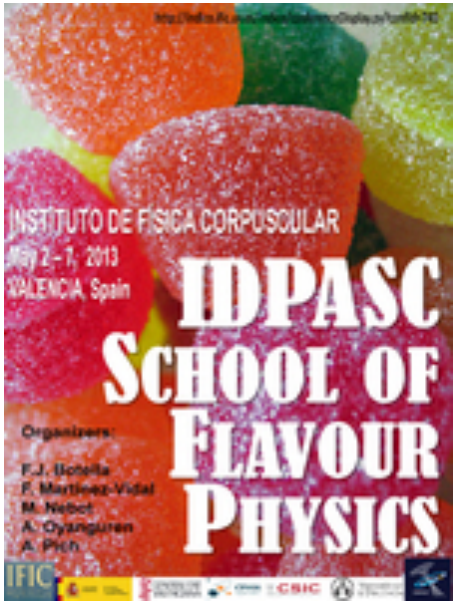
Statistical Analysis Tools for Particle Physics



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

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

Software

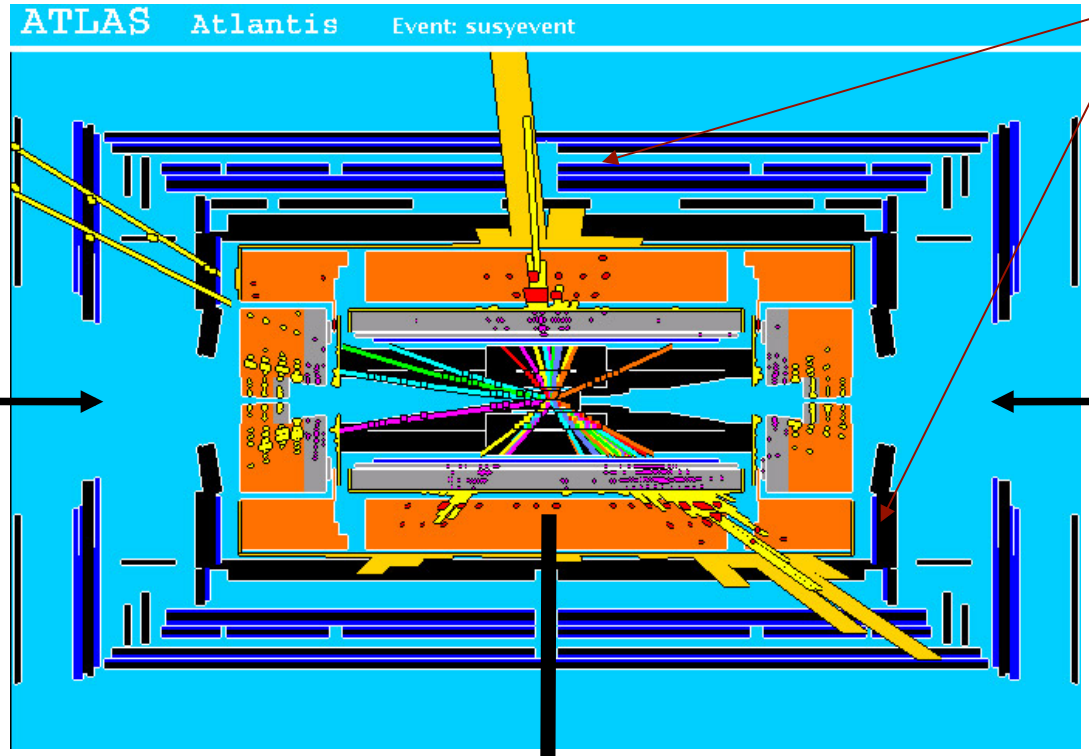
TMVA, Höcker, Stelzer, Tegenfeldt, Voss, Voss, physics/0703039

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



high p_T muons

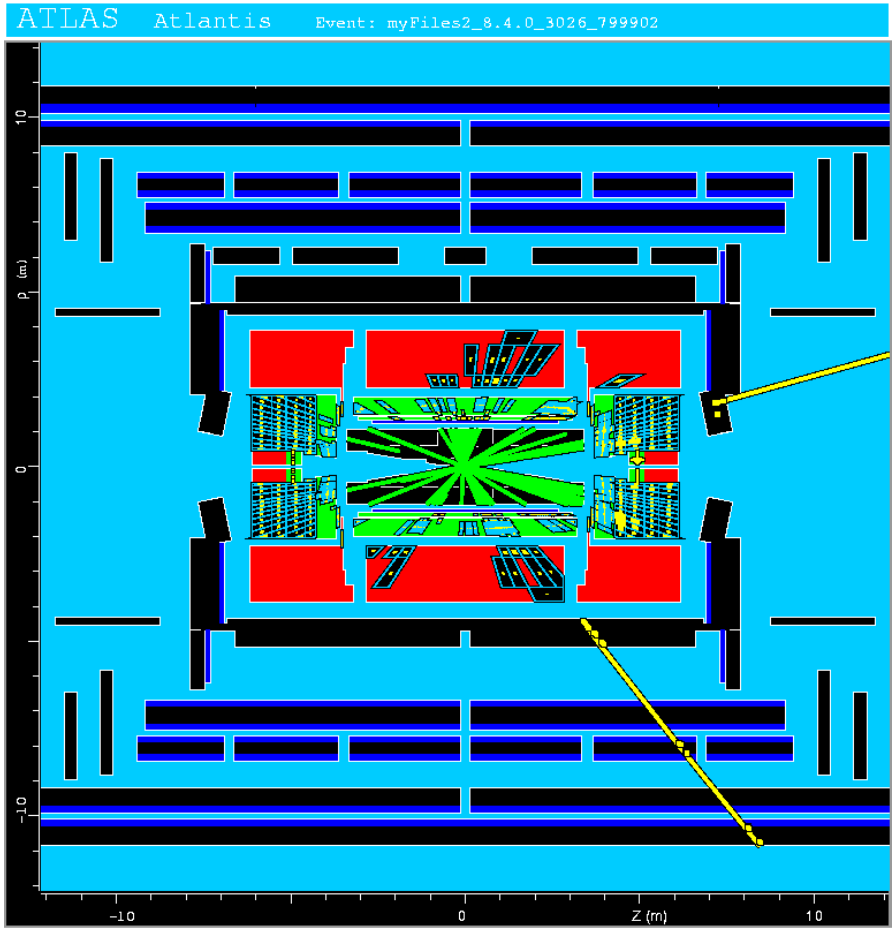
high p_T jets of hadrons

p

p

missing transverse energy

Background events



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

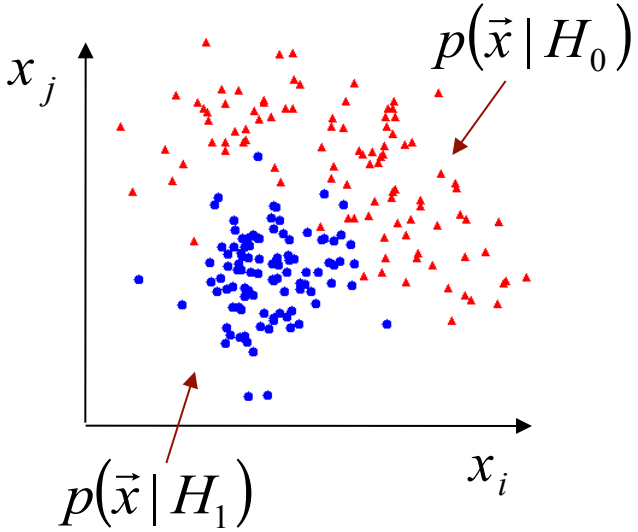
→ can easily mimic a SUSY event.

Event selection as a statistical test

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

x_1 = jet p_T
x_2 = missing energy
x_3 = 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 $pp \to t\bar{t}$, $pp \to \tilde{g}\tilde{g}$,…

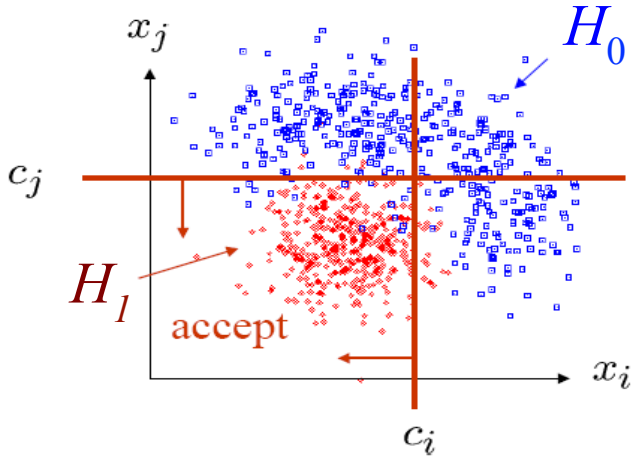


E.g. hypotheses H_0, H_1, ...
Often simply "signal", "background"

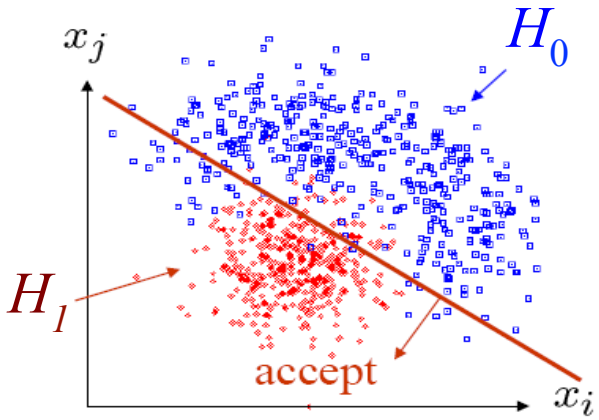
Finding an optimal decision boundary

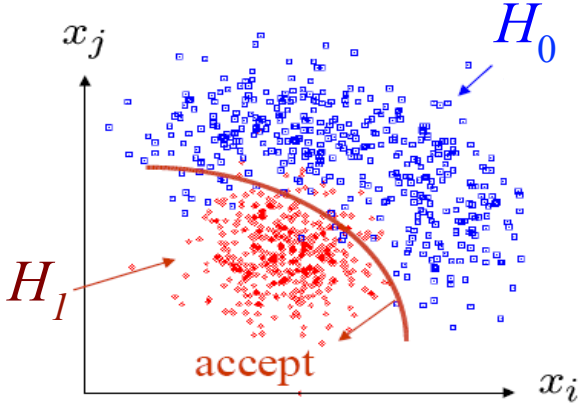
In particle physics usually start by making simple "cuts":

$$x_i < c_i$$
$$x_j < c_j$$



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):

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

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:

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

Test statistics

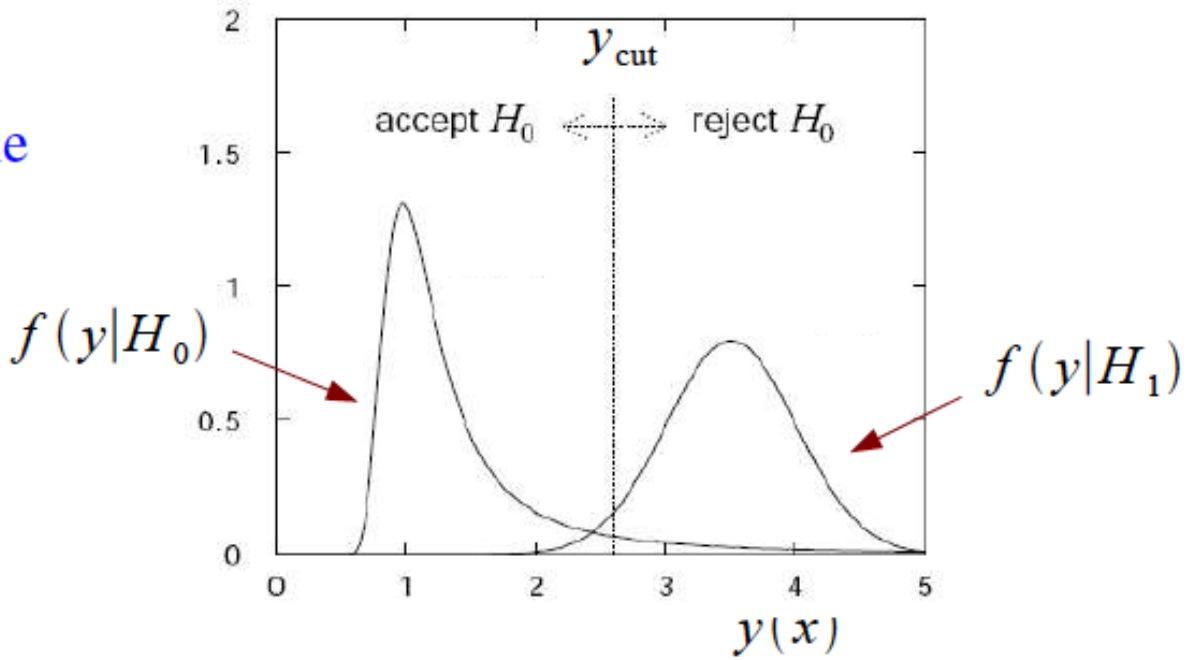
The decision boundary is a surface in the n-dimensional space of input variables, e.g., $y(\vec{x}) = \text{const}$.

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 $y(\boldsymbol{x})$, dividing $\boldsymbol{x}$-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): $\alpha = \int_{R_1} f(y|H_0)\,dy$

α = 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:

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

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

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

posterior probability

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

Constructing a test statistic

The Neyman-Pearson lemma states: to obtain the highest background rejection for a given signal efficiency (highest power for a given significance level), choose the acceptance region for signal such that

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

where c is a constant that determines the signal efficiency.

Equivalently, the optimal discriminating function is given by the likelihood ratio:

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

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 ~ $s/\sqrt{b}$

Neyman-Pearson doesn't always help

The problem is that we usually don't have explicit formulae for the pdfs $p(\boldsymbol{x}|\mathrm{s})$, $p(\boldsymbol{x}|\mathrm{b})$, so for a given $\boldsymbol{x}$ 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 $\vec{x} \sim p(\vec{x}|\mathrm{s})$ $\longrightarrow$ $\vec{x}_1, \ldots, \vec{x}_{N_s}$

generate $\vec{x} \sim p(\vec{x}|\mathrm{b})$ $\longrightarrow$ $\vec{x}_1, \ldots, \vec{x}_{N_b}$

"training data"
events of known type

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 $y(\boldsymbol{x})$ with fewer parameters; determine them (using MC) to give best discrimination between signal and background.

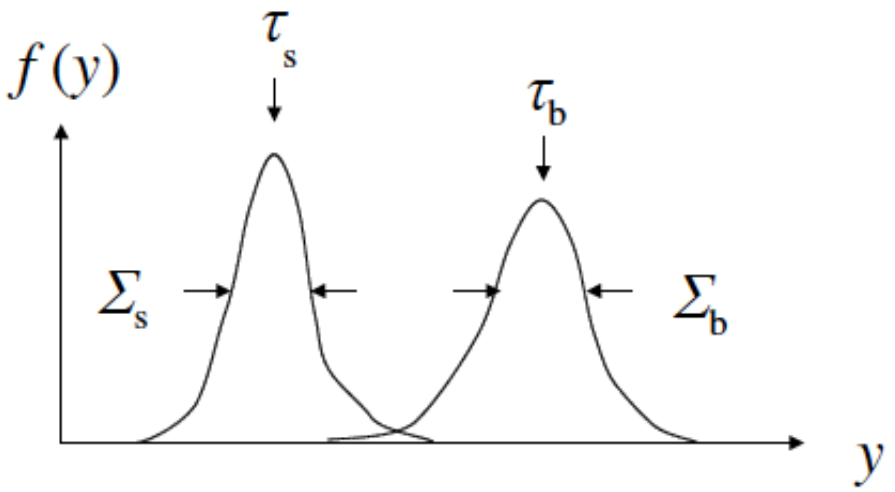
Alternatively, try to estimate the probability densities $p(\boldsymbol{x}|\mathrm{s})$, $p(\boldsymbol{x}|\mathrm{b})$ and use the estimated pdfs to compute the likelihood ratio at the desired $\boldsymbol{x}$ values (for real data).

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, ..., w_n so that the pdfs $f(y|\mathrm{s}), f(y|\mathrm{b})$ have maximum 'separation'. We want:

large distance between mean values, small widths



→ Fisher: maximize $J(\vec{w}) = \dfrac{(\tau_\mathrm{s} - \tau_\mathrm{b})^2}{\Sigma_\mathrm{s}^2 + \Sigma_\mathrm{b}^2}$

Coefficients for maximum separation

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

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

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 $\boldsymbol{w}$

The numerator of $J(\boldsymbol{w})$ is

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

'between' classes

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

and the denominator is

'within' classes

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

→ maximize $\quad J(\vec{w})=\dfrac{\vec{w}^T B\,\vec{w}}{\vec{w}^T W\,\vec{w}}=\dfrac{\text{separation between classes}}{\text{separation within classes}}$

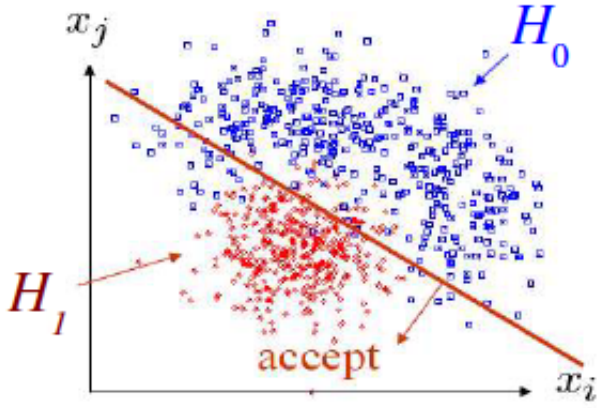
Fisher discriminant function

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

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

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.,

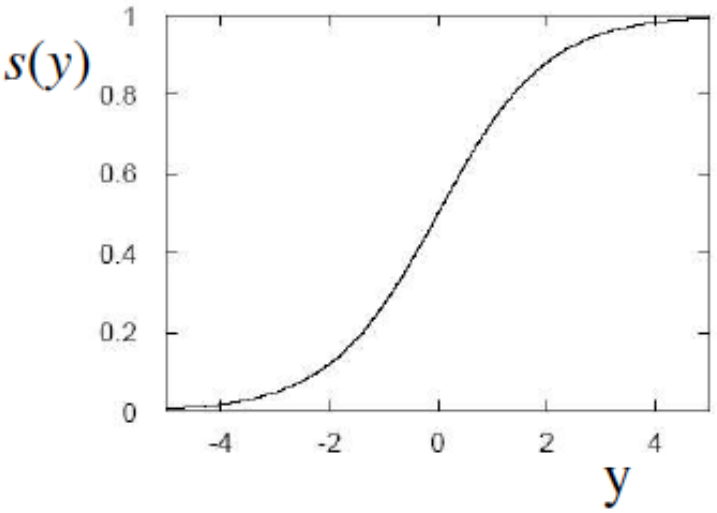
$$P(H_0|\vec{x}) = \frac{p(\vec{x}|H_0)\,P(H_0)}{p(\vec{x}|H_0)\,P(H_0) + p(\vec{x}|H_1)\,P(H_1)}$$

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.)



Transformation of inputs

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

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

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

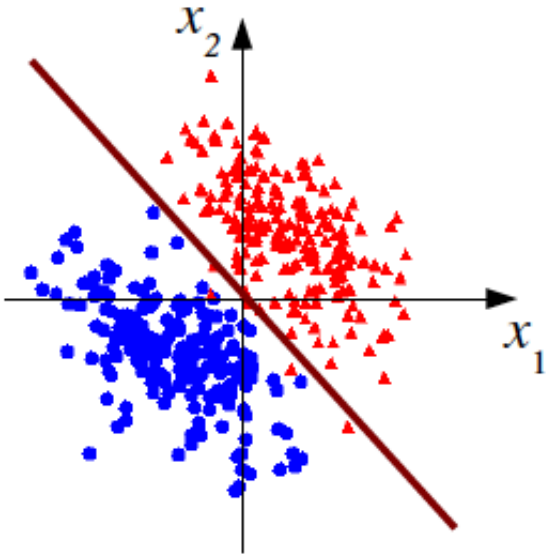
In other cases we will see that the basis functions only enter as dot products

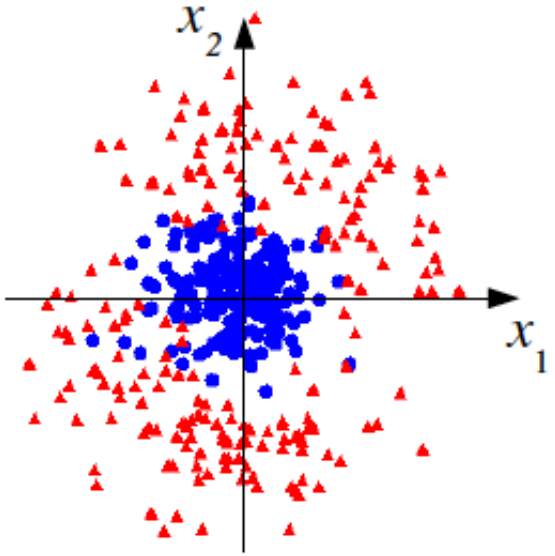
$$\vec{\varphi}(\vec{x}_i)\cdot\vec{\varphi}(\vec{x}_j) = K(\vec{x}_i, \vec{x}_j)$$

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

Linear decision boundaries

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





For some other cases a linear boundary is almost useless.

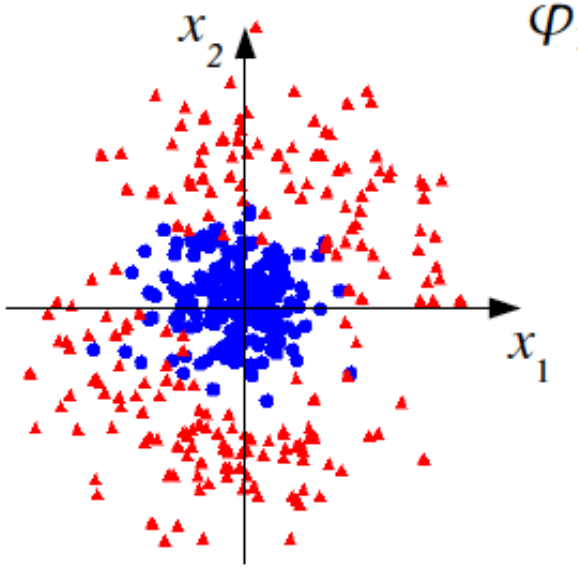
Nonlinear transformation of inputs

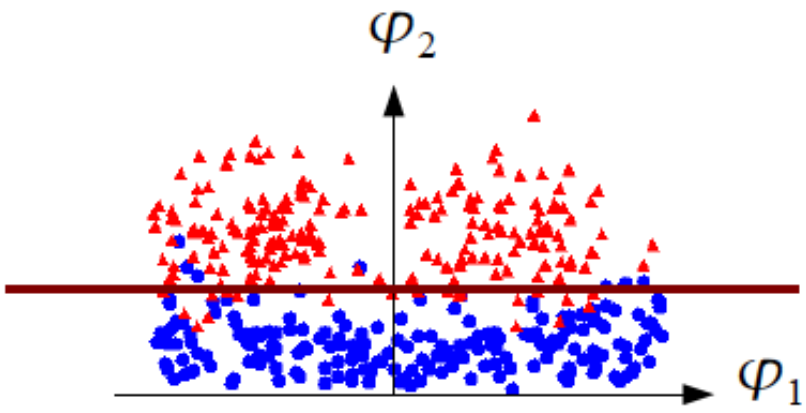
We can try to find a transformation, $x_1, \ldots, x_n \to \varphi_1(\vec{x}), \ldots, \varphi_m(\vec{x})$ so that the transformed "feature space" variables can be separated better by a linear boundary:

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

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

Here, guess fixed basis functions (no free parameters)





Neural networks

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

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

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

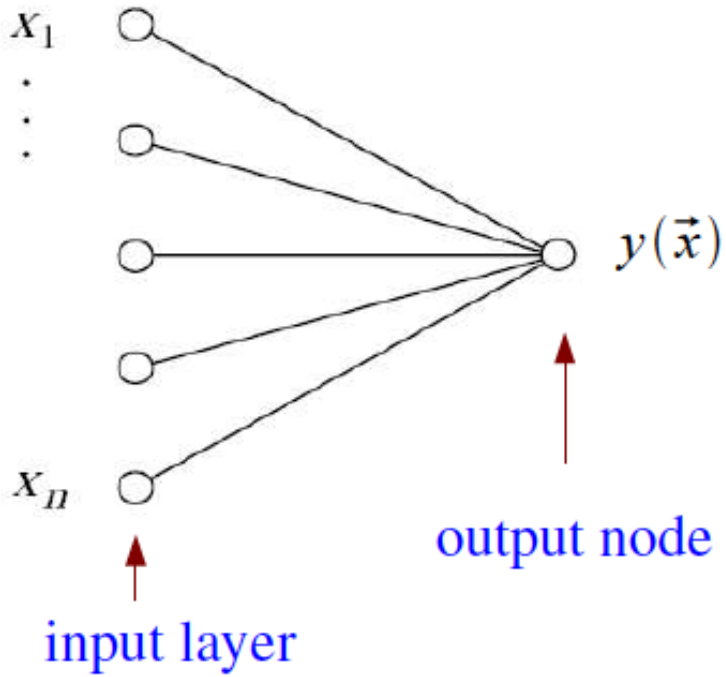
The training data are then used to adjust the parameters so that the resulting discriminant function has the best performance.

The single layer perceptron

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

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

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



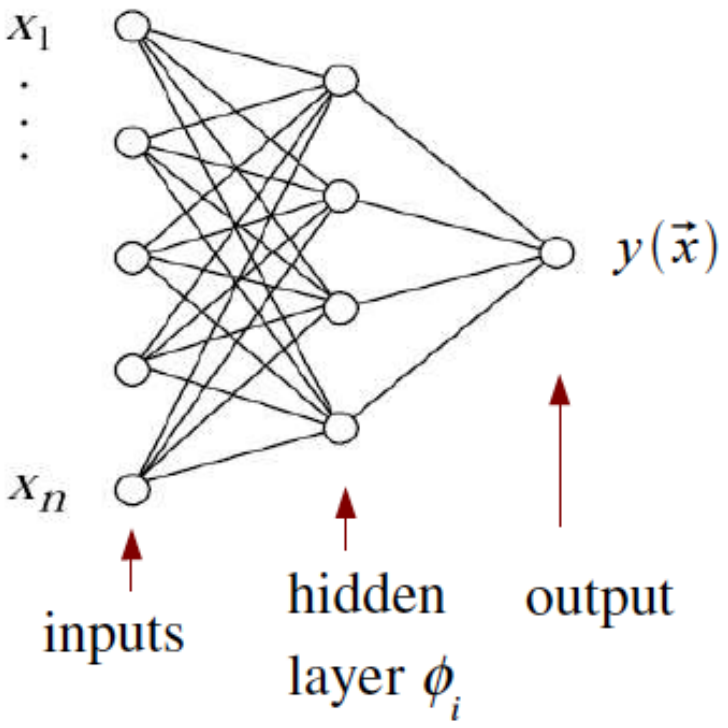
The multilayer perceptron

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

Superscript for weights indicates layer number

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

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



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

Network architecture: one hidden layer

Theorem: An MLP with a single hidden layer having a sufficiently large number of nodes can approximate arbitrarily well the Bayes 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"

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

Contribution to error function from each event

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

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

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

learning rate ($\eta>0$)

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

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

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

Error backpropagation

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

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

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

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

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

derivative of activation function

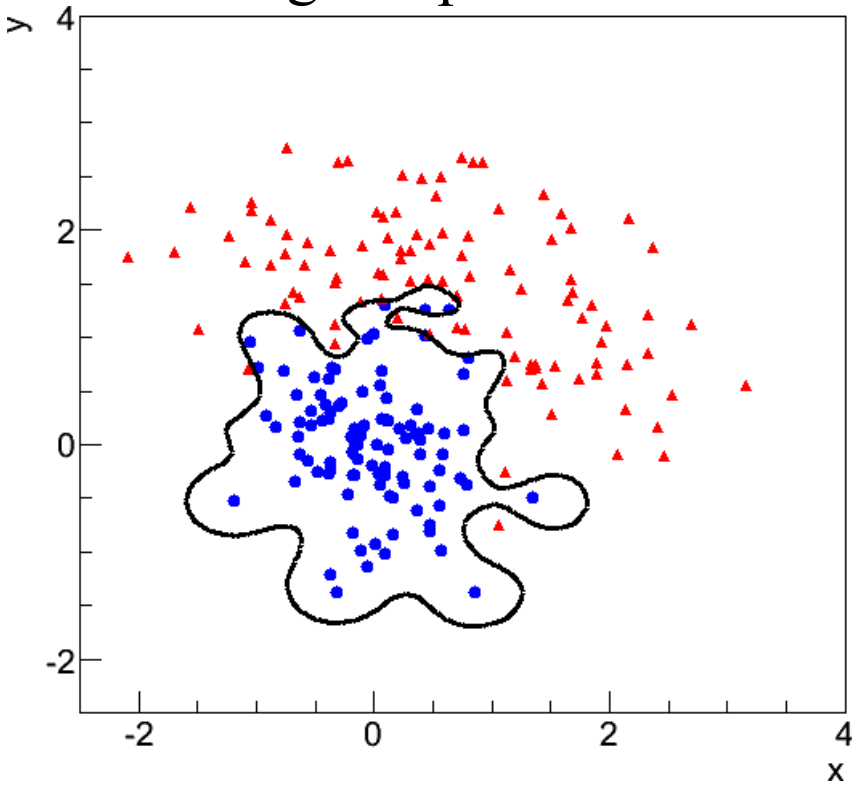
Chain rule gives all the needed derivatives.

Overtraining

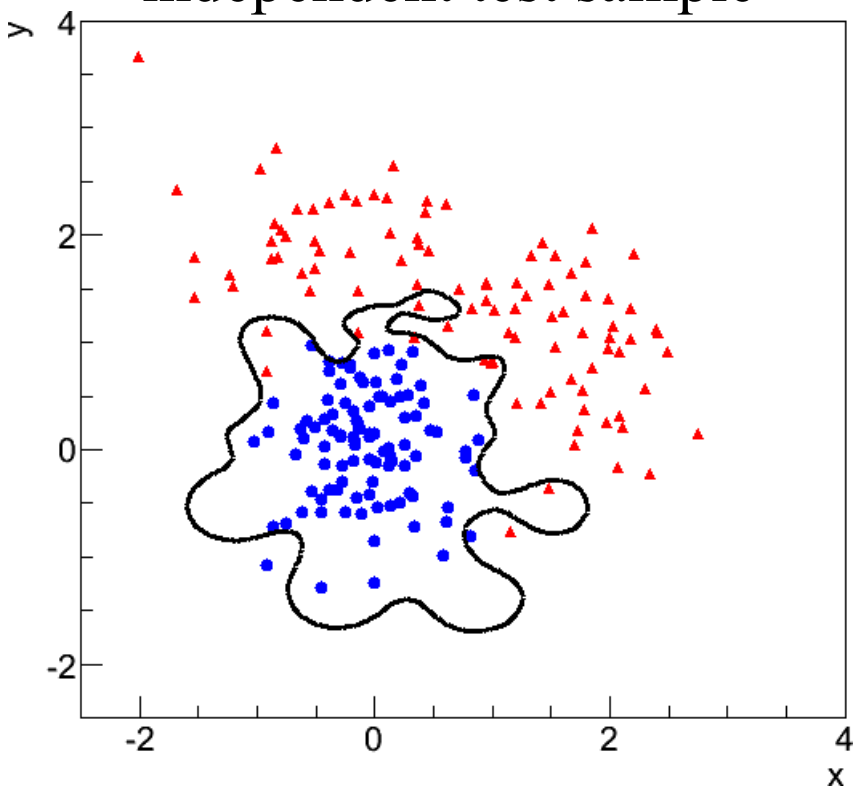
If decision boundary is too flexible it will conform too closely to the training points → overtraining.

Monitor by applying classifier to independent test sample.

training sample

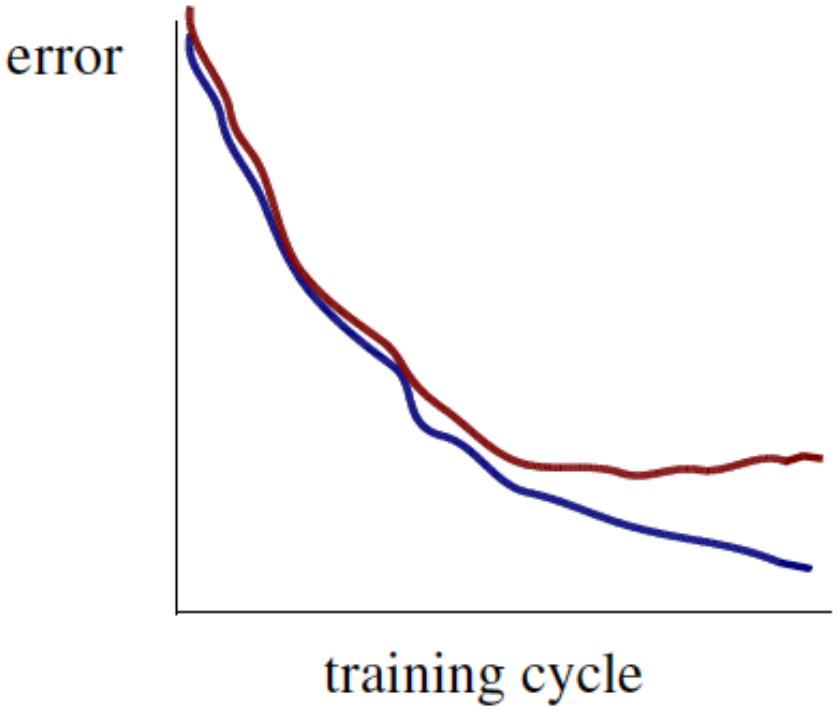


independent test sample



Monitoring overtraining

If we monitor the value of the error function $E(\boldsymbol{w})$ 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

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 may 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 too expensive (Narsky):	train : validate : test 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.,

regularization parameter

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

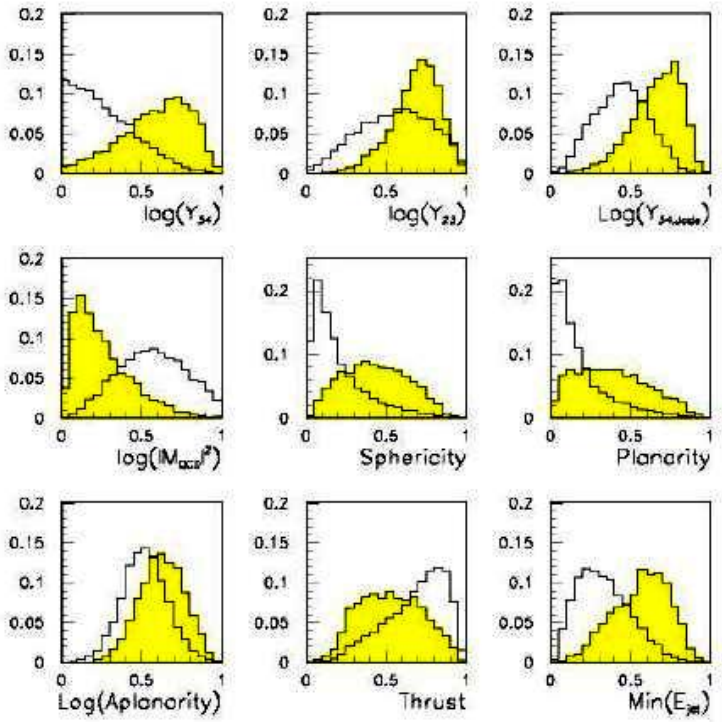
Increasing λ gives a smoother boundary (higher bias, lower variance)

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

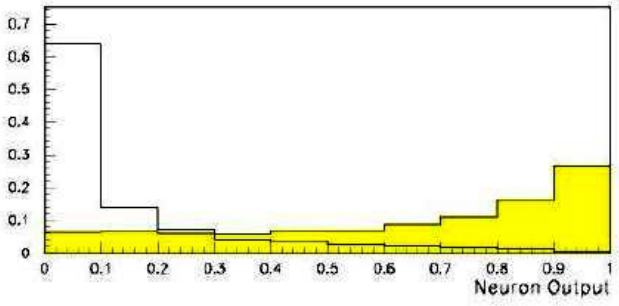
Signal: $e^+e^- \to W^+W^-$ (often 4 well separated hadron jets)

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



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

Neural network output:



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

Probability Density Estimation (PDE)

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

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

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

Correlation vs. independence

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

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

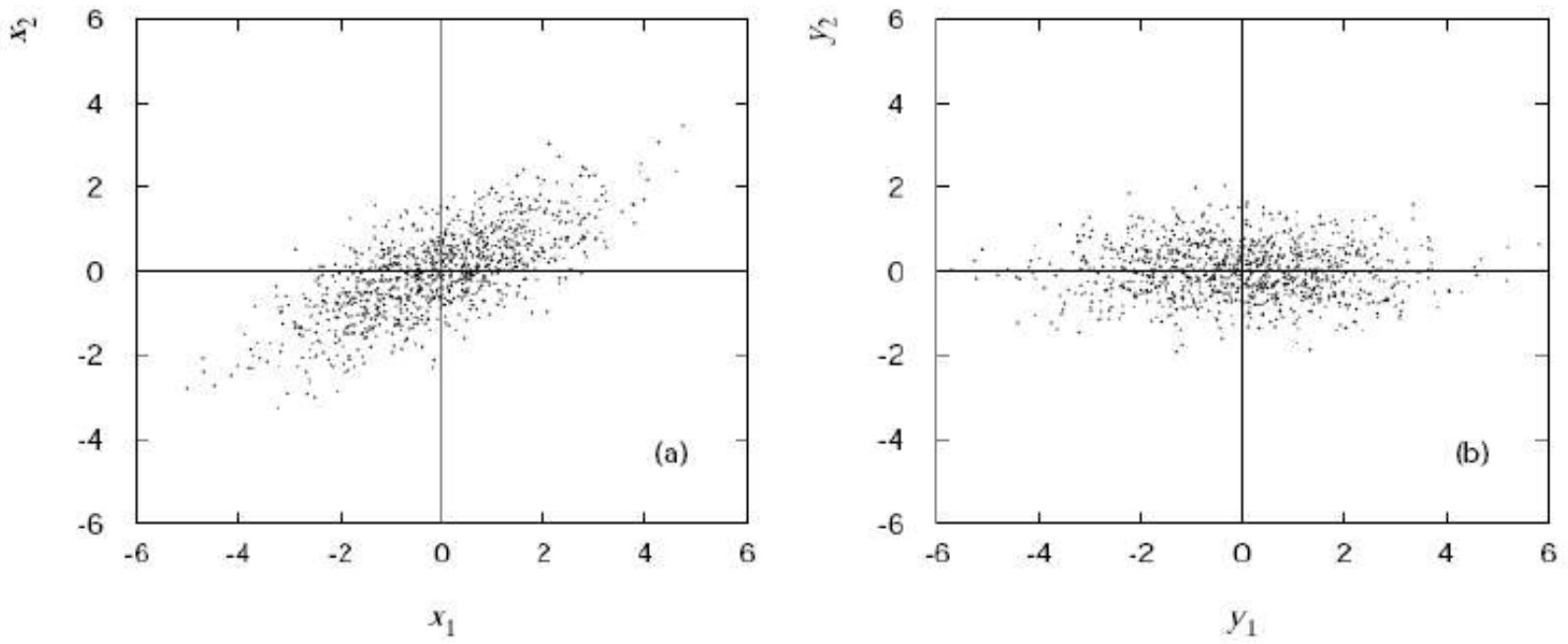
holds only if the components of $\boldsymbol{x}$ are independent

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

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

Decorrelation of input variables

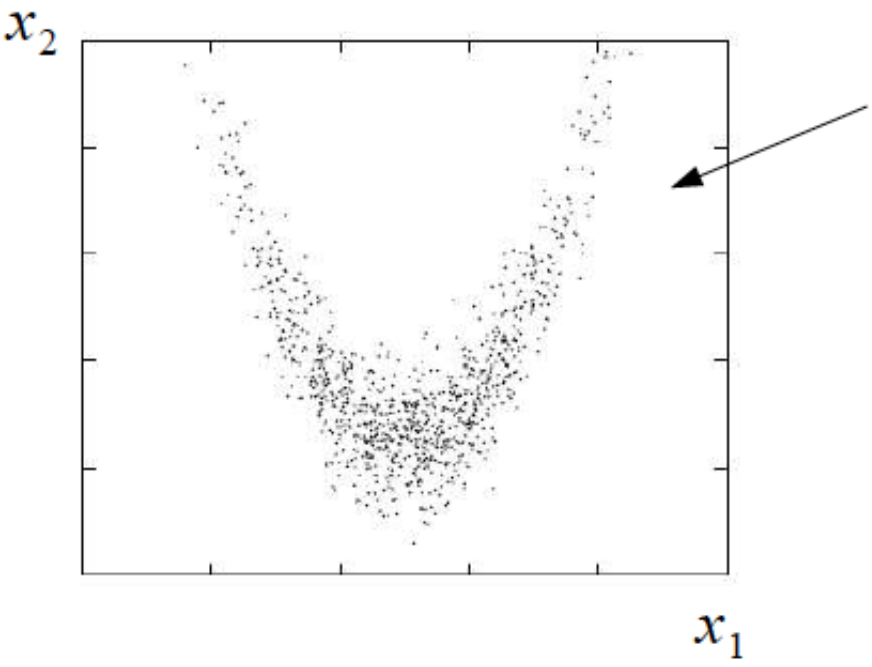
But we can define a set of uncorrelated input variables by a linear transformation, i.e., find the matrix A such that for $\vec{y} = A\vec{x}$ the covariances cov[y_i, y_j] = 0:



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

Decorrelation is not enough

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



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

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

and therefore

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

Naive Bayes

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

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

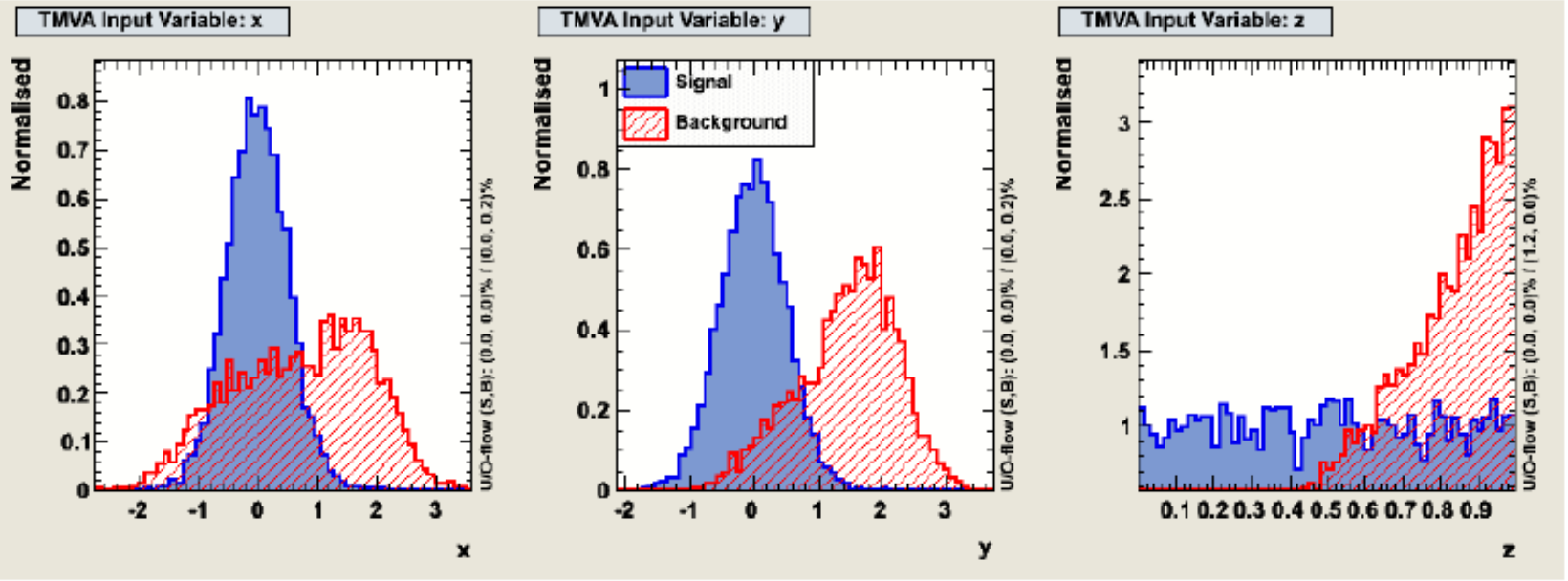
kernel

bandwidth (smoothing parameter)

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

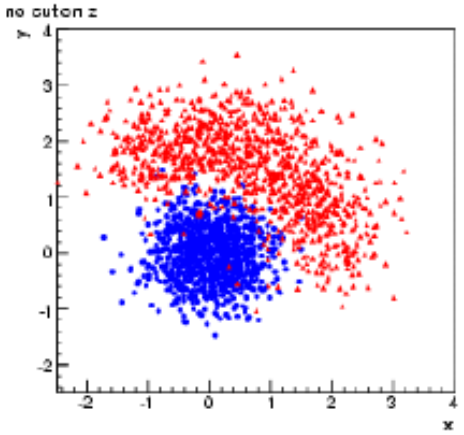
Need to sum N terms to evaluate function (slow); faster algorithms only count events in vicinity of $\boldsymbol{x}$ (k-nearest neighbor, range search).

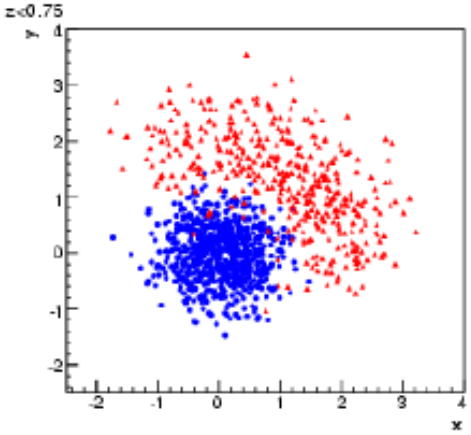
Test example with TMVA



Test example, x vs. y with cuts on z

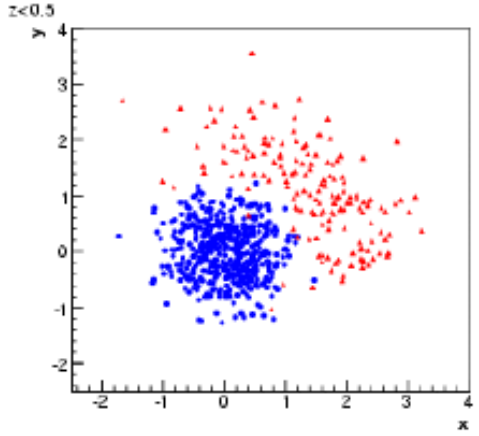
no cut on z

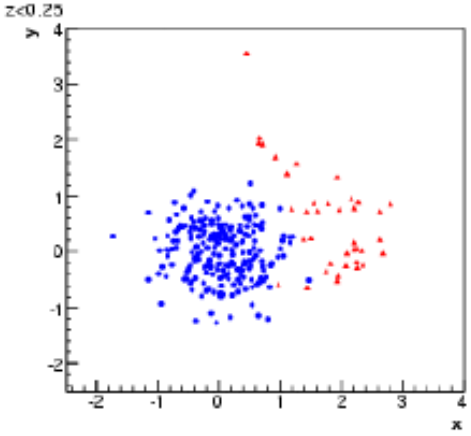




$z < 0.75$

$z < 0.5$

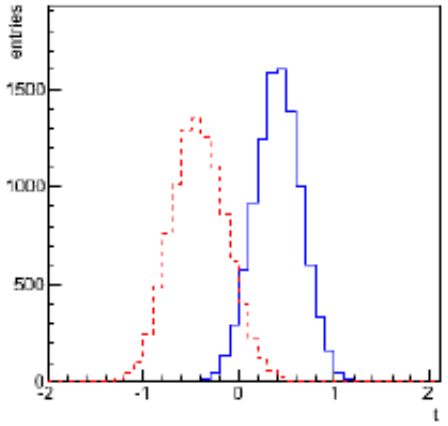


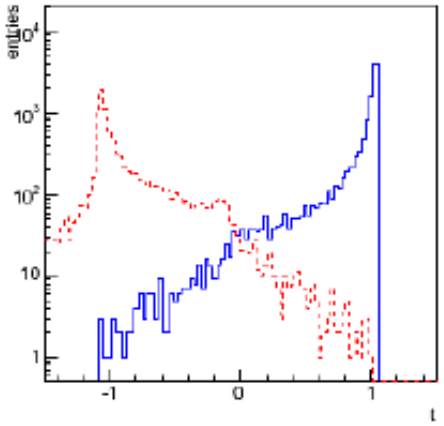


$z < 0.25$

Test example results

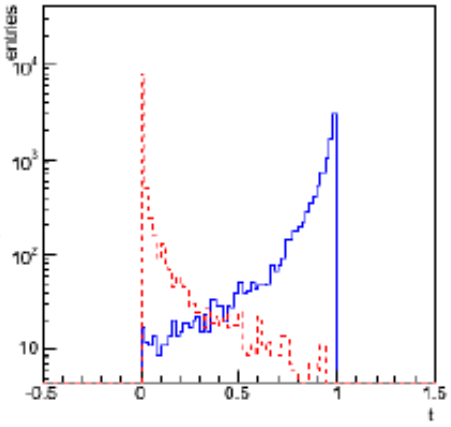
Fisher discriminant

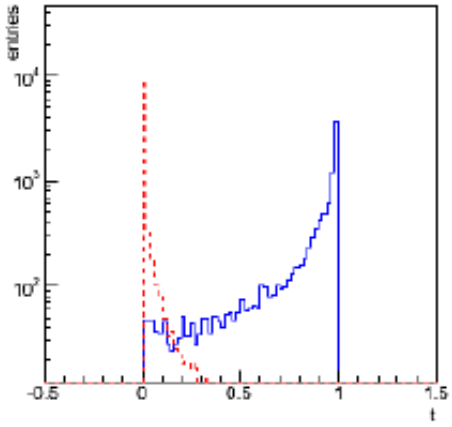




Multilayer perceptron

Naive Bayes, no decorrelation



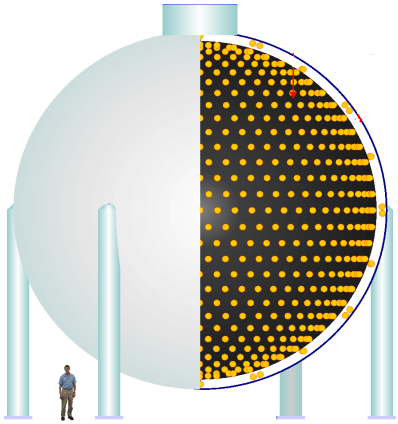


Naive Bayes with decorrelation

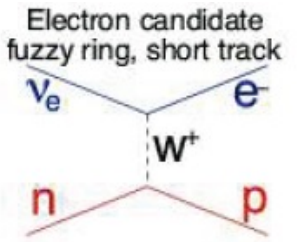
Particle i.d. in MiniBooNE

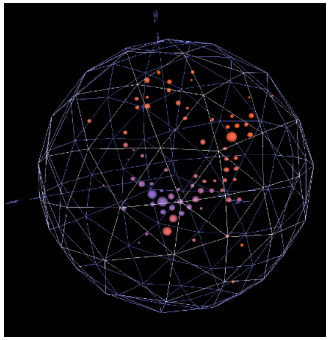
Detector is a 12-m diameter tank of mineral oil exposed to a beam of neutrinos and viewed by 1520 photomultiplier tubes:

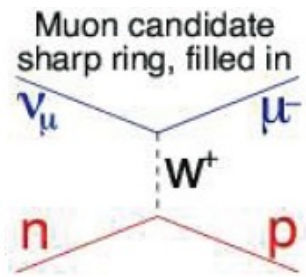
MiniBooNE Detector

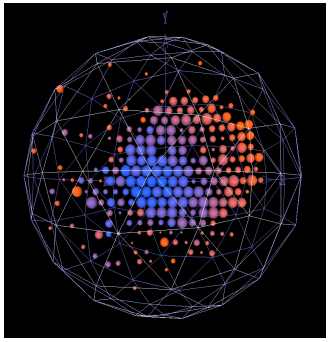


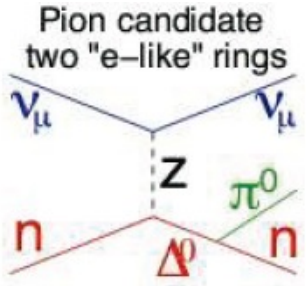
Search for ν_μ to ν_e oscillations required particle i.d. using information from the PMTs.

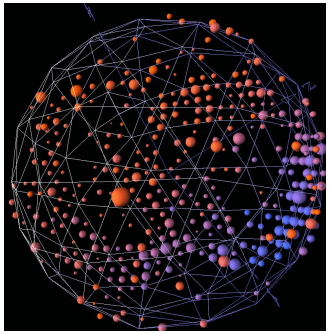












H.J. Yang, MiniBooNE PID, DNP06

Decision trees

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

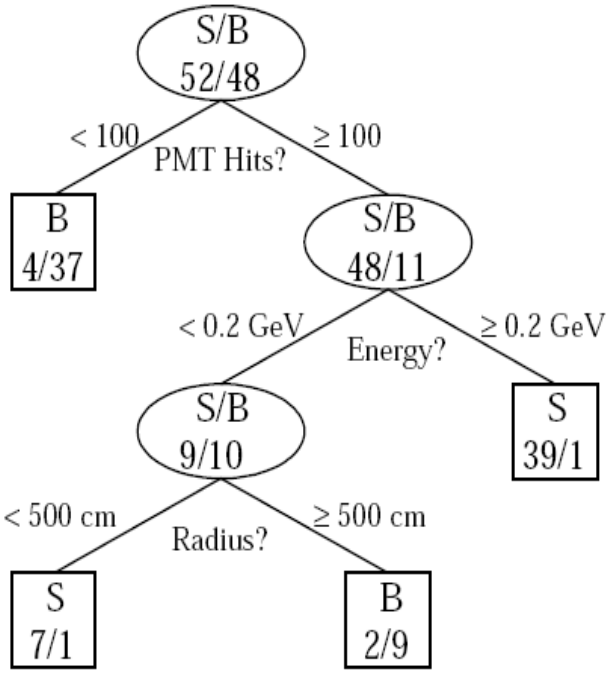
$$P = \frac{\sum_{\text{signal}} w_i}{\sum_{\text{signal}} w_i + \sum_{\text{background}} w_i}$$

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 as 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 the discriminant function

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

The same variable may be used at several nodes; others may not be used at all.

In principle a decision tree classifier can have perfect separation between the event classes (will happen e.g. if the terminal nodes have a single event).

This would clearly be a very over-trained classifier.

Decision tree size and stability

Usually one grows the tree first to a very large (e.g. maximum) size and then applies pruning.

For example one can recombine leaves based on some measure of generalization performance (e.g. using statistical error of purity estimates).

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 $\boldsymbol{x}_i$,

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

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

Updating the event weights

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

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

i = event index

k = training sample index

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

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

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

Error rate of the kth classifier

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

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

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

Assigning the classifier score

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

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

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

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

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

AdaBoost error rate

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

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

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

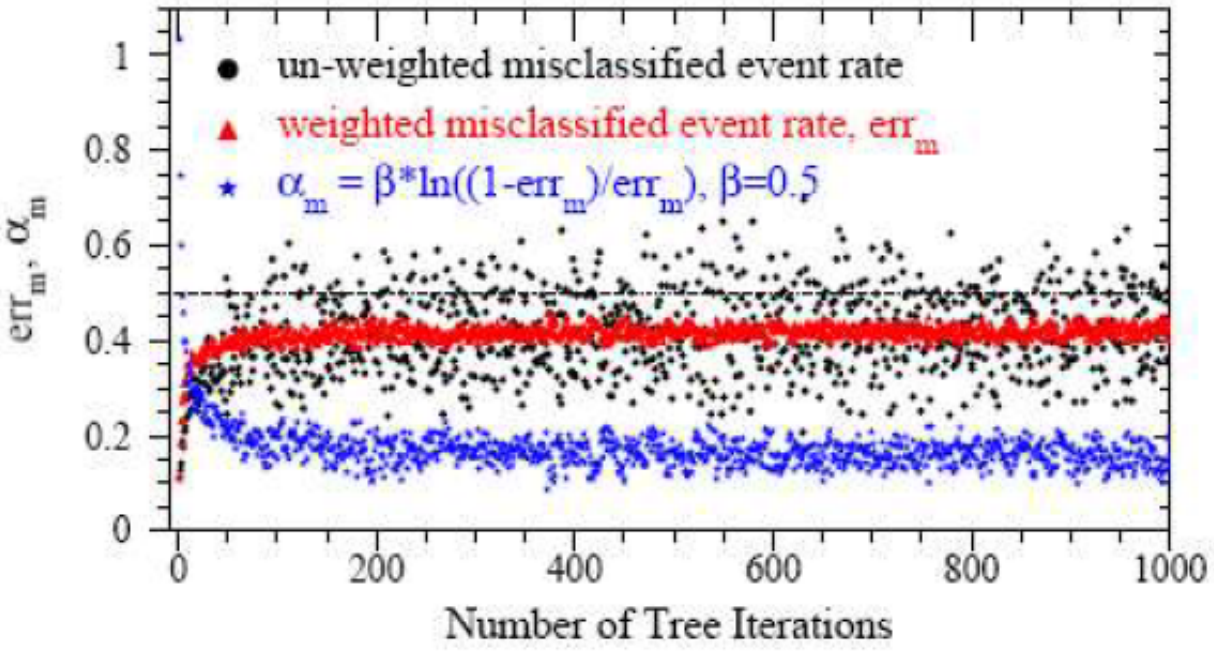
So the procedure is to monitor the error rate of the combined classifier at each step with a validation sample and to stop before it starts to rise.

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

BDT example from MiniBooNE

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

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

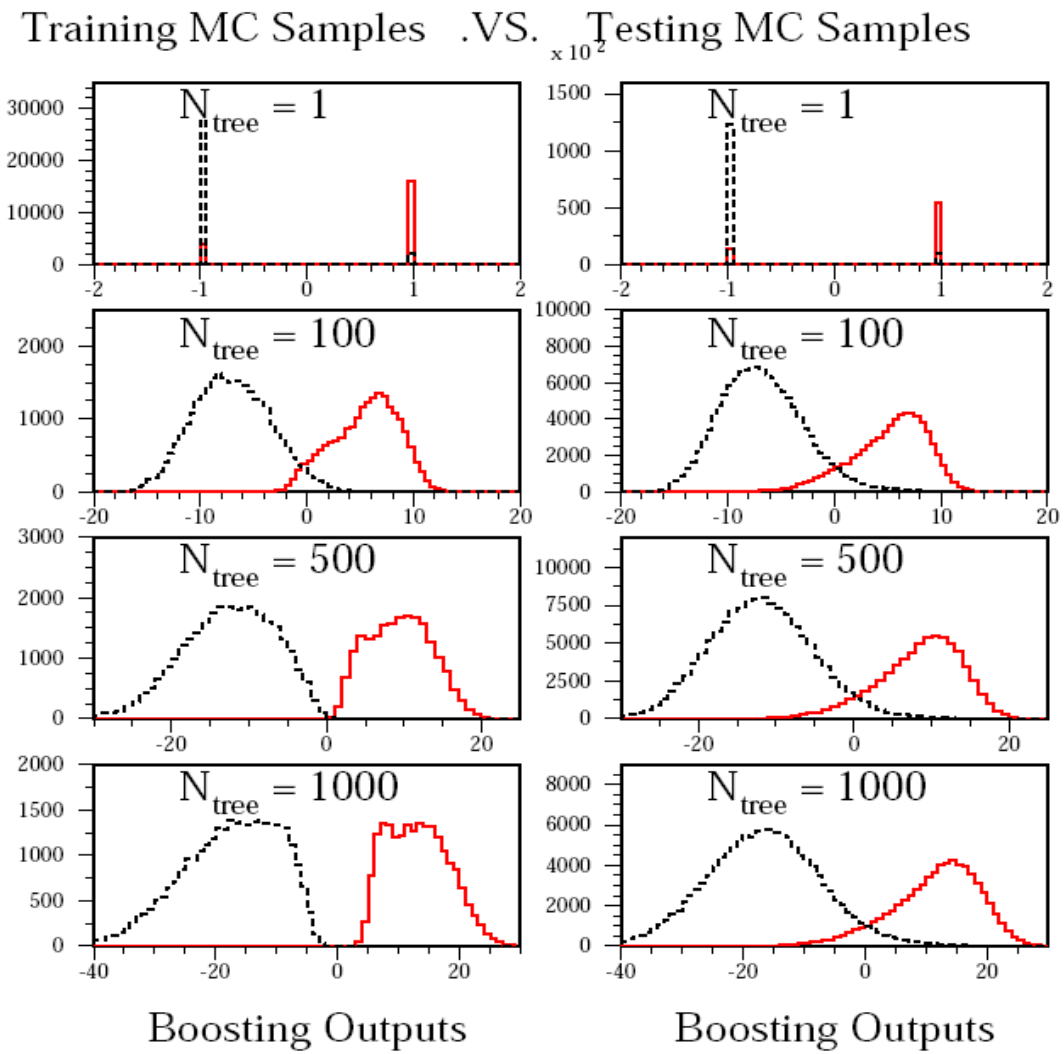


B. Roe et al., NIM 543 (2005) 577

Monitoring overtraining

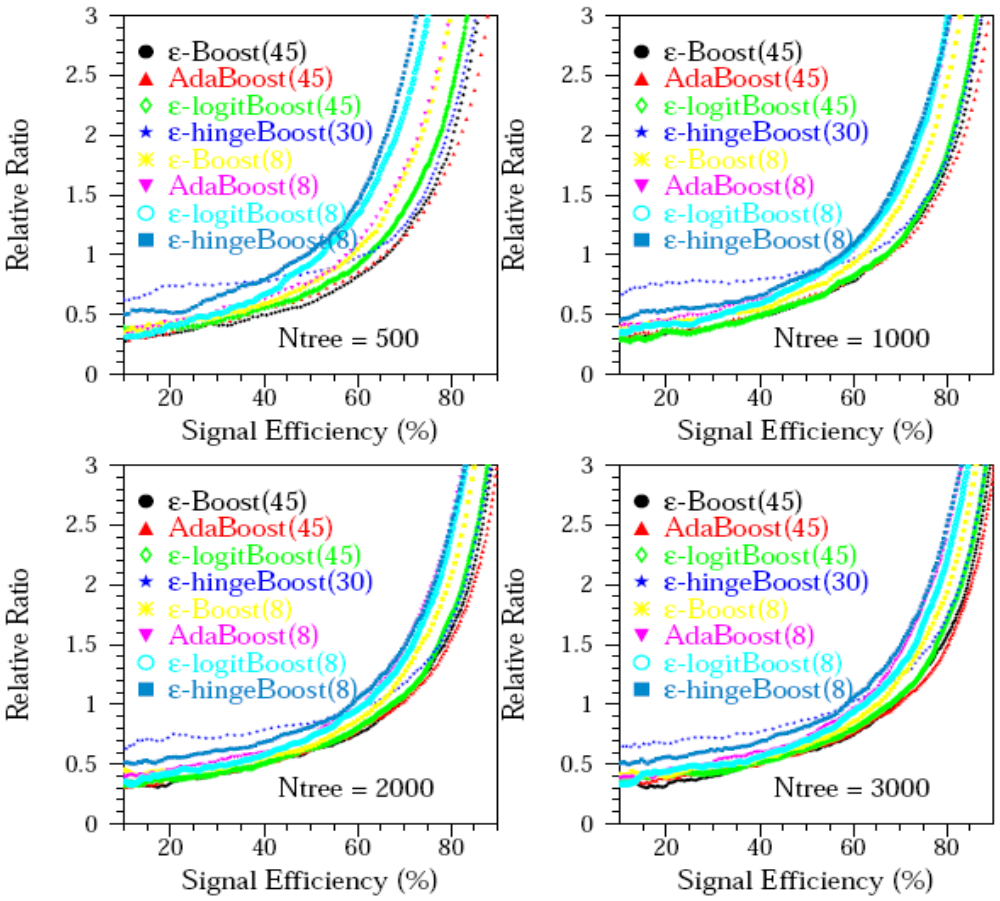
From MiniBooNE example:

Performance stable after a few hundred trees.



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 (Boostrap-Aggregating), generates the ensemble of classifiers by random sampling with replacement from the full training sample.

Support Vector Machines

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

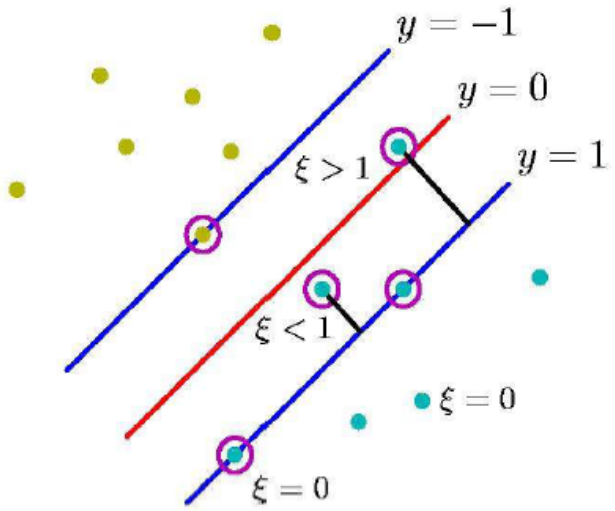
Maximize distance between separating hyperplanes (margin) subject to constraints allowing for some misclassification.

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

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

So only need kernel

$$K(\boldsymbol{x}, \boldsymbol{x}') = \vec{\varphi}(\boldsymbol{x}) \cdot \vec{\varphi}(\boldsymbol{x}')$$



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 on Monday

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 (!)

Linear SVMs

Consider a training data set consisting of

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

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

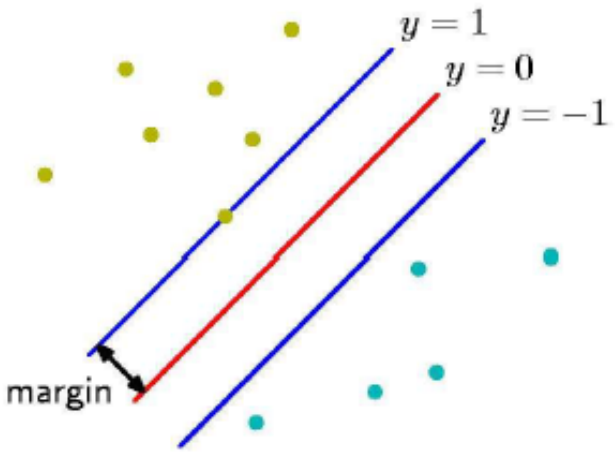
Suppose the classes can be separated by a hyperplane defined by a normal vector $\boldsymbol{w}$ and scalar offset b (the "bias"). We have

$\boldsymbol{x}_i \cdot \boldsymbol{w} + b \geqslant +1$ for all $y_i = +1$

$\boldsymbol{x}_i \cdot \boldsymbol{w} + b \leqslant -1$ for all $y_i = -1$

or equivalently

$y_i(\boldsymbol{x}_i \cdot \boldsymbol{w} + b) - 1 \geqslant 0$ for all i

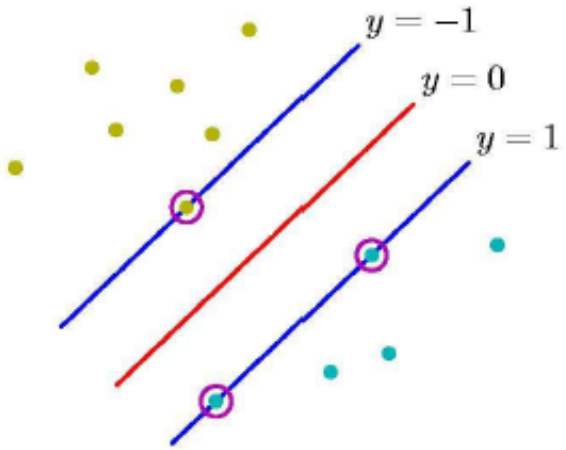


Bishop Ch. 7

Margin and support vectors

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

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



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 $\boldsymbol{w}$). These points are called support vectors.

Linear SVM classifier

We can define the classifier using

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

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

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

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

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

Lagrangian formulation

This constrained minimization problem can be reformulated using a Lagrangian

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

positive Lagrange multipliers α_i

We need to minimize L with respect to $\boldsymbol{w}$ 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

$$\boldsymbol{w} = \sum_i \alpha_i y_i \boldsymbol{x}_i$$

(sum only contains support vectors)

Dual formulation

The classifier function is thus

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

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

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

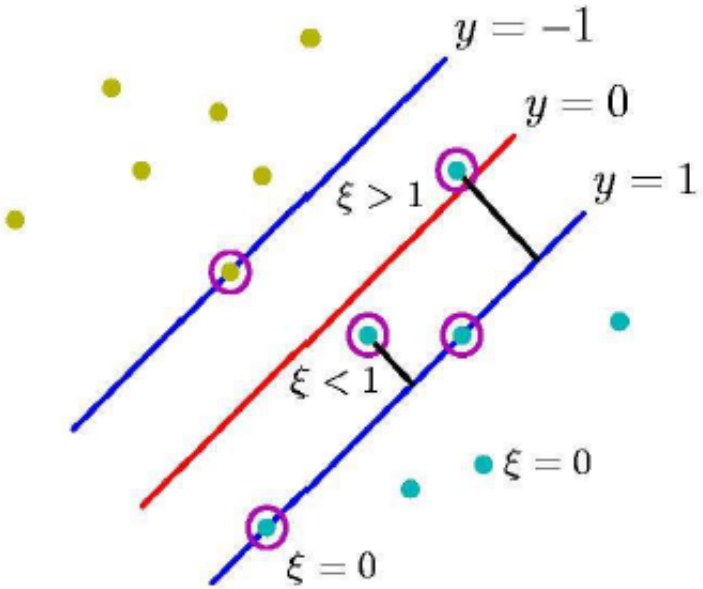
So this means that both the classifier function and the Lagrangian only involve dot products of vectors in the input variable space.

Nonseparable data

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

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

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



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 $\boldsymbol{w}$ to be

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

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

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

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

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:

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

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 $\boldsymbol{\phi}(\boldsymbol{x})$ appearing in the place of $\boldsymbol{x}$.

Only dot products

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

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

so in the feature space this becomes

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

The Kernel trick

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

$$K(\boldsymbol{x}, \boldsymbol{x}') = \vec{\varphi}(\boldsymbol{x}) \cdot \vec{\varphi}(\boldsymbol{x}')$$

Functions having this property must satisfy Mercer's condition

$$\int K(\boldsymbol{x}, \boldsymbol{x}') g(\boldsymbol{x}) g(\boldsymbol{x}') d\boldsymbol{x} d\boldsymbol{x}' \geqslant 0$$

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

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

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

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

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

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 the last several 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

Two distinct event selection problems

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

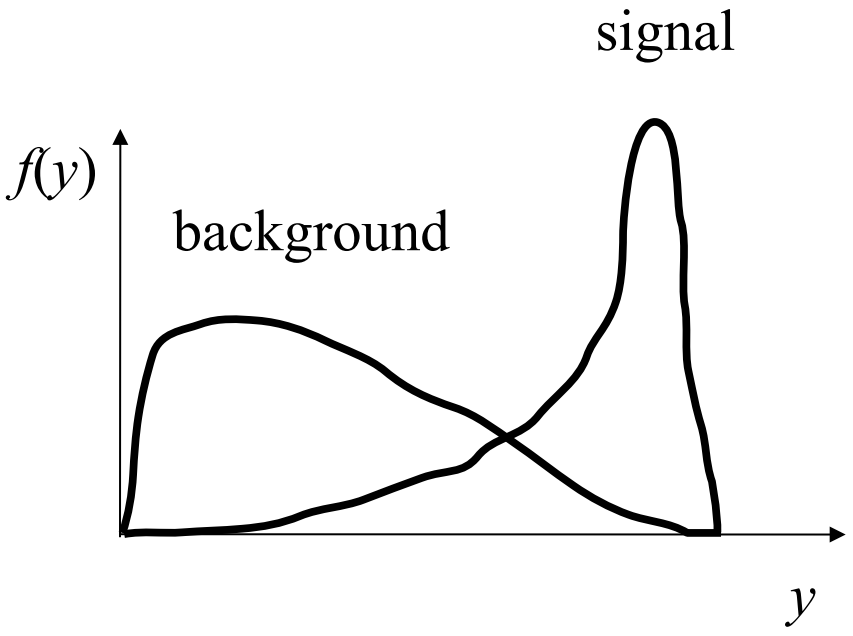
Example: separation of different particle types (electron vs muon)
Use the selected sample for further study.

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).

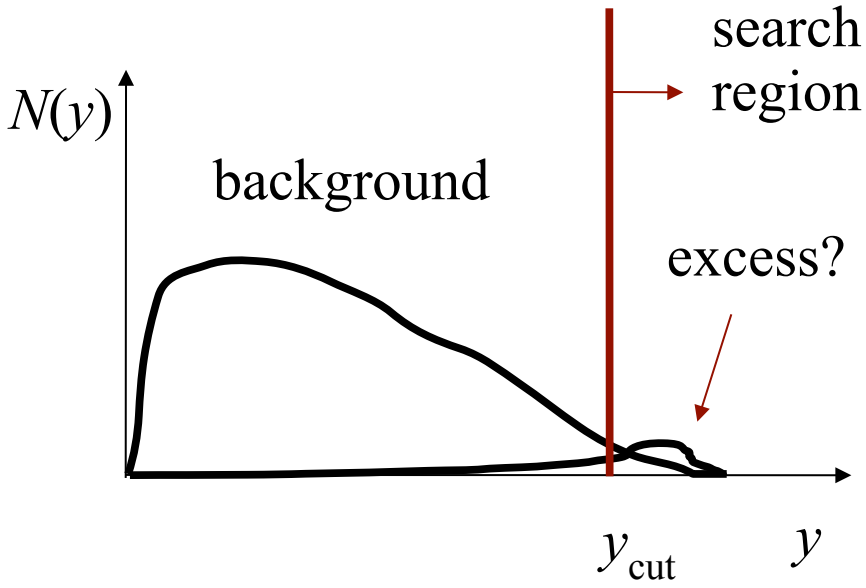
Many subtle issues here, mainly related to the heavy burden of proof required to establish presence of a new phenomenon.

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

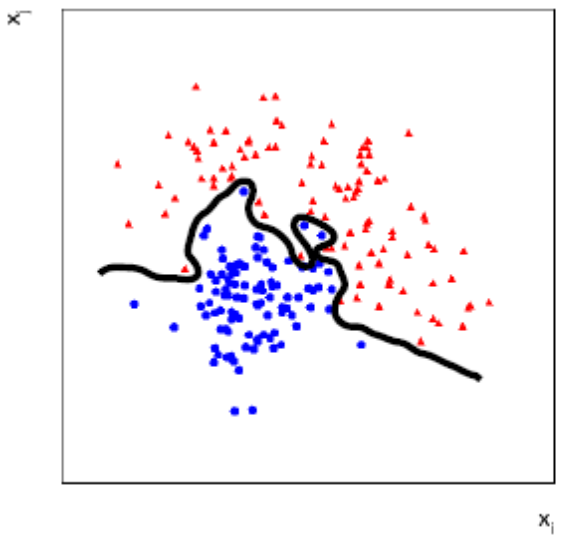
Discovery = number of events found in search region incompatible with background-only hypothesis.

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

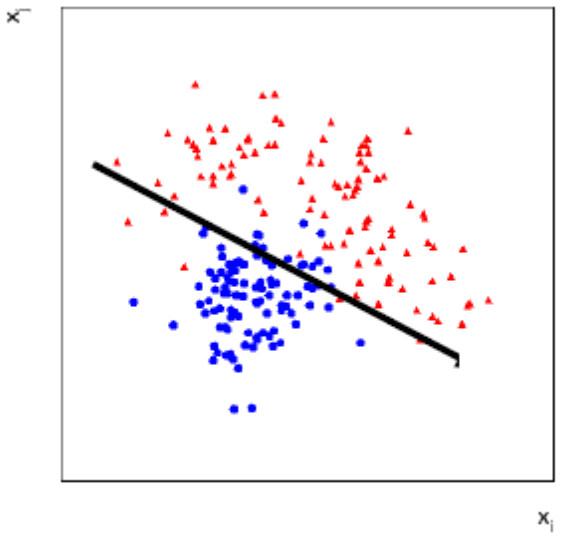
Decision boundary flexibility

The decision boundary will be defined by some free parameters that we adjust using training data (of known type) to achieve the best separation between the event types.

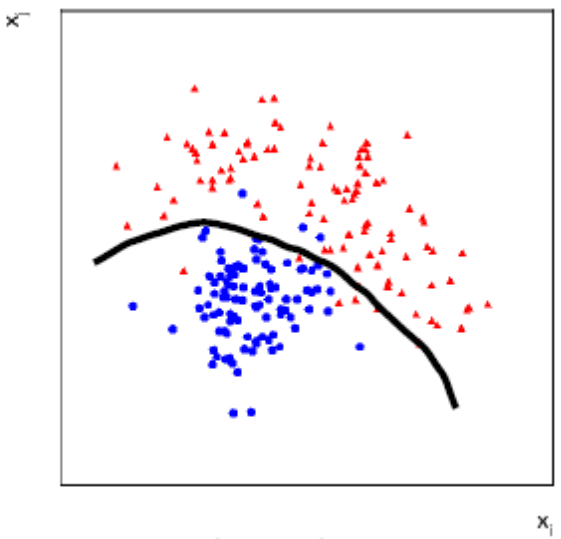
Goal is to determine the boundary using a finite amount of training data so as to best separate between the event types for an unseen data sample.



overtraining



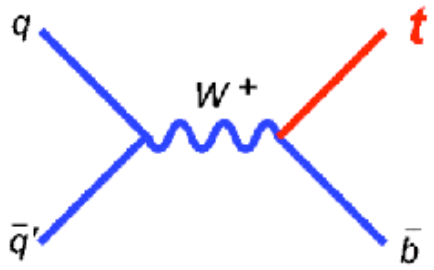
boundary too rigid



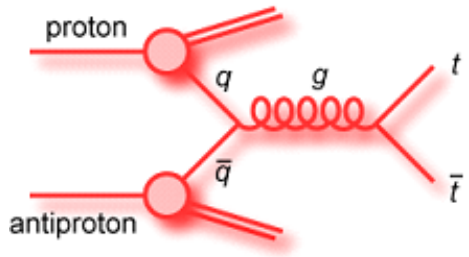
good trade-off

Single top quark production (CDF/D0)

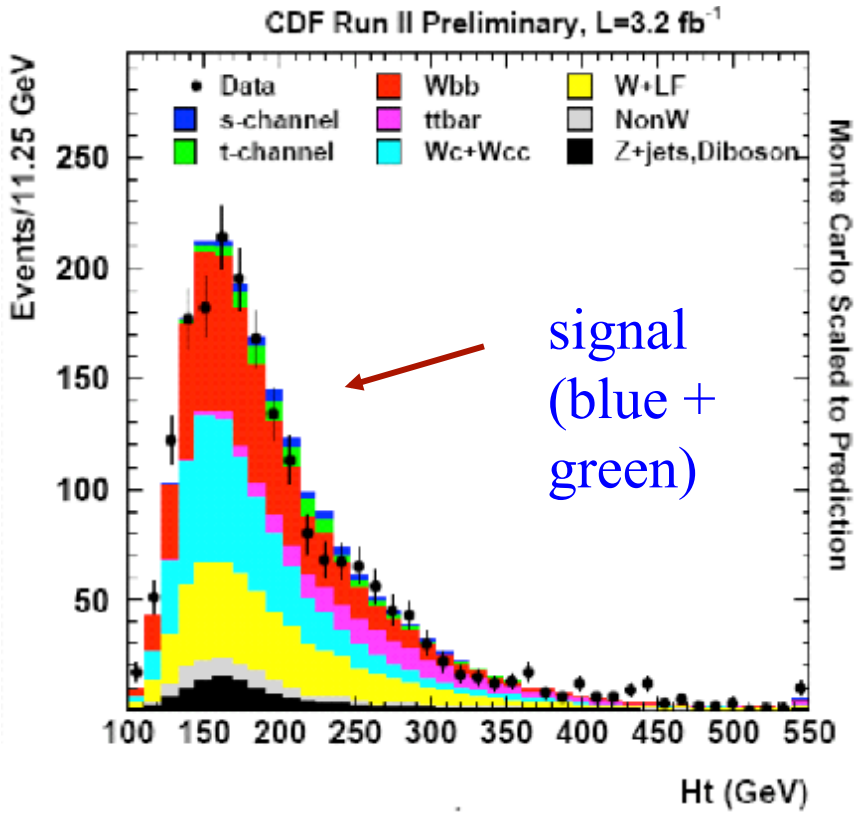
Top quark discovered in pairs, but
SM predicts single top production.



Pair-produced tops are now
a background process.

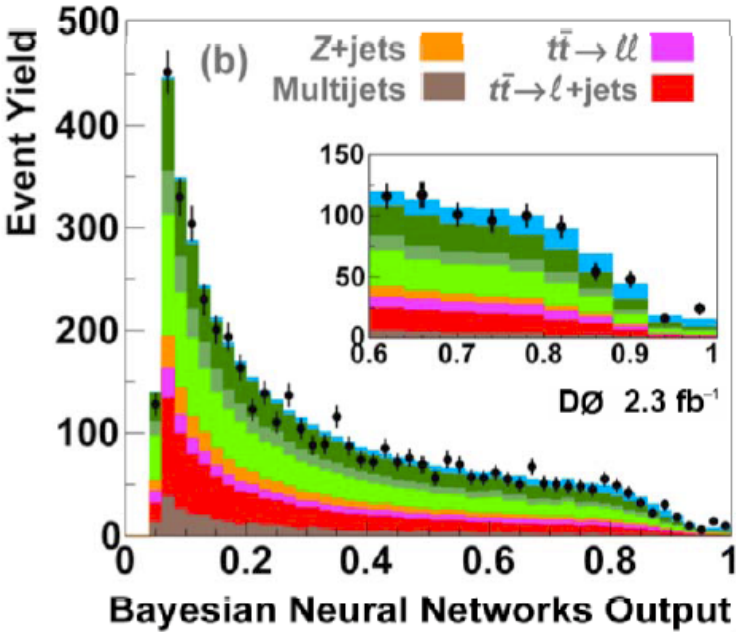


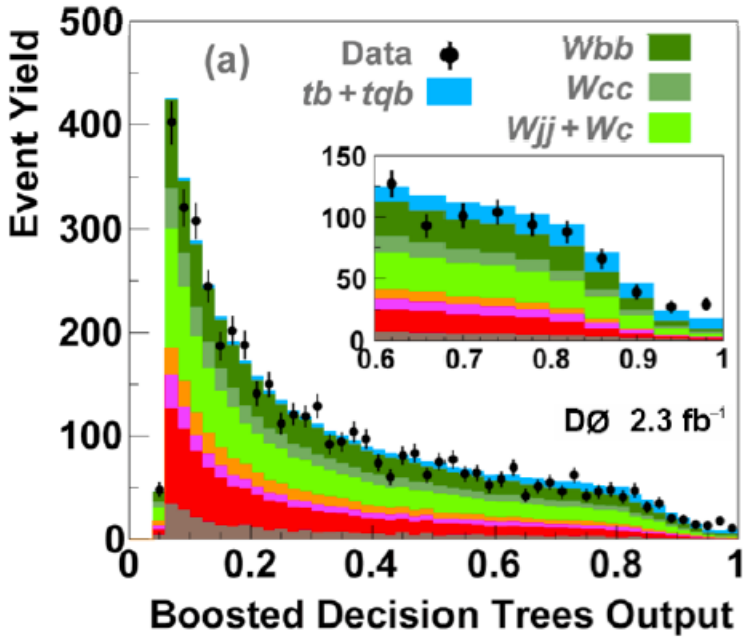
Use many inputs based on
jet properties, particle i.d., ...



signal
(blue +
green)

Different classifiers for single top





Also Naive Bayes and various approximations to likelihood ratio,....

Final combined result is statistically significant (>5σ level) but not easy to understand classifier outputs.