbourhood over which we average to vary depending on the local density.

In sample point adaptive KDE the bandwidth becomes a function of $p(\boldsymbol{x})$:

$$h = \frac{h_0}{\sqrt{(p(\boldsymbol{x}))}}$$

# KDE boundary issues

Some components of $\boldsymbol{x}$ may have finite limits. But if we use e.g. a Gaussian kernel, then some of the probability "spills out" of the allowed range.

The probability inside the range is therefore underestimated.

One solution is to renormalize the kernel so that its integral inside the allowed range is equal to unity.

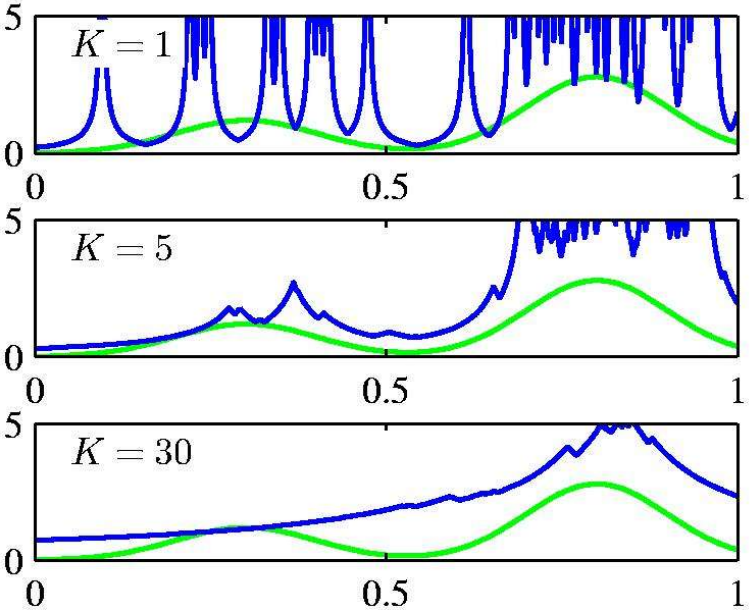
Another option is to "mirror" the distribution about the boundary. The events from outside spilling in compensate those spilling out.

# *K*-nearest neighbour method

Instead of fixing *V*, consider a fixed number of events *K*
and find the appropriate *V* such that it just contains *K* events.

The density estimate is then simply $\hat{p}(\boldsymbol{x}) = \dfrac{K}{N\,V}$

*K* plays role of smoothing parameter.
Large *K* means lower statistical error
in the estimate of the density,
e.g., *K*=100 gives 10% accuracy.
But large *K* means you need a bigger
volume, estimate is less local.




Bishop Section 2.5

Estimate of $p(\boldsymbol{x})$ is not a true pdf – integral over entire space diverges.

# *K*-nearest neighbour method

Example from TMVA manual – here the algorithm is used directly as a classifier. The event type is assigned based on majority vote within the volume.

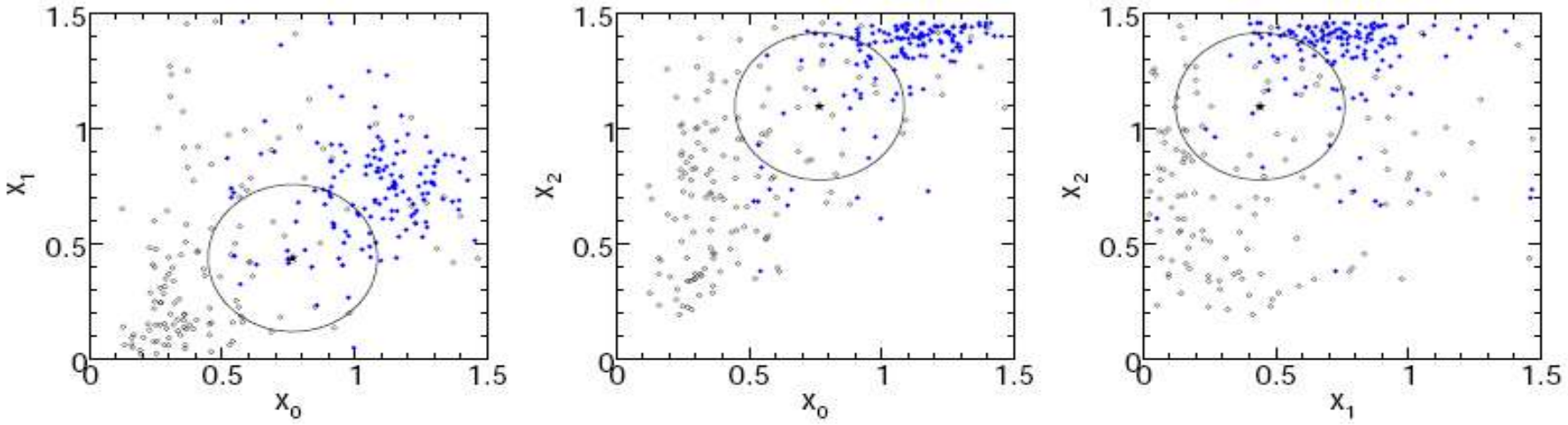



Figure 11: Example for the k-nearest neighbour algorithm in a three-dimensional space (i.e., for three discriminating input variables). The three plots are projections upon the two-dimensional coordinate planes. The full (open) circles are the signal (background) events. The k-NN algorithm searches for 20 nearest points in the *nearest neighborhood* (circle) of the query event, shown as a star. The nearest neighborhood counts 13 signal and 7 background points so that query event may be classified as a signal candidate.

# Feature normalization

*K*-NN algorithms rely on a metric in the input variable space to define the volume.

If there is a great difference in the ranges spanned by some of the variables, then they are implicitly given different weights.

Typically scale the input variables so as to give approximately equal distances between relevant features. Try e.g.

Linear scaling in unit range: $x' = \dfrac{x - x_{\min}}{x_{\max} - x_{\min}}$ $\quad (0 \leqslant x' \leqslant 1)$

Standardization: $x' = \dfrac{x - \mu}{\sigma}$ (zero mean, unit variance)

# Curse of dimensionality for *K*-NN

Suppose out data are uniformly distributed in a *D*-dimensional unit cube.

We want a "small" volume to capture a fraction $r = K/N$ of the events.

Make a hypercube local neighbourhood with side $e$, volume $e^D$

Its volume fraction is $r = e^D$

The side of an edge of the small volume is $e = r^{1/D}$

For e.g. $r$=0.001 and $D$ = 30 the side of the "neighborhood" is 0.8, almost the entire range of the input.

(cf. Hastie, Tibshirani and Friedman p 23.)

# Lecture 3 summary

Many important classification methods rely on obtaining estimates of the multivariate probability densities. We can use e.g.

Histograms

Kernel density estimation (parzen window)

*K*-nearest neighbour

Both KDE and *K*-NN methods require the entire data set to be retained, and all become problematic in higher dimensions.

But if the pdfs are not too nonlinear, the density estimation can be done in lower dimension (Naive Bayes, Lancaster models).

# Extra slides

# *K*, *V* vs. data sample size

Take the volume smaller and the number of events *K* larger if the total size of the data sample is can be increased, since statistical error in estimate scales as $1/\sqrt{N}$.