Statistical Methods for Particle Physics
Lecture 2: statistical tests, multivariate methods

www.pp.rhul.ac.uk/~cowan/stat_aachen.html

Graduierten-Kolleg
RWTH Aachen
10-14 February 2014

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

Outline

Hypotheses

A hypothesis H specifies the probability for the data, i.e., the outcome of the observation, here symbolically: x.

x could be uni-/multivariate, continuous or discrete.

E.g. write $x \sim f(x|H)$.

x could represent e.g. observation of a single particle, a single event, or an entire "experiment".

Possible values of x form the sample space S (or "data space").

Simple (or "point") hypothesis: $f(x|H)$ completely specified.

Composite hypothesis: H contains unspecified parameter(s).

The probability for x given H is also called the likelihood of the hypothesis, written $L(x|H)$.

Frequentist statistical tests

Consider a hypothesis H_0 and alternative H_1.

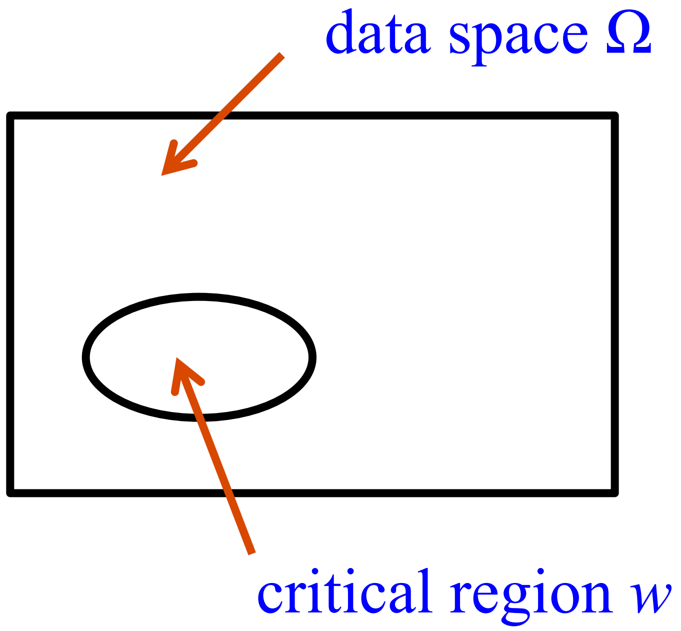
A test of H_0 is defined by specifying a critical region w of the data space such that there is no more than some (small) probability α, assuming H_0 is correct, to observe the data there, i.e.,

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

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.

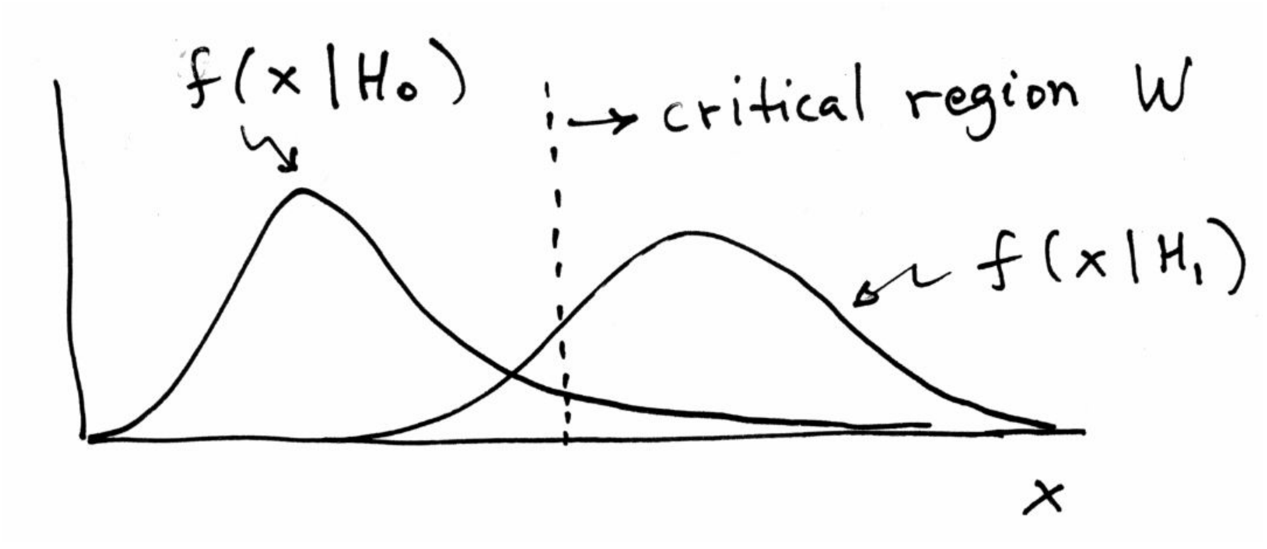


Definition of a test (2)

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:



Type-I, Type-II errors

Rejecting the hypothesis H_0 when it is true is a Type-I error.

The maximum probability for this is the size of the test:

$$P(x \in W \mid H_0) \leq \alpha$$

But we might also accept H_0 when it is false, and an alternative H_1 is true.

This is called a Type-II error, and occurs with probability

$$P(x \in S - W \mid H_1) = \beta$$

One minus this is called the power of the test with respect to the alternative H_1:

$$\text{Power} = 1 - \beta$$

Rejecting a hypothesis

Note that rejecting H_0 is not necessarily equivalent to the statement that we believe it is false and H_1 true. In frequentist statistics only associate probability with outcomes of repeatable observations (the data).

In Bayesian statistics, probability of the hypothesis (degree of belief) would be found using Bayes' theorem:

$$P(H|x) = \frac{P(x|H)\pi(H)}{\int P(x|H)\pi(H)\,dH}$$

which depends on the prior probability $\pi(H)$.

What makes a frequentist test useful is that we can compute the probability to accept/reject a hypothesis assuming that it is true, or assuming some alternative is true.

Physics context of a statistical test

Event Selection: the event types in question are both known to exist.

Example: separation of different particle types (electron vs muon) or known event types (ttbar vs QCD multijet).
Use the selected sample for further study.

Search for New Physics: the null hypothesis is

H_0 : all events correspond to Standard Model (background only),

and the alternative is

H_1 : events include a type whose existence is not yet established (signal plus background)

Many subtle issues here, mainly related to the high standard of proof required to establish presence of a new phenomenon. The optimal statistical test for a search is closely related to that used for event selection.

Example of a multivariate statistical test

Suppose the result of a measurement for an individual event is a collection of numbers $\vec{x} = (x_1, \dots, x_n)$

x_1 = number of muons,

x_2 = mean p_T of jets,

x_3 = missing energy, ...

$\vec{x}$ follows some n-dimensional joint pdf, which depends on the type of event produced, i.e., was it

$$\mathrm{pp} \to t\bar{t}\ , \qquad \mathrm{pp} \to \tilde{g}\tilde{g}\ , \dots$$

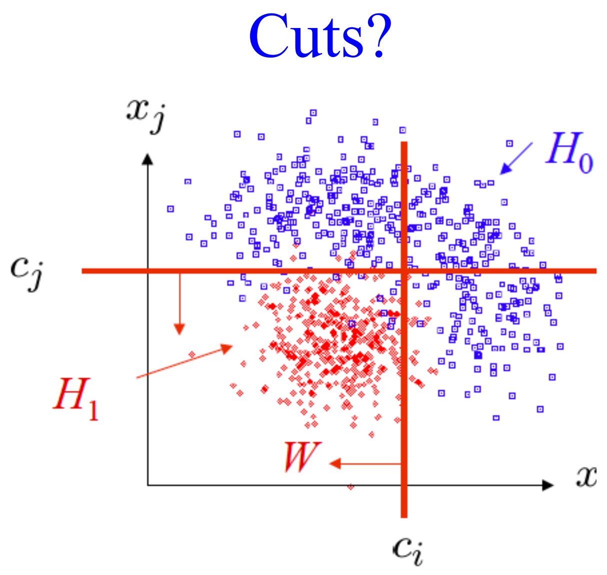
For each reaction we consider we will have a hypothesis for the pdf of $\vec{x}$, e.g., $f(\vec{x}|H_0)$, $f(\vec{x}|H_1)$, etc.

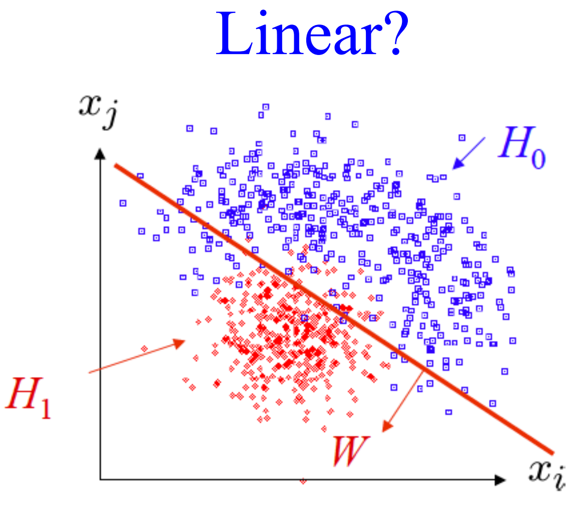
E.g. call H_0 the background hypothesis (the event type we want to reject); H_1 is signal hypothesis (the type we want).

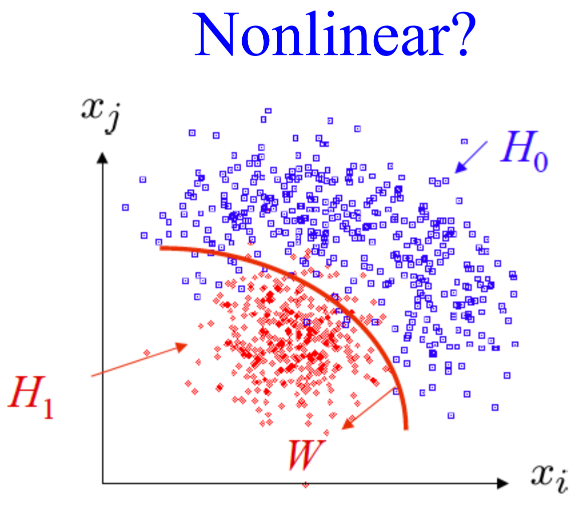
Defining a multivariate critical region

Each event is a point in ***x***-space; critical region is now defined by a 'decision boundary' in this space.

What kind of boundary is best?







Test statistics

The critical region's boundary can be defined by

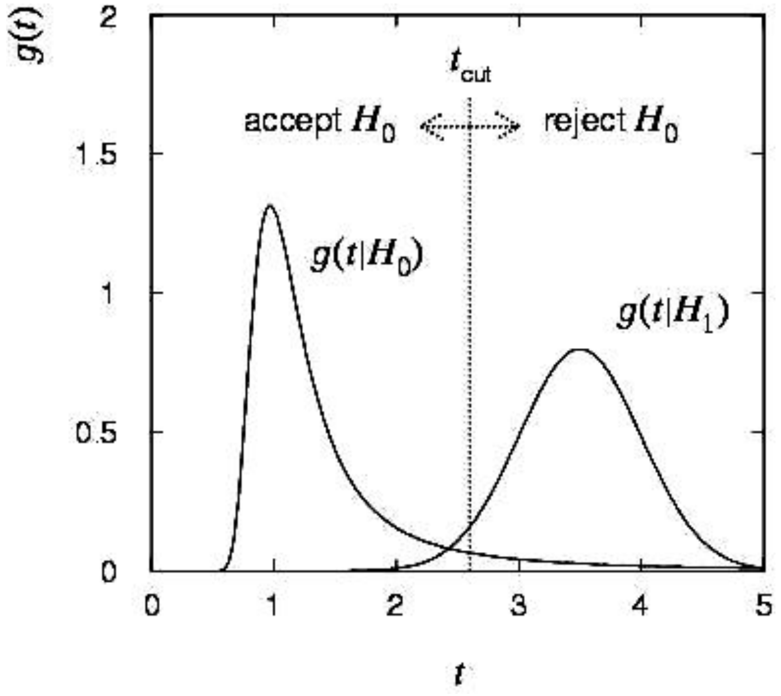
$$t(x_1, \ldots, x_n) = t_{\text{cut}}$$

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

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

Critical region's boundary is now a single 'cut' on t.

So the original n-dimensional problem becomes in effect 1-dimensional.



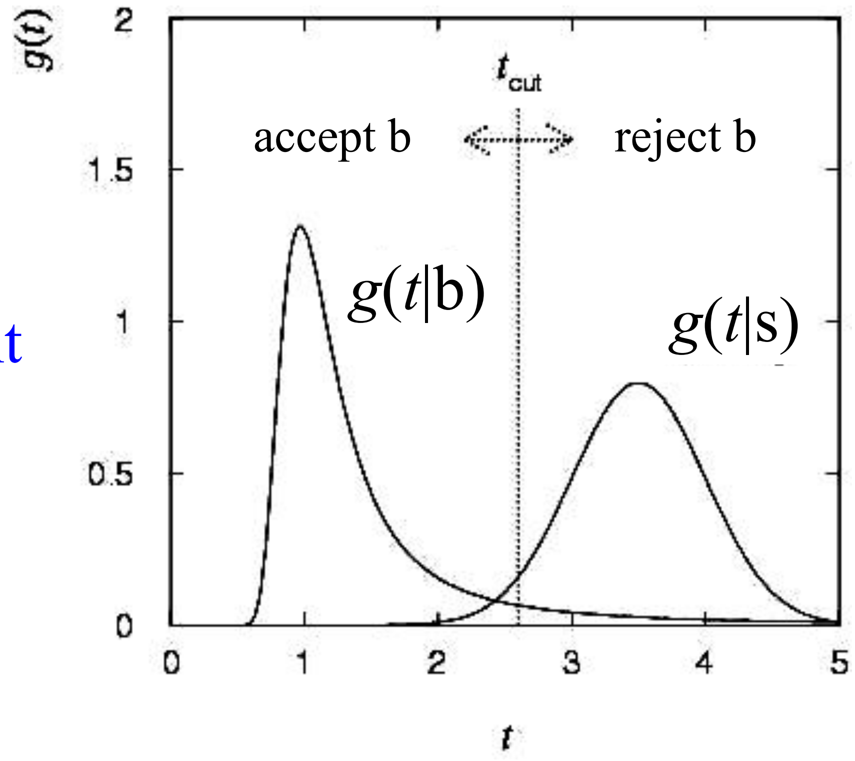
Signal/background efficiency

Probability to reject background hypothesis for background event (background efficiency):

$$\varepsilon_{\rm b} = \int_{t_{\rm cut}}^{\infty} g(t|{\rm b})\, dt = \alpha$$

Probability to accept a signal event as signal (signal efficiency):

$$\varepsilon_{\rm s} = \int_{t_{\rm cut}}^{\infty} g(t|{\rm s})\, dt = 1 - \beta$$



Purity of event selection

Suppose only one background type b; overall fractions of signal and background events are π_s and π_b (prior probabilities).

Suppose we select signal events with $t > t_{cut}$. What is the 'purity' of our selected sample?

Here purity means the probability to be signal given that the event was accepted. Using Bayes' theorem we find:

$$
\begin{aligned}
P(\mathrm{s}|t > t_{\mathrm{cut}}) &= \frac{P(t > t_{\mathrm{cut}}|\mathrm{s})\pi_{\mathrm{s}}}{P(t > t_{\mathrm{cut}}|\mathrm{s})\pi_{\mathrm{s}} + P(t > t_{\mathrm{cut}}|\mathrm{b})\pi_{\mathrm{b}}} \\
&= \frac{\varepsilon_{\mathrm{s}}\pi_{\mathrm{s}}}{\varepsilon_{\mathrm{s}}\pi_{\mathrm{s}} + \varepsilon_{\mathrm{b}}\pi_{\mathrm{b}}}
\end{aligned}
$$

So the purity depends on the prior probabilities as well as on the signal and background efficiencies.

Constructing a test statistic

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

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

inside the region, and $\leq c$ outside, where c is a constant which determines the power.

Equivalently, optimal scalar test statistic is
$$\boxed{t(\mathbf{x}) = \frac{P(\mathbf{x}|H_1)}{P(\mathbf{x}|H_0)}}$$

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

Why Neyman-Pearson doesn't always help

The problem is that we usually don't have explicit formulae for the pdfs $P(\boldsymbol{x}|H_0)$, $P(\boldsymbol{x}|H_1)$.

Instead we may have Monte Carlo models for signal and background processes, so we can produce simulated data, and enter each event into an n-dimensional histogram.

Use e.g. M bins for each of the n dimensions, total of M^n cells.

But n is potentially large, $\rightarrow$ prohibitively large number of cells to populate with Monte Carlo data.

Compromise: make Ansatz for form of test statistic $t(\vec{x})$ with fewer parameters; determine them (e.g. using MC) to give best discrimination between signal and background.

Multivariate methods

Many new (and some old) methods:

- Fisher discriminant
- Neural networks
- Kernel density methods
- Support Vector Machines
- Decision trees
  - Boosting
  - Bagging

New software for HEP, e.g.,

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

`StatPatternRecognition`, I. Narsky, physics/0507143

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, 2001

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`

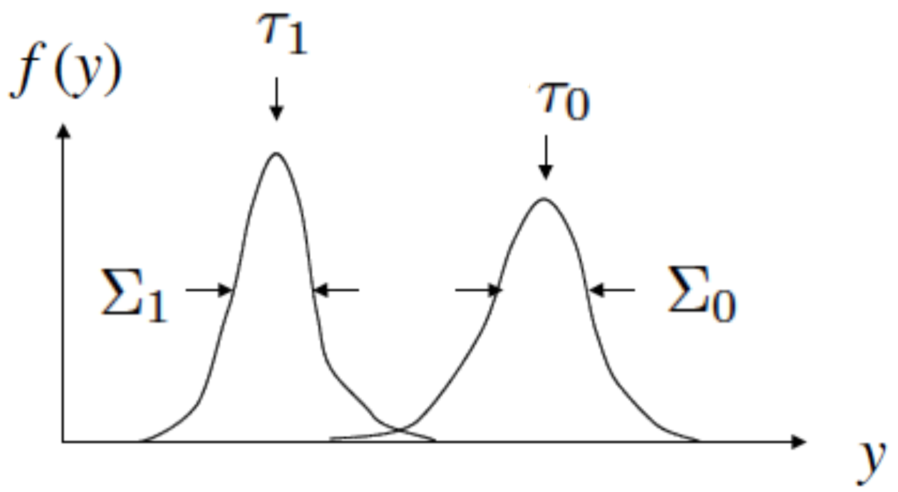
SLAC Lectures on Machine Learning by Ilya Narsky (2006)
`www-group.slac.stanford.edu/sluo/Lectures/Stat2006_Lectures.html`

Linear test statistic

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

Choose the parameters $w_1, \ldots, w_n$ so that the pdfs $f(y|H_0),\ f(y|H_1)$ have maximum 'separation'. We want:

large distance between mean values, small widths



→ Fisher: maximize $J(\mathbf{w}) = \dfrac{(\tau_1 - \tau_0)^2}{\Sigma_0^2 + \Sigma_1^2}$

Coefficients for maximum separation

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

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

where $k = 0, 1$ (hypothesis)

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

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 $J(\boldsymbol{w})$ is

$$(\tau_1 - \tau_0)^2 = \sum_{i,j=1}^{n} w_i w_j (\boldsymbol{\mu}_0 - \boldsymbol{\mu}_1)_i (\boldsymbol{\mu}_0 - \boldsymbol{\mu}_1)_j = \sum_{i,j=1}^{n} w_i w_j B_{ij}$$

where $B_{ij} = (\boldsymbol{\mu}_0 - \boldsymbol{\mu}_1)_i (\boldsymbol{\mu}_0 - \boldsymbol{\mu}_1)_j$

The denominator is $\Sigma_0^2 + \Sigma_1^2 = \sum_{i,j=1}^{n} (V_0 + V_1)_{ij} = \sum_{i,j=1}^{n} w_i w_j W_{ij}$

where $W_{ij} = (V_0 + V_1)_{ij}$. So the quantity to maximize is:

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

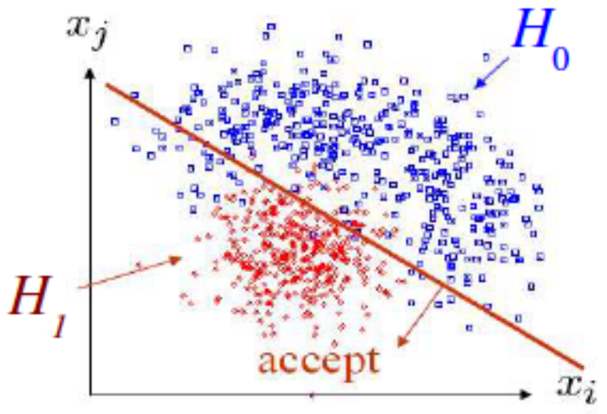
Fisher discriminant function

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

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

Gives linear decision boundary.

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



Fisher discriminant for Gaussian data

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

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

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

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

The likelihood ratio is thus

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

$$=e^y$$

Fisher for Gaussian data (2)

That is, $y(\boldsymbol{x})$ 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 discrimimant.

Fisher and Gaussian data (3)

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

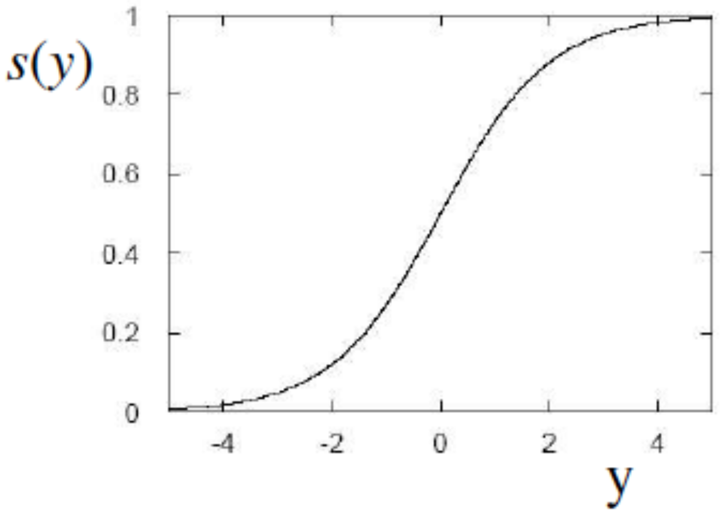
$$P(H_0|\vec{x}) = \frac{p(\vec{x}|H_0)\,P(H_0)}{p(\vec{x}|H_0)\,P(H_0) + p(\vec{x}|H_1)\,P(H_1)}$$

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

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

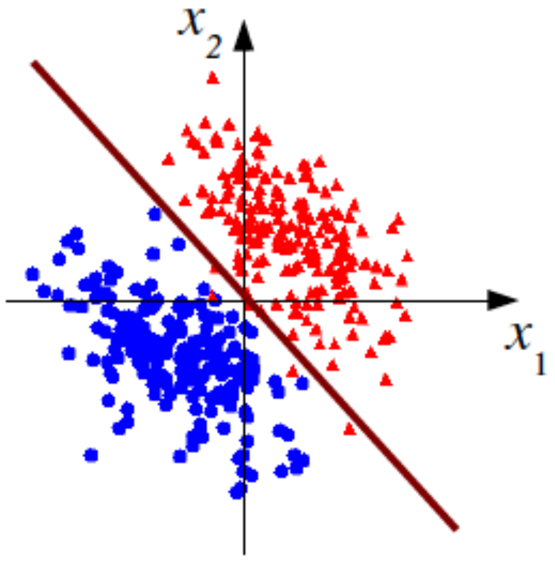
which is the logistic sigmoid function:

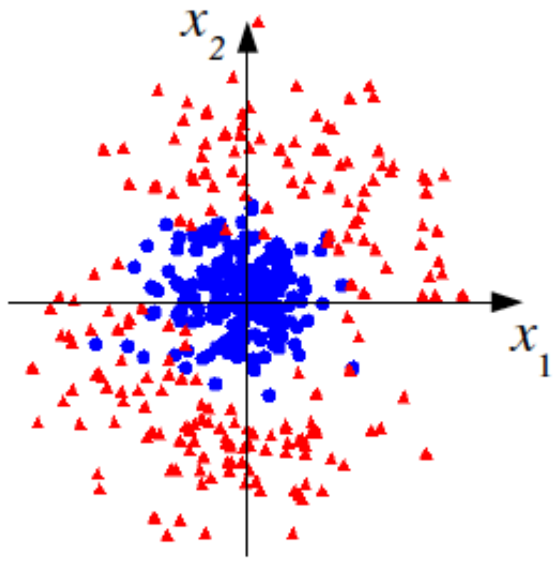
(We will use this later in connection with Neural Networks.)



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.

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" $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$

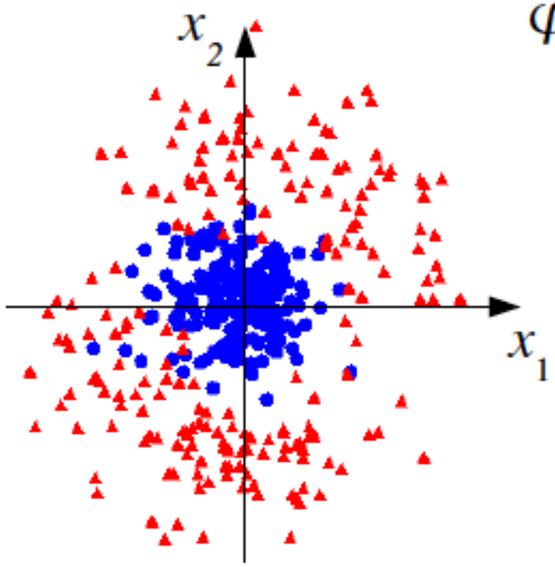
Nonlinear transformation of inputs

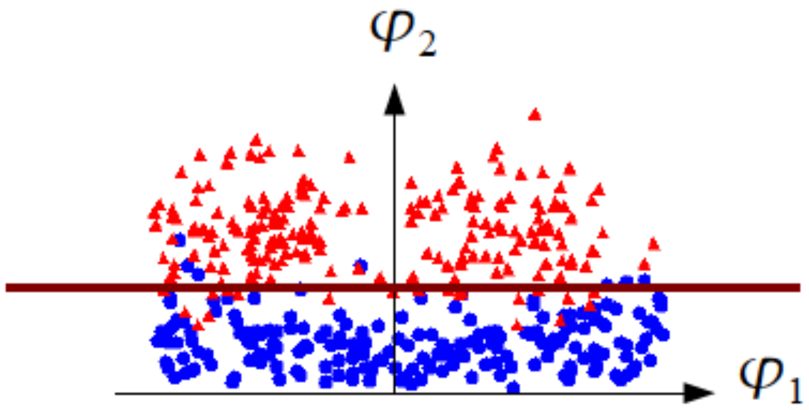
We can try to find a transformation, $x_1, \ldots, x_n \rightarrow \varphi_1(\vec{x}), \ldots, \varphi_m(\vec{x})$ so that the transformed "feature space" variables can be separated better by a linear boundary:

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

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

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.

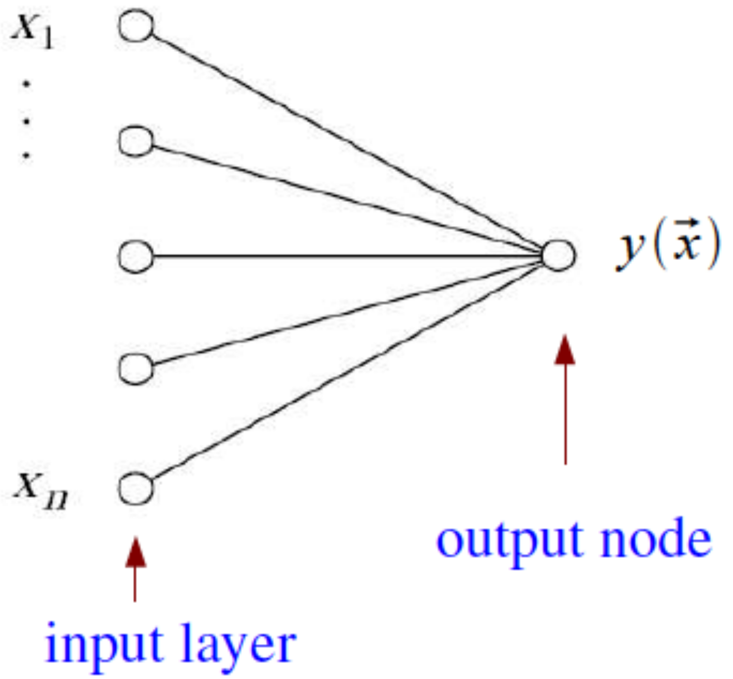
The training data are then used to adjust the parameters so that the resulting discriminant function has the best performance.

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.



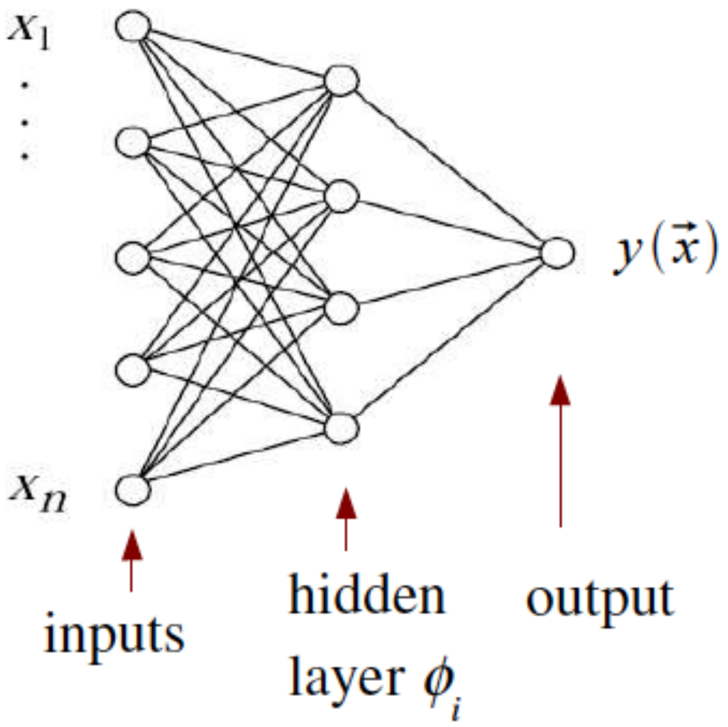
The multilayer perceptron

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

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

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

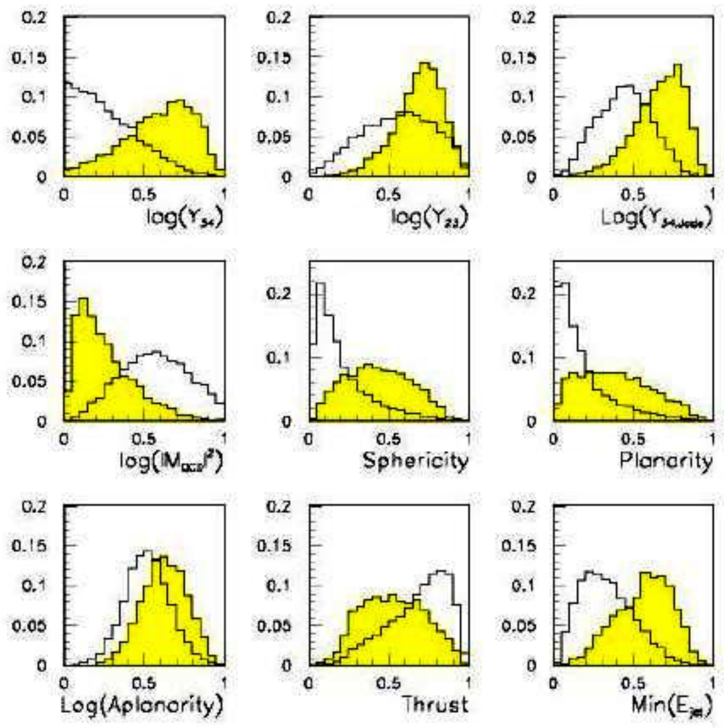


This is the multilayer perceptron, our basic neural network model; straightforward to generalize to multiple hidden layers.

Neural network example from LEP II

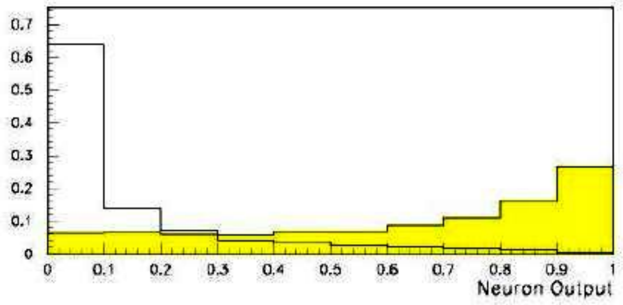
Signal: $e^+e^- \to W^+W^-$ (often 4 well separated hadron jets)

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



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

Neural network output:



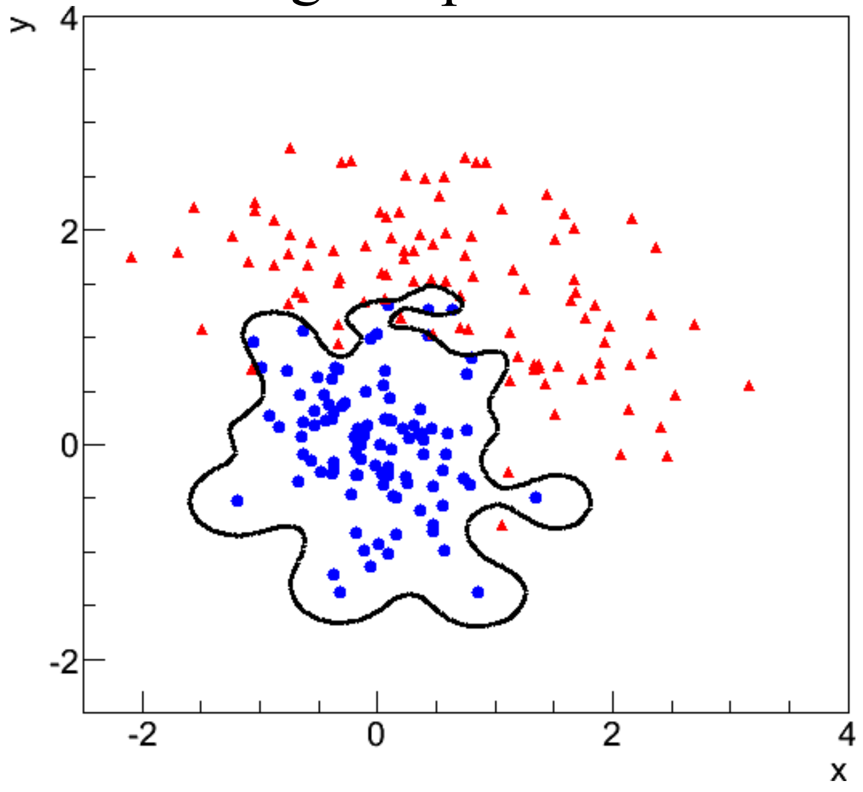
(Garrido, Juste and Martinez, ALEPH 96-144)

Overtraining

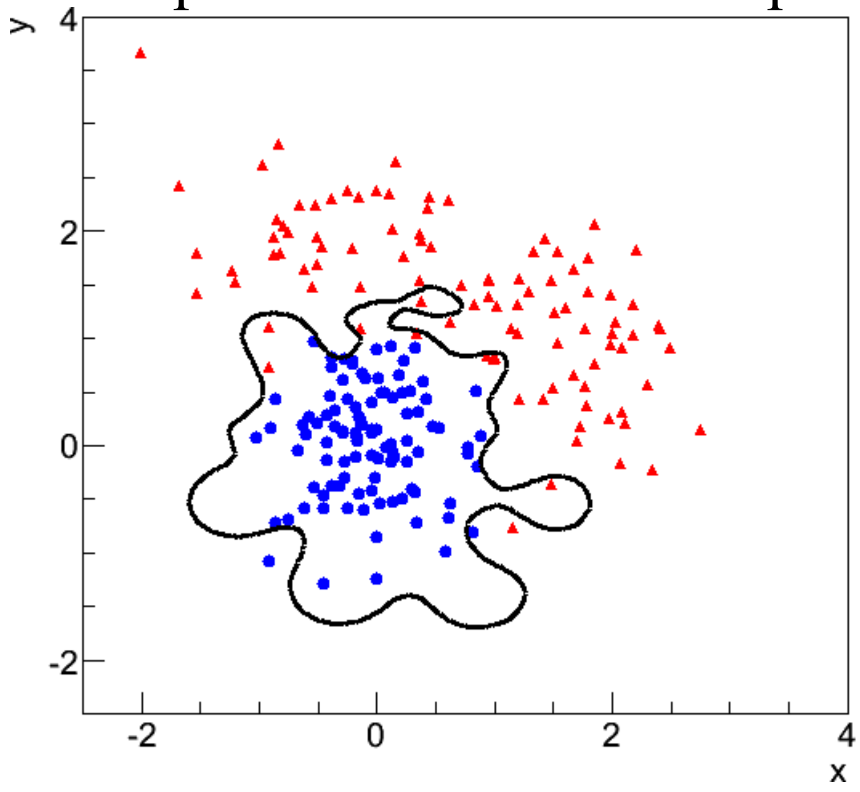
If decision boundary is too flexible it will conform too closely to the training points $\rightarrow$ overtraining.

Monitor by applying classifier to independent validation sample.

training sample

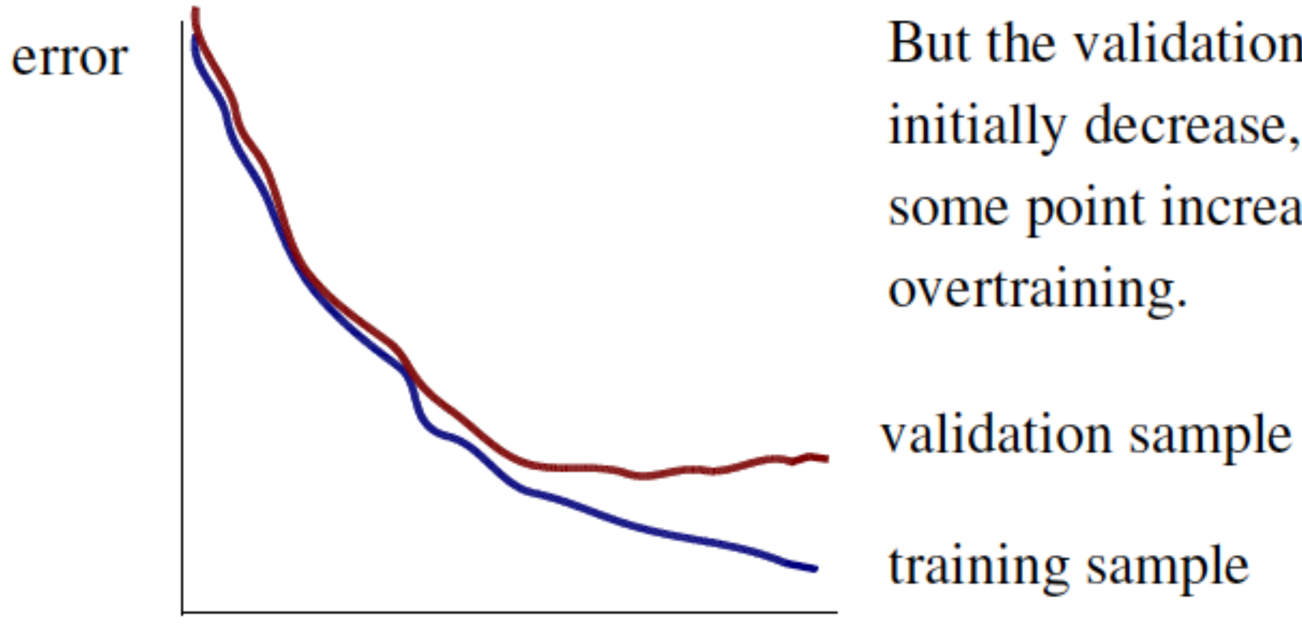


independent validation sample



Monitoring overtraining

Monitor misclassification (error) rate versus "flexibility" of the boundary. For training sample this will continue to decrease.



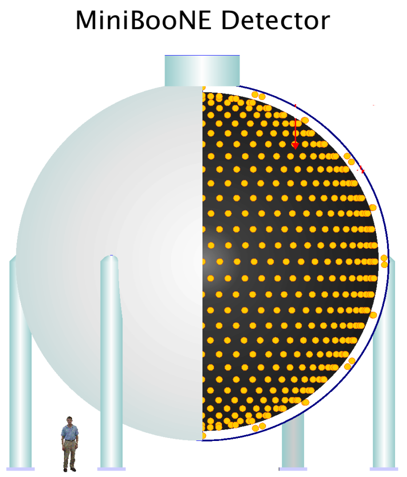
But the validation sample it may initially decrease, and then at some point increase, indicating overtraining.

# of adjustable parameters in classifier ~ "flexibility" of boundary

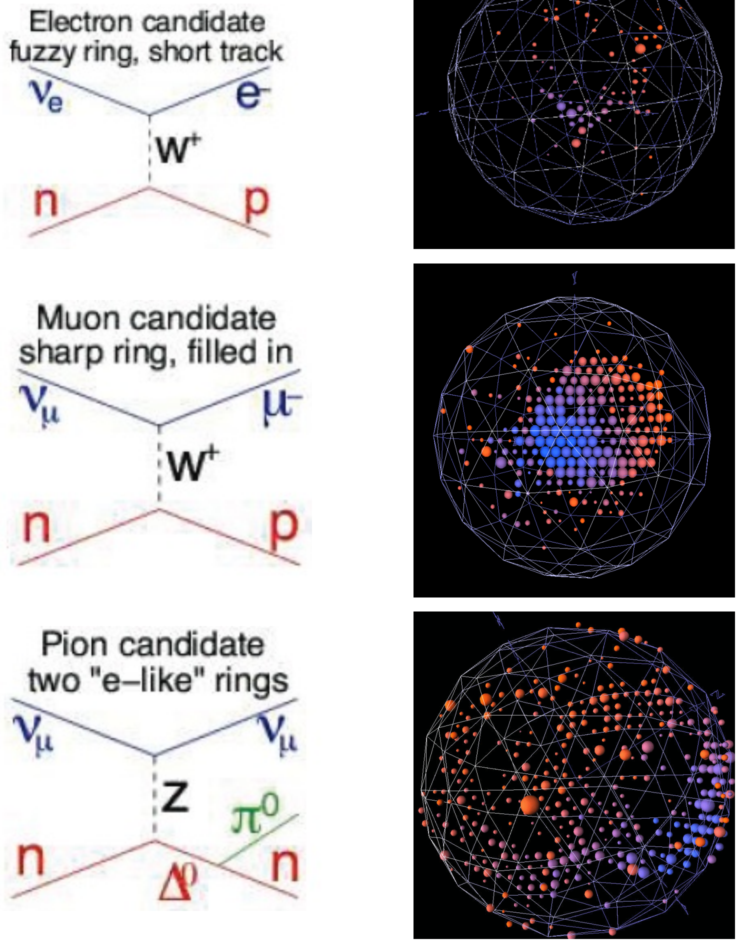
Choose classifier that minimizes error function for validation sample.

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:

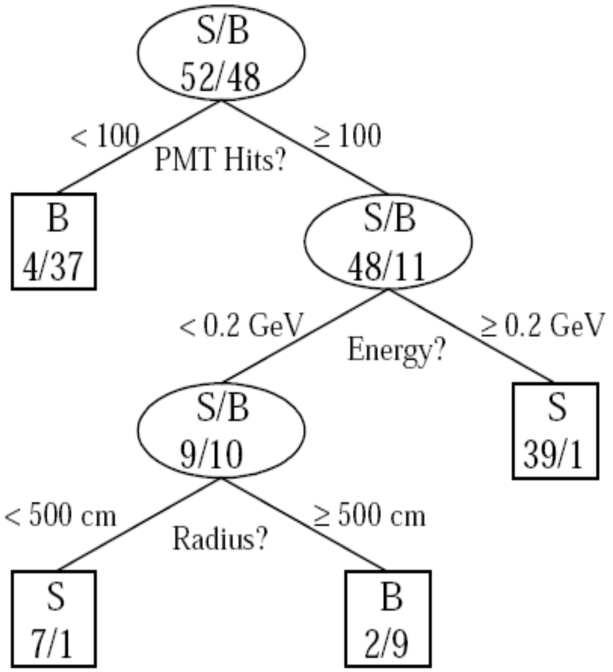
$$P = \frac{\sum_{\text{signal}} w_i}{\sum_{\text{signal}} w_i + \sum_{\text{background}} w_i}$$

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

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

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 $f_k(x)$ are characterized by a given 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 *k*th classifier based on its error rate,

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

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

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.

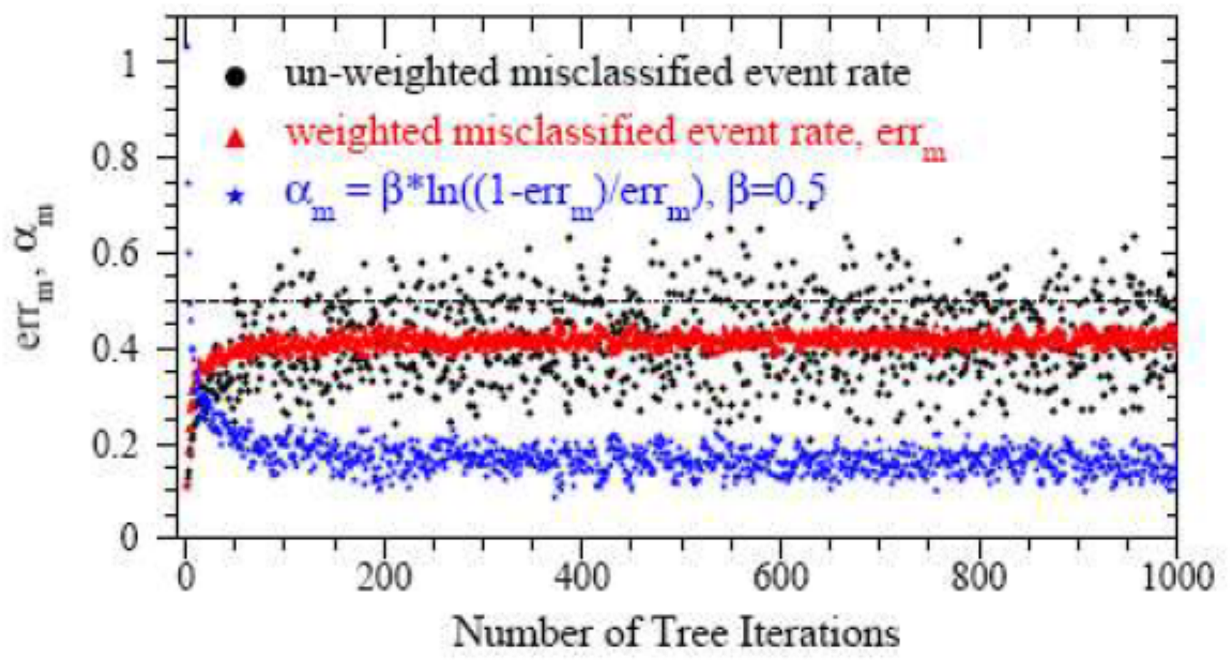
So the procedure is to monitor the error rate of the combined classifier at each step with a validation sample and to stop before it starts to rise.

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

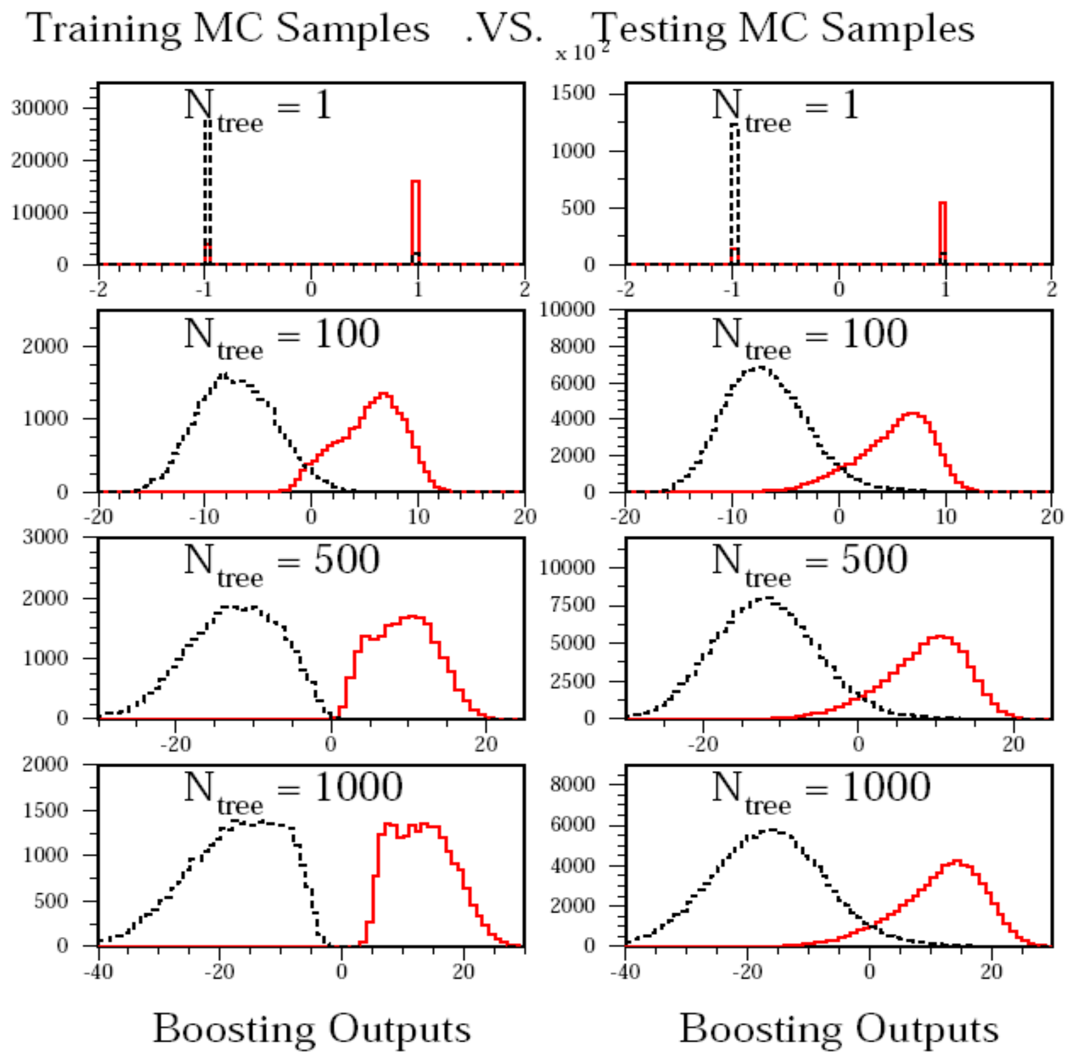


B. Roe et al., NIM 543 (2005) 577

Monitoring overtraining

From MiniBooNE example:

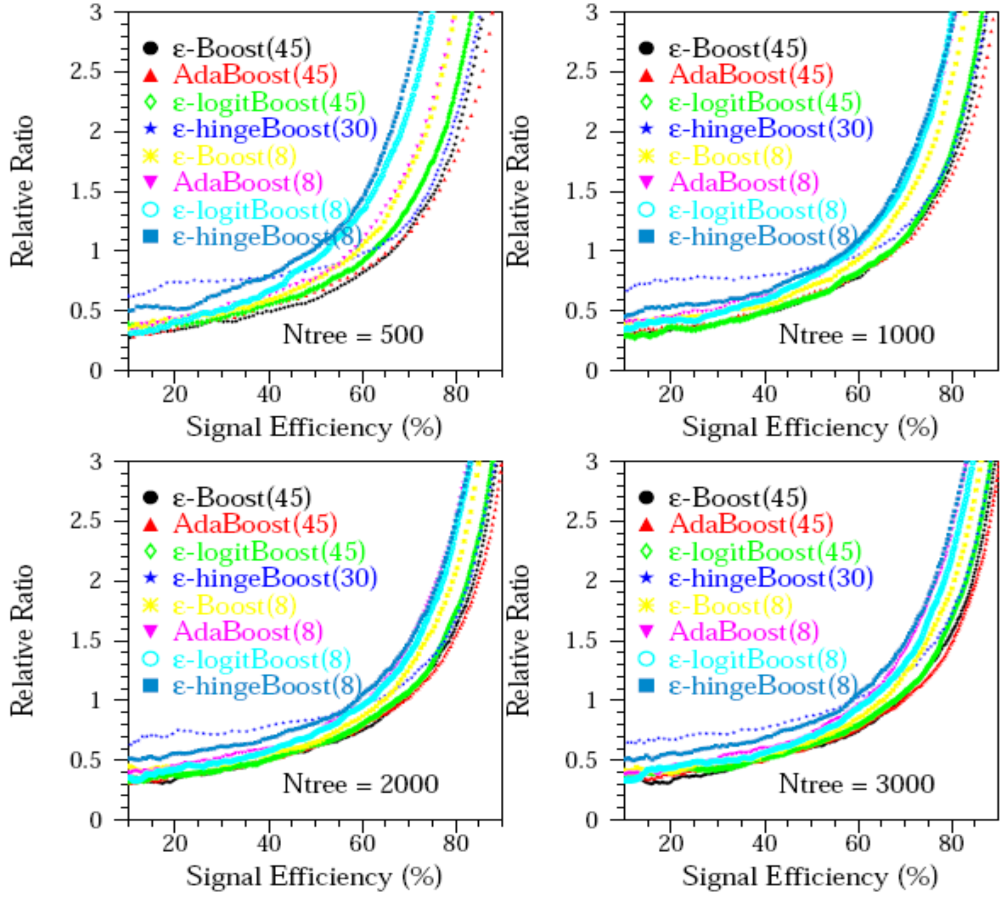
Performance stable after a few hundred trees.



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

Comparison of boosting algorithms

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



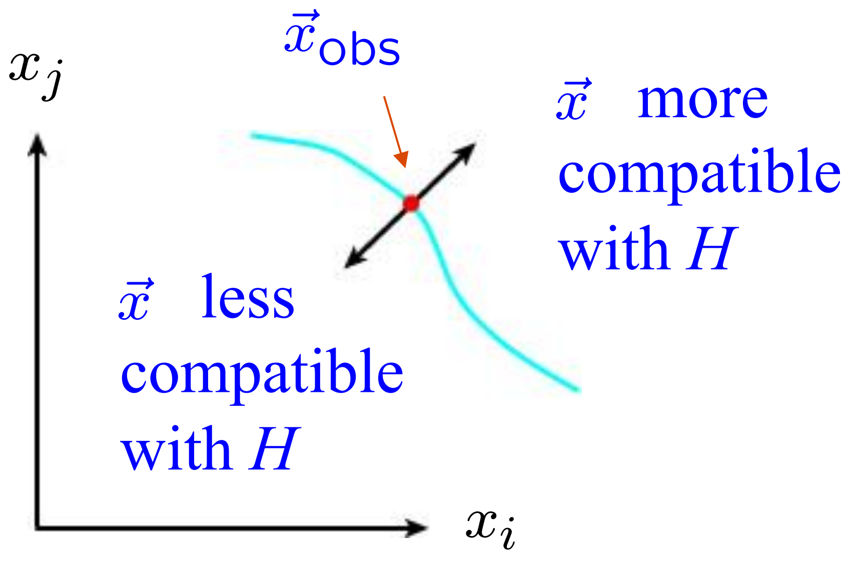
Testing significance / goodness-of-fit

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

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

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

Decide what part of the data space represents less compatibility with *H* than does the point $\vec{x}_{\mathrm{obs}}$.
(Not unique!)



x_j
$\vec{x}_{\mathrm{obs}}$
$\vec{x}$ more compatible with H
$\vec{x}$ less compatible with H
x_i

p-values

Express ‘goodness-of-fit’ 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!

In frequentist statistics we don’t talk about *P*(*H*) (unless *H* represents a repeatable observation). In Bayesian statistics we do; use Bayes’ theorem to obtain

$$P(H|\vec{x}) = \frac{P(\vec{x}|H)\pi(H)}{\int P(\vec{x}|H)\pi(H)\,dH}$$

where $\pi(H)$ is the prior probability for *H*.

For now stick with the frequentist approach;
result is *p*-value, regrettably easy to misinterpret as *P*(*H*).

p-value example: testing whether a coin is 'fair'

Probability to observe *n* heads in *N* coin tosses is binomial:

$$P(n; p, N) = \frac{N!}{n!(N-n)!} p^n (1-p)^{N-n}$$

Hypothesis *H*: the coin is fair ($p = 0.5$).

Suppose we toss the coin $N = 20$ times and get $n = 17$ heads.

Region of data space with equal or lesser compatibility with *H* relative to $n = 17$ is: $n = 17, 18, 19, 20, 0, 1, 2, 3$. Adding up the probabilities for these values gives:

$$P(n = 0, 1, 2, 3, 17, 18, 19, \text{ or } 20) = 0.0026 \, .$$

i.e. $p = 0.0026$ is the probability of obtaining such a bizarre result (or more so) 'by chance', under the assumption of *H*.

Distribution of the *p*-value

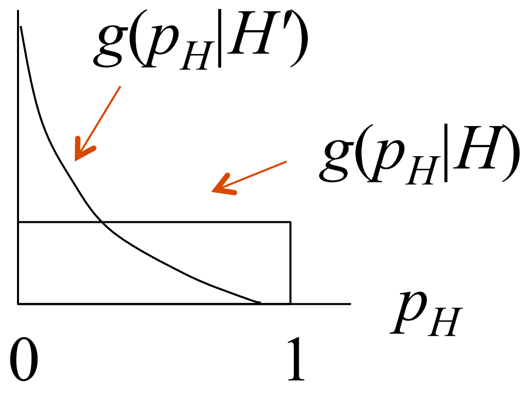
The *p*-value is a function of the data, and is thus itself a random variable with a given distribution. Suppose the *p*-value of *H* is found from a test statistic $t(\boldsymbol{x})$ as

$$p_H = \int_t^{\infty} f(t'|H)dt'$$

The pdf of p_H under assumption of *H* is

$$g(p_H|H) = \frac{f(t|H)}{|\partial p_H/\partial t|} = \frac{f(t|H)}{f(t|H)} = 1 \quad (0 \le p_H \le 1)$$

In general for continuous data, under assumption of *H*, $p_H \sim$ Uniform[0,1] and is concentrated toward zero for Some (broad) class of alternatives.



Using a p-value to define test of H_0

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 started by defining critical region in the original data space ($\boldsymbol{x}$), then reformulated this in terms of a scalar test statistic $t(\boldsymbol{x})$.

We can take this one step further and 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.

The significance of an observed signal

Suppose we observe n events; these can consist of:

n_b events from known processes (background)
n_s events from a new process (signal)

If n_s, n_b are Poisson r.v.s with means s, b, then $n = n_s + n_b$ is also Poisson, mean = $s + b$:

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

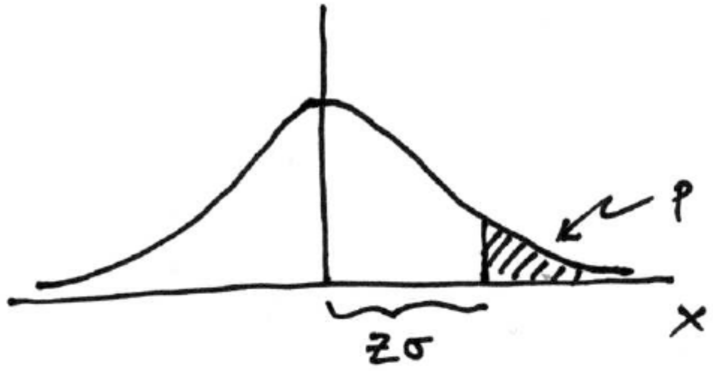
Suppose $b = 0.5$, and we observe $n_{obs} = 5$. Should we claim evidence for a new discovery?

Give p-value for hypothesis $s = 0$:

$$\begin{aligned} p\text{-value} &= P(n \geq 5; b = 0.5, s = 0) \\ &= 1.7 \times 10^{-4} \neq P(s = 0)! \end{aligned}$$

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.



$$p = \int_Z^\infty \frac{1}{\sqrt{2\pi}} e^{-x^2/2}\,dx = 1 - \Phi(Z)$$

```
1 - TMath::Freq
```

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

```
TMath::NormQuantile
```

Pearson's χ^2 statistic

Test statistic for comparing observed data $\vec{n} = (n_1, \ldots, n_N)$ (n_i independent) to predicted mean values $\vec{\nu} = (\nu_1, \ldots, \nu_N)$:

$$\chi^2 = \sum_{i=1}^{N} \frac{(n_i - \nu_i)^2}{\sigma_i^2} \,, \quad \text{where } \sigma_i^2 = V[n_i] \,.$$

(Pearson's χ^2 statistic)

χ^2 = sum of squares of the deviations of the *i*th measurement from the *i*th prediction, using σ_i as the 'yardstick' for the comparison.

For $n_i \sim$ Poisson(ν_i) we have $V[n_i] = \nu_i$, so this becomes

$$\chi^2 = \sum_{i=1}^{N} \frac{(n_i - \nu_i)^2}{\nu_i} \,.$$

Pearson's χ^2 test

If n_i are Gaussian with mean ν_i and std. dev. σ_i, i.e., $n_i \sim N(\nu_i\ ,\ \sigma_i^2)$, then Pearson's χ^2 will follow the χ^2 pdf (here for $\chi^2 = z$):

$$f_{\chi^2}(z; N) = \frac{1}{2^{N/2}\Gamma(N/2)} z^{N/2-1} e^{-z/2}$$

If the n_i are Poisson with $\nu_i >> 1$ (in practice OK for $\nu_i >$ half dozen) then the Poisson dist. becomes Gaussian and therefore Pearson's χ^2 statistic here as well follows the χ^2 pdf.

The χ^2 value obtained from the data then gives the p-value:

$$p = \int_{\chi^2}^{\infty} f_{\chi^2}(z; N)\, dz\ .$$

The 'χ^2 per degree of freedom'

Recall that for the chi-square pdf for N degrees of freedom,

$$E[z] = N \,, \qquad V[z] = 2N \,.$$

This makes sense: if the hypothesized ν_i are right, the rms deviation of n_i from ν_i is σ_i, so each term in the sum contributes ~ 1.

One often sees χ^2/N reported as a measure of goodness-of-fit. But... better to give $\chi^2$and N separately. Consider, e.g.,

$$\begin{aligned} \chi^2 &= 15,\; N = 10 \rightarrow p-\text{value} = 0.13 \,, \\ \chi^2 &= 150,\; N = 100 \rightarrow p-\text{value} = 9.0 \times 10^{-4} \,. \end{aligned}$$

i.e. for N large, even a χ^2 per dof only a bit greater than one can imply a small p-value, i.e., poor goodness-of-fit.

Pearson's χ^2 with multinomial data

If $n_{\rm tot} = \sum_{i=1}^{N}$ is fixed, then we might model $n_i \sim$ binomial

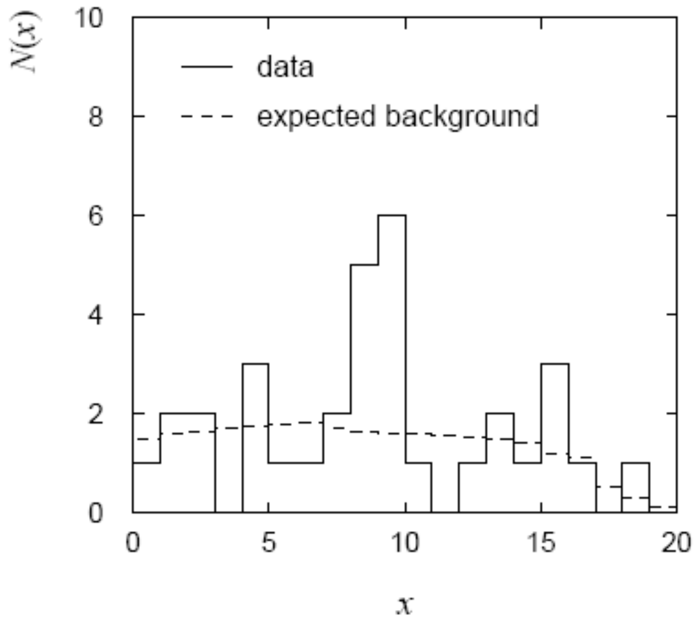
with $p_i = n_i / n_{\rm tot}$. I.e. $\vec{n} = (n_1, \ldots, n_N) \sim$ multinomial.

In this case we can take Pearson's χ^2 statistic to be

$$\chi^2 = \sum_{i=1}^{N} \frac{(n_i - p_i n_{\rm tot})^2}{p_i n_{\rm tot}}$$

If all $p_i\, n_{\rm tot} >> 1$ then this will follow the chi-square pdf for $N-1$ degrees of freedom.

Example of a χ^2 test



← This gives

$$\chi^2 = \sum_{i=1}^{N} \frac{(n_i - \nu_i)^2}{\nu_i} = 29.8$$

for $N = 20$ dof.

Now need to find p-value, but... many bins have few (or no) entries, so here we do not expect χ^2 to follow the chi-square pdf.

Using MC to find distribution of χ^2 statistic

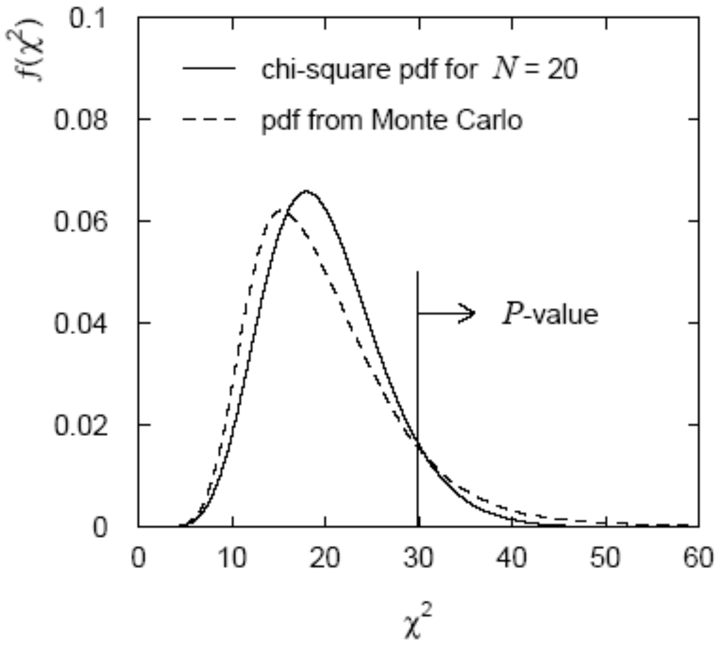
The Pearson χ^2 statistic still reflects the level of agreement between data and prediction, i.e., it is still a 'valid' test statistic.

To find its sampling distribution, simulate the data with a Monte Carlo program: $n_i \sim \text{Poisson}(\nu_i)\,, \; i = 1, N\,.$

Here data sample simulated 10^6 times. The fraction of times we find $\chi^2 > 29.8$ gives the p-value:

$$p = 0.11$$

If we had used the chi-square pdf we would find $p = 0.073$.



Extra slides

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 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:

$\vec{x}_a = (x_1, \ldots, x_n)$	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"

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

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.

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:

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

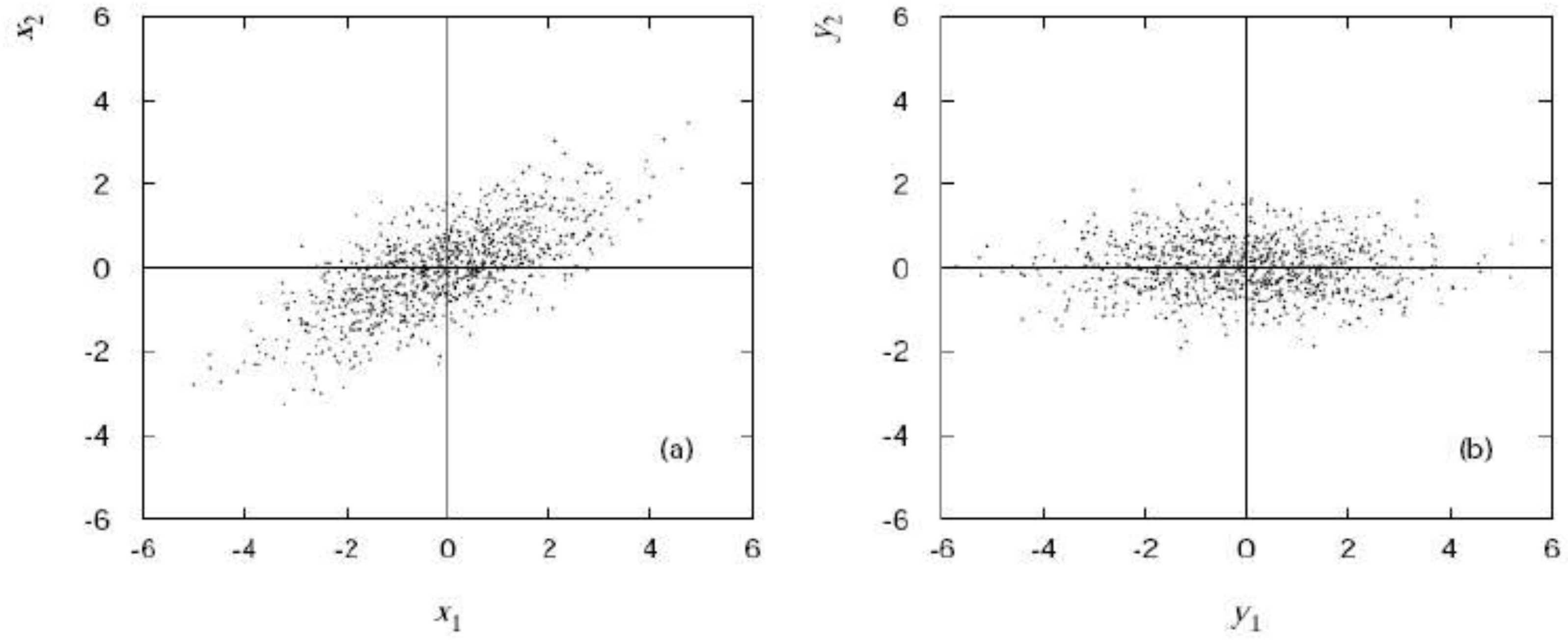
holds only if the components of $\boldsymbol{x}$ are independent

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

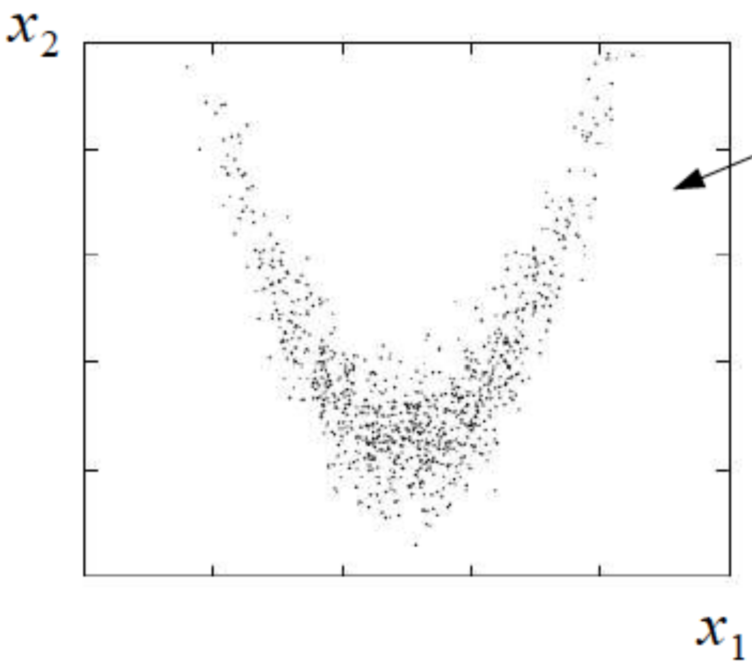
But we can define a set of uncorrelated input variables by a linear transformation, i.e., find the matrix A such that for $\vec{y} = A\vec{x}$ the covariances $\mathrm{cov}[y_i, y_\mathrm{i}] = 0$:



For the following suppose that the variables are "decorrelated" in this way for each of $p(\mathbf{x}|H_0)$ and $p(\boldsymbol{x}|H_1)$ 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

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, $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, estimate $f(\boldsymbol{x})$ 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 $\boldsymbol{x}$ (*k*-nearest neighbor, range search).