Statistical Methods for Particle Physics

Lecture 2: multivariate methods

http://indico.ihep.ac.cn/event/4902/

iSTEP 2015
Shandong University, Jinan
August 11-19, 2015

Glen Cowan (谷林·科恩)
Physics Department
Royal Holloway, University of London
g.cowan@rhul.ac.uk
www.pp.rhul.ac.uk/~cowan

Outline

Lecture 1: Introduction and review of fundamentals

- Probability, random variables, pdfs
- Parameter estimation, maximum likelihood
- Statistical tests for discovery and limits

→ Lecture 2: Multivariate methods

- Neyman-Pearson lemma
- Fisher discriminant, neural networks
- Boosted decision trees

Lecture 3: Systematic uncertainties and further topics

- Nuisance parameters (Bayesian and frequentist)
- Experimental sensitivity
- The look-elsewhere effect

Resources on multivariate methods

C.M. Bishop, Pattern Recognition and Machine Learning, Springer, 2006

T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning, 2nd ed., Springer, 2009

R. Duda, P. Hart, D. Stork, Pattern Classification, 2nd ed., Wiley, 2001

A. Webb, Statistical Pattern Recognition, 2nd ed., Wiley, 2002.

Ilya Narsky and Frank C. Porter, *Statistical Analysis Techniques in Particle Physics*, Wiley, 2014.

朱永生（编著），实验数据多元统计分析，科学出版社，北京，2009。

Software

TMVA, Höcker, Stelzer, Tegenfeldt, Voss, Voss, physics/0703039

From **tmva.sourceforge.net**, also distributed with ROOT

Variety of classifiers

Good manual

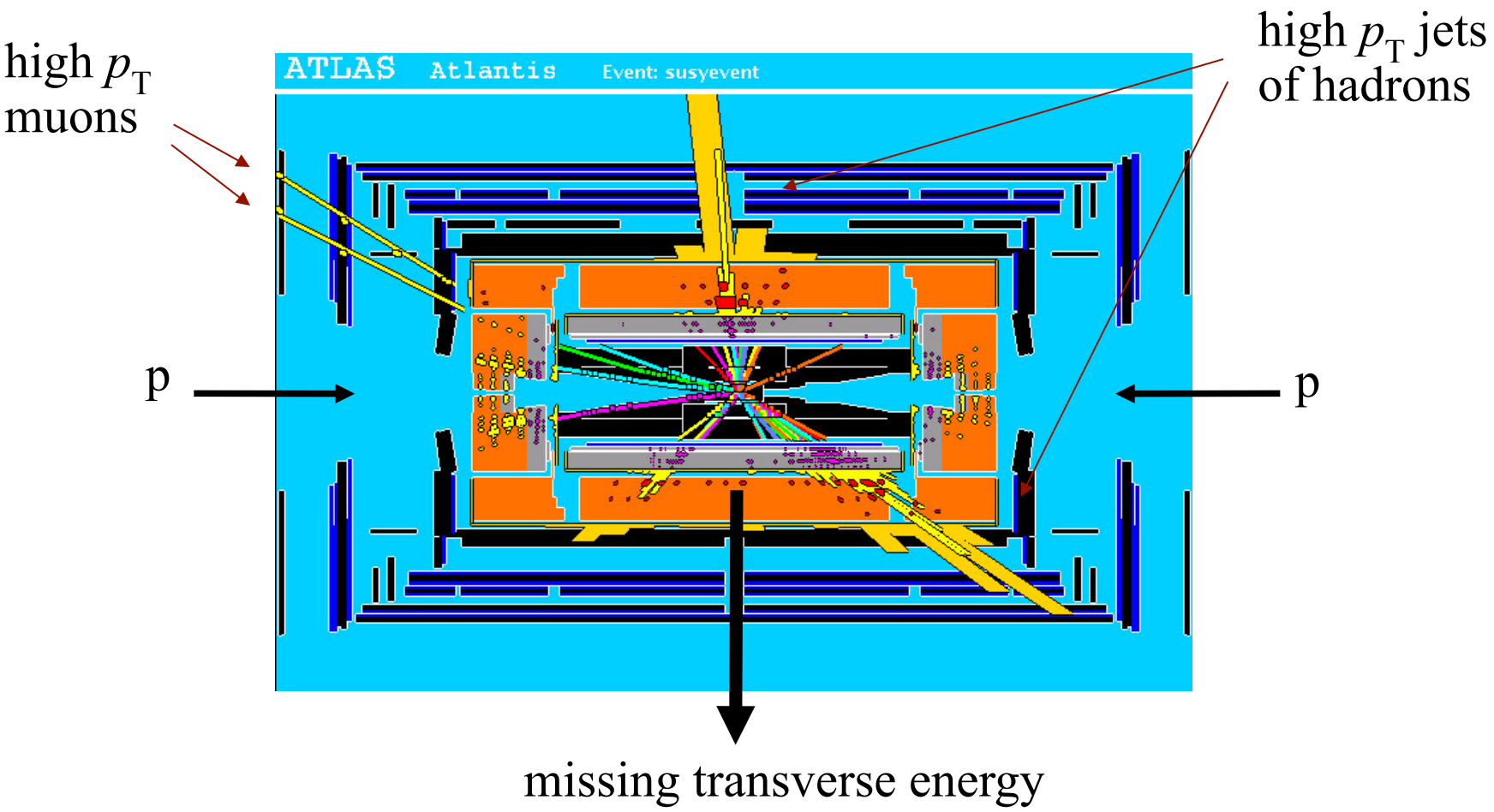
StatPatternRecognition, I. Narsky, physics/0507143

Further info from **statpatrec.sourceforge.net**

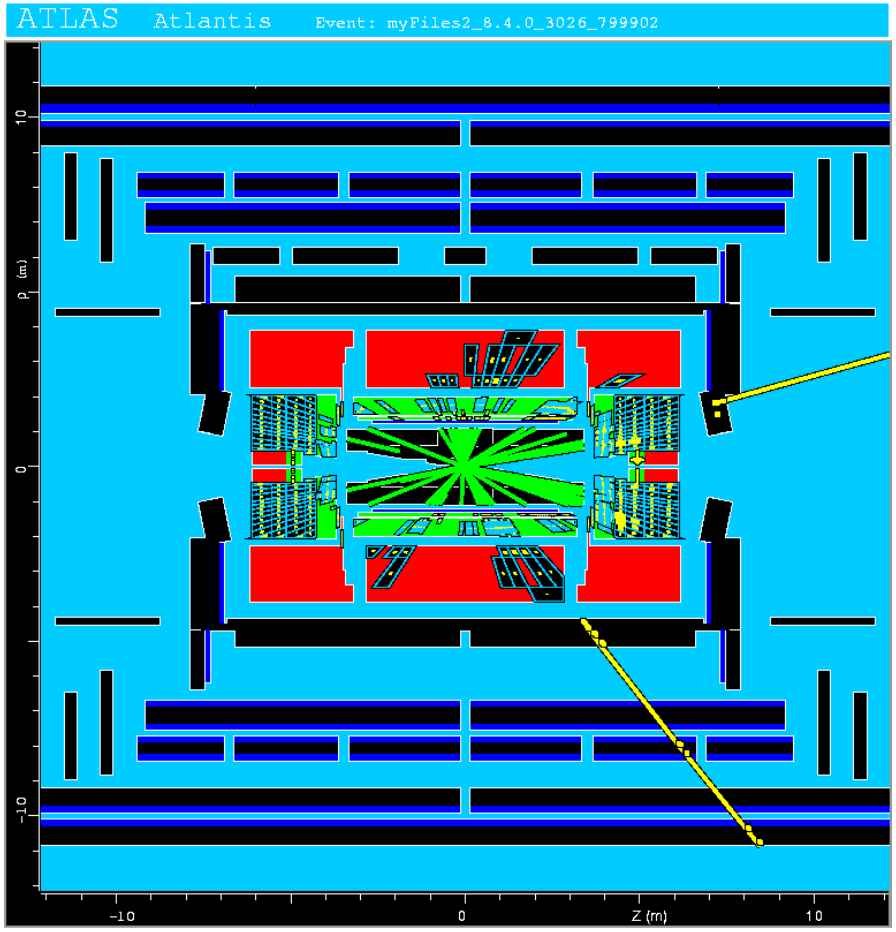
Also wide variety of methods, many complementary to **TMVA**

Future support for project not clear.

A simulated SUSY event in ATLAS



Background events



This event from Standard Model ttbar production also has high p_T jets and muons, and some missing transverse energy.

→ can easily mimic a SUSY event.

Defining a multivariate critical region

For each event, measure, e.g.,

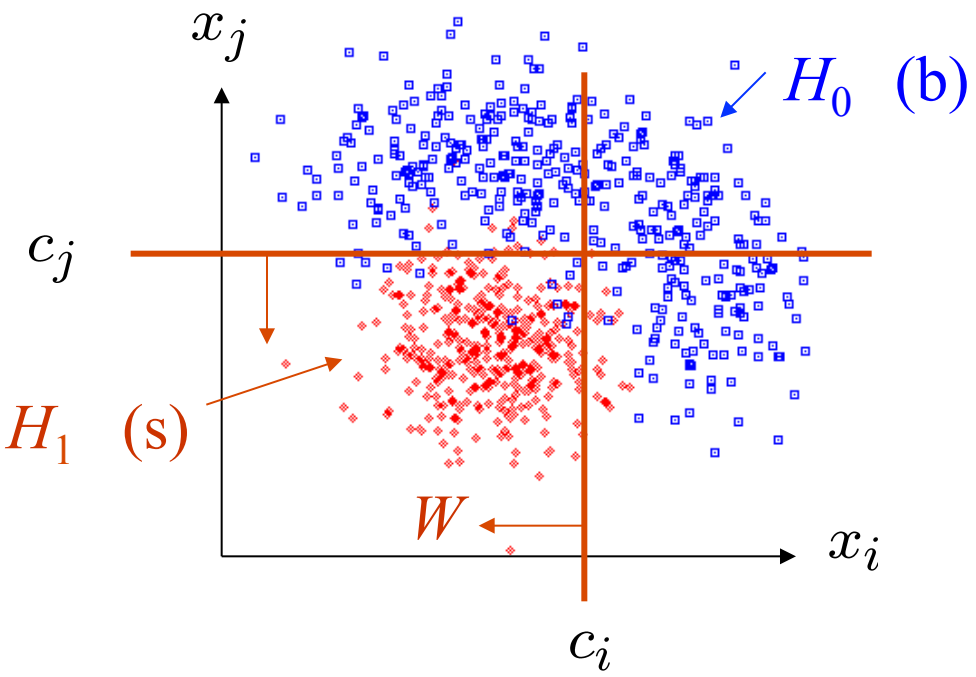
x_1 = missing energy, x_2 = electron p_T, x_3 = ...

Each event is a point in n-dimensional $\boldsymbol{x}$-space; critical region is now defined by a 'decision boundary' in this space.
What is best way to determine the boundary?

Perhaps with 'cuts':

$$x_i < c_i$$

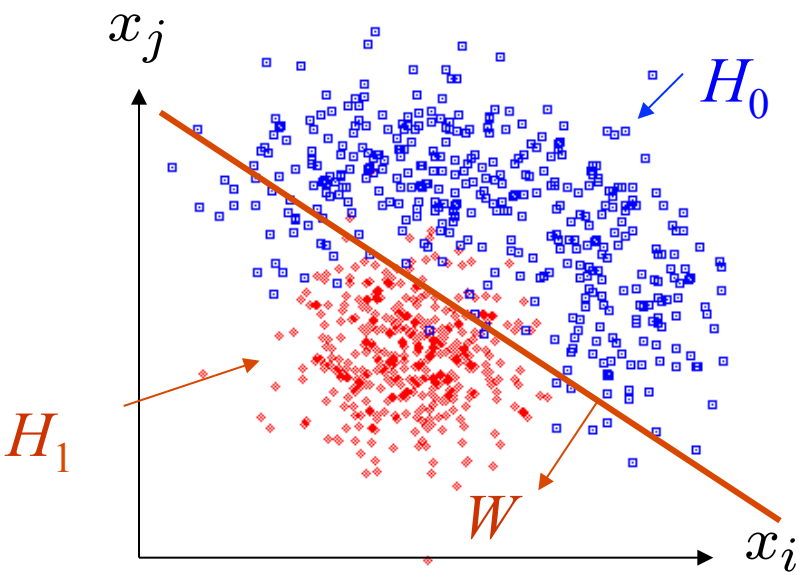
$$x_j < c_j$$



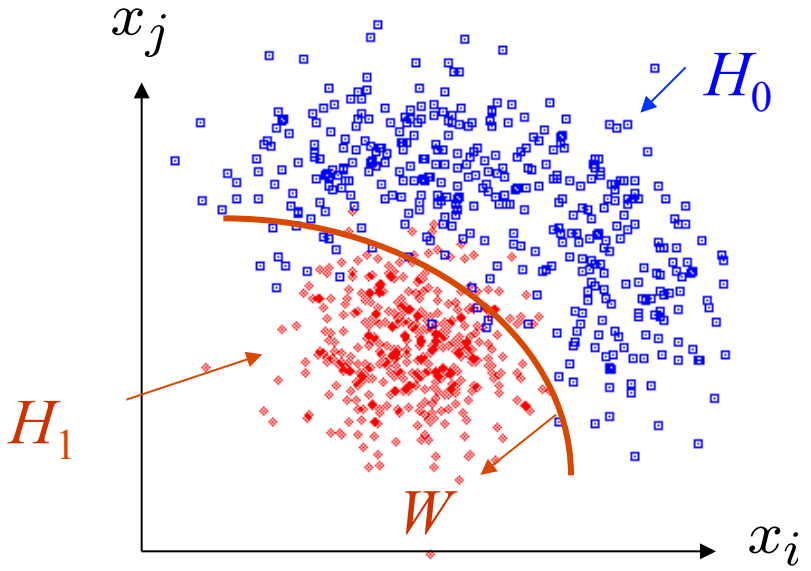
Other multivariate decision boundaries

Or maybe use some other sort of decision boundary:

linear



or nonlinear



Multivariate methods for finding optimal critical region have become a Big Industry (neural networks, boosted decision trees,...), benefitting from recent advances in Machine Learning.

Test statistics

The boundary of the critical region for an n-dimensional data space $\boldsymbol{x} = (x_1,..., x_n)$ can be defined by an equation of the form

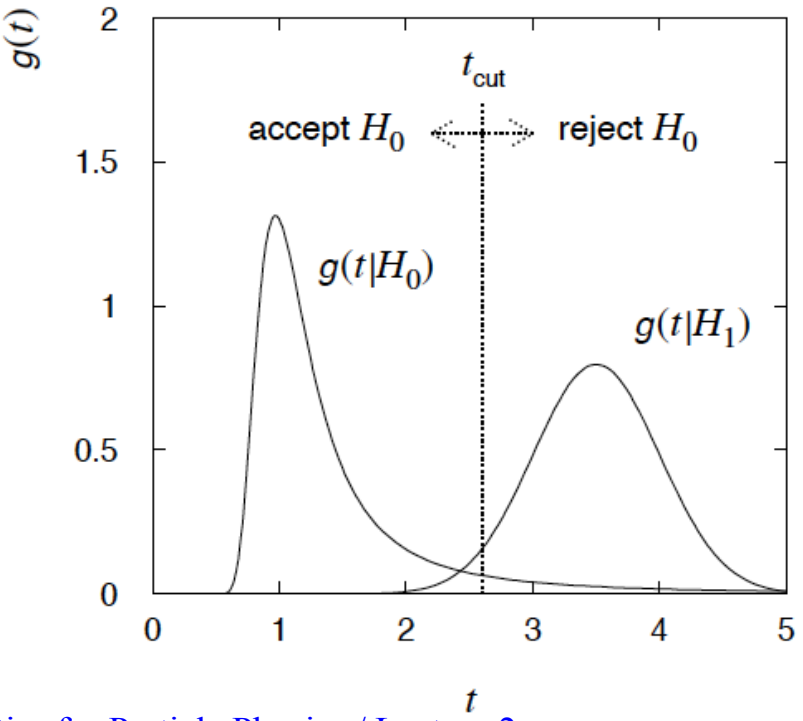
$$t(x_1, \ldots, x_n) = t_{\text{cut}}$$

where $t(x_1,\ldots, x_n)$ is a scalar test statistic.

We can work out the pdfs $g(t|H_0),\ g(t|H_1),\ \ldots$

Decision boundary is now a single 'cut' on t, defining the critical region.

So for an n-dimensional problem we have a corresponding 1-d problem.



Test statistic based on likelihood ratio

How can we choose a test's critical region in an 'optimal way'?

Neyman-Pearson lemma states:

To get the highest power for a given significance level in a test of H_0, (background) versus H_1, (signal) the critical region should have

$$\frac{f(\mathbf{x}|H_1)}{f(\mathbf{x}|H_0)} > c$$

inside the region, and $\leq c$ outside, where c is a constant chosen to give a test of the desired size.

Equivalently, optimal scalar test statistic is

$$t(\mathbf{x}) = \frac{f(\mathbf{x}|H_1)}{f(\mathbf{x}|H_0)}$$

N.B. any monotonic function of this is leads to the same test.

Classification viewed as a statistical test

Probability to reject H_0 if true (type I error): $\alpha = \int_W f(\mathbf{x}|H_0)d\mathbf{x}$

α = size of test, significance level, false discovery rate

Probability to accept H_0 if H_1 true (type II error) $\beta = \int_{\overline{W}} f(\mathbf{x}|H_1)d\mathbf{x}$

$1 - \beta$ = power of test with respect to H_1

Equivalently if e.g. H_0 = background, H_1 = signal, use efficiencies:

$$\varepsilon_\mathrm{b} = \int_W f(\mathbf{x}|H_0) = \alpha$$

$$\varepsilon_\mathrm{s} = \int_W f(\mathbf{x}|H_1) = 1 - \beta = \text{power}$$

Purity / misclassification rate

Consider the probability that an event of signal (s) type classified correctly (i.e., the event selection purity),

Use Bayes' theorem:

Here W is signal region

ε_s

prior probability

$$P(\mathrm{s}|\mathbf{x} \in W) = \frac{P(\mathbf{x} \in W|\mathrm{s})P(\mathrm{s})}{P(\mathbf{x} \in W|\mathrm{s})P(\mathrm{s}) + P(\mathbf{x} \in W|\mathrm{b})P(\mathrm{b})}$$

ε_b

posterior probability = signal purity

= 1 – signal misclassification rate

Note purity depends on the prior probability for an event to be signal or background as well as on s/b efficiencies.

Neyman-Pearson doesn't usually help

We usually don't have explicit formulae for the pdfs $f(\boldsymbol{x}|\mathrm{s})$, $f(\boldsymbol{x}|\mathrm{b})$, so for a given $\boldsymbol{x}$ we can't evaluate the likelihood ratio

$$t(\mathbf{x}) = \frac{f(\mathbf{x}|s)}{f(\mathbf{x}|b)}$$

Instead we may have Monte Carlo models for signal and background processes, so we can produce simulated data:

$$\text{generate } \boldsymbol{x} \sim f(\boldsymbol{x}|\mathrm{s}) \quad \rightarrow \quad \boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$$

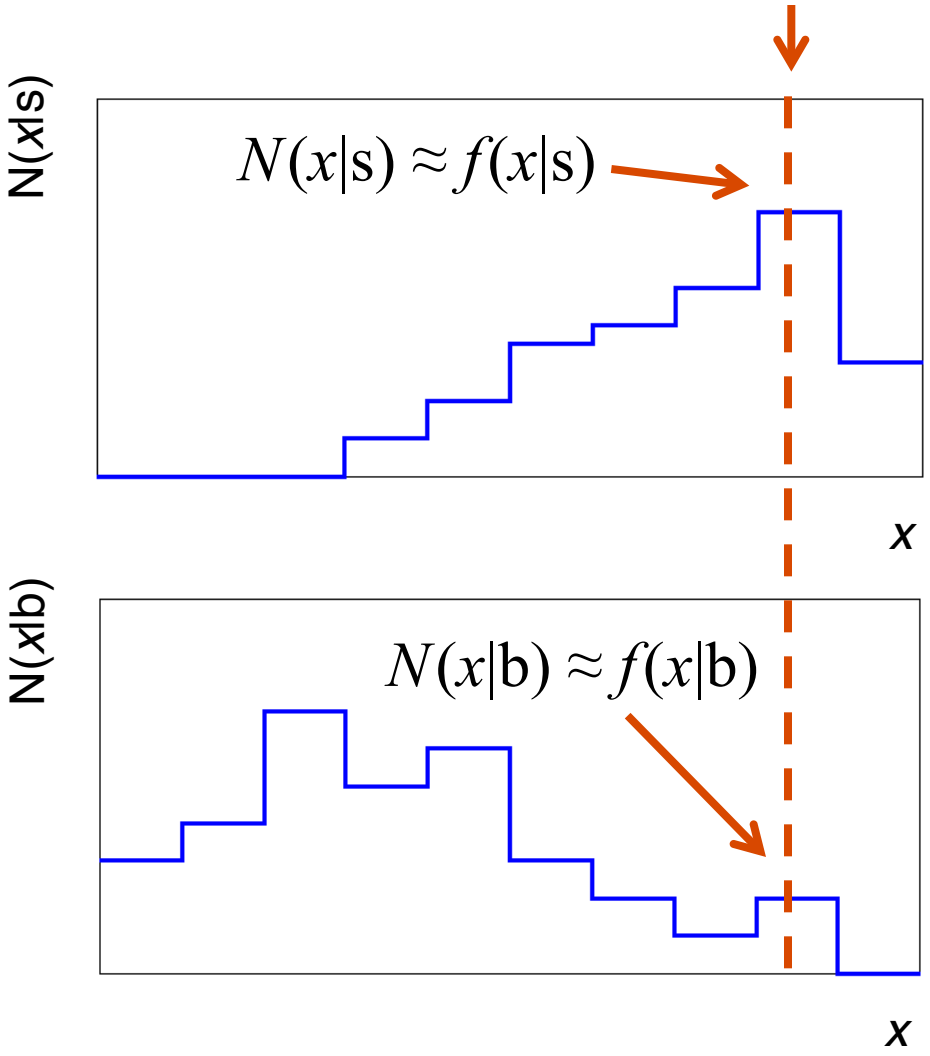
$$\text{generate } \boldsymbol{x} \sim f(\boldsymbol{x}|\mathrm{b}) \quad \rightarrow \quad \boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$$

This gives samples of "training data" with events of known type.

Can be expensive (1 fully simulated LHC event ~ 1 CPU minute).

Approximate LR from histograms

Want $t(x) = f(x|\mathrm{s})/f(x|\mathrm{b})$ for x here



One possibility is to generate MC data and construct histograms for both signal and background.

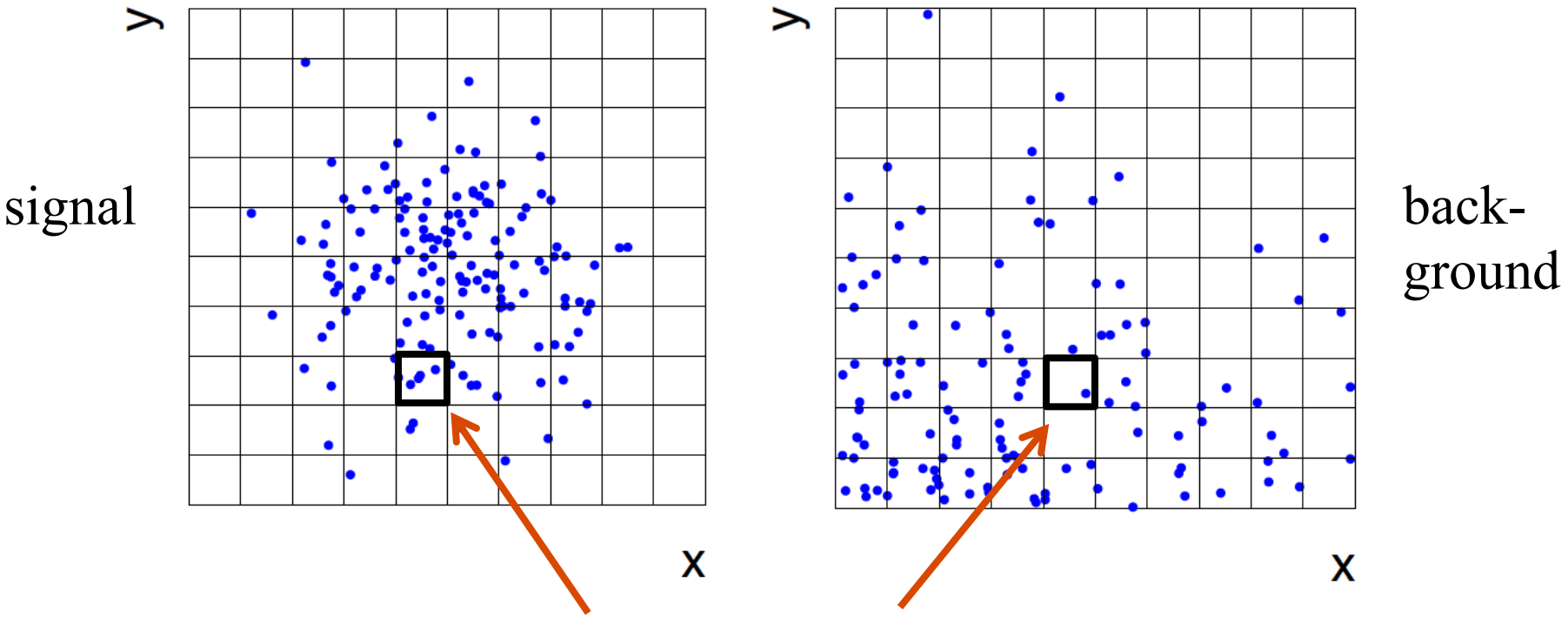
Use (normalized) histogram values to approximate LR:

$$t(x) \approx \frac{N(x|s)}{N(x|b)}$$

Can work well for single variable.

Approximate LR from 2D-histograms

Suppose problem has 2 variables. Try using 2-D histograms:



Approximate pdfs using $N(x,y|\mathrm{s})$, $N(x,y|\mathrm{b})$ in corresponding cells.

But if we want M bins for each variable, then in n-dimensions we have M^n cells; can't generate enough training data to populate.

→ Histogram method usually not usable for $n > 1$ dimension.

Strategies for multivariate analysis

Neyman-Pearson lemma gives optimal answer, but cannot be used directly, because we usually don't have $f(\boldsymbol{x}|\mathrm{s})$, $f(\boldsymbol{x}|\mathrm{b})$.

Histogram method with M bins for n variables requires that we estimate M^n parameters (the values of the pdfs in each cell), so this is rarely practical.

A compromise solution is to assume a certain functional form for the test statistic $t(\boldsymbol{x})$ with fewer parameters; determine them (using MC) to give best separation between signal and background.

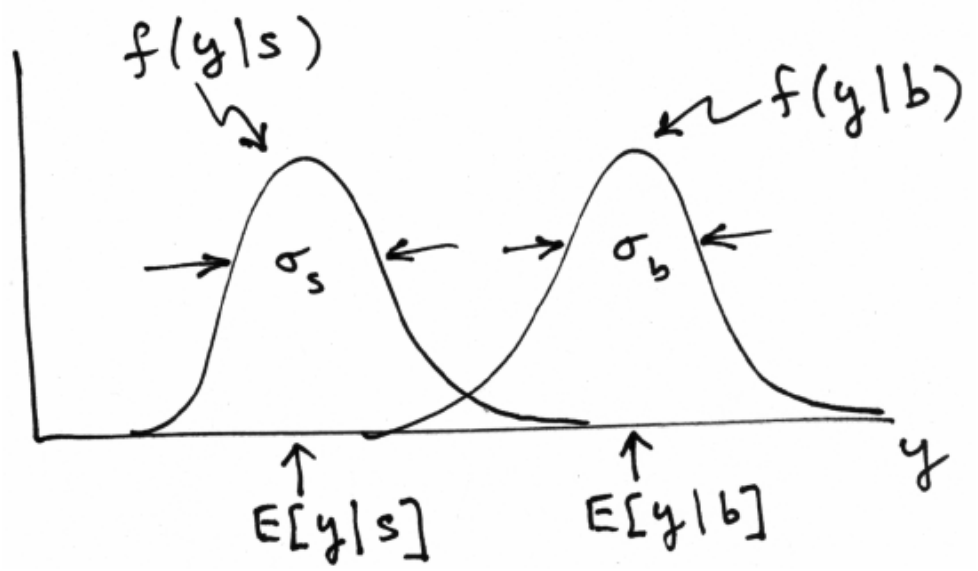
Alternatively, try to estimate the probability densities $f(\boldsymbol{x}|\mathrm{s})$ and $f(\boldsymbol{x}|\mathrm{b})$ (with something better than histograms) and use the estimated pdfs to construct an approximate likelihood ratio.

Linear test statistic

Suppose there are n input variables: $\boldsymbol{x} = (x_1, \ldots, x_n)$.

Consider a linear function: $$y(\mathbf{x}) = \sum_{i=1}^{n} w_i x_i$$

For a given choice of the coefficients $\boldsymbol{w} = (w_1, \ldots, w_n)$ we will get pdfs $f(y|\mathrm{s})$ and $f(y|\mathrm{b})$:



Linear test statistic

Fisher: to get large difference between means and small widths for $f(y|\mathrm{s})$ and $f(y|\mathrm{b})$, maximize the difference squared of the expectation values divided by the sum of the variances:

$$J(\mathbf{w}) = \frac{(E[y|s] - E[y|b])^2}{V[y|s] + V[y|b]}$$

Setting $\partial J / \partial w_i = 0$ gives for $\boldsymbol{w} = (w_1, \ldots w_n)$:

$$\mathbf{w} \propto W^{-1}(\boldsymbol{\mu}_\mathrm{b} - \boldsymbol{\mu}_\mathrm{s})$$

$$W_{ij} = \mathrm{cov}[x_i, x_j|\mathrm{s}] + \mathrm{cov}[x_i, x_j|\mathrm{b}]$$

$$\mu_{i,\mathrm{s}} = E[x_i|s], \qquad \mu_{i,\mathrm{b}} = E[x_i|b]$$

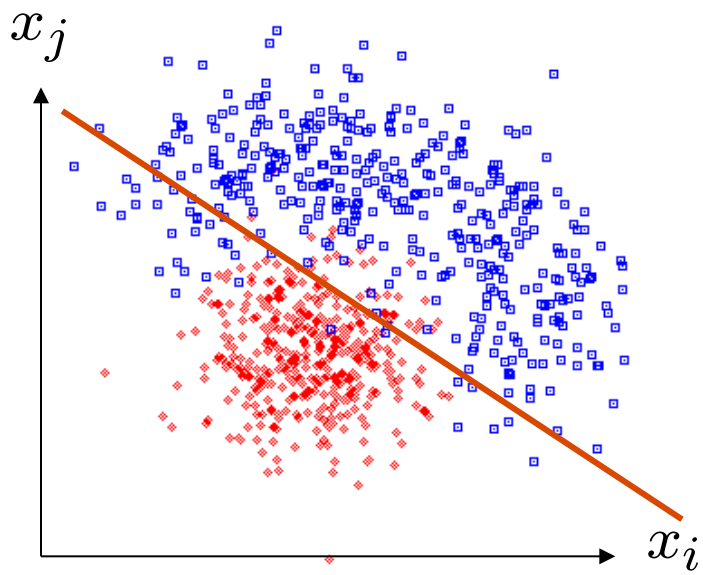
The Fisher discriminant

The resulting coefficients w_i define a Fisher discriminant.

Coefficients defined up to multiplicative constant; can also add arbitrary offset, i.e., usually define test statistic as

$$y(\mathbf{x}) = w_0 + \sum_{i=1}^{n} w_i x_i$$

Boundaries of the test's critical region are surfaces of constant $y(\boldsymbol{x})$, here linear (hyperplanes):



Fisher discriminant for Gaussian data

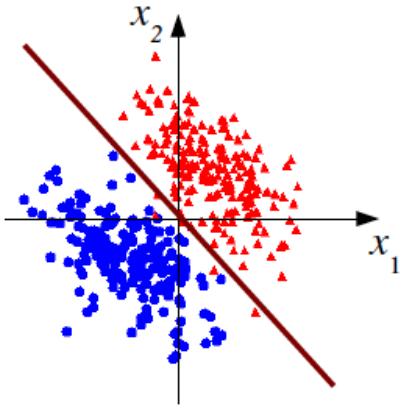
Suppose the pdfs of the input variables, $f(\boldsymbol{x}|\mathrm{s})$ and $f(\boldsymbol{x}|\mathrm{b})$, are both multivariate Gaussians with same covariance but different means:

$$f(\boldsymbol{x}|\mathrm{s}) = \mathrm{Gauss}(\boldsymbol{\mu}_\mathrm{s}, V)$$

$$f(\boldsymbol{x}|\mathrm{b}) = \mathrm{Gauss}(\boldsymbol{\mu}_\mathrm{b}, V)$$

Same covariance

$V_{ij} = \mathrm{cov}[x_i, x_j]$



In this case it can be shown that the Fisher discriminant is

$$y(\mathbf{x}) \sim \ln \frac{f(\mathbf{x}|\mathrm{s})}{f(\mathbf{x}|\mathrm{b})}$$

i.e., it is a monotonic function of the likelihood ratio and thus leads to the same critical region. So in this case the Fisher discriminant provides an optimal statistical test.

Transformation of inputs

If the data are not Gaussian with equal covariance, a linear decision boundary is not optimal. But we can try to subject the data to a transformation

$$\varphi_1(\vec{x}), \ldots, \varphi_m(\vec{x})$$

and then treat the ϕ_i as the new input variables. This is often called "feature space" and the ϕ_i are "basis functions". The basis functions can be fixed or can contain adjustable parameters which we optimize with training data (cf. neural networks).

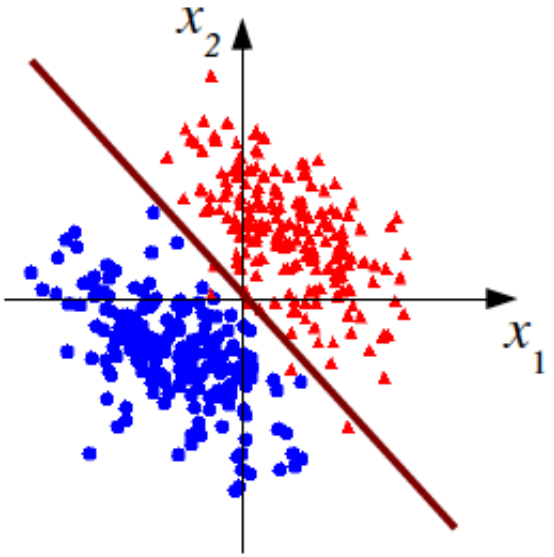
In other cases we will see that the basis functions only enter as dot products

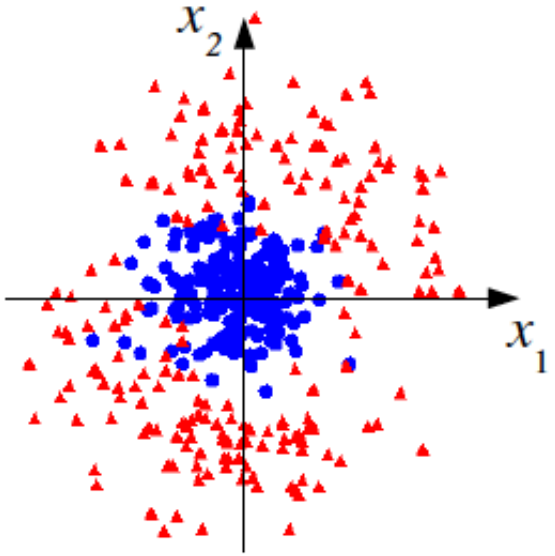
$$\vec{\varphi}(\vec{x}_i)\cdot\vec{\varphi}(\vec{x}_j) = K(\vec{x}_i, \vec{x}_j)$$

and thus we will only need the "kernel function" $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$

Linear decision boundaries

A linear decision boundary is only optimal when both classes follow multivariate Gaussians with equal covariances and different means.





For some other cases a linear boundary is almost useless.

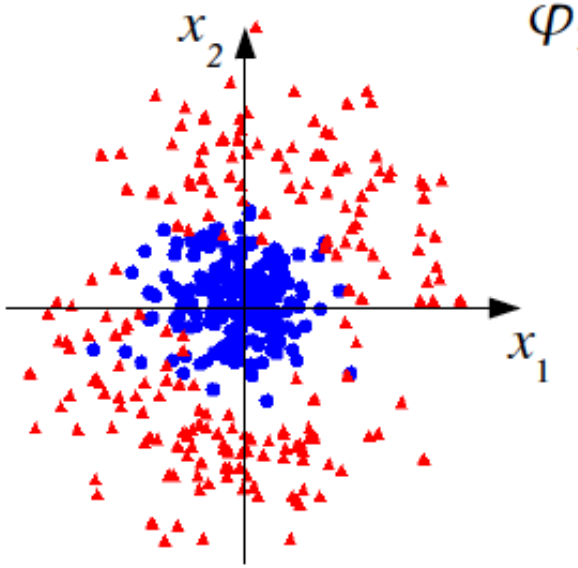
Nonlinear transformation of inputs

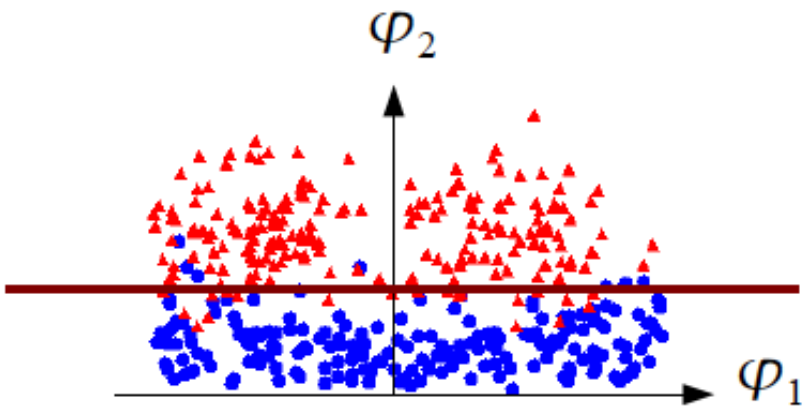
We can try to find a transformation, $x_1, \ldots, x_n \to \varphi_1(\vec{x}), \ldots, \varphi_m(\vec{x})$ so that the transformed "feature space" variables can be separated better by a linear boundary:

$$\varphi_1 = \tan^{-1}(x_2/x_1)$$

$$\varphi_2 = \sqrt{x_1^2 + x_2^2}$$

Here, guess fixed basis functions (no free parameters)





Neural networks

Neural networks originate from attempts to model neural processes (McCulloch and Pitts, 1943; Rosenblatt, 1962).

Widely used in many fields, and for many years the only "advanced" multivariate method popular in HEP.

We can view a neural network as a specific way of parametrizing the basis functions used to define the feature space transformation.

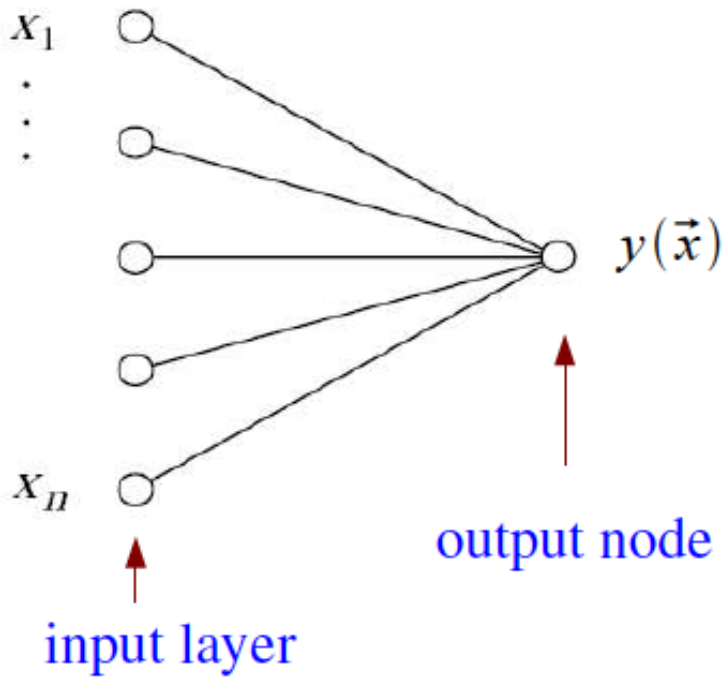
The training data are then used to adjust the parameters so that the resulting discriminant function has the best performance.

The single layer perceptron

Define the discriminant using $y(\vec{x}) = h\left(w_0 + \sum_{i=1}^{n} w_i x_i\right)$

where h is a nonlinear, monotonic activation function; we can use e.g. the logistic sigmoid $h(x) = (1 + e^{-x})^{-1}$.

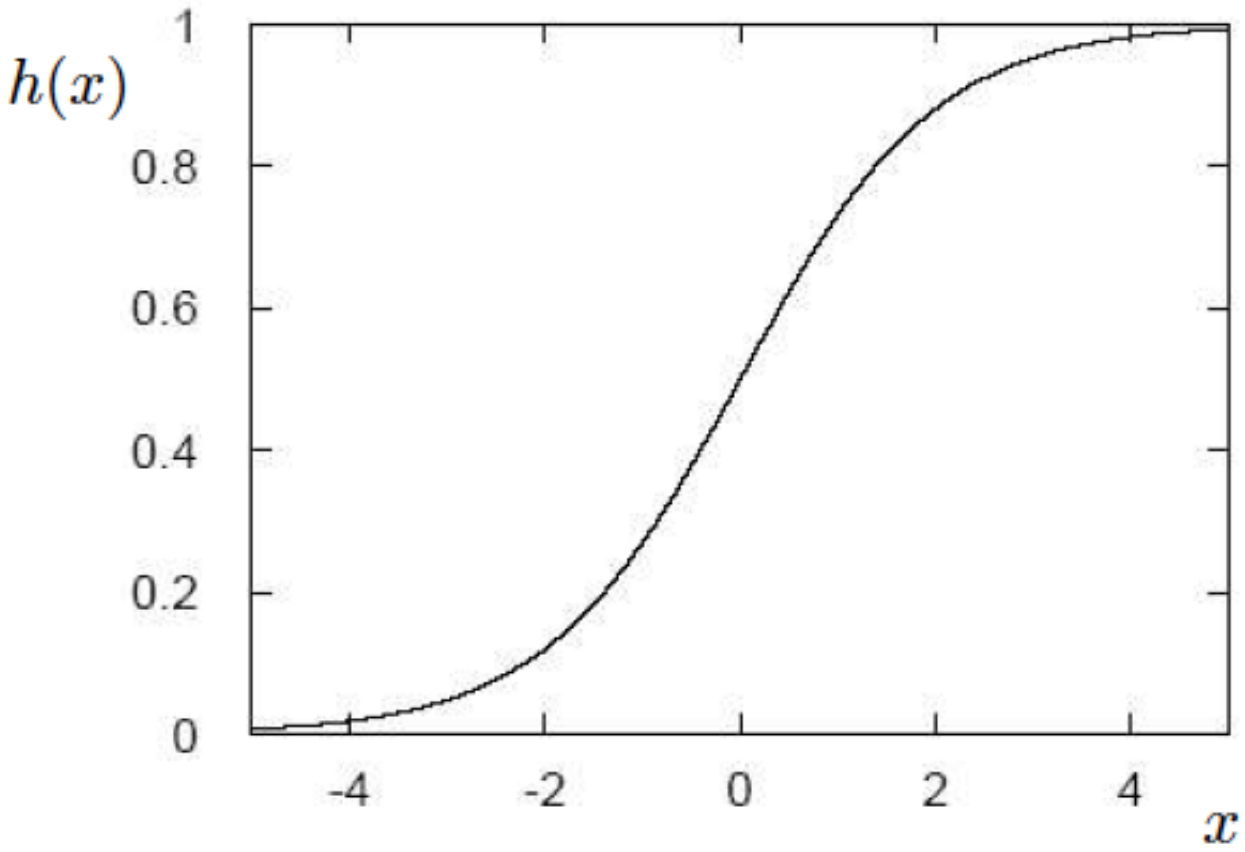
If the activation function is monotonic, the resulting $y(\boldsymbol{x})$ is equivalent to the original linear discriminant. This is an example of a "generalized linear model" called the single layer perceptron.



The activation function

For activation function $h(\cdot)$ often use logistic sigmoid:

$$h(x) = \frac{1}{1 + e^{-x}}$$



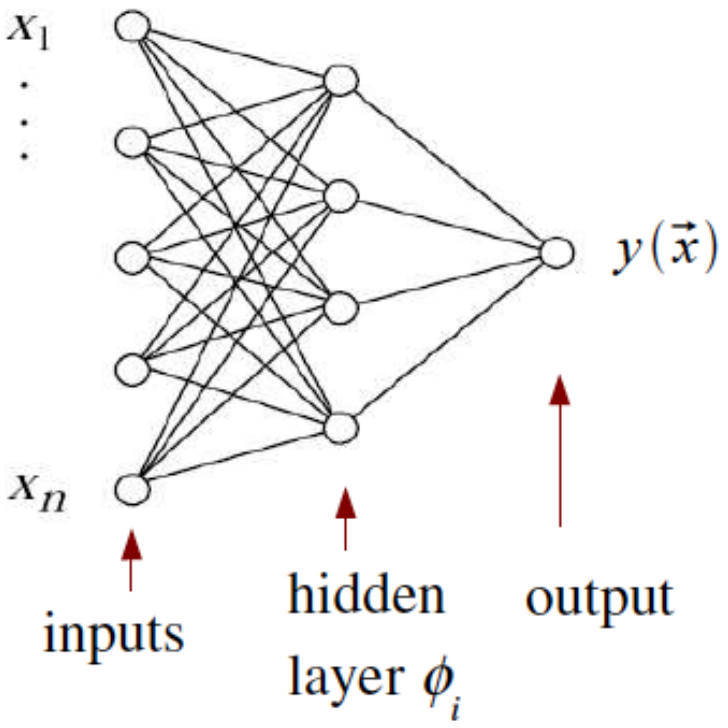
The multilayer perceptron

Now use this idea to define not only the output $y(\boldsymbol{x})$, but also the set of transformed inputs $\varphi_1(\vec{x}), \ldots, \varphi_m(\vec{x})$ that form a "hidden layer":

Superscript for weights indicates layer number

$$\varphi_i(\vec{x}) = h\left(w_{i0}^{(1)} + \sum_{j=1}^{n} w_{ij}^{(1)} x_j\right)$$

$$y(\vec{x}) = h\left(w_{10}^{(2)} + \sum_{j=1}^{n} w_{1j}^{(2)} \varphi_j(\vec{x})\right)$$



This is the multilayer perceptron, our basic neural network model; straightforward to generalize to multiple hidden layers.

Network architecture

Theorem: An MLP with a single hidden layer having a sufficiently large number of nodes can approximate arbitrarily well the optimal decision boundary.

Holds for any continuous non-polynomial activation function
Leshno, Lin, Pinkus and Schocken (1993) Neural Networks 6, 861-867

However, the number of required nodes may be very large; cannot train well with finite samples of training data.

Recent advances in *Deep Neural Networks* have shown important advantages in having multiple hidden layers.

For a particle physics application of Deep Learning, see e.g.
Baldi, Sadowski and Whiteson, *Nature Communications* 5 (2014); arXiv:1402.4735.

Network training

The type of each training event is known, i.e., for event a we have:

$\vec{x}_a = (x_1, \ldots, x_n)$ the input variables, and

$t_a = 0, 1$ a numerical label for event type ("target value")

Let $\boldsymbol{w}$ denote the set of all of the weights of the network. We can determine their optimal values by minimizing a sum-of-squares "error function"

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{a=1}^{N} |y(\vec{x}_a, \boldsymbol{w}) - t_a|^2 = \sum_{a=1}^{N} E_a(\boldsymbol{w})$$

Contribution to error function from each event

Numerical minimization of $E(\boldsymbol{w})$

Consider gradient descent method: from an initial guess in weight space $\boldsymbol{w}^{(1)}$ take a small step in the direction of maximum decrease. I.e. for the step τ to $\tau+1$,

$$\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} - \eta \nabla E(\boldsymbol{w}^{(\tau)})$$

learning rate ($\eta>0$)

If we do this with the full error function $E(\boldsymbol{w})$, gradient descent does surprisingly poorly; better to use "conjugate gradients".

But gradient descent turns out to be useful with an online (sequential) method, i.e., where we update $\boldsymbol{w}$ for each training event a, (cycle through all training events):

$$\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} - \eta \nabla E_a(\boldsymbol{w}^{(\tau)})$$

Error backpropagation

Error backpropagation ("backprop") is an algorithm for finding the derivatives required for gradient descent minimization.

The network output can be written $y(\boldsymbol{x}) = h(u(\boldsymbol{x}))$ where

$$u(\vec{x}) = \sum_{j=0} w_{1j}^{(2)} \varphi_j(\vec{x}), \qquad \varphi_j(\vec{x}) = h\left(\sum_{k=0} w_{jk}^{(1)} x_k \right)$$

where we defined $\phi_0 = x_0 = 1$ and wrote the sums over the nodes in the preceding layers starting from 0 to include the offsets.

So e.g. for event a we have $\dfrac{\partial E_a}{\partial w_{1j}^{(2)}} = (y_a - t_a)\, h'(u(\vec{x}))\, \varphi_j(\vec{x})$

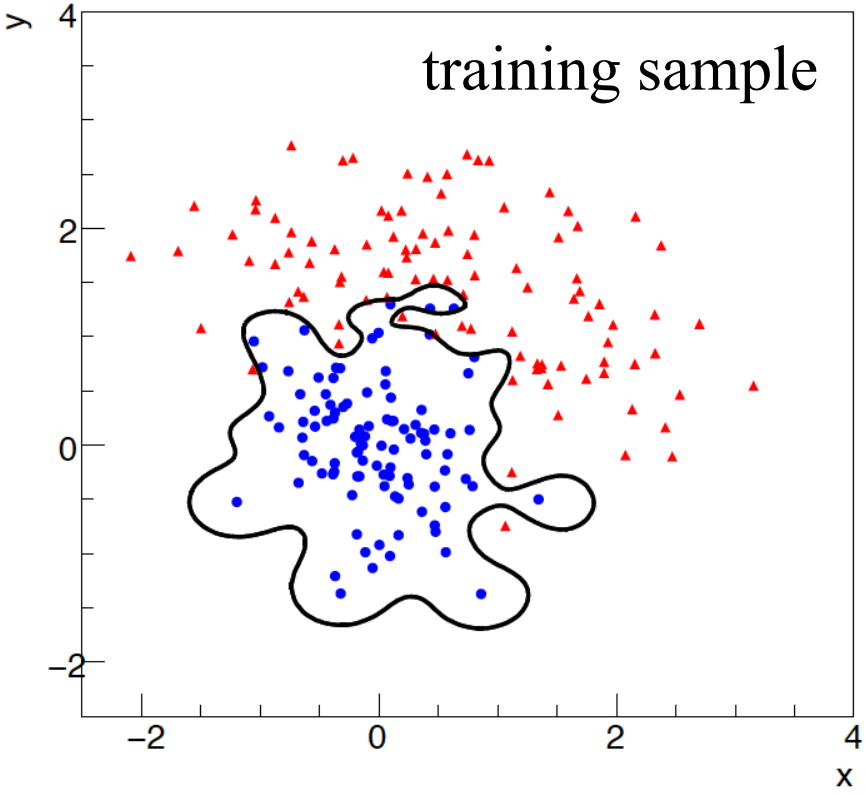
derivative of activation function

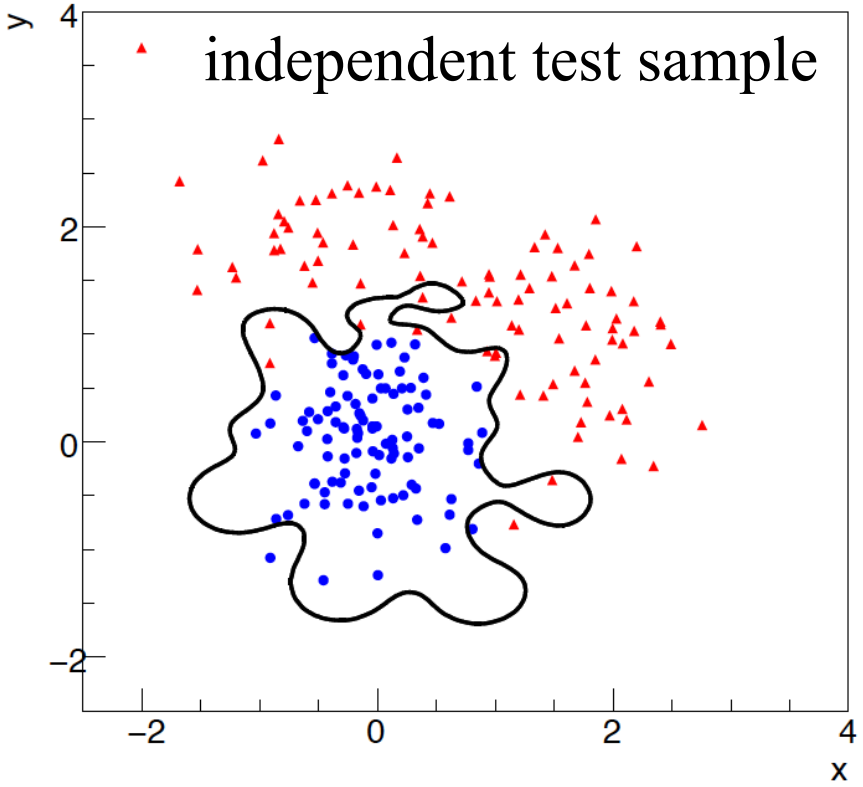
Chain rule gives all the needed derivatives.

Overtraining

Including more parameters in a classifier makes its decision boundary increasingly flexible, e.g., more nodes/layers for a neural network.

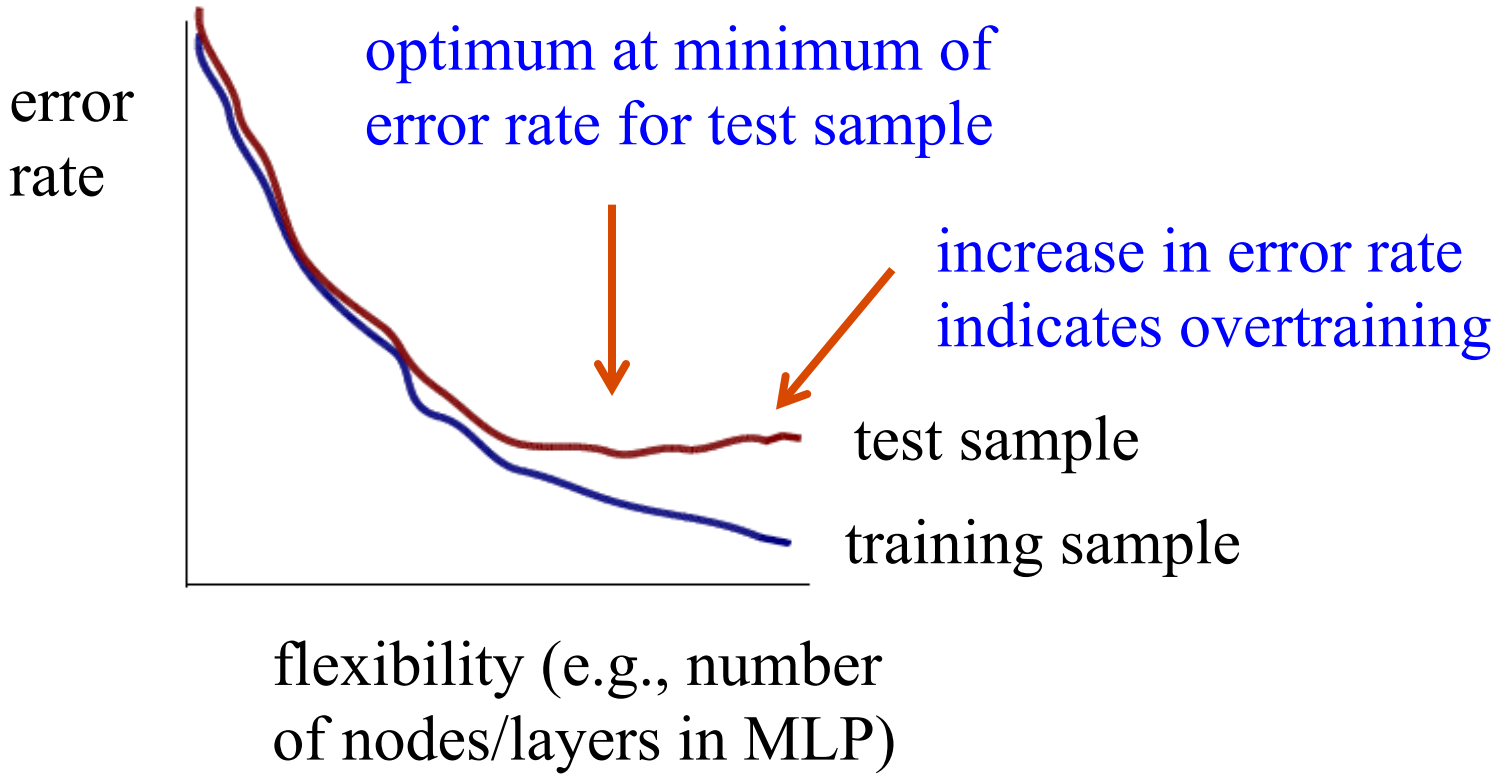
A "flexible" classifier may conform too closely to the training points; the same boundary will not perform well on an independent test data sample (→ "overtraining").





Monitoring overtraining

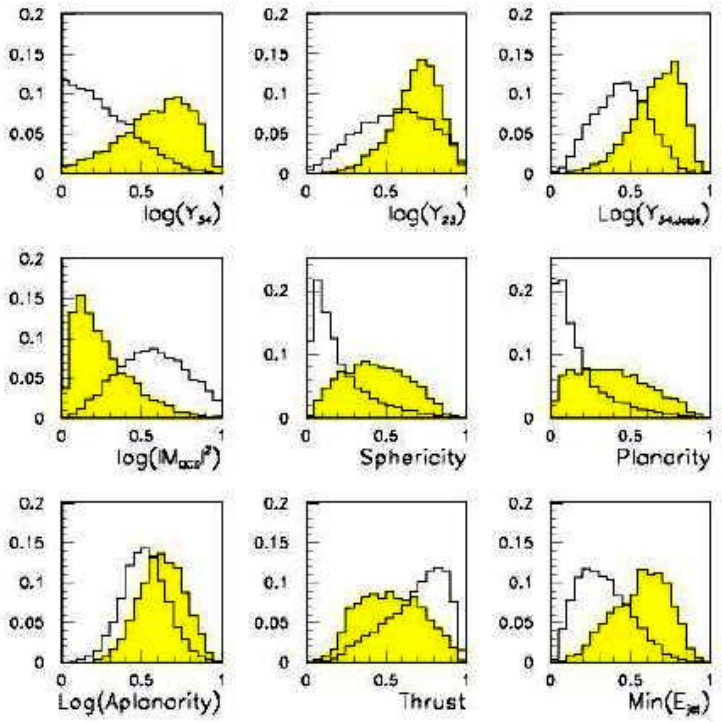
If we monitor the fraction of misclassified events (or similar, e.g., error function $E(\boldsymbol{w})$) for test and training samples, it will usually decrease for both as the boundary is made more flexible:



Neural network example from LEP II

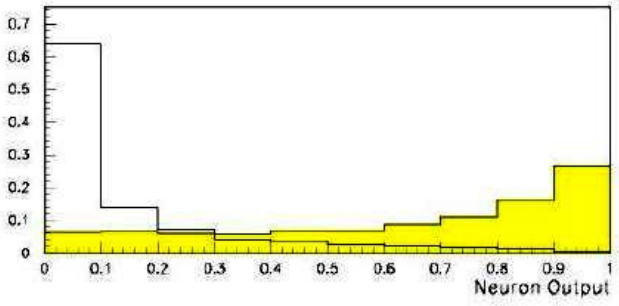
Signal: $e^+e^- \to W^+W^-$ (often 4 well separated hadron jets)

Background: $e^+e^- \to qqgg$ (4 less well separated hadron jets)



← input variables based on jet structure, event shape, ...
none by itself gives much separation.

Neural network output:



(Garrido, Juste and Martinez, ALEPH 96-144)

Probability Density Estimation (PDE)

Construct non-parametric estimators for the pdfs of the data $\boldsymbol{x}$ for the two event classes, $p(\boldsymbol{x}|H_0)$, $p(\boldsymbol{x}|H_1)$ and use these to construct the likelihood ratio, which we use for the discriminant function:

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

n-dimensional histogram is a brute force example of this; we will see a number of ways that are much better.

Correlation vs. independence

In a general a multivariate distribution $p(\boldsymbol{x})$ does not factorize into a product of the marginal distributions for the individual variables:

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

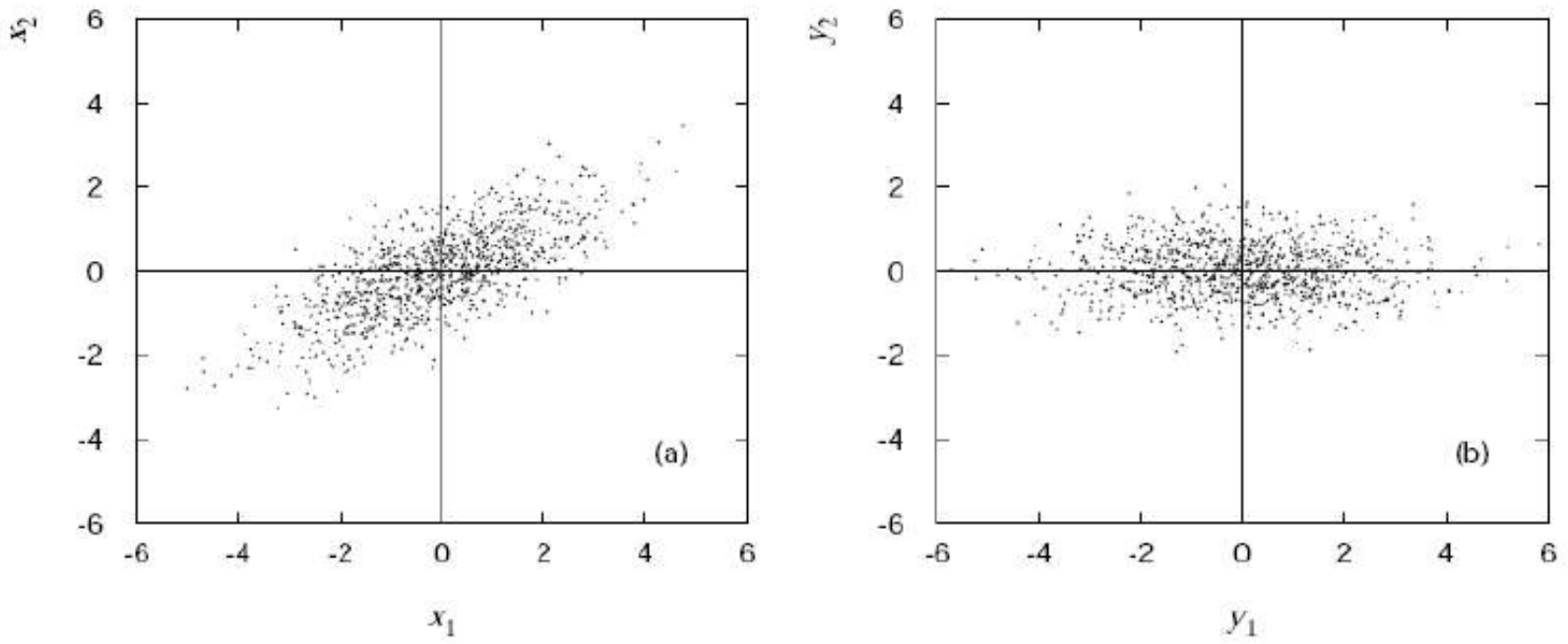
holds only if the components of $\boldsymbol{x}$ are independent

Most importantly, the components of $\boldsymbol{x}$ will generally have nonzero covariances (i.e. they are correlated):

$$V_{ij} = \mathrm{cov}[x_i, x_j] = E[x_i x_j] - E[x_i]E[x_j] \neq 0$$

Decorrelation of input variables

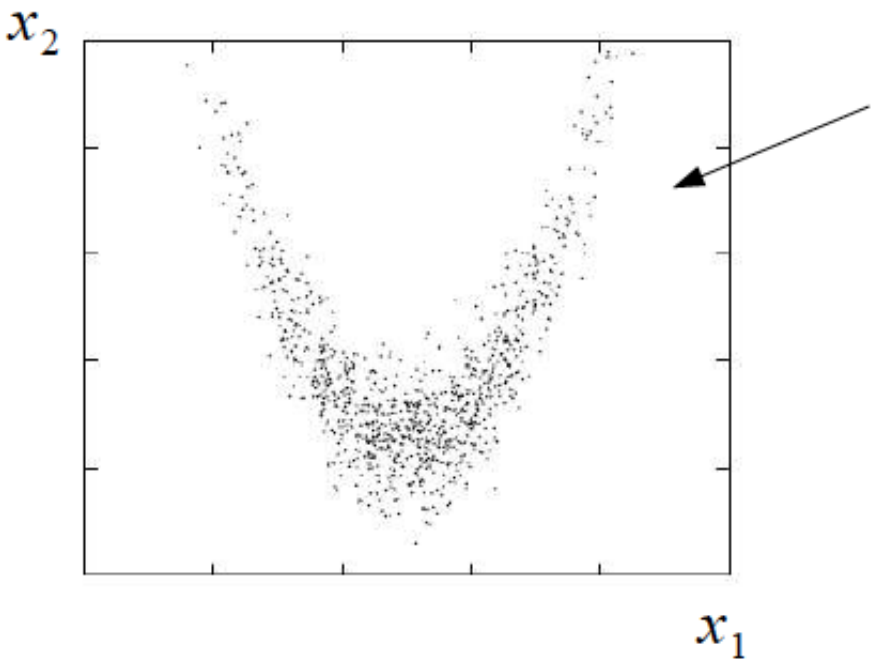
But we can define a set of uncorrelated input variables by a linear transformation, i.e., find the matrix A such that for $\vec{y} = A\vec{x}$ the covariances cov[y_i, y_j] = 0:



For the following suppose that the variables are "decorrelated" in this way for each of $p(\mathbf{x}|H_0)$ and $p(\mathbf{x}|H_1)$ separately (since in general their correlations are different).

Decorrelation is not enough

But even with zero correlation, a multivariate pdf $p(\boldsymbol{x})$ will in general have nonlinearities and thus the decorrelated variables are still not independent.



pdf with zero covariance but components still not independent, since clearly

$$p(x_2|x_1) \equiv \frac{p(x_1, x_2)}{p_1(x_1)} \neq p_2(x_2)$$

and therefore

$$p(x_1, x_2) \neq p_1(x_1)\, p_2(x_2)$$

Naive Bayes method

First decorrelate $\boldsymbol{x}$, i.e., find $\boldsymbol{y} = A\boldsymbol{x}$, with $\mathrm{cov}[y_i, y_j] = V[y_i]\,\delta_{ij}$.

Pdfs of $\boldsymbol{x}$ and $\boldsymbol{y}$ are then related by

$$f(\mathbf{x}) = |J| g(\mathbf{y}(\mathbf{x})) \qquad \text{where} \qquad J = \det(A)$$

If nonlinear features of g($\boldsymbol{y}$) not too important, estimate using product of marginal pdfs:

$$\hat{f}(\mathbf{x}) = |J| \prod_{i=1}^{n} g_i(y_i(\mathbf{x})) = |\det(A)| \prod_{i=1}^{n} g_i((A\mathbf{x})_i)$$

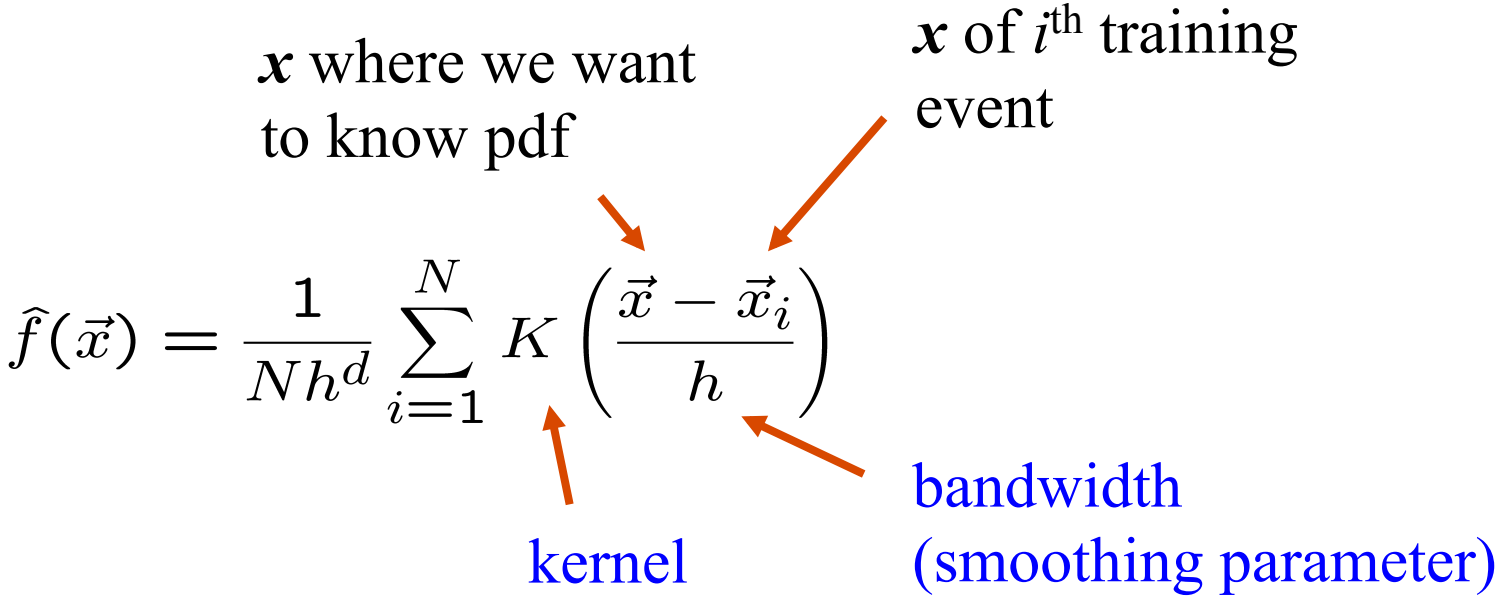
Do separately for the two hypotheses s and b (separate matrices A_s and A_b and marginal pdfs $g_{\mathrm{s},i}$, $g_{\mathrm{b},i}$). Then define test statistic as

$$y(\mathbf{x}) = \frac{\hat{f}_\mathrm{s}(\mathbf{x})}{\hat{f}_\mathrm{b}(\mathbf{x})}$$

Called Naive Bayes classifier. Reduces problem of estimating an n-dimensional pdf to finding n one-dimensional marginal pdfs.

Kernel-based PDE (KDE)

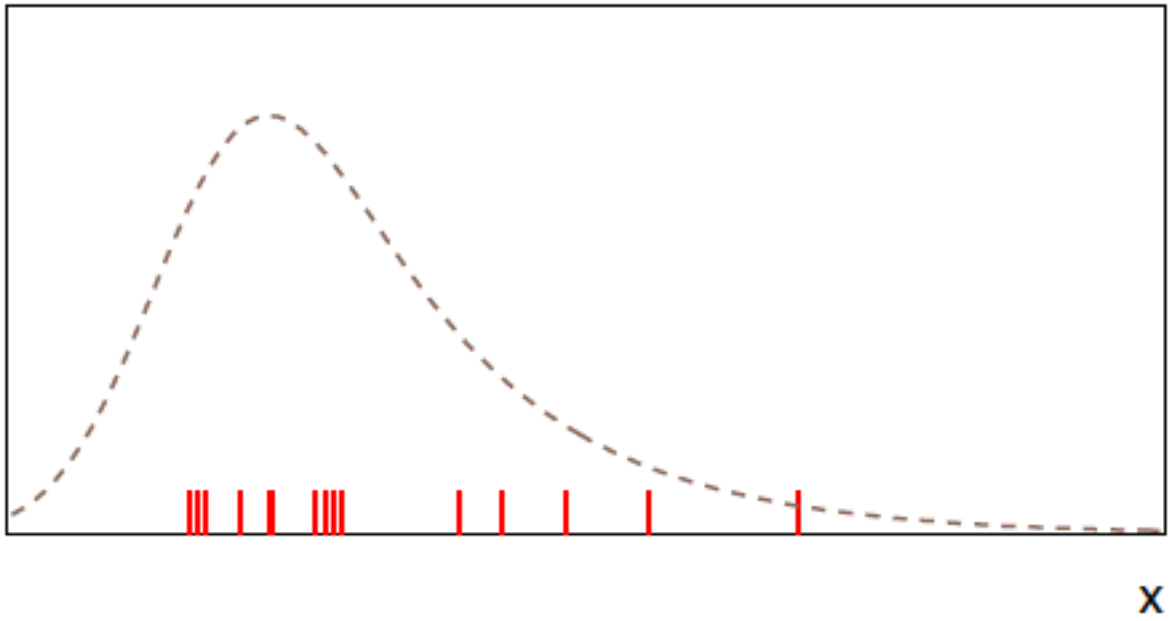
Consider d dimensions, N training events, $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, estimate $f(\boldsymbol{x})$ with



Use e.g. Gaussian kernel: $K(\vec{x}) = \frac{1}{(2\pi)^{d/2}} e^{-|\vec{x}|^2/2}$

Gaussian KDE in 1-dimension

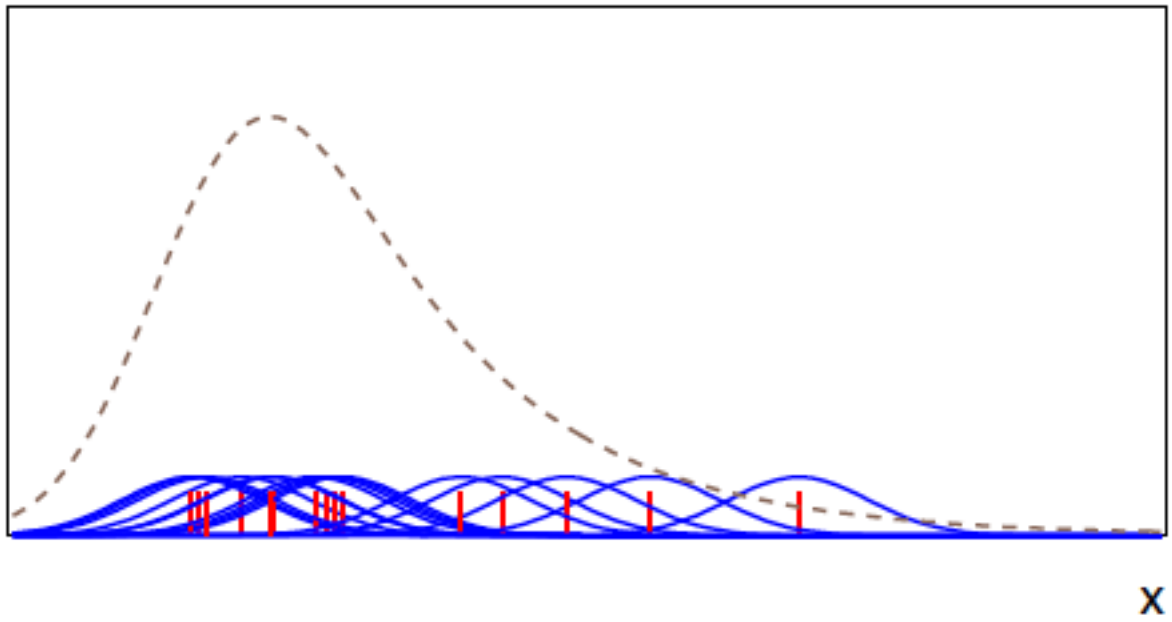
Suppose the pdf (dashed line) below is not known in closed form, but we can generate events that follow it (the red tick marks):



Goal is to find an approximation to the pdf using the generated date values.

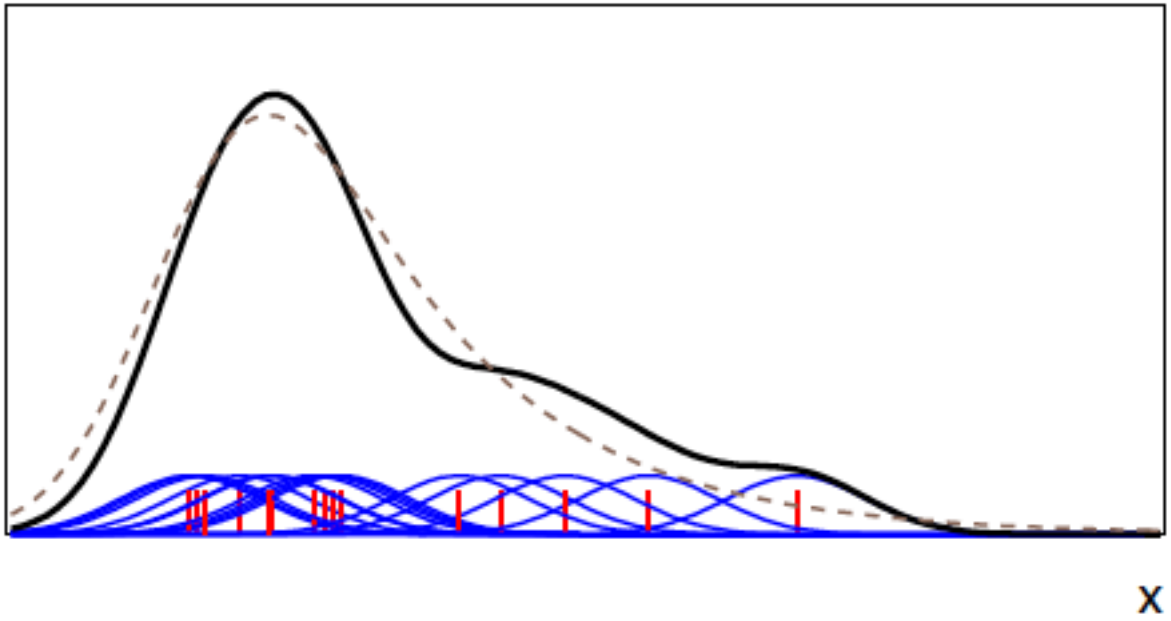
Gaussian KDE in 1-dimension (cont.)

Place a kernel pdf (here a Gaussian) centred around each generated event weighted by $1/N_{\text{event}}$:



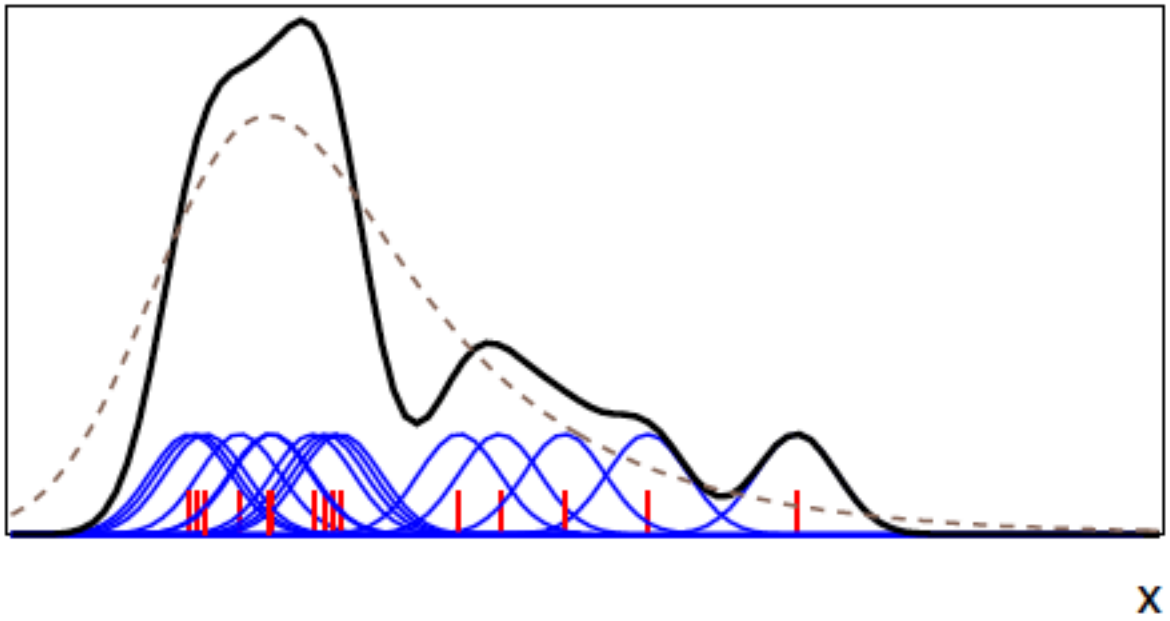
Gaussian KDE in 1-dimension (cont.)

The KDE estimate the pdf is given by the sum of all of the Gaussians:



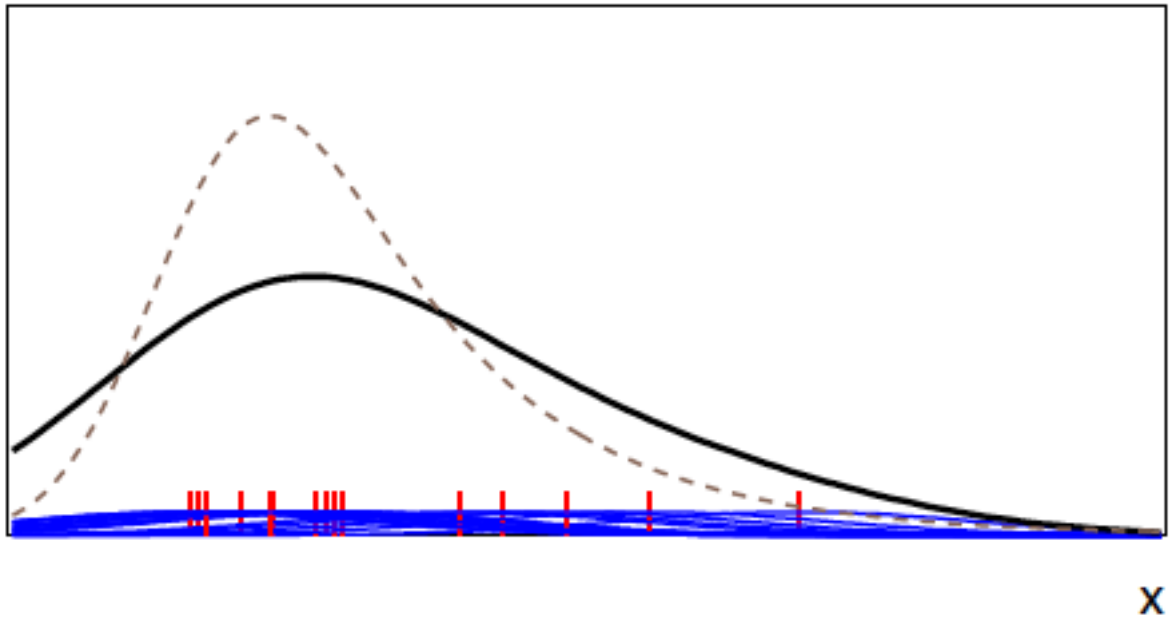
Choice of kernel width

The width h of the Gaussians is analogous to the bin width of a histogram. If it is too small, the estimator has noise:



Choice of kernel width (cont.)

If width of Gaussian kernels too large, structure is washed out:



KDE discussion

Various strategies can be applied to choose width h of kernel based trade-off between bias and variance (noise).

Adaptive KDE allows width of kernel to vary, e.g., wide where target pdf is low (few events); narrow where pdf is high.

Advantage of KDE: no training!

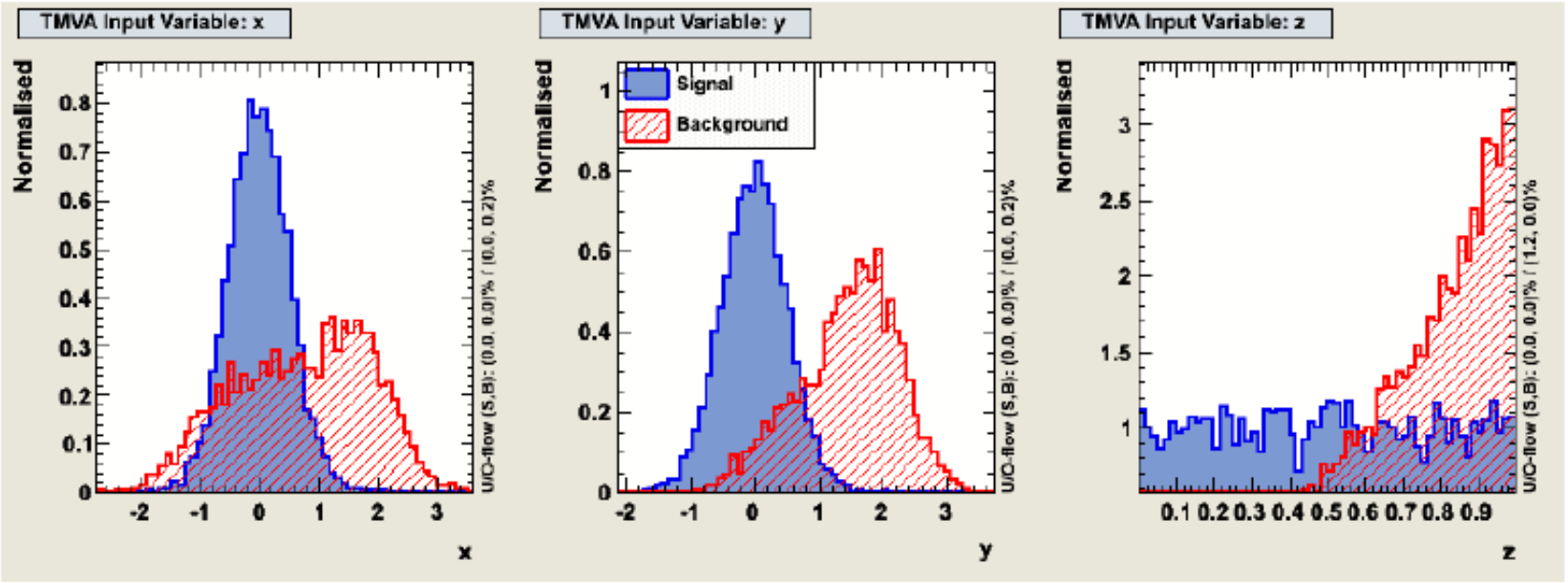
Disadvantage of KDE: to evaluate we need to sum N_{event} terms, so if we have many events this can be slow.

Special treatment required if kernel extends beyond range where pdf defined. Can e.g., renormalize the kernels to unity inside the allowed range; alternatively "mirror" the events about the boundary (contribution from the mirrored events exactly compensates the amount lost outside the boundary).

Software in ROOT: RooKeysPdf (K. Cranmer, CPC 136:198,2001)

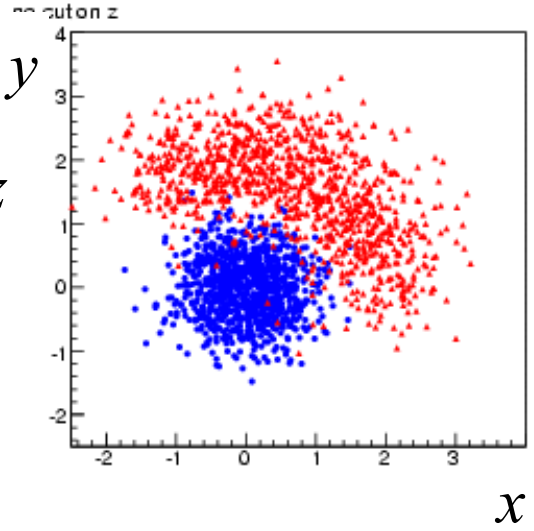
Test example with TMVA

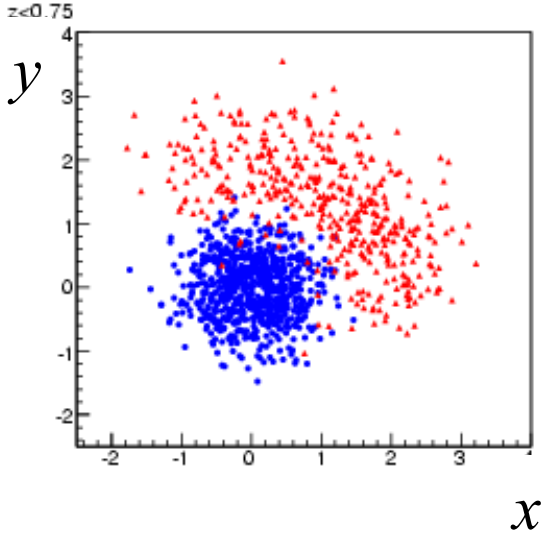
Each event characterized by 3 variables, x, y, z:



Test example (x, y, z)

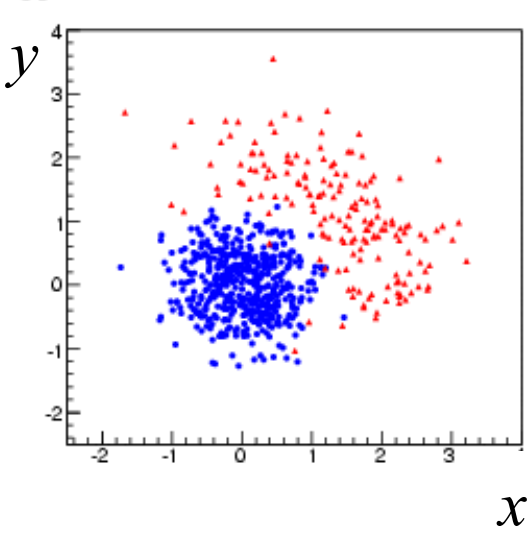
no cut on z

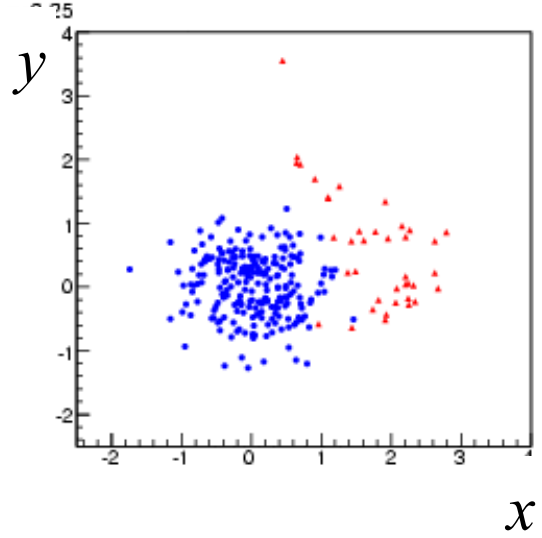




$z < 0.75$

$z < 0.5$

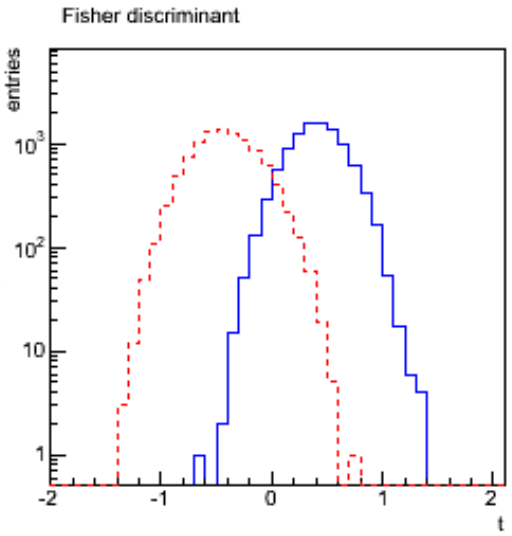


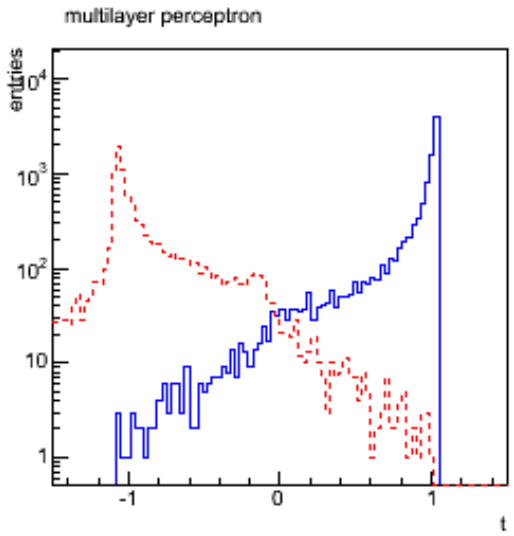


$z < 0.25$

Test example results

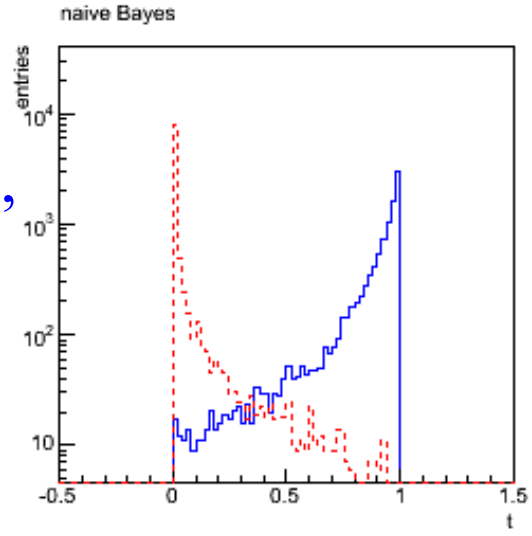
Fisher discriminant

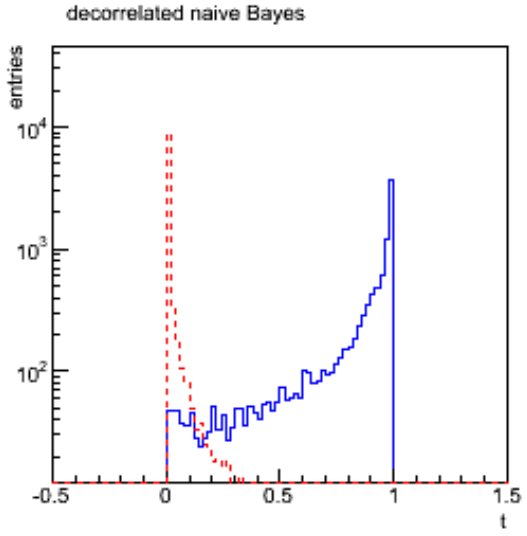




Multilayer perceptron

Naive Bayes, no decor-relation

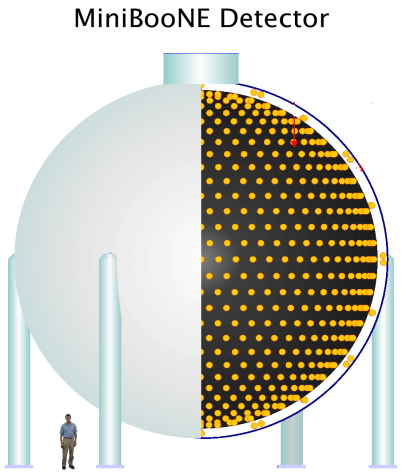




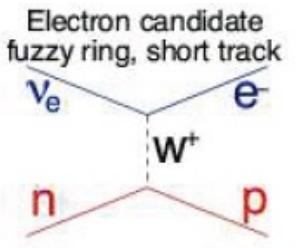
Naive Bayes with decor-relation

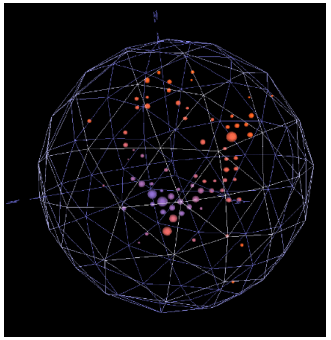
Particle i.d. in MiniBooNE

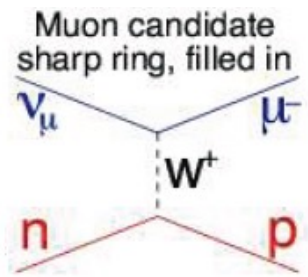
Detector is a 12-m diameter tank of mineral oil exposed to a beam of neutrinos and viewed by 1520 photomultiplier tubes:

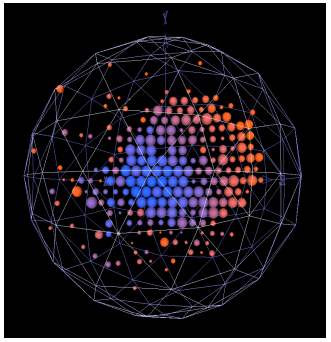


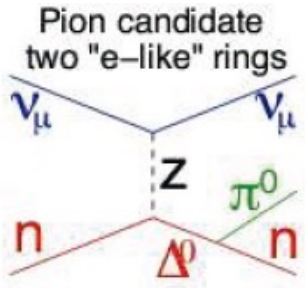
Search for ν_μ to ν_e oscillations required particle i.d. using information from the PMTs.

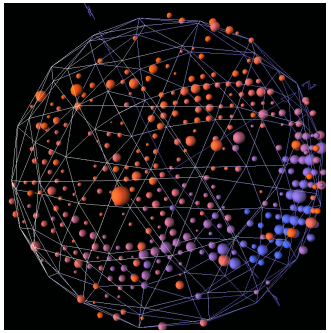












H.J. Yang, MiniBooNE PID, DNP06

Decision trees

Out of all the input variables, find the one for which with a single cut gives best improvement in signal purity:

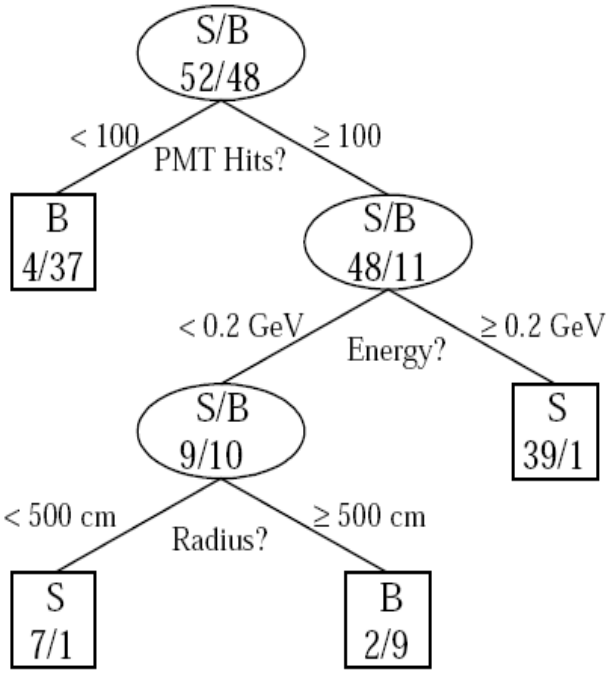
$$P = \frac{\sum_{\text{signal}} w_i}{\sum_{\text{signal}} w_i + \sum_{\text{background}} w_i}$$

where w_i. is the weight of the ith event.

Resulting nodes classified as either signal/background.

Iterate until stop criterion reached based on e.g. purity or minimum number of events in a node.

The set of cuts defines the decision boundary.



Example by MiniBooNE experiment, B. Roe et al., NIM 543 (2005) 577

Finding the best single cut

The level of separation within a node can, e.g., be quantified by the *Gini coefficient*, calculated from the (s or b) purity as:

$$G = p(1-p)$$

For a cut that splits a set of events a into subsets b and c, one can quantify the improvement in separation by the change in weighted Gini coefficients:

$$\Delta = W_a G_a - W_b G_b - W_c G_c \quad \text{where, e.g.,} \quad W_a = \sum_{i \in a} w_i$$

Choose e.g. the cut to the maximize Δ; a variant of this scheme can use instead of Gini e.g. the misclassification rate:

$$\varepsilon = 1 - \max(p, 1-p)$$

Decision trees (2)

The terminal nodes (leaves) are classified a signal or background depending on majority vote (or e.g. signal fraction greater than a specified threshold).

This classifies every point in input-variable space as either signal or background, a decision tree classifier, with discriminant function

$$f(\boldsymbol{x}) = 1 \text{ if } \boldsymbol{x} \text{ in signal region, } -1 \text{ otherwise}$$

Decision trees tend to be very sensitive to statistical fluctuations in the training sample.

Methods such as boosting can be used to stabilize the tree.

Boosting

Boosting is a general method of creating a set of classifiers which can be combined to achieve a new classifier that is more stable and has a smaller error than any individual one.

Often applied to decision trees but, can be applied to any classifier.

Suppose we have a training sample T consisting of N events with

$\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ event data vectors (each $\boldsymbol{x}$ multivariate)

$y_1, \ldots, y_N$ true class labels, +1 for signal, −1 for background

$w_1, \ldots, w_N$ event weights

Now define a rule to create from this an ensemble of training samples $T_1, T_2, \ldots$, derive a classifier from each and average them.

Trick is to create modifications in the training sample that give classifiers with smaller error rates than those of the preceding ones.

A successful example is AdaBoost (Freund and Schapire, 1997).

AdaBoost

First initialize the training sample T_1 using the original

$\boldsymbol{x}_1,\ldots,\boldsymbol{x}_N$ event data vectors

$y_1,\ldots,y_N$ true class labels (+1 or -1)

$w_1^{(1)},\ldots,w_N^{(1)}$ event weights

with the weights equal and normalized such that

$$\sum_{i=1}^{N} w_i^{(1)} = 1$$

Then train the classifier $f_1(\boldsymbol{x})$ (e.g. a decision tree) with a method that incorporates the event weights. For an event with data $\boldsymbol{x}_i$,

$f_1(\boldsymbol{x}_i) > 0$ classify as signal

$f_1(\boldsymbol{x}_i) < 0$ classify as background

Updating the event weights

Define the training sample for step k+1 from that of k by updating the event weights according to

$$w_i^{(k+1)} = w_i^{(k)} \frac{e^{-\alpha_k f_k(x_i) y_i / 2}}{Z_k}$$

i = event index

k = training sample index

where Z_k is a normalization factor defined such that the sum of the weights over all events is equal to one.

Therefore event weight for event i is increased in the k+1 training sample if it was classified incorrectly in sample k.

Idea is that next time around the classifier should pay more attention to this event and try to get it right.

Error rate of the kth classifier

At each step the classifiers $f_k(\boldsymbol{x})$ are defined so as to minimize the error rate ε_k,

$$\varepsilon_k = \sum_{i=1}^{N} w_i^{(k)} I\left(y_i f_k(\boldsymbol{x}_i) \leqslant 0\right)$$

where $I(X) = 1$ if X is true and is zero otherwise.

Assigning the classifier score

Assign a score to the kth classifier based on its error rate,

$$\alpha_k = \ln \frac{1-\varepsilon_k}{\varepsilon_k}$$

If we define the final classifier as $f(\boldsymbol{x}) = \sum_{k=1}^{K} \alpha_k f_k(\boldsymbol{x}, T_k)$

then one can show that its error rate on the training data satisfies the bound

$$\varepsilon \leqslant \prod_{k=1}^{K} 2\sqrt{\varepsilon_k(1-\varepsilon_k)}$$

AdaBoost error rate

So providing each classifier in the ensemble has $\varepsilon_k < ½$, i.e., better than random guessing, then the error rate for the final classifier on the training data (not on unseen data) drops to zero.

That is, for sufficiently large K the training data will be over fitted.

The error rate on a validation sample would reach some minimum after a certain number of steps and then could rise.

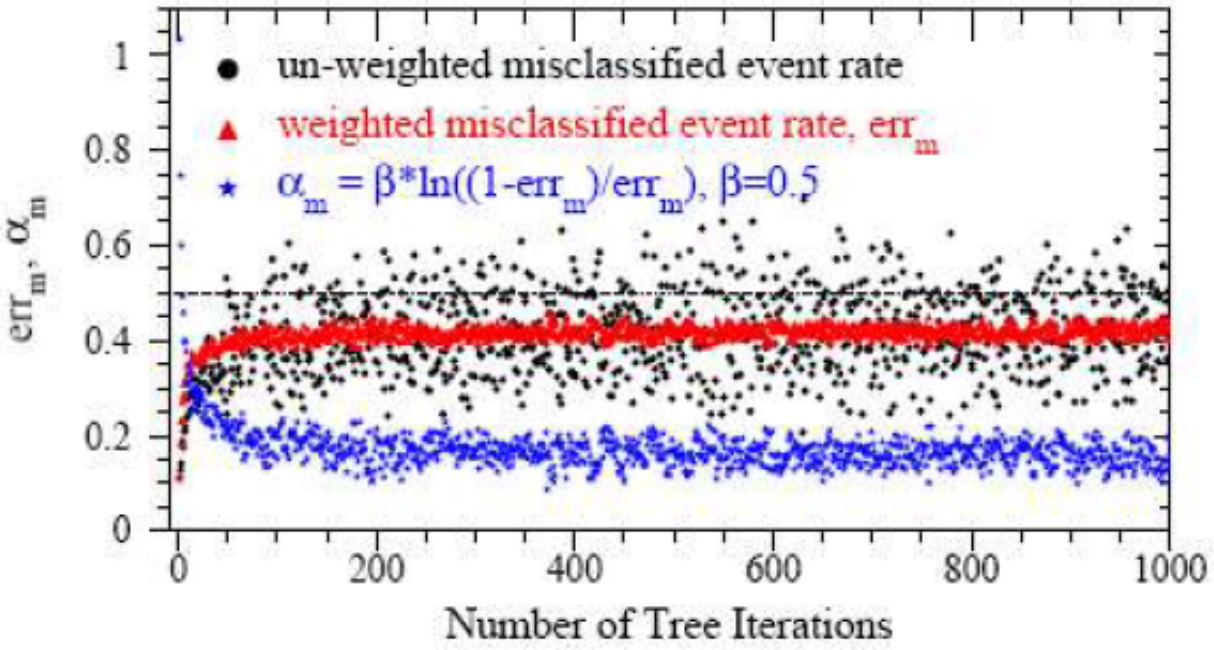
So the procedure is to monitor the error rate of the combined classifier at each step with a validation sample and to stop before it starts to rise.

Although in principle AdaBoost must overfit, in practice following this procedure overtraining is not a big problem.

BDT example from MiniBooNE

~200 input variables for each event (ν interaction producing e, μ or π).

Each individual tree is relatively weak, with a misclassification error rate ~ 0.4 – 0.45

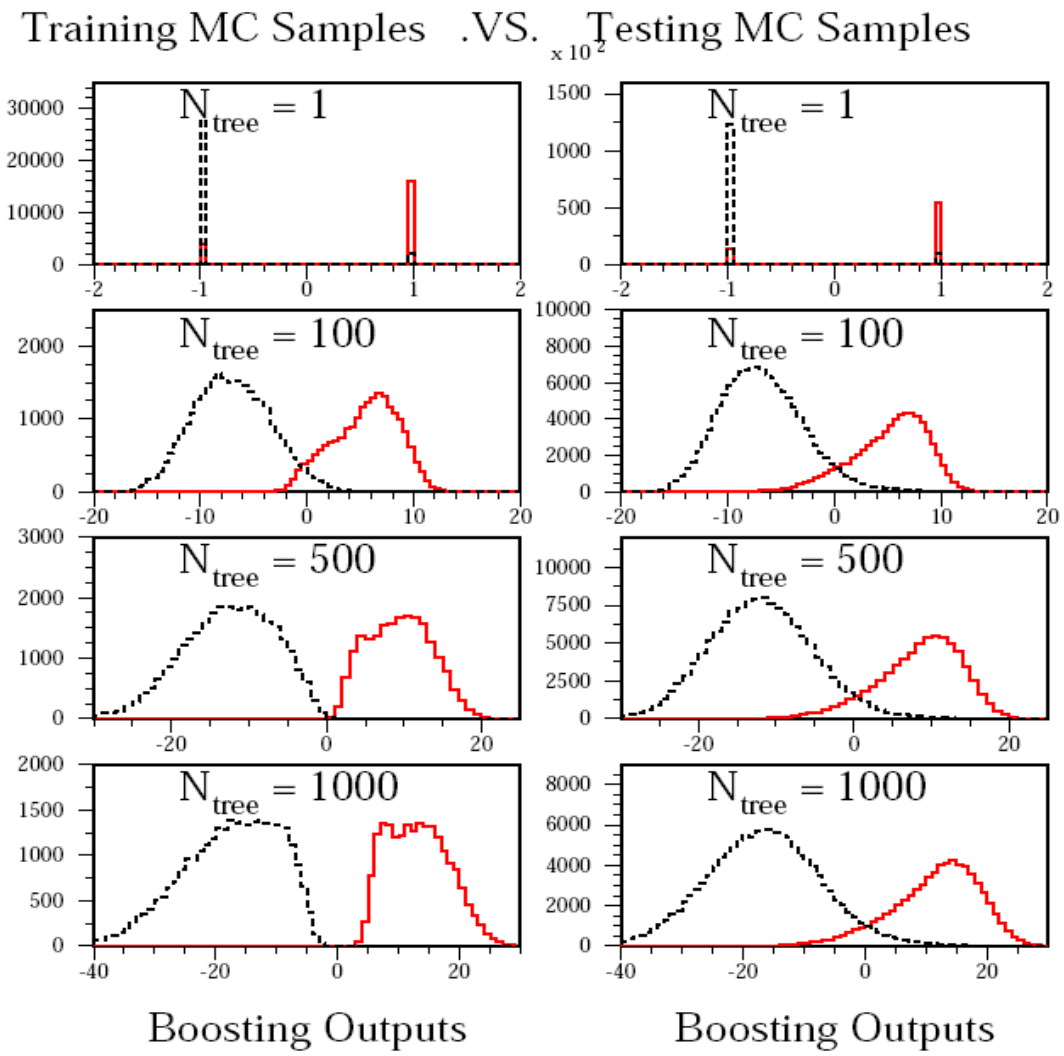


B. Roe et al., NIM 543 (2005) 577

Monitoring overtraining

From MiniBooNE example:

Performance stable after a few hundred trees.



A simple example (2D)

Consider two variables, x_1 and x_2, and suppose we have formulas for the joint pdfs for both signal (s) and background (b) events (in real problems the formulas are usually not available).

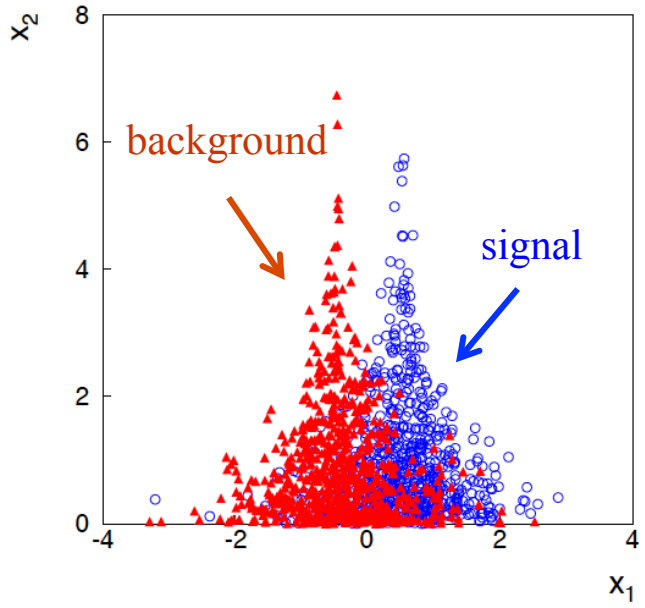
$f(x_1|x_2)$ ~ Gaussian, different means for s/b,
Gaussians have same σ, which depends on x_2,
$f(x_2)$ ~ exponential, same for both s and b,
$f(x_1, x_2) = f(x_1|x_2)\, f(x_2)$:

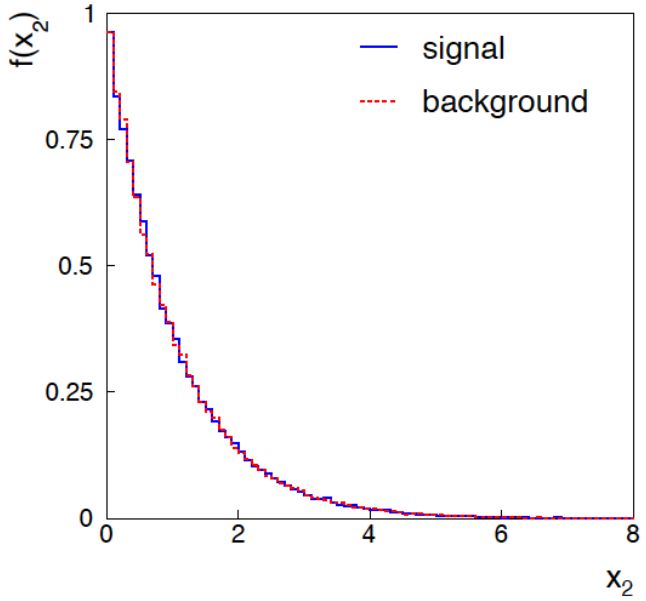
$$f(x_1, x_2|\mathrm{s}) = \frac{1}{\sqrt{2\pi}\sigma(x_2)} e^{-(x_1-\mu_\mathrm{s})^2/2\sigma^2(x_2)} \frac{1}{\lambda} e^{-x_2/\lambda}$$

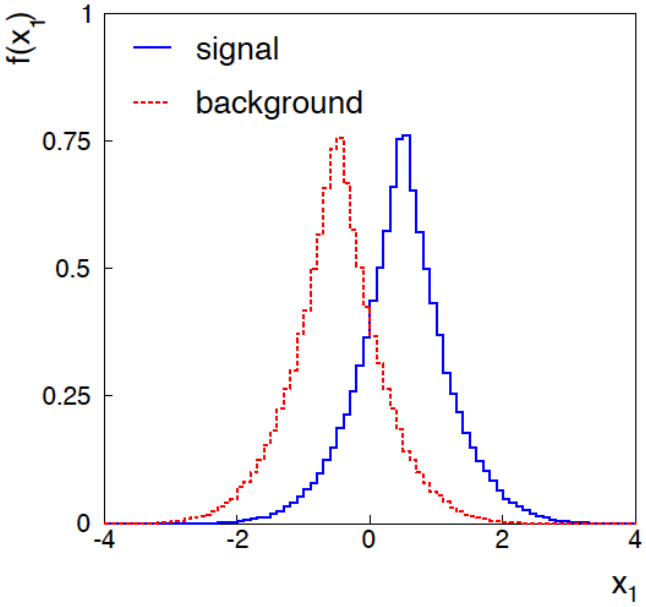
$$f(x_1, x_2|\mathrm{b}) = \frac{1}{\sqrt{2\pi}\sigma(x_2)} e^{-(x_1-\mu_\mathrm{b})^2/2\sigma^2(x_2)} \frac{1}{\lambda} e^{-x_2/\lambda}$$

$$\sigma(x_2) = \sigma_0 e^{-x_2/\xi}$$

Joint and marginal distributions of x_1, x_2







Distribution $f(x_2)$ same for s, b.

So does x_2 help discriminate between the two event types?

Likelihood ratio for 2D example

Neyman-Pearson lemma says best critical region is determined by the likelihood ratio:

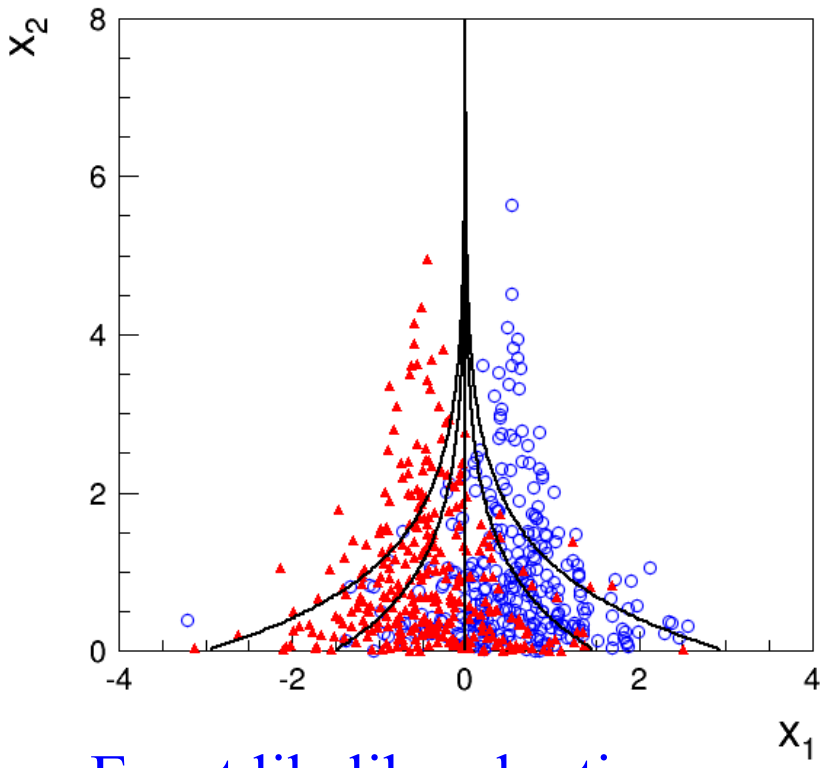
$$t(x_1, x_2) = \frac{f(x_1, x_2|\mathrm{s})}{f(x_1, x_2|\mathrm{b})}$$

Equivalently we can use any monotonic function of this as a test statistic, e.g.,

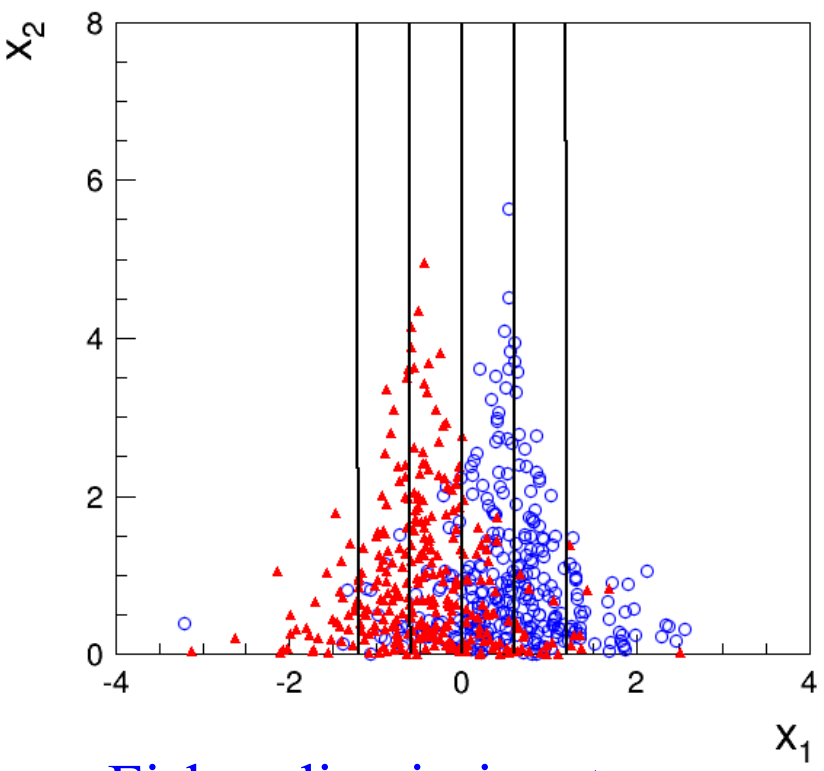
$$\ln t = \frac{\frac{1}{2}(\mu_\mathrm{b}^2 - \mu_\mathrm{s}^2) + (\mu_\mathrm{s} - \mu_\mathrm{b})x_1}{\sigma_0^2 e^{-2x_2/\xi}}$$

Boundary of optimal critical region will be curve of constant ln t, and this depends on x_2!

Contours of constant MVA output

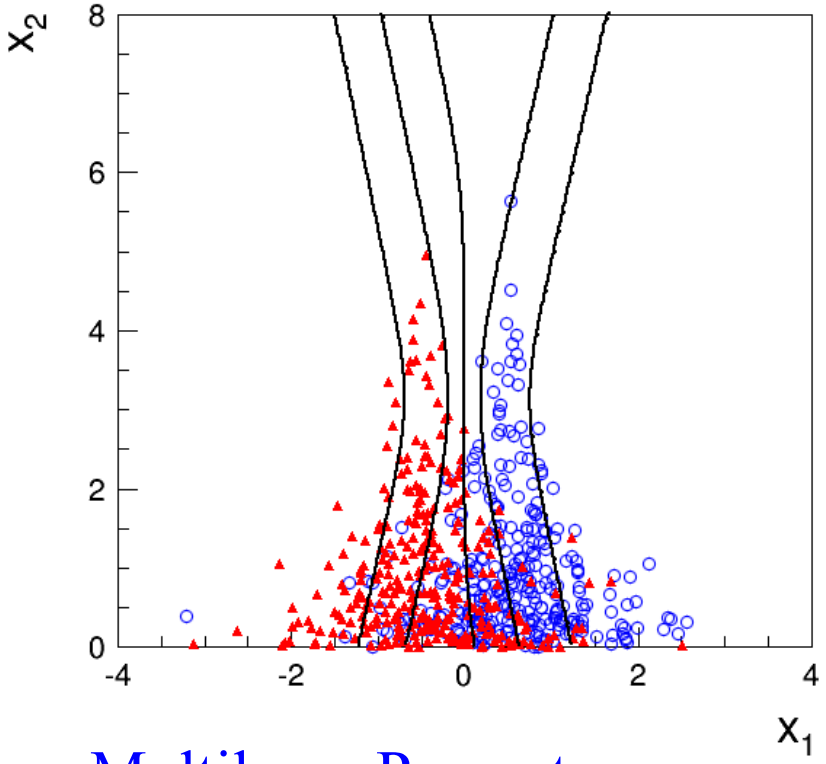


Exact likelihood ratio

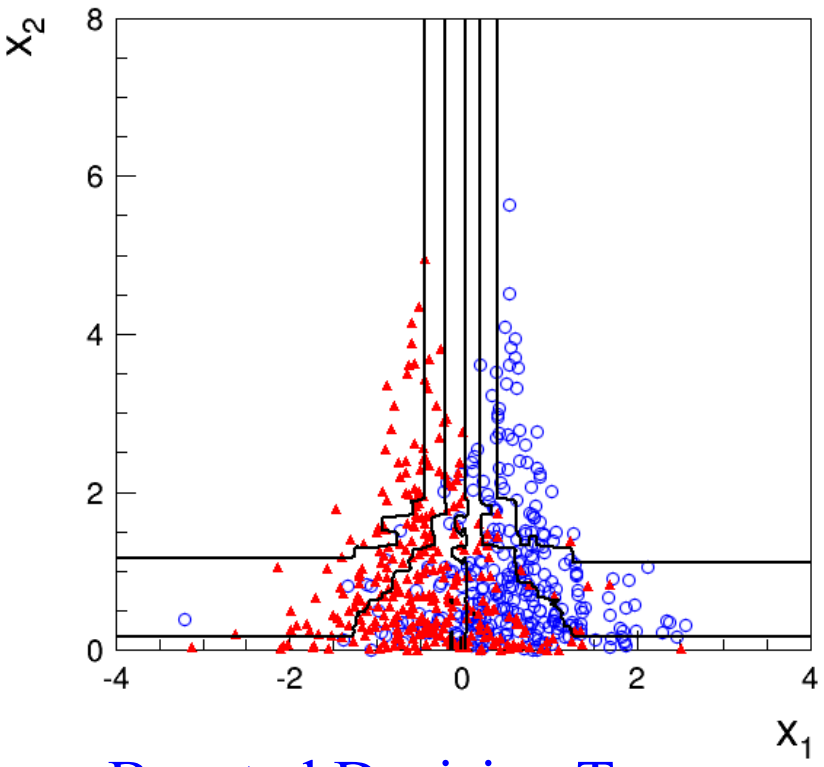


Fisher discriminant

Contours of constant MVA output



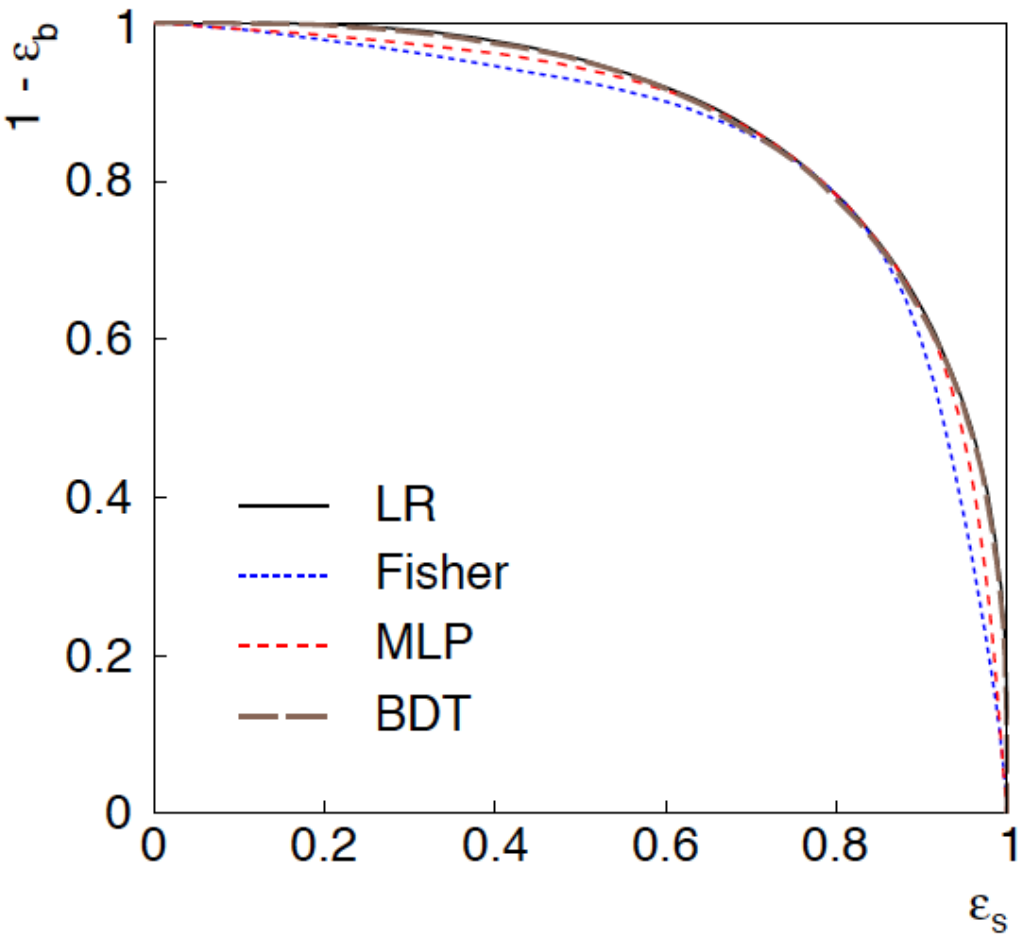
Multilayer Perceptron
1 hidden layer with 2 nodes



Boosted Decision Tree
200 iterations (AdaBoost)

Training samples: 10^5 signal and 10^5 background events

ROC curve



ROC = "receiver operating characteristic" (term from signal processing).

Shows (usually) background rejection ($1-\varepsilon_b$) versus signal efficiency ε_s.

Higher curve is better; usually analysis focused on a small part of the curve.

2D Example: discussion

Even though the distribution of x_2 is same for signal and background, x_1 and x_2 are not independent, so using x_2 as an input variable helps.

Here we can understand why: high values of x_2 correspond to a smaller σ for the Gaussian of x_1. So high x_2 means that the value of x_1 was well measured.

If we don't consider x_2, then all of the x_1 measurements are lumped together. Those with large σ (low x_2) "pollute" the well measured events with low σ (high x_2).

Often in HEP there may be variables that are characteristic of how well measured an event is (region of detector, number of pile-up vertices,...). Including these variables in a multivariate analysis preserves the information carried by the well-measured events, leading to improved performance.

Summary on multivariate methods

Particle physics has used several multivariate methods for many years:

linear (Fisher) discriminant
neural networks
naive Bayes

and has in recent years started to use a few more:

boosted decision trees
support vector machines
kernel density estimation
k-nearest neighbour

The emphasis is often on controlling systematic uncertainties between the modeled training data and Nature to avoid false discovery.

Although many classifier outputs are "black boxes", a discovery at 5σ significance with a sophisticated (opaque) method will win the competition if backed up by, say, 4σ evidence from a cut-based method.

Extra slides

Example of optimal selection region: measurement of signal cross section

Suppose that for a given event selection region, the expected numbers of signal and background events are:

$$b = \sigma_b \varepsilon_b \int L\, dt$$

$$s = \sigma_s \varepsilon_s \int L\, dt$$

cross section — efficiency — luminosity

The number n of selected events will follow a Poisson distribution with mean value $s + b$:

$$P(n|s,b) = \frac{(s+b)^n}{n!} e^{-(s+b)}$$

Optimal selection for measurement of signal rate

Suppose only unknown is s (or equivalently, σ_s) and goal is to measure this with best possible accuracy by counting the number of events n observed in data. The (log-)likelihood function is

$$L(s) = \frac{(s+b)^n}{n!} e^{-(s+b)}$$

Set derivative of $\ln L(s)$ with respect to s equal to zero and solve to find maximum-likelihood estimator:

$$\hat{s} = n - b$$

Variance of $\hat{s}$ is: $\sigma^2_{\hat{s}} = V[\hat{s}] = V[n-b] = V[n] = E[n] = s + b$

So "relative precision" of measurement is: $\dfrac{\sigma_{\hat{s}}}{s} = \dfrac{\sqrt{s+b}}{s}$

Optimal selection (continued)

So if our goal is best relative precision of a *measurement*, then choose the event selection region to maximize

$$\frac{s}{\sqrt{s+b}}$$

In other analyses, we may not know whether the signal process exists (e.g., SUSY), and goal is to search for it.

Then we try to maximize the probability, assuming the signal exists, of *discovery*, i.e., rejecting background-only hypothesis.

To do this we can maximize, e.g.,

$$\frac{s}{\sqrt{b}}$$

or similar (depending on details of problem; more on this later).

In general, optimal trade-off between efficiency and purity will depend on the goals of the analysis.