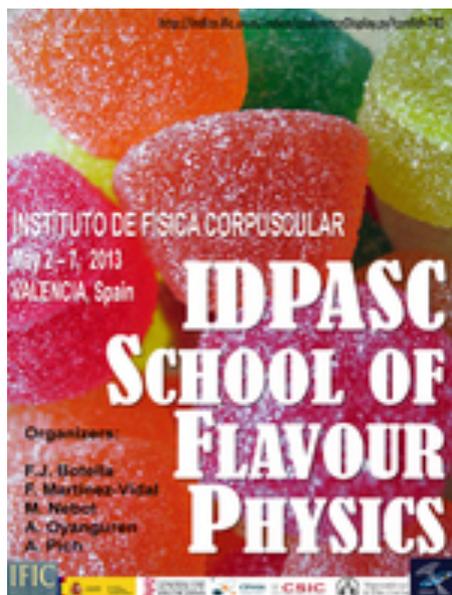


Statistical Analysis Tools for Particle Physics



IDPASC School of Flavour Physics
Valencia, 2-7 May, 2013

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



Outline

Multivariate methods for HEP

Event selection as a statistical test

Neyman-Pearson lemma and likelihood ratio test

Some multivariate classifiers:

Linear

Neural networks

Boosted Decision Trees

Support Vector Machines

Use of multivariate methods in a search (brief overview)

Resources on multivariate methods

Books:

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

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

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

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

Materials from some recent meetings:

PHYSTAT conference series (2002, 2003, 2005, 2007,...) see
www.phystat.org

Caltech workshop on multivariate analysis, 11 February, 2008
indico.cern.ch/conferenceDisplay.py?confId=27385

SLAC Lectures on Machine Learning by Ilya Narsky (2006)
www-group.slac.stanford.edu/sluo/Lectures/Stat2006_Lectures.html

Software

TMVA, Höcker, Stelzer, Tegenfeldt, Voss, Voss, [physics/0703039](#)

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

Variety of classifiers

Good manual

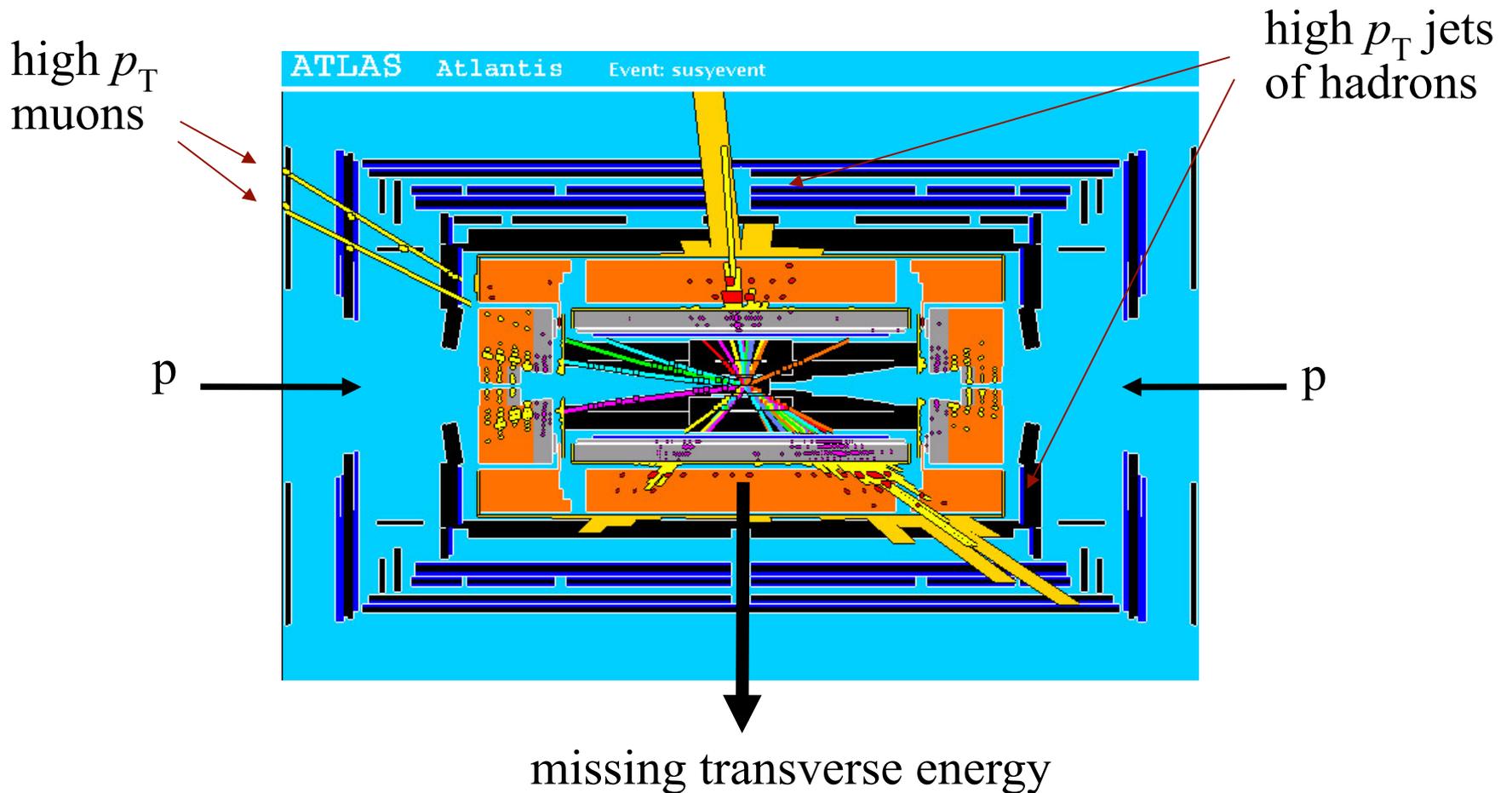
StatPatternRecognition, I. Narsky, [physics/0507143](#)

Further info from `www.hep.caltech.edu/~narsky/spr.html`

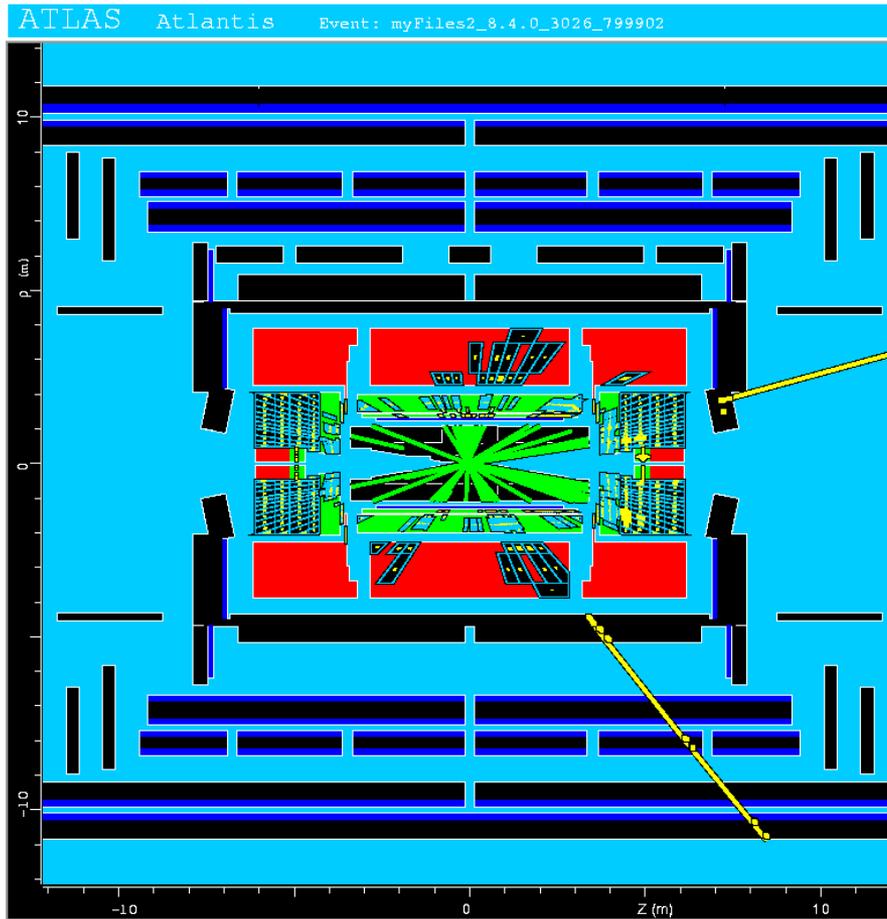
Also wide variety of methods, many complementary to **TMVA**

Currently appears project no longer to be supported

A simulated SUSY event in ATLAS



Background events



This event from Standard Model $t\bar{t}$ production also has high p_T jets and muons, and some missing transverse energy.

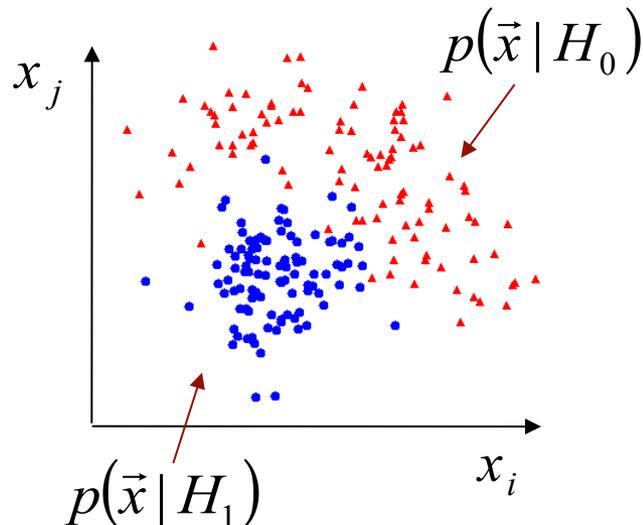
→ can easily mimic a SUSY event.

Event selection as a statistical test

For each event we measure a set of numbers: $\vec{x} = (x_1, \dots, x_n)$

$x_1 = \text{jet } p_T$
 $x_2 = \text{missing energy}$
 $x_3 = \text{particle i.d. measure, ...}$

\vec{x} follows some n -dimensional joint probability density, which depends on the type of event produced, i.e., was it $pp \rightarrow t\bar{t}$, $pp \rightarrow \tilde{g}\tilde{g}, \dots$

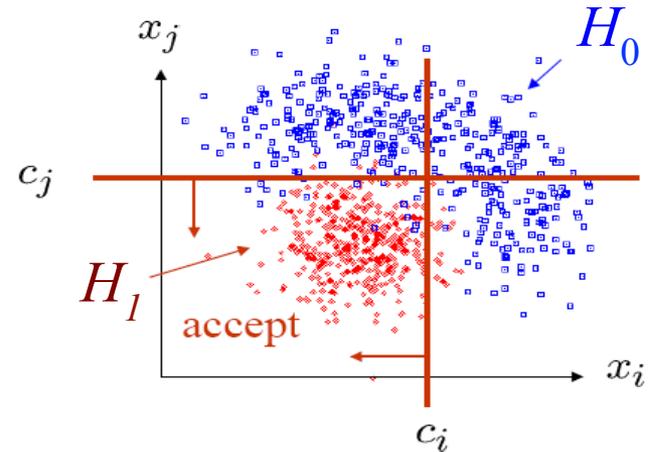


E.g. hypotheses H_0, H_1, \dots
Often simply “signal” (s),
“background” (b)

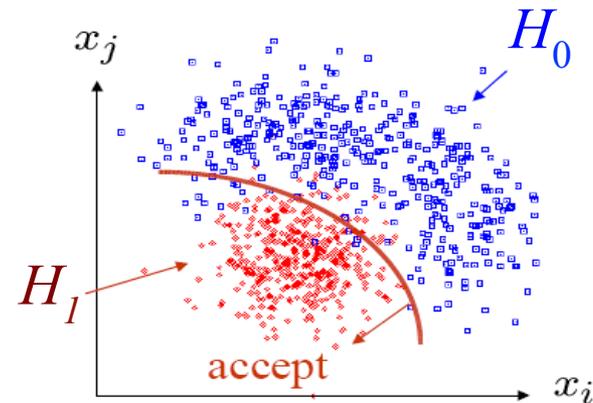
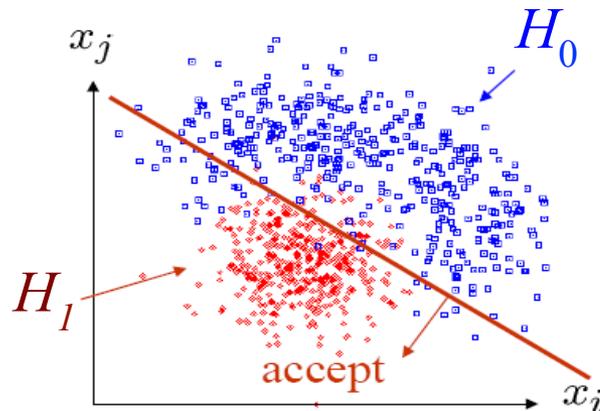
Finding an optimal decision boundary

In particle physics usually start by making simple “cuts”:

$$\begin{aligned}x_i &< c_i \\x_j &< c_j\end{aligned}$$



Maybe later try some other type of decision boundary:



General considerations

In all multivariate analyses we must consider e.g.

- Choice of variables to use

- Functional form of decision boundary (type of classifier)

- Computational issues

- Trade-off between sensitivity and complexity

- Trade-off between statistical and systematic uncertainty

Our choices can depend on goals of the analysis, e.g.,

- Event selection for further study

- Searches for new event types

Probability – quick review

Frequentist ($A =$ outcome of repeatable observation):

$$P(A) = \lim_{n \rightarrow \infty} \frac{\text{outcome is } A}{n}$$

Subjective ($A =$ hypothesis):

$$P(A) = \text{degree of belief that } A \text{ is true}$$

Conditional probability:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Bayes' theorem:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} = \frac{P(B|A)P(A)}{\sum_i P(B|A_i)P(A_i)}$$

Test statistics

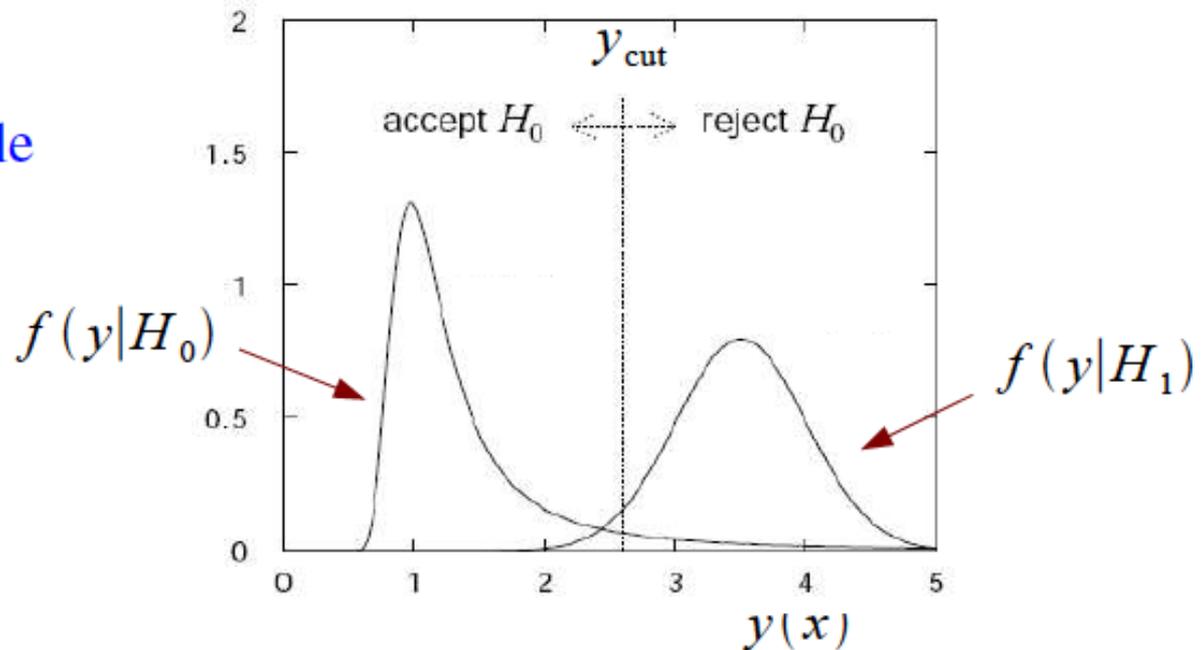
The decision boundary is a surface in the n -dimensional space of input variables, e.g., $y(\vec{x}) = \text{const}$.

We can treat the $y(x)$ as a scalar **test statistic** or discriminating function, and try to define this function so that its distribution has the maximum possible separation between the event types:

The decision boundary is now effectively a single cut on $y(\mathbf{x})$, dividing \mathbf{x} -space into two regions:

R_0 (accept H_0)

R_1 (reject H_0)



Classification viewed as a statistical test

Probability to reject H_0 if it is true (type I error): $\alpha = \int_{R_1} f(y|H_0) dy$

$\alpha =$ significance level, size of test, false discovery rate

Probability to accept H_0 if H_1 is true (type II error): $\beta = \int_{R_0} f(y|H_1) dy$

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

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

$$\varepsilon_s = \int_{R_1} f(y|H_1) dy = 1 - \beta = \text{power} \quad \varepsilon_b = \int_{R_0} f(y|H_0) dy = \alpha$$

Purity / misclassification rate

Consider the probability that an event assigned to a certain category is classified correctly (i.e., the purity).

Use Bayes' theorem:

Here R_1 is signal region prior probability

$$P(s|\mathbf{x} \in R_1) = \frac{P(\mathbf{x} \in R_1|s)P(s)}{P(\mathbf{x} \in R_1|s)P(s) + P(\mathbf{x} \in R_1|b)P(b)}$$

posterior probability

N.B. purity depends on the prior probabilities for an event to be signal or background ($\sim s, b$ cross sections).

Constructing a test statistic

The **Neyman-Pearson lemma** states: to obtain the highest signal efficiency for a given background efficiency (highest power for a test of a given size), choose the critical region for a test of the background hypothesis (acceptance region for signal) such that

$$\frac{p(\mathbf{x}|\mathbf{s})}{p(\mathbf{x}|\mathbf{b})} > c$$

where c is a constant determined by the background efficiency.

Equivalently, the optimal test statistic is given by the **likelihood ratio**:

$$y(\mathbf{x}) = \frac{p(\mathbf{x}|\mathbf{s})}{p(\mathbf{x}|\mathbf{b})}$$

N.B. any monotonic function of this is just as good.

Purity vs. efficiency trade-off

The actual choice of signal efficiency (and thus purity) will depend on goal of analysis, e.g.,

Trigger selection (high efficiency)

Event sample used for precision measurement (high purity)

Measurement of signal cross section: maximize $s/\sqrt{s+b}$

Discovery of signal: maximize expected significance $\sim s/\sqrt{b}$

Neyman-Pearson doesn't always help

The problem is that we usually don't have explicit formulae for the pdfs $p(\mathbf{x}|s)$, $p(\mathbf{x}|b)$, so for a given \mathbf{x} we can't evaluate the likelihood ratio.

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

generate $\vec{x} \sim p(\vec{x}|s)$ \longrightarrow $\vec{x}_1, \dots, \vec{x}_{N_s}$  “training data”
events of known type
generate $\vec{x} \sim p(\vec{x}|b)$ \longrightarrow $\vec{x}_1, \dots, \vec{x}_{N_b}$

Naive try: enter each (s,b) event into an n -dimensional histogram, use e.g. M bins for each of the n dimensions, total of M^n cells.

n is potentially large \rightarrow prohibitively large number of cells to populate, can't generate enough training data.

Strategies for event classification

A compromise solution is to make an Ansatz for the form of the test statistic $y(\mathbf{x})$ with fewer parameters; determine them (using MC) to give best discrimination between signal and background.

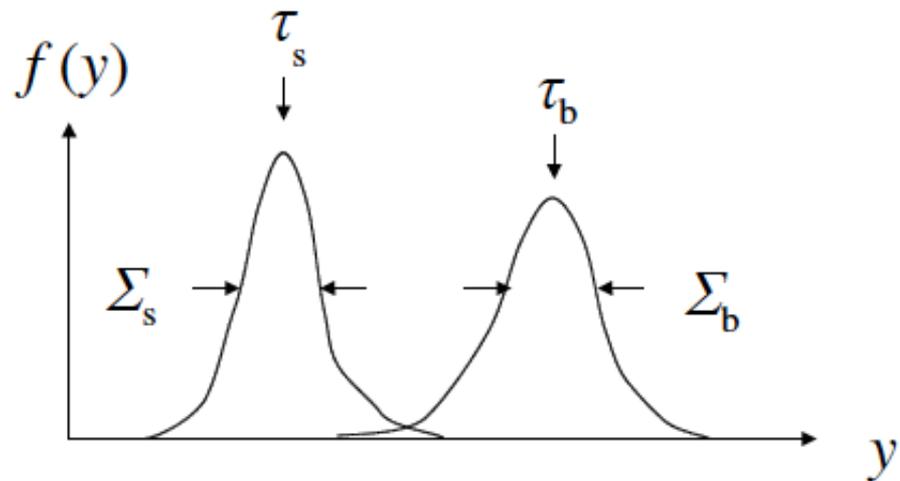
Alternatively, try to estimate the probability densities $p(\mathbf{x}|s)$, $p(\mathbf{x}|b)$ and use the estimated pdfs to compute the likelihood ratio at the desired \mathbf{x} values (for real data).

Linear test statistic

Ansatz:
$$y(\vec{x}) = \sum_{i=1}^n w_i x_i = \vec{w}^T \vec{x}$$

Choose the parameters w_1, \dots, w_n so that the pdfs $f(y|s), f(y|b)$ have maximum 'separation'. We want:

large distance between
mean values, small widths



→ Fisher: maximize
$$J(\vec{w}) = \frac{(\tau_s - \tau_b)^2}{\Sigma_s^2 + \Sigma_b^2}$$

Coefficients for maximum separation

We have $(\mu_k)_i = \int x_i p(\vec{x}|H_k) d\vec{x}$ ← mean, covariance of \mathbf{x}

$$(V_k)_{ij} = \int (x - \mu_k)_i (x - \mu_k)_j p(\vec{x}|H_k) d\vec{x}$$

where $k = 0, 1$ (hypothesis)

and $i, j = 1, \dots, n$ (component of \mathbf{x})

For the mean and variance of $y(\vec{x})$ we find

$$\tau_k = \int y(\vec{x}) p(\vec{x}|H_k) d\vec{x} = \vec{w}^T \vec{\mu}_k$$

$$\Sigma_k^2 = \int (y(\vec{x}) - \tau_k)^2 p(\vec{x}|H_k) d\vec{x} = \vec{w}^T V_k \vec{w}$$

Determining the coefficients w

The numerator of $J(w)$ is

$$(\tau_0 - \tau_1)^2 = \sum_{i,j=1}^n w_i w_j (\mu_0 - \mu_1)_i (\mu_0 - \mu_1)_j$$

$$= \sum_{i,j=1}^n w_i w_j B_{ij} = \vec{w}^T B \vec{w}$$

← ‘between’ classes

and the denominator is

$$\Sigma_0^2 + \Sigma_1^2 = \sum_{i,j=1}^n w_i w_j (V_0 + V_1)_{ij} = \vec{w}^T W \vec{w}$$

← ‘within’ classes

→ maximize

$$J(\vec{w}) = \frac{\vec{w}^T B \vec{w}}{\vec{w}^T W \vec{w}} = \frac{\text{separation between classes}}{\text{separation within classes}}$$

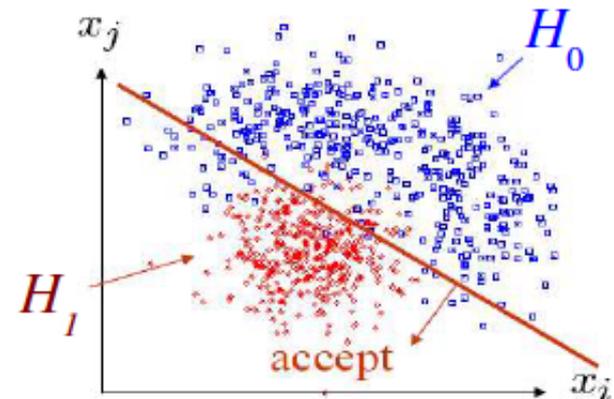
Fisher discriminant function

Setting $\frac{\partial J}{\partial w_i} = 0$ gives Fisher's linear discriminant function:

$$y(\vec{x}) = \vec{w}^T \vec{x} \quad \text{with } \vec{w} \propto W^{-1}(\vec{\mu}_0 - \vec{\mu}_1)$$

Gives linear decision boundary.

Projection of points in direction of decision boundary gives maximum separation.



Fisher discriminant for Gaussian data

Suppose $f(\mathbf{x}|H_k)$ is a multivariate Gaussian with mean values

$$E_0[\vec{x}] = \vec{\mu}_0 \text{ for } H_0 \quad E_1[\vec{x}] = \vec{\mu}_1 \text{ for } H_1$$

and covariance matrices $V_0 = V_1 = V$ for both. We can write the Fisher's discriminant function (with an offset) is

$$y(\vec{x}) = w_0 + (\vec{\mu}_0 - \vec{\mu}_1)^T V^{-1} \vec{x}$$

The likelihood ratio is thus

$$\begin{aligned} \frac{p(\vec{x}|H_0)}{p(\vec{x}|H_1)} &= \exp\left[-\frac{1}{2}(\vec{x} - \vec{\mu}_0)^T V^{-1}(\vec{x} - \vec{\mu}_0) + \frac{1}{2}(\vec{x} - \vec{\mu}_1)^T V^{-1}(\vec{x} - \vec{\mu}_1)\right] \\ &= e^y \end{aligned}$$

Fisher for Gaussian data (2)

That is, $y(\mathbf{x})$ is a monotonic function of the likelihood ratio, so for this case the Fisher discriminant is equivalent to using the likelihood ratio, and is therefore optimal.

For non-Gaussian data this no longer holds, but linear discriminant function may be simplest practical solution.

Often try to transform data so as to better approximate Gaussian before constructing Fisher discriminant.

Fisher and Gaussian data (3)

Multivariate Gaussian data with equal covariance matrices also gives a simple expression for posterior probabilities, e.g.,

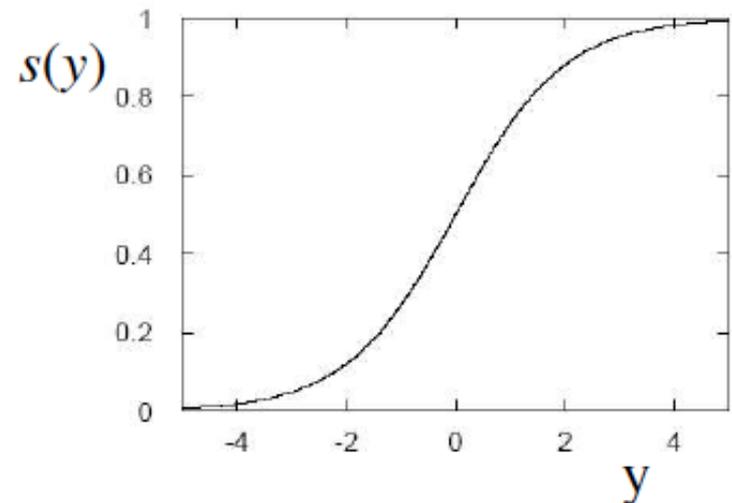
$$P(H_0|\vec{x}) = \frac{p(\vec{x}|H_0)P(H_0)}{p(\vec{x}|H_0)P(H_0) + p(\vec{x}|H_1)P(H_1)}$$

For Gaussian \mathbf{x} and a particular choice of the offset w_0 this becomes:

$$P(H_0|\vec{x}) = \frac{1}{1 + e^{-y(\vec{x})}} \equiv s(y(\vec{x}))$$

which is the **logistic sigmoid function**:

(We will use this later in connection with Neural Networks.)



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

$$\phi_1(\vec{x}), \dots, \phi_m(\vec{x})$$

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).

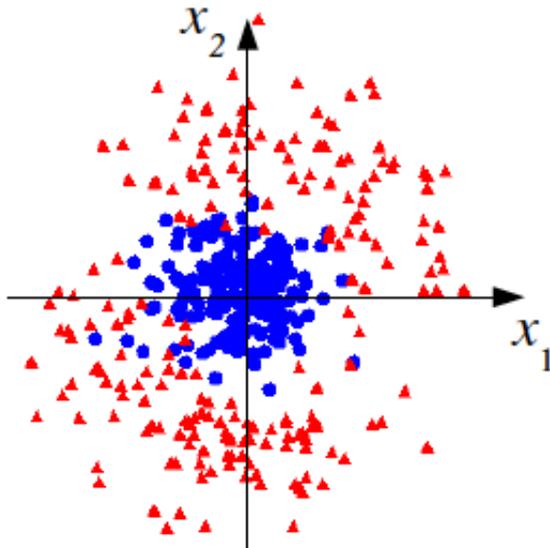
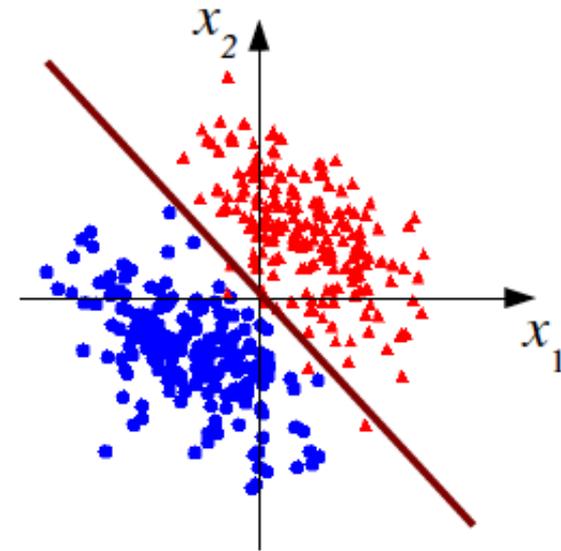
In other cases we will see that the basis functions only enter as dot products

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

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

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.

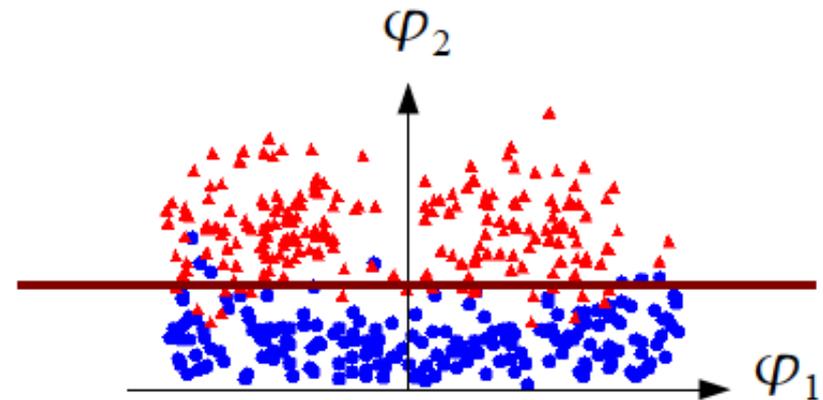
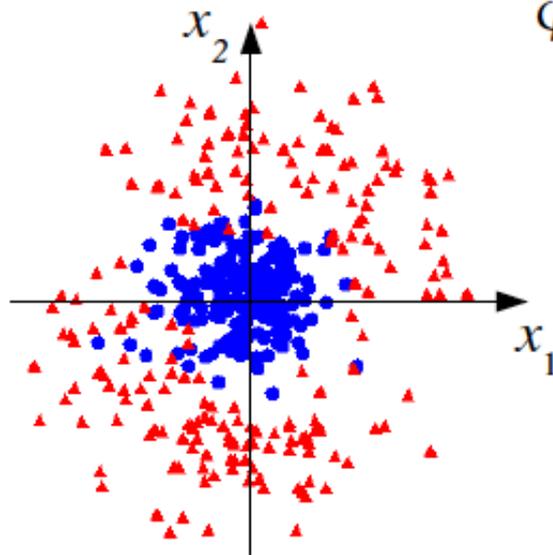
Nonlinear transformation of inputs

We can try to find a transformation, $x_1, \dots, x_n \rightarrow \varphi_1(\vec{x}), \dots, \varphi_m(\vec{x})$ 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.

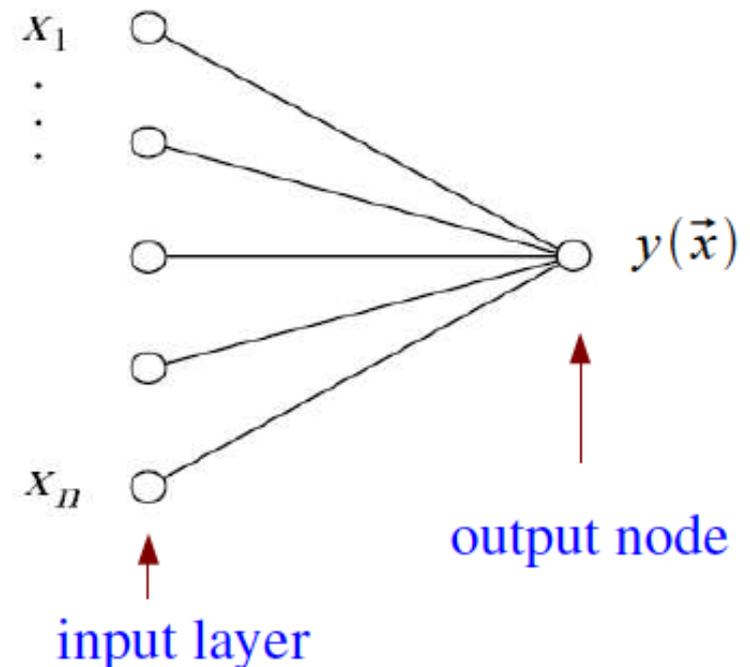
The training data are then used to adjust the parameters so that the resulting discriminant function has the best performance.

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 $y(\mathbf{x})$ is equivalent to the original linear discriminant. This is an example of a “generalized linear model” called the **single layer perceptron**.



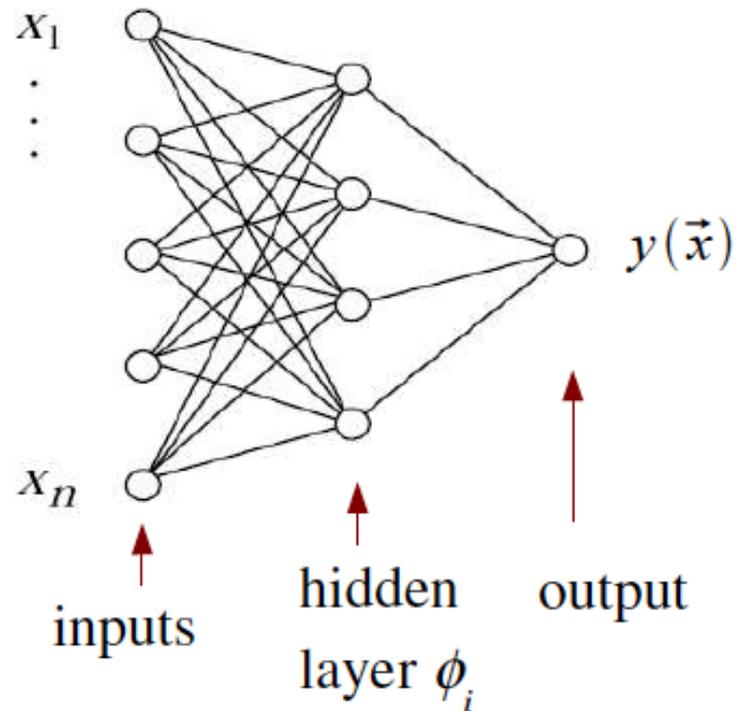
The multilayer perceptron

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

Superscript for weights indicates layer number

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

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



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

Network architecture: one hidden layer

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

In practice often choose a single hidden layer and try increasing the number of nodes until no further improvement in performance is found.

More than one hidden layer

“Relatively little is known concerning the advantages and disadvantages of using a single hidden layer with many units (neurons) over many hidden layers with fewer units. The mathematics and approximation theory of the MLP model with more than one hidden layer is not well understood.”

“Nonetheless there seems to be reason to conjecture that the two hidden layer model may be significantly more promising than the single hidden layer model, ...”

A. Pinkus, *Approximation theory of the MLP model in neural networks*, Acta Numerica (1999), pp. 143—195.

Network training

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

$$\vec{x}_a = (x_1, \dots, x_n) \quad \text{the input variables, and}$$
$$t_a = 0, 1 \quad \text{a numerical label for event type (“target value”)}$$

Let \mathbf{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”

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


Contribution to error function
from each event

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

Consider gradient descent method: from an initial guess in weight space $\mathbf{w}^{(1)}$ take a small step in the direction of maximum decrease.

I.e. for the step τ to $\tau+1$,

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



learning rate ($\eta > 0$)

If we do this with the full error function $E(\mathbf{w})$, 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 \mathbf{w} for each training event a , (cycle through all training events):

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

Error backpropagation

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

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

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

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 $\frac{\partial E_a}{\partial w_{1j}^{(2)}} = (y_a - t_a) h'(u(\vec{x})) \varphi_j(\vec{x})$

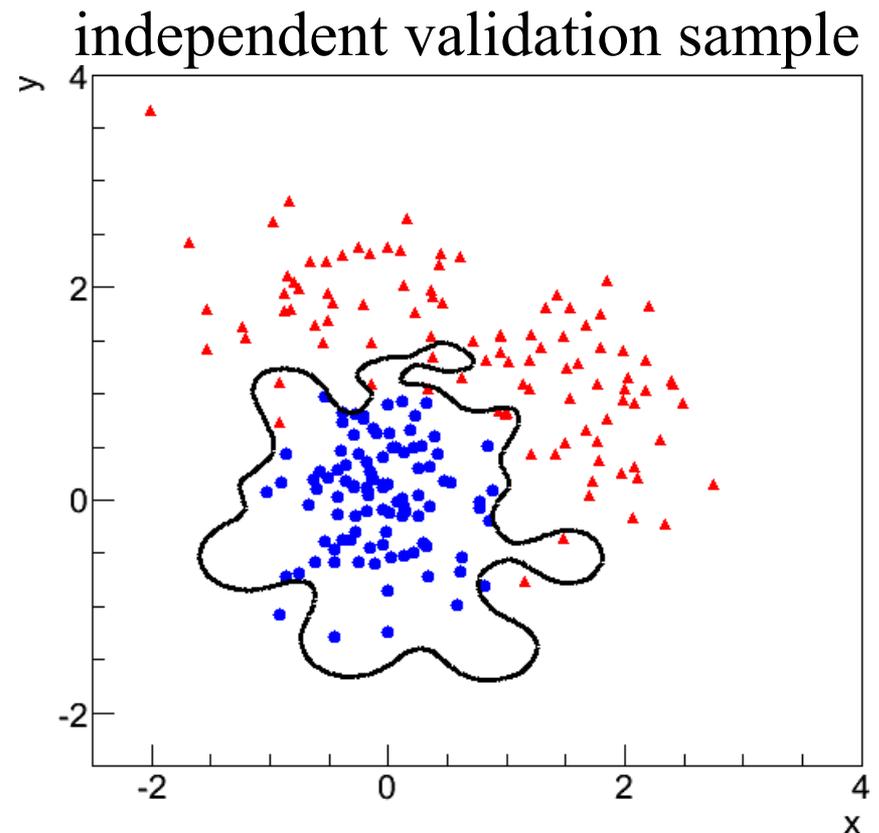
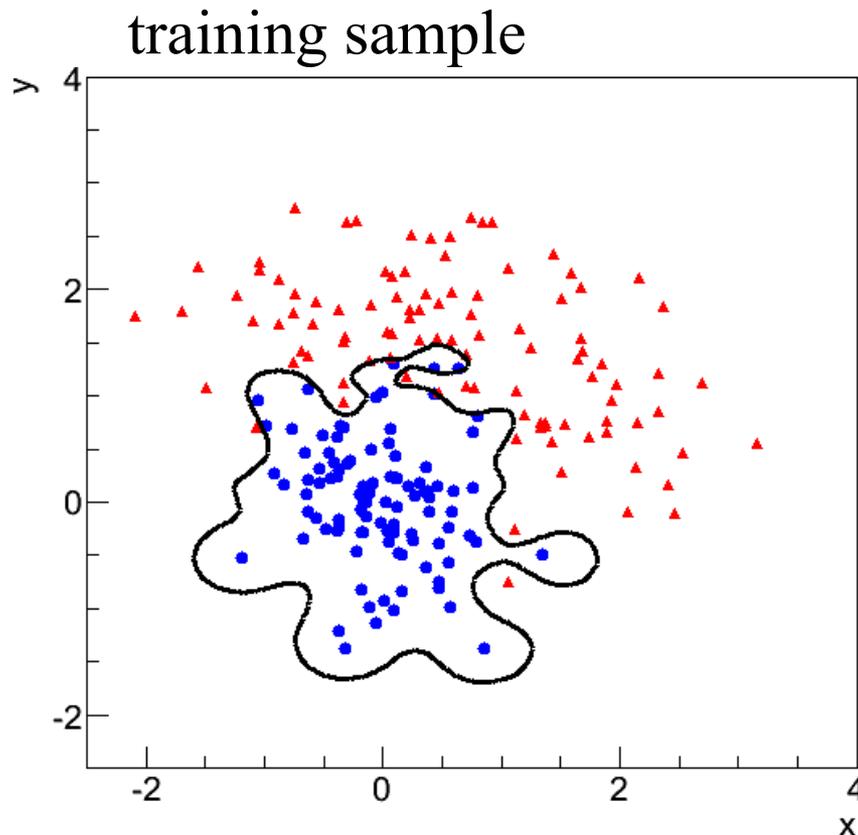
Chain rule gives all the needed derivatives.

 derivative of
activation function

Overtraining

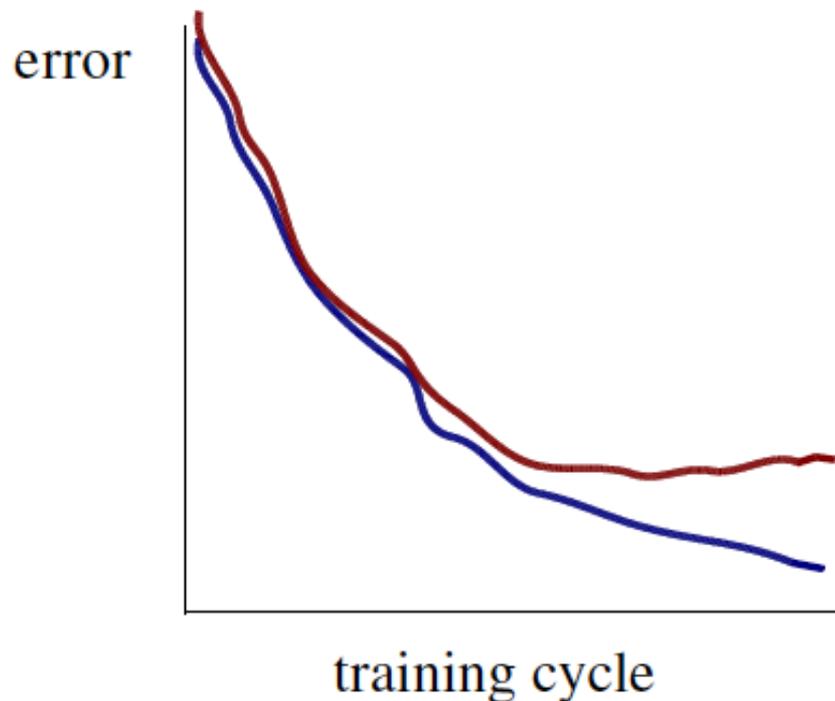
If decision boundary is too flexible it will conform too closely to the training points \rightarrow **overtraining**.

Monitor by applying classifier to independent validation sample.



Monitoring overtraining

If we monitor the value of the error function $E(\mathbf{w})$ at every cycle of the minimization, for the training sample it will continue to decrease.



But the validation sample it may initially decrease, and then at some point increase, indicating overtraining.

validation sample

training sample

Choose classifier that minimizes error function for validation sample.

Validation and testing

The validation sample can be used to make various choices about the network architecture, e.g., adjust the number of hidden nodes so as to obtain good “**generalization performance**” (ability to correctly classify unseen data).

If the validation stage is iterated many times, the estimated error rate based on the validation sample has a bias, so strictly speaking one should finally estimate the error rate with an independent **test sample**.

Rule of thumb if data not	train	:	validate	:	test
too expensive (Narsky):	50	:	25	:	25

But this depends on the type of classifier. Often the bias in the error rate from the validation sample is small and one can omit the test step.

Regularized neural networks

Often one uses the test sample to optimize the number of hidden nodes.

Alternatively one may use a relatively large number of hidden nodes but include in the error function a regularization term that penalizes overfitting, e.g.,

$$\tilde{E}(\mathbf{w}) = E(\mathbf{w}) + \frac{\lambda}{2} \mathbf{w}^T \mathbf{w}$$

regularization parameter 

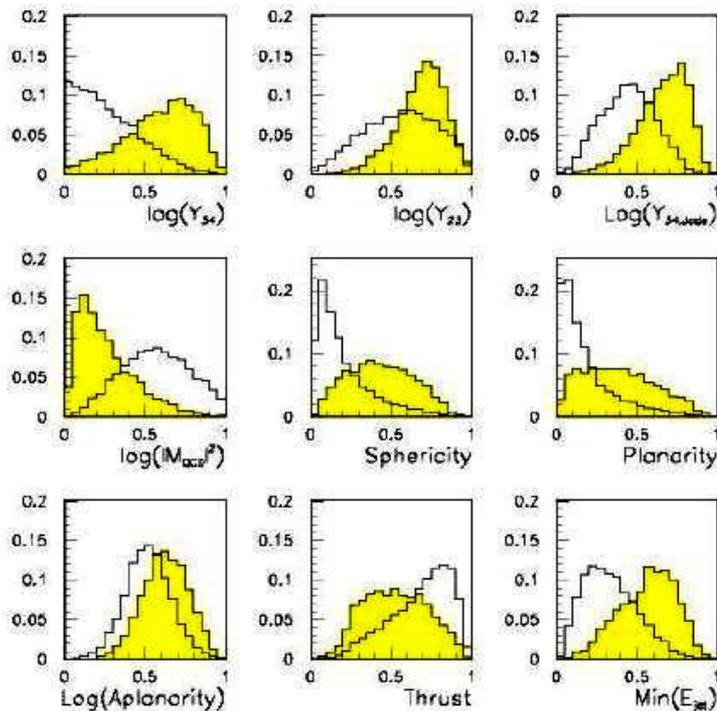
Increasing λ gives a smoother boundary (higher bias, lower variance)

Known as “weight decay”, since the weights are driven to zero unless supported by the data (an example of “parameter shrinkage”).

Neural network example from LEP II

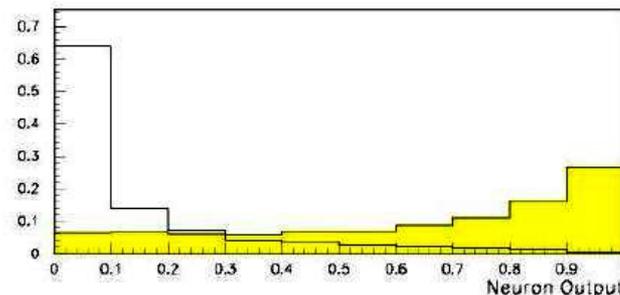
Signal: $e^+e^- \rightarrow W^+W^-$ (often 4 well separated hadron jets)

Background: $e^+e^- \rightarrow q\bar{q}g$ (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 \mathbf{x} for the two event classes, $p(\mathbf{x}|H_0)$, $p(\mathbf{x}|H_1)$ 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 $p(\mathbf{x})$ does **not** factorize into a product of the marginal distributions for the individual variables:

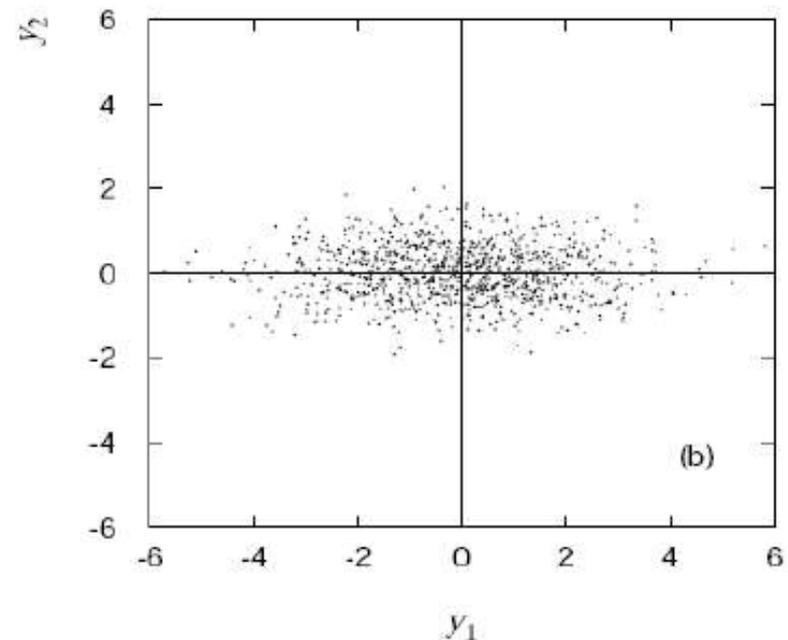
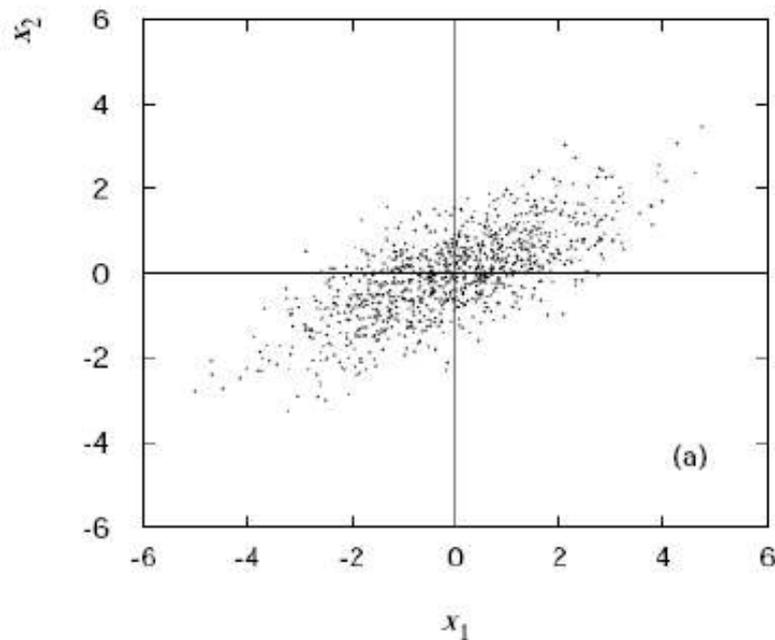
$$p(\vec{x}) = \prod_{i=1}^n p_i(x_i) \quad \leftarrow \text{holds only if the components of } \mathbf{x} \text{ are independent}$$

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

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

Decorrelation of input variables

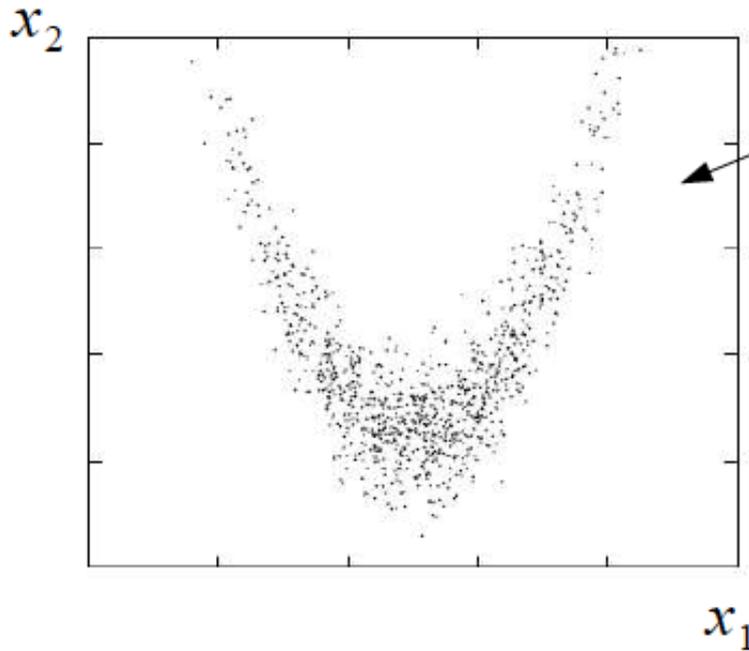
But we can define a set of uncorrelated input variables by a linear transformation, i.e., find the matrix A such that for $\vec{y} = A \vec{x}$ the covariances $\text{cov}[y_i, y_j] = 0$:



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 $p(\mathbf{x})$ 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

But if the nonlinearities are not too great, it is reasonable to first decorrelate the inputs and take as our estimator for each pdf

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

So this at least reduces the problem to one of finding estimates of one-dimensional pdfs.

The resulting estimated likelihood ratio gives the **Naive Bayes classifier** (in HEP sometimes called the “likelihood method”).

Kernel-based PDE (KDE, Parzen window)

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

$$\hat{f}(\vec{x}) = \frac{1}{Nh^d} \sum_{i=1}^N K\left(\frac{\vec{x} - \vec{x}_i}{h}\right)$$

kernel

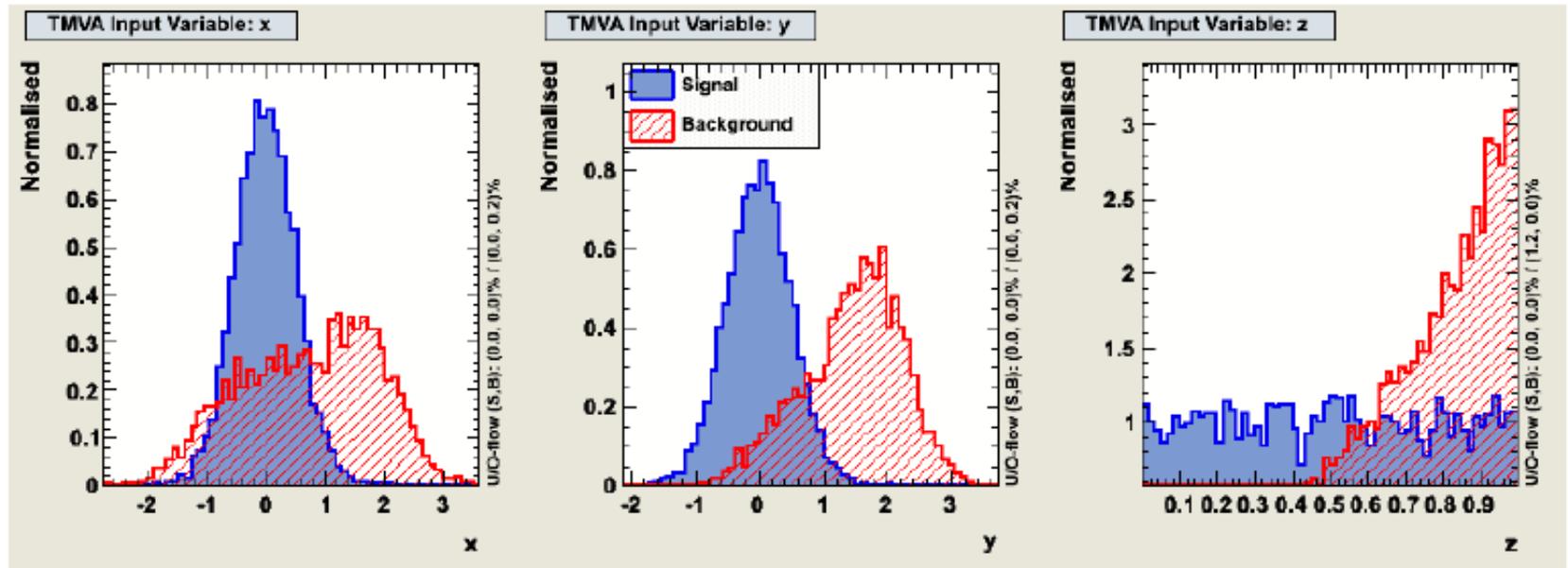
bandwidth
(smoothing parameter)

Use e.g. Gaussian kernel:

$$K(\vec{x}) = \frac{1}{(2\pi)^{d/2}} e^{-|\vec{x}|^2/2}$$

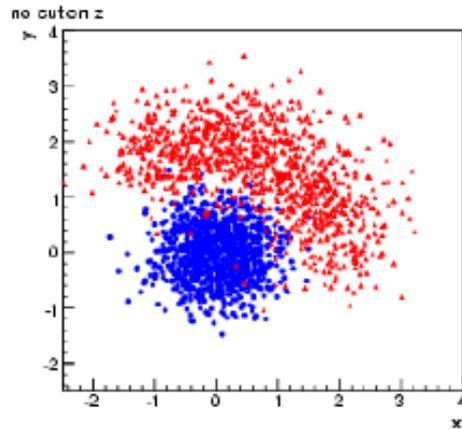
Need to sum N terms to evaluate function (slow);
faster algorithms only count events in vicinity of \mathbf{x}
(k -nearest neighbor, range search).

Test example with TMVA

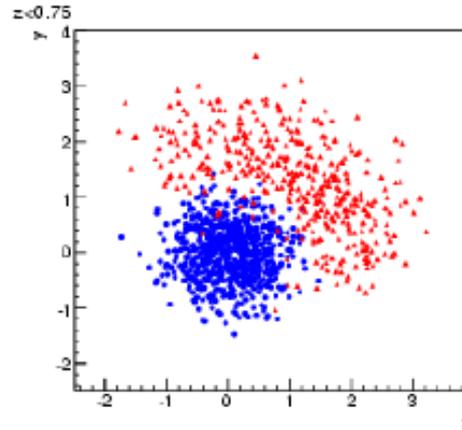


Test example, x vs. y with cuts on z

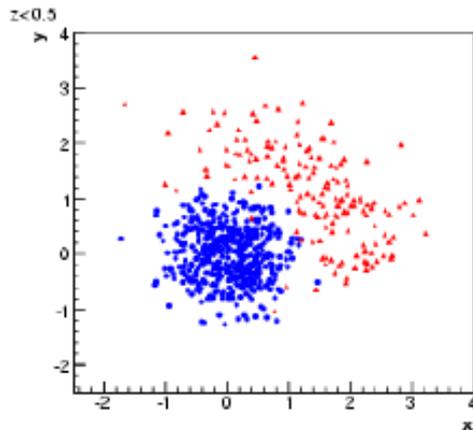
no cut on z



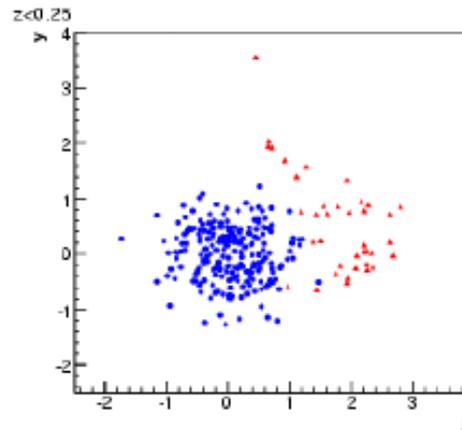
$z < 0.75$



$z < 0.5$

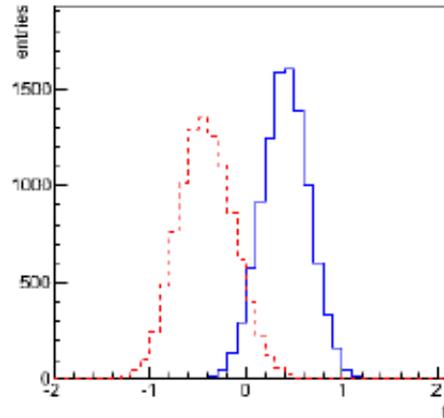


$z < 0.25$

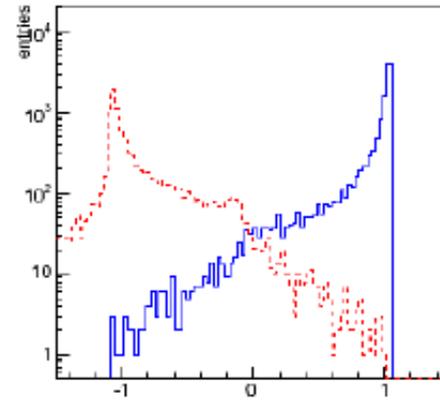


Test example results

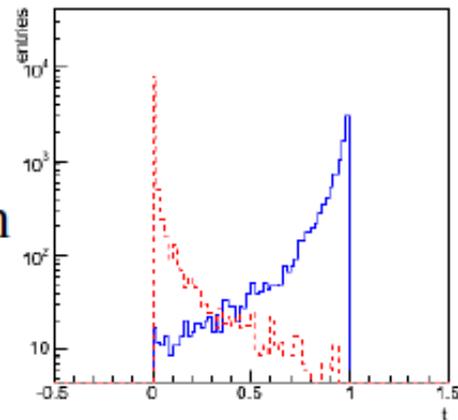
Fisher
discriminant



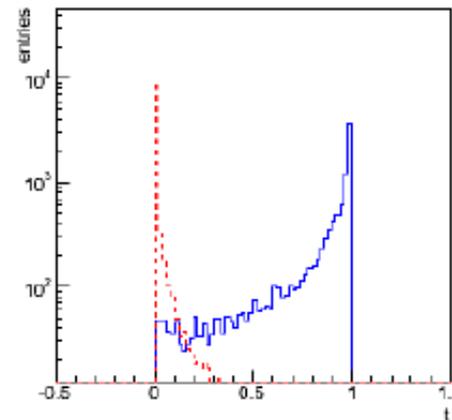
Multilayer
perceptron



Naive Bayes,
no decorrelation

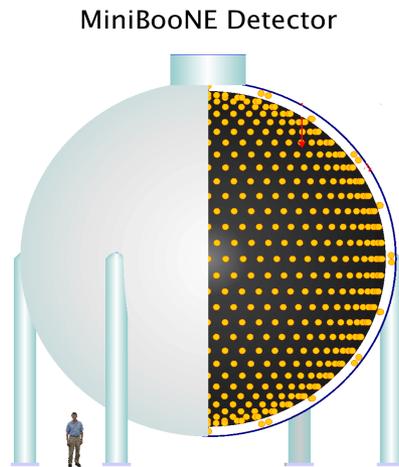


Naive Bayes with
decorrelation

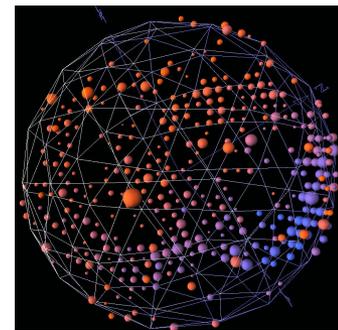
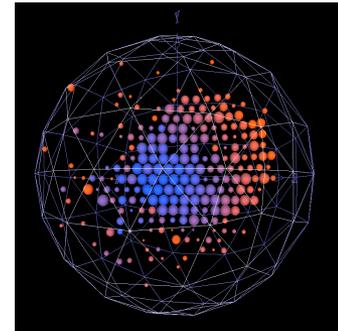
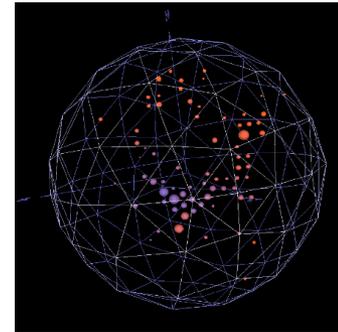
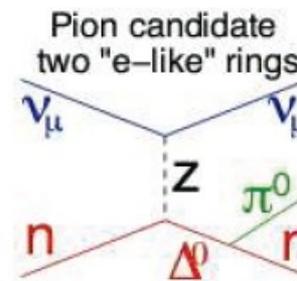
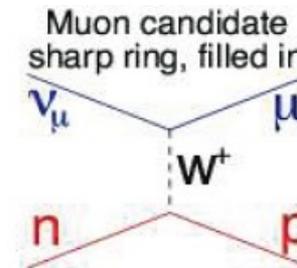
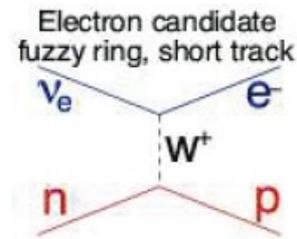


Particle i.d. in MiniBooNE

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:

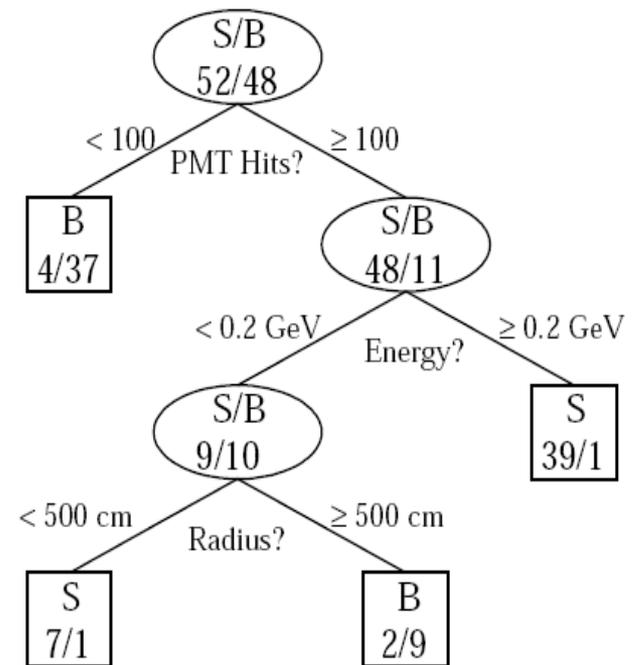
$$P = \frac{\sum_{\text{signal}} w_i}{\sum_{\text{signal}} w_i + \sum_{\text{background}} w_i}$$

where w_i is the weight of the i th 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

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

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

$\mathbf{x}_1, \dots, \mathbf{x}_N$ event data vectors (each \mathbf{x} multivariate)

y_1, \dots, y_N true class labels, +1 for signal, -1 for background

w_1, \dots, w_N event weights

Now define a rule to create from this an ensemble of training samples T_1, T_2, \dots , 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

$\mathbf{x}_1, \dots, \mathbf{x}_N$ event data vectors

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

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

with the weights equal and normalized such that

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

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

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

$f_1(\mathbf{x}_i) < 0$ 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 k th classifier

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

$$\varepsilon_k = \sum_{i=1}^N w_i^{(k)} I(y_i f_k(\mathbf{x}_i) \leq 0)$$

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

Assigning the classifier score

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

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

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

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

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

AdaBoost error rate

So providing each classifier in the ensemble has $\epsilon_k < 1/2$, 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.

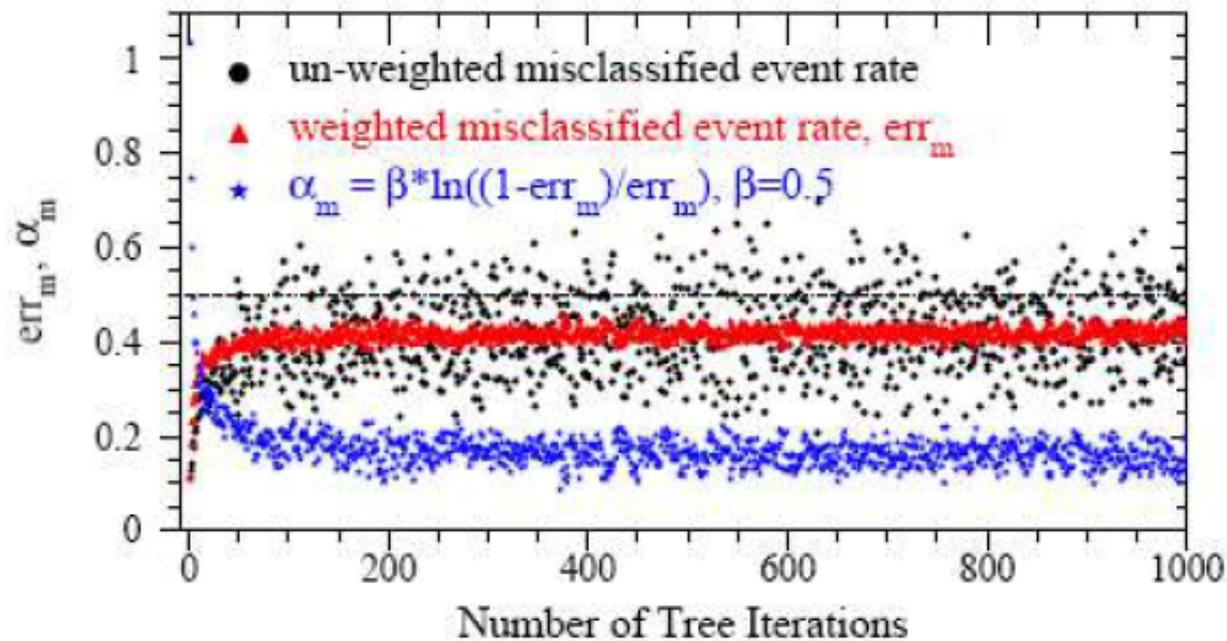
So the procedure is to monitor the error rate of the combined classifier at each step with a validation sample and to stop before it starts to rise.

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 $\sim 0.4 - 0.45$



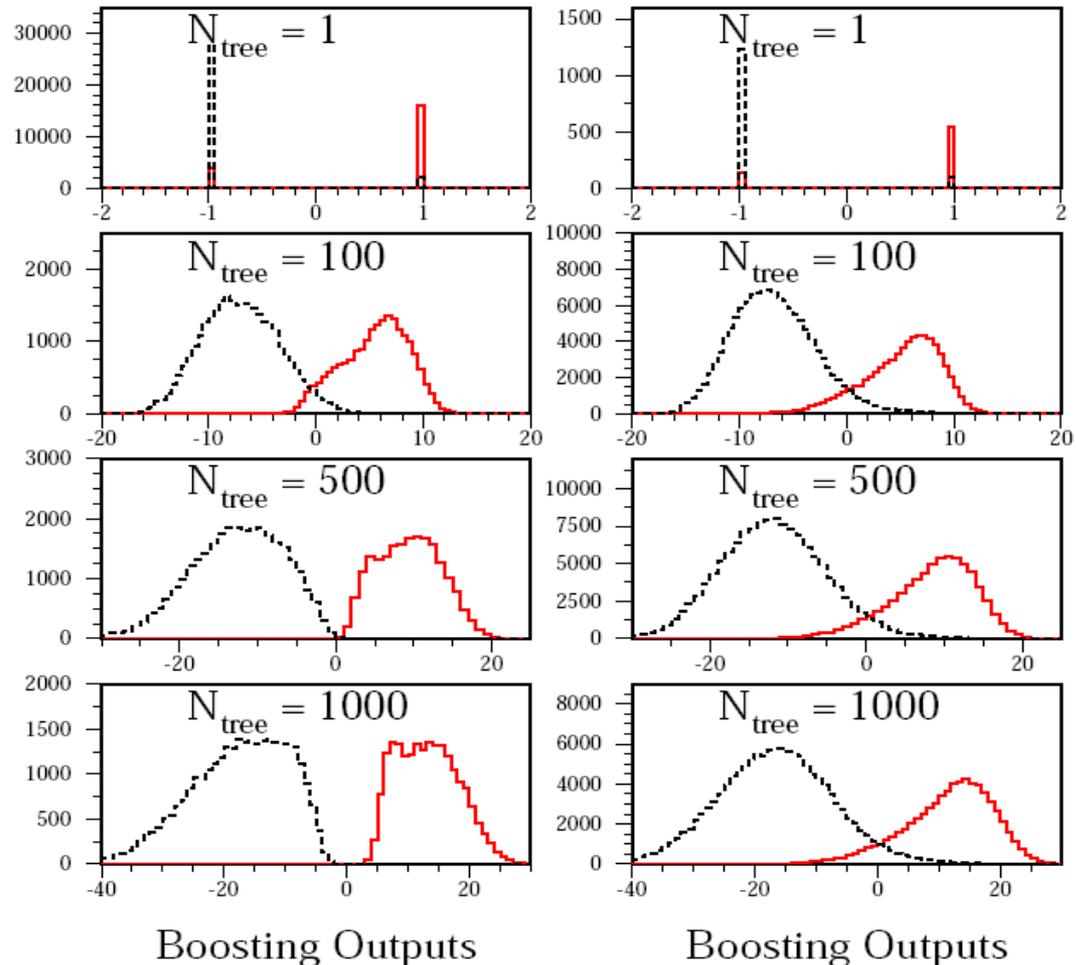
B. Roe et al., NIM 543 (2005) 577

Monitoring overtraining

From MiniBooNE
example:

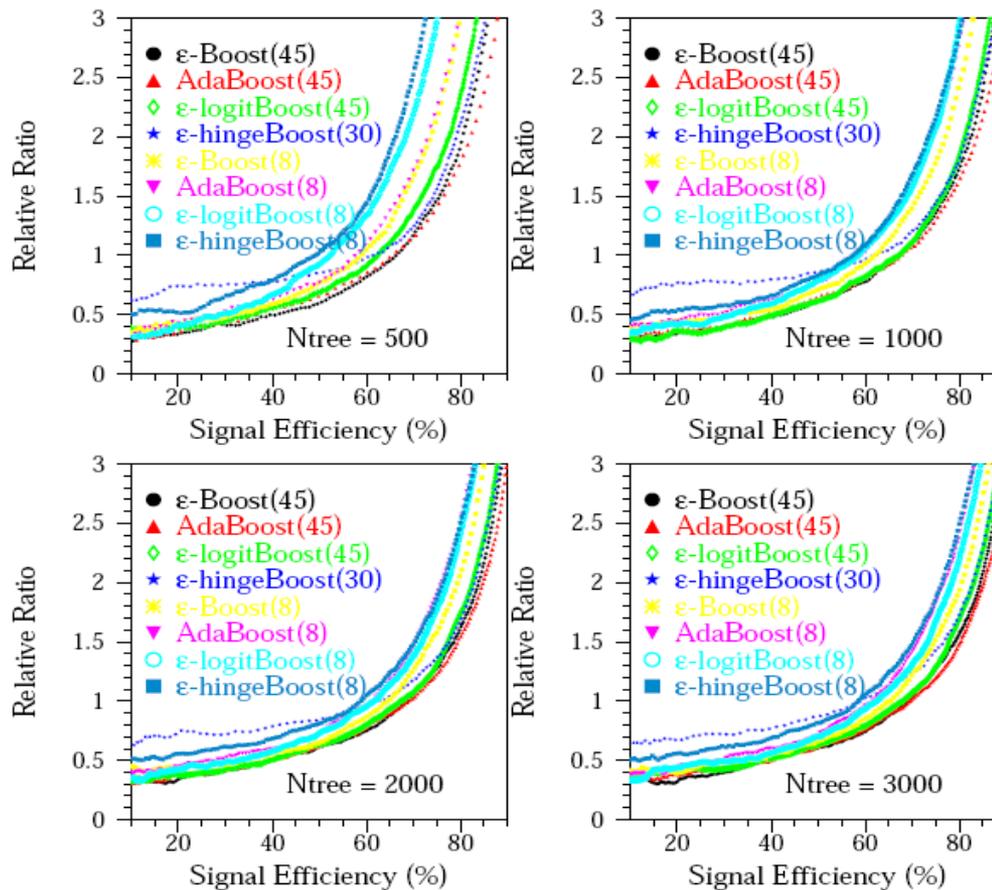
Performance stable
after a few hundred
trees.

Training MC Samples .VS. Testing MC Samples



Comparison of boosting algorithms

A number of boosting algorithms on the market; differ in the update rule for the weights.



Boosted decision tree summary

Advantage of boosted decision tree is it can handle a large number of inputs. Those that provide little/no separation are rarely used as tree splitters are effectively ignored.

Easy to deal with inputs of mixed types (real, integer, categorical...).

If a tree has only a few leaves it is easy to visualize (but rarely use only a single tree).

There are a number of boosting algorithms, which differ primarily in the rule for updating the weights (ϵ -Boost, LogitBoost,...)

Other ways of combining weaker classifiers: Bagging (Bootstrap-Aggregating), generates the ensemble of classifiers by random sampling with replacement from the full training sample.

Support Vector Machines

Support Vector Machines (SVMs) are an example of a kernel-based classifier, which exploits a nonlinear mapping of the input variables onto a higher dimensional feature space.

The SVM finds a linear decision boundary in the higher dimensional space.

But thanks to the “kernel trick” one does not every have to write down explicitly the feature space transformation.

Some references for kernel methods and SVMs:

The books mentioned earlier

C. Burges, A Tutorial on Support Vector Machines for Pattern Recognition,
research.microsoft.com/~cburges/papers/SVMTutorial.pdf

N. Cristianini and J.Shawe-Taylor. An Introduction to Support Vector Machines and other kernel-based learning methods. Cambridge University Press, 2000.

The TMVA manual (!)

Support Vector Machines

Map input variables into high dimensional feature space: $\mathbf{x} \rightarrow \phi$

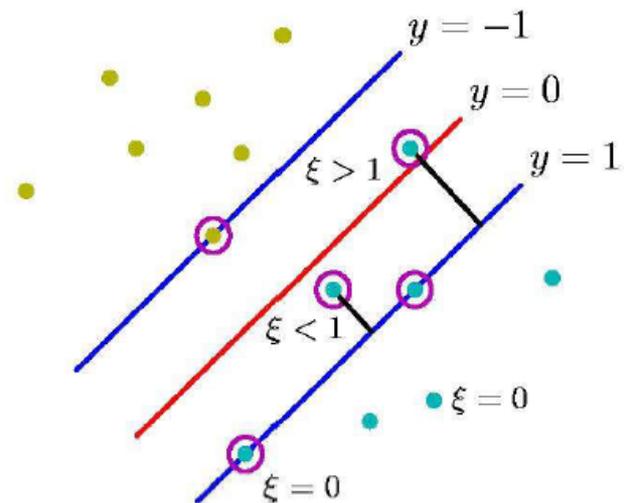
Maximize distance between separating hyperplanes (margin)
subject to constraints allowing for some misclassification.

Final classifier only depends on scalar
products of $\phi(\mathbf{x})$:

$$y(\mathbf{x}) = \text{sign} \left(\sum_i \alpha_i y_i \vec{\phi}(\mathbf{x}) \cdot \vec{\phi}(\mathbf{x}_i) + b \right)$$

So only need kernel

$$K(\mathbf{x}, \mathbf{x}') = \vec{\phi}(\mathbf{x}) \cdot \vec{\phi}(\mathbf{x}')$$



Linear SVMs

Consider a training data set consisting of

$\mathbf{x}_1, \dots, \mathbf{x}_N$ event data vectors

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

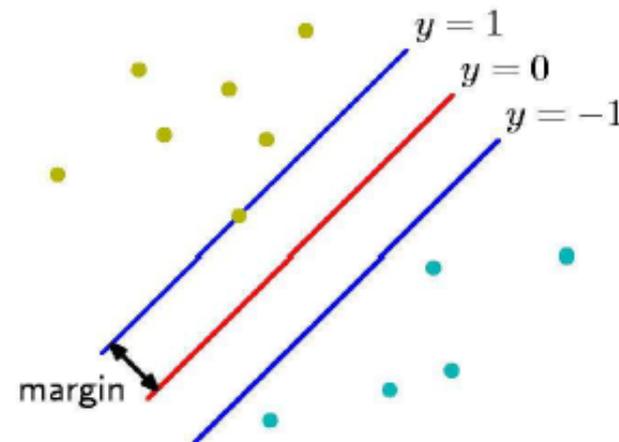
Suppose the classes can be separated by a hyperplane defined by a normal vector \mathbf{w} and scalar offset b (the “bias”). We have

$$\mathbf{x}_i \cdot \mathbf{w} + b \geq +1 \quad \text{for all } y_i = +1$$

$$\mathbf{x}_i \cdot \mathbf{w} + b \leq -1 \quad \text{for all } y_i = -1$$

or equivalently

$$y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 \geq 0 \quad \text{for all } i$$

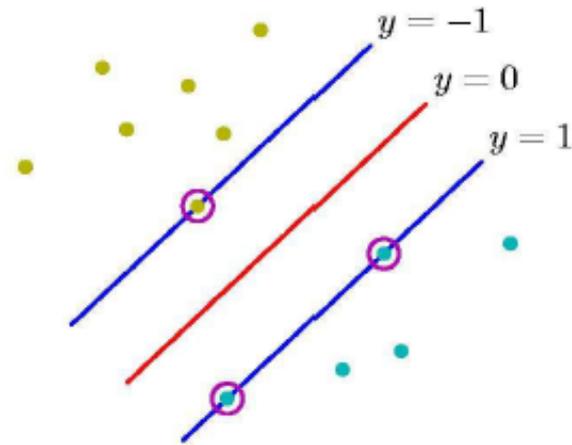


Bishop Ch. 7

Margin and support vectors

The distance between the hyperplanes defined by $y(\mathbf{x}) = +1$ and $y(\mathbf{x}) = -1$ is called the **margin**, which is:

$$\text{margin} = \frac{2}{\|\mathbf{w}\|}$$



If the training data are perfectly separated then this means there are no points inside the margin.

Suppose there are points on the margin (this is equivalent to defining the scale of \mathbf{w}). These points are called **support vectors**.

Linear SVM classifier

We can define the classifier using

$$y(\mathbf{x}) = \text{sign}(\mathbf{x} \cdot \mathbf{w} + b)$$

which is +1 for points on one side of the hyperplane and -1 on the other.

The best classifier should have a large margin, so to maximize

$$\text{margin} = \frac{2}{\|\mathbf{w}\|}$$

we can minimize $\|\mathbf{w}\|^2$ subject to the constraints

$$y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 \geq 0 \quad \text{for all } i$$

Lagrangian formulation

This constrained minimization problem can be reformulated using a Lagrangian

$$L = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^N \alpha_i (y_i (\mathbf{x}_i \cdot \mathbf{w} + b) - 1)$$

positive Lagrange multipliers α_i

We need to minimize L with respect to \mathbf{w} and b and maximize with respect to α_i .

There is an α_i for every training point. Those that lie on the margin (the support vectors) have $\alpha_i > 0$, all others have $\alpha_i = 0$. The solution can be written

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i \quad (\text{sum only contains support vectors})$$

Dual formulation

The classifier function is thus

$$y(\mathbf{x}) = \text{sign}(\mathbf{x} \cdot \mathbf{w} + b) = \text{sign}\left(\sum_i \alpha_i y_i \mathbf{x} \cdot \mathbf{x}_i + b\right)$$

It can be shown that one finds the same solution a by minimizing the dual Lagrangian

$$L_D = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j$$

So this means that both the classifier function and the Lagrangian only involve dot products of vectors in the input variable space.

Nonseparable data

If the training data points cannot be separated by a hyperplane, one can redefine the constraints by adding slack variables ξ_i :

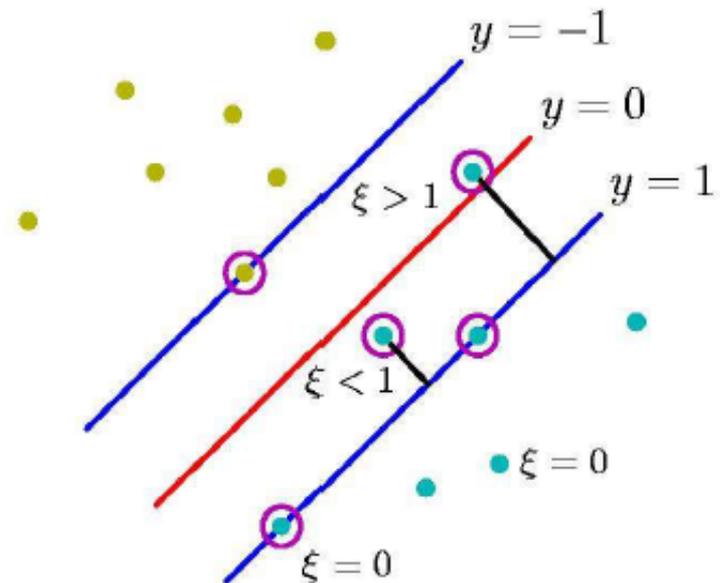
$$y_i(\mathbf{x}_i \cdot \mathbf{w} + b) + \xi_i - 1 \geq 0 \text{ with } \xi_i \geq 0 \text{ for all } i$$

Thus the training point \mathbf{x}_i is allowed to be up to a distance ξ_i on the wrong side of the margin, and $\xi_i = 0$ at or on the right side of the margin.

For an error to occur we have $\xi_i > 1$, so

$$\sum_i \xi_i$$

is an upper bound on the number of training errors.



Cost function for nonseparable case

To limit the magnitudes of the ξ_i we can define the error function that we minimize to determine \mathbf{w} to be

$$E(\mathbf{w}) = \frac{1}{2} \|\mathbf{w}\|^2 + C \left(\sum_i \xi_i \right)^k$$

where C is a cost parameter we must choose that limits the amount of misclassification. It turns out that for $k=1$ or 2 this is a quadratic programming problem and furthermore for $k=1$ it corresponds to minimizing the same dual Lagrangian

$$L_D = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j$$

where the constraints on the α_i become $0 \leq \alpha_i \leq C$.

Nonlinear SVM

So far we have only reformulated a way to determine a linear classifier, which we know is useful only in limited circumstances.

But the important extension to nonlinear classifiers comes from first transforming the input variables to feature space:

$$\vec{\phi}(\mathbf{x}) = (\varphi_1(\mathbf{x}), \dots, \varphi_m(\mathbf{x}))$$

These will behave just as our new “input variables”. Everything about the mathematical formulation of the SVM will look the same as before except with $\phi(\mathbf{x})$ appearing in the place of \mathbf{x} .

Only dot products

Recall the SVM problem was formulated entirely in terms of dot products of the input variables, e.g., the classifier is

$$y(\mathbf{x}) = \text{sign} \left(\sum_i \alpha_i y_i \mathbf{x} \cdot \mathbf{x}_i + b \right)$$

so in the feature space this becomes

$$y(\mathbf{x}) = \text{sign} \left(\sum_i \alpha_i y_i \vec{\varphi}(\mathbf{x}) \cdot \vec{\varphi}(\mathbf{x}_i) + b \right)$$

The Kernel trick

How do the dot products help? It turns out that a broad class of **kernel functions** can be written in the form:

$$K(\mathbf{x}, \mathbf{x}') = \vec{\phi}(\mathbf{x}) \cdot \vec{\phi}(\mathbf{x}')$$

Functions having this property must satisfy Mercer's condition

$$\int \int K(\mathbf{x}, \mathbf{x}') g(\mathbf{x}) g(\mathbf{x}') d\mathbf{x} d\mathbf{x}' \geq 0$$

for any function g where $\int g^2(\mathbf{x}) d\mathbf{x}$ is finite.

So we don't even need to find explicitly the feature space transformation $\phi(\mathbf{x})$, we only need a kernel.

Finding kernels

There are a number of techniques for finding kernels, e.g., constructing new ones from known ones according to certain rules (cf. Bishop Ch 6).

Frequently used kernels to construct classifiers are e.g.

$$K(\mathbf{x}, \mathbf{x}') = (\mathbf{x} \cdot \mathbf{x}' + \theta)^p \quad \text{polynomial}$$

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(\frac{-\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right) \quad \text{Gaussian}$$

$$K(\mathbf{x}, \mathbf{x}') = \tanh(\kappa(\mathbf{x} \cdot \mathbf{x}') + \theta) \quad \text{sigmoidal}$$

Using an SVM

To use an SVM the user must as a minimum choose

- a kernel function (e.g. Gaussian)

- any free parameters in the kernel (e.g. the σ of the Gaussian)

- the cost parameter C (plays role of regularization parameter)

The training is relatively straightforward because, in contrast to neural networks, the function to be minimized has a single global minimum.

Furthermore evaluating the classifier only requires that one retain and sum over the support vectors, a relatively small number of points.

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

- k -nearest neighbour

- boosted decision trees

- support vector machines

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.

Quotes I like

*“Keep it simple.
As simple as possible.
Not any simpler.”*

– A. Einstein

*“If you believe in something
you don't understand, you suffer...”*

– Stevie Wonder

Extra slides

Frequentist statistical tests – quick review

Consider a hypothesis H_0 and alternative H_1 , which each specify the probability for data x ., i.e., $P(x|H_0)$, $P(x|H_1)$.

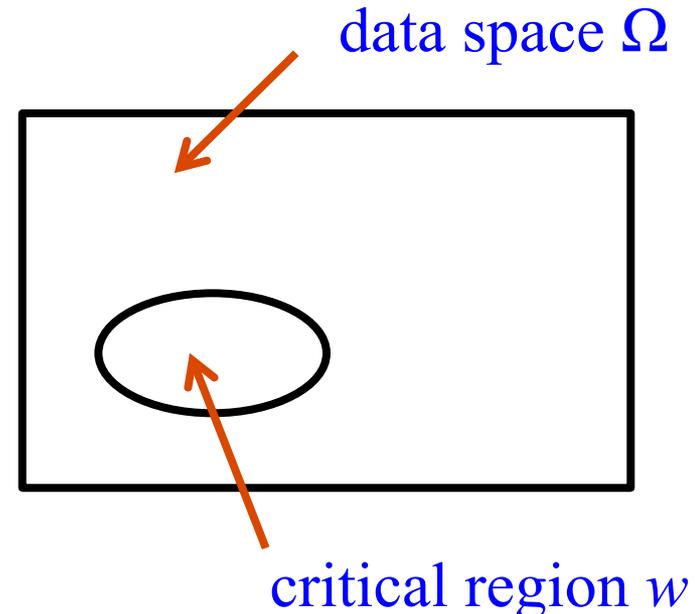
A **test** of H_0 is defined by specifying a **critical region** w of the data space such that there is no more than some (small) probability α , assuming H_0 is correct, to observe the data there, i.e.,

$$P(x \in w | H_0) \leq \alpha$$

Need inequality if data are discrete.

α is called the **size** or **significance level** of the test.

If x is observed in the critical region, reject H_0 .

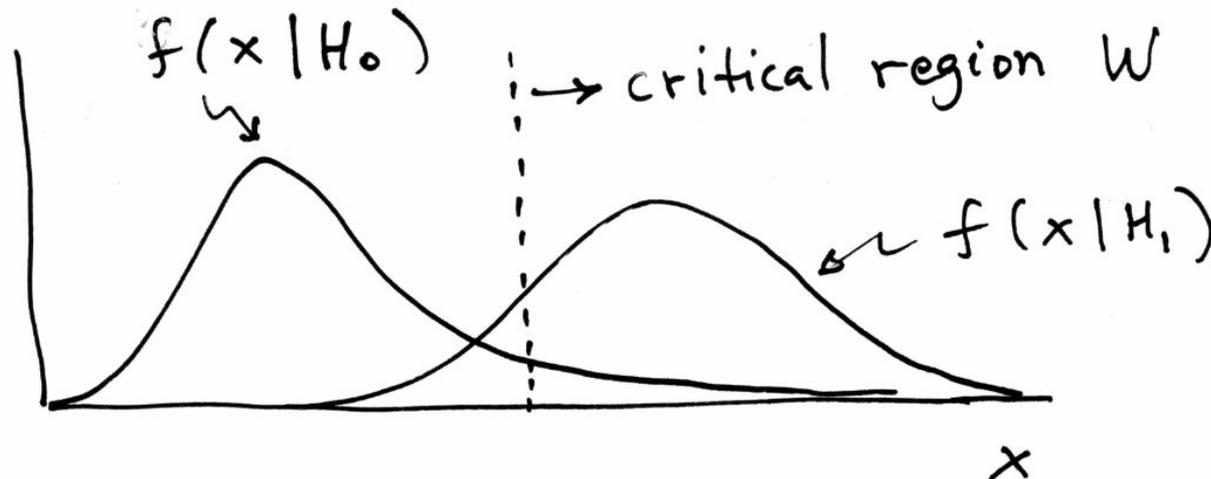


Role of the alternative hypothesis

But in general there are an infinite number of possible critical regions that give the same significance level α .

So the choice of the critical region for a test of H_0 needs to take into account the alternative hypothesis H_1 .

Roughly speaking, place the critical region where there is a low probability to be found if H_0 is true, but high if H_1 is true:



p-values

Suppose hypothesis H predicts pdf $f(\vec{x}|H)$ for a set of observations $\vec{x} = (x_1, \dots, x_n)$.

We observe a single point in this space: \vec{x}_{obs}

What can we say about the validity of H in light of the data?

Express level of compatibility by giving the *p*-value for H :

p = probability, under assumption of H , to observe data with equal or lesser compatibility with H relative to the data we got.



This is not the probability that H is true!

Requires one to say what part of data space constitutes lesser compatibility with H than the observed data (implicitly this means that region gives better agreement with some alternative).

Using a p -value to define test of H_0

One can show the distribution of the p -value of H , under assumption of H , is uniform in $[0,1]$.

So the probability to find the p -value of H_0 , p_0 , less than α is

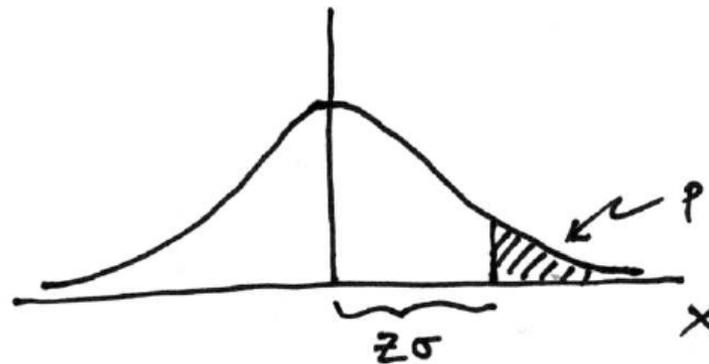
$$P(p_0 \leq \alpha | H_0) = \alpha$$

We can define the critical region of a test of H_0 with size α as the set of data space where $p_0 \leq \alpha$.

Formally the p -value relates only to H_0 , but the resulting test will have a given power with respect to a given alternative H_1 .

Significance from p -value

Often define significance Z as the number of standard deviations that a Gaussian variable would fluctuate in one direction to give the same p -value.



$$p = \int_Z^{\infty} \frac{1}{\sqrt{2\pi}} e^{-x^2/2} dx = 1 - \Phi(Z) \quad \mathbf{1 - TMath::Freq}$$

$$Z = \Phi^{-1}(1 - p) \quad \mathbf{TMath::NormQuantile}$$

E.g. $Z = 5$ (a “5 sigma effect”) corresponds to $p = 2.9 \times 10^{-7}$.

Two distinct event selection problems

In some cases, the event types in question are both known to exist.

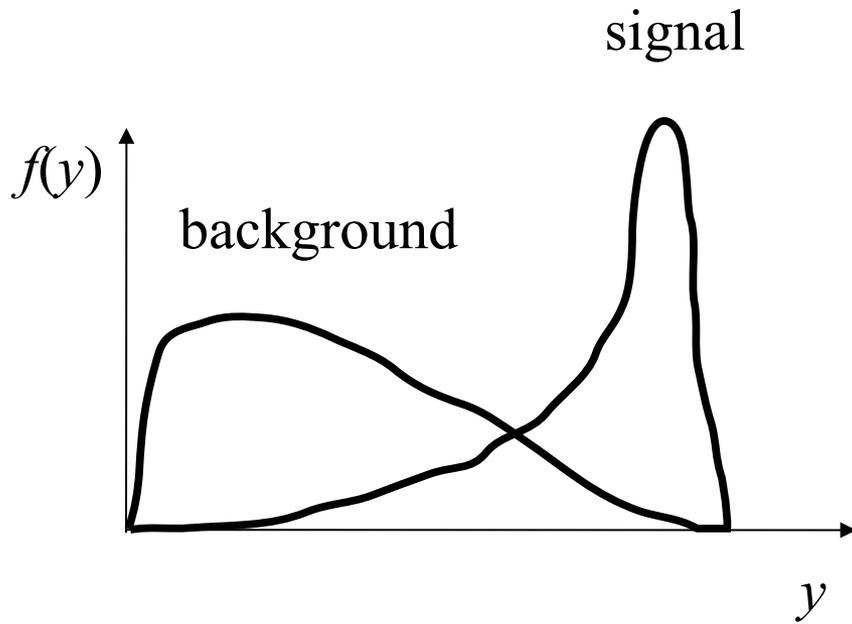
Example: separation of different particle types (electron vs muon)
Use the selected sample for further study.

In other cases, the null hypothesis H_0 means "Standard Model" events, and the alternative H_1 means "events of a type whose existence is not yet established" (to do so is the goal of the analysis).

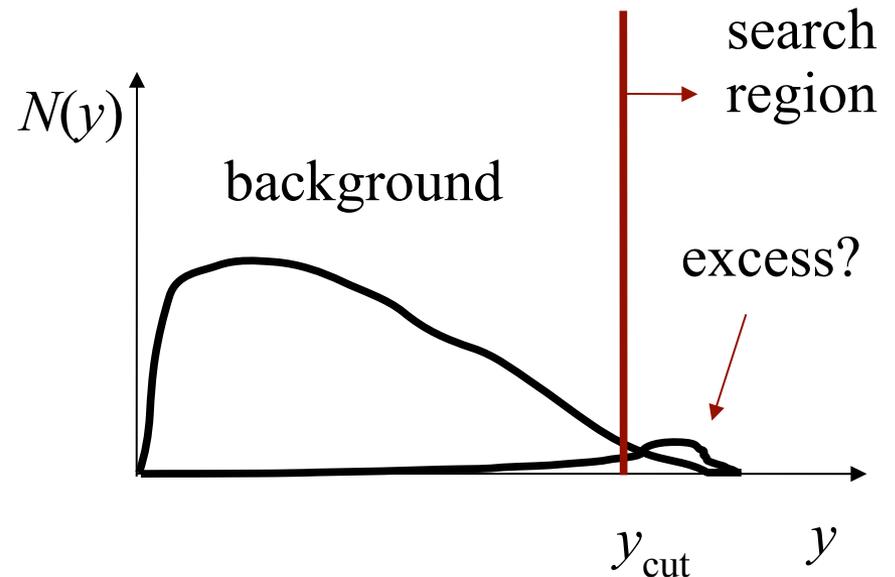
Many subtle issues here, mainly related to the heavy burden of proof required to establish presence of a new phenomenon.

Typically require p -value of background-only hypothesis below $\sim 10^{-7}$ (a 5 sigma effect) to claim discovery of "New Physics".

Using classifier output for discovery



Normalized to unity



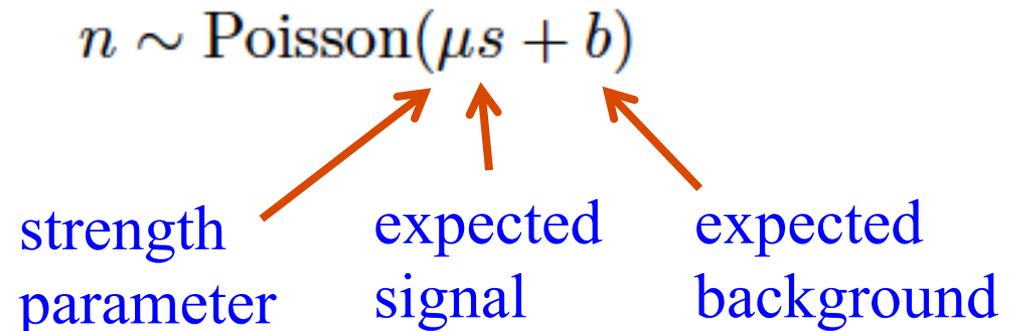
Normalized to expected number of events

Discovery = number of events found in search region incompatible with background-only hypothesis.

p -value of background-only hypothesis can depend crucially on distribution $f(y|b)$ in the "search region".

Prototype search

Suppose # of events
found in search region:



For each event measure x :
(classifier output or kin. var.)

$$x_1, \dots, x_n$$

where each x follows

$$f(x|\mu) = \frac{\mu s}{\mu s + b} f(x|s) + \frac{b}{\mu s + b} f(x|b)$$

Likelihood function is:

$$L(\mu) = \frac{(\mu s + b)^n}{n!} e^{-(\mu s + b)} \prod_{i=1}^n f(x_i|\mu)$$

Test of signal strength μ

Can test a value of μ using: $\lambda(\mu) = \frac{L(\mu)}{L(\hat{\mu})}$, $\hat{\mu} = \operatorname{argmax}_{\mu} L(\mu)$

E.g. to test
 $H_0 : \mu = 0$ versus
 $H_1 : \mu > 0$ use

$$q_0 = \begin{cases} -2 \ln \lambda(0) & \hat{\mu} \geq 0 \\ 0 & \hat{\mu} < 0 \end{cases}$$

I.e., large q_0 means increasing disagreement between data and hypothesis of $\mu = 0$.

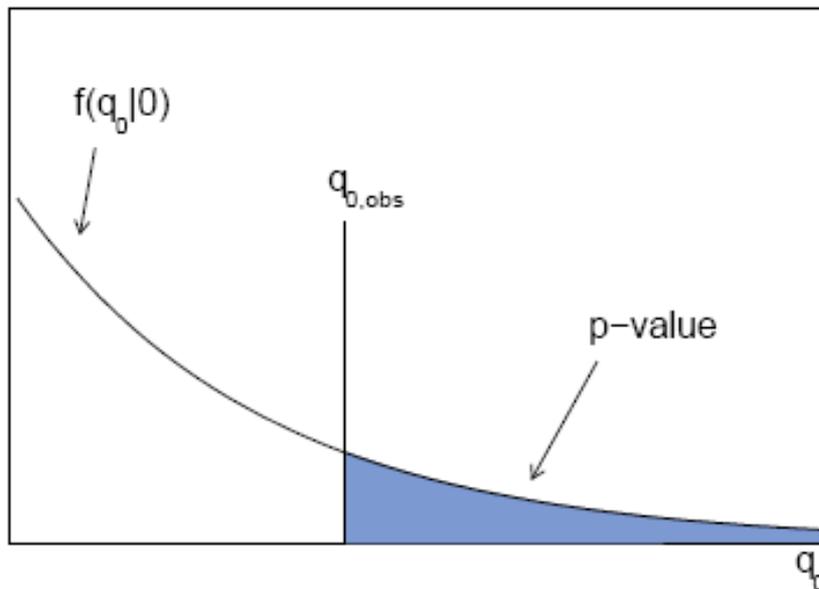
In this case, we are assuming alternative to background-only hypothesis is a positive signal strength.

p-value for discovery

Large q_0 means increasing incompatibility between the data and hypothesis, therefore *p*-value for an observed $q_{0,\text{obs}}$ is

$$p_0 = \int_{q_{0,\text{obs}}}^{\infty} f(q_0|0) dq_0$$

For large sample related to chi-square distribution.



From *p*-value get equivalent significance,

$$Z = \Phi^{-1}(1 - p)$$

Cumulative distribution of q_0 , significance

From the pdf, the cumulative distribution of q_0 is found to be

$$F(q_0|\mu') = \Phi\left(\sqrt{q_0} - \frac{\mu'}{\sigma}\right)$$

The special case $\mu' = 0$ is

$$F(q_0|0) = \Phi(\sqrt{q_0})$$

The p -value of the $\mu = 0$ hypothesis is

$$p_0 = 1 - F(q_0|0)$$

Therefore the discovery significance Z is simply

$$Z = \Phi^{-1}(1 - p_0) = \sqrt{q_0}$$

Alternative likelihood ratio for full experiment

We can define a test statistic Q monotonic in the likelihood ratio as

$$Q = -2 \ln \frac{L_{s+b}}{L_b} = -s + \sum_{i=1}^n \ln \left(1 + \frac{s}{b} \frac{f(\mathbf{x}_i|s)}{f(\mathbf{x}_i|b)} \right)$$

To compute p -values for the b and $s+b$ hypotheses given an observed value of Q we need the distributions $f(Q|b)$ and $f(Q|s+b)$.

Note that the term $-s$ in front is a constant and can be dropped.

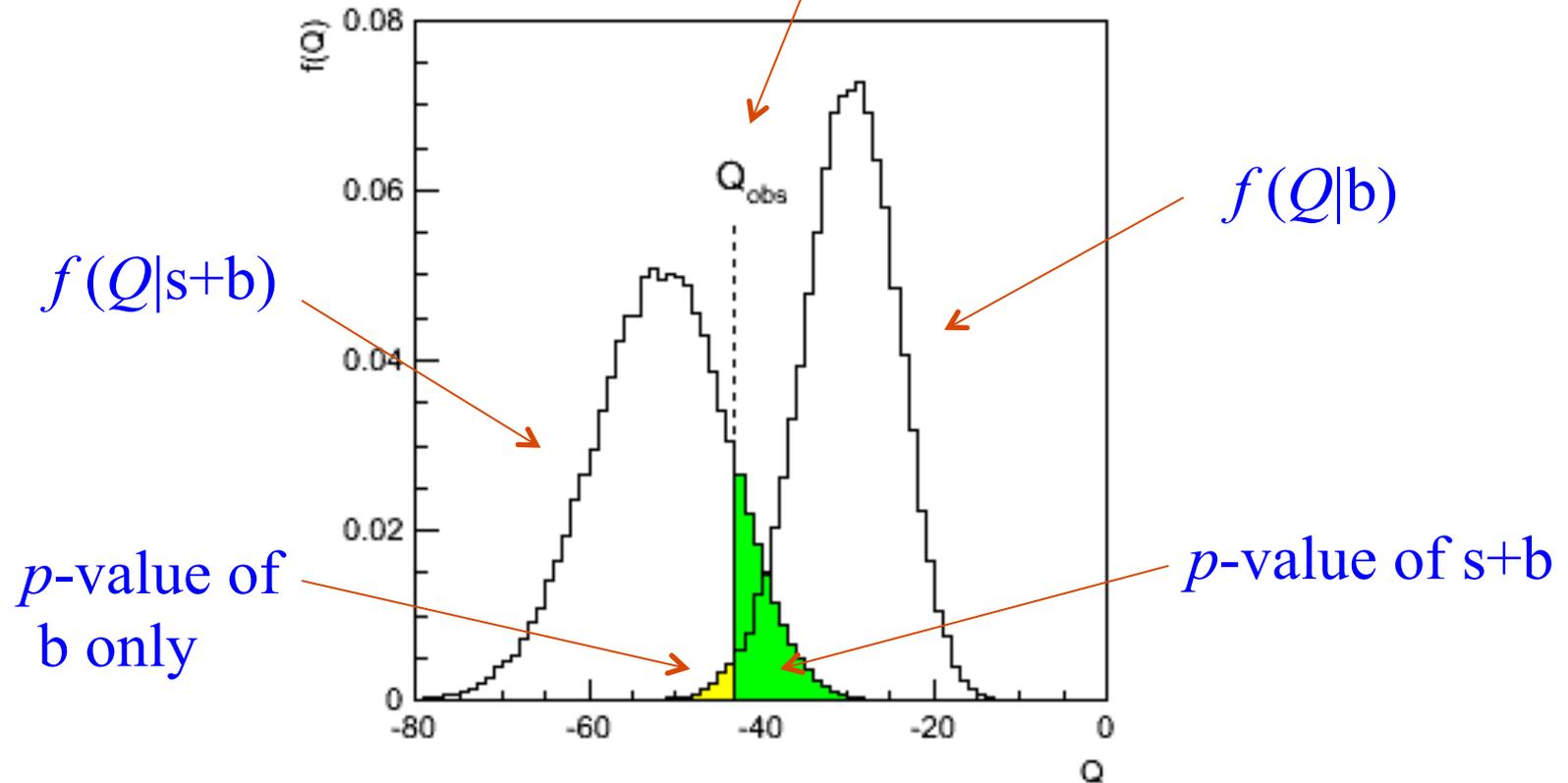
The rest is a sum of contributions for each event, and each term in the sum has the same distribution.

Can exploit this to relate distribution of Q to that of single event terms using (Fast) Fourier Transforms (Hu and Nielsen, physics/9906010).

Distribution of Q

Take e.g. $b = 100$, $s = 20$.

Suppose in real experiment Q is observed here.



If $p_{s+b} < \alpha$, reject signal model s at confidence level $1 - \alpha$.

If $p_b < 2.9 \times 10^{-7}$, reject background-only model (signif. $Z = 5$).