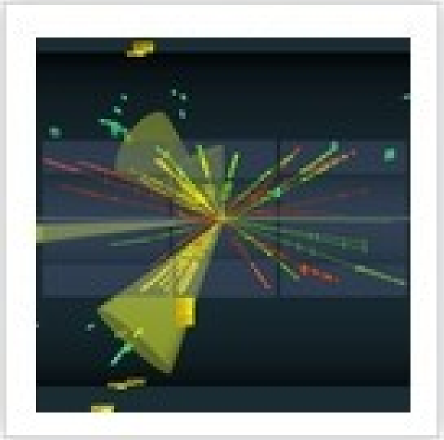
Statistical Methods for HEP
Lecture 2: Multivariate Methods



Taller de Altas Energías 2010

Universitat de Barcelona

1-11 September 2010

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

Outline

Lecture 1: Introduction and basic formalism
Probability, statistical tests, parameter estimation.

→ Lecture 2: Multivariate methods
General considerations
Example of a classifier: Boosted Decision Trees

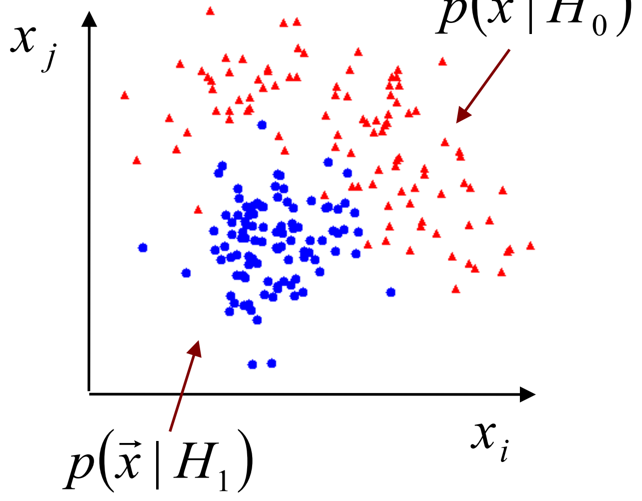
Lecture 3: Discovery and Exclusion limits
Quantifying significance and sensitivity
Systematic uncertainties (nuisance parameters)

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 $\mathrm{pp} \rightarrow \mathrm{t}\bar{\mathrm{t}}$, $\mathrm{pp} \rightarrow \tilde{\mathrm{g}}\tilde{\mathrm{g}}, \ldots$

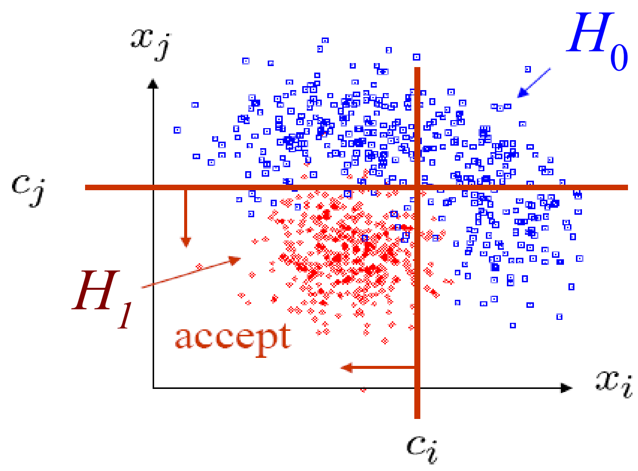


E.g. hypotheses H_0, H_1, ...
Often simply “signal”, “background”

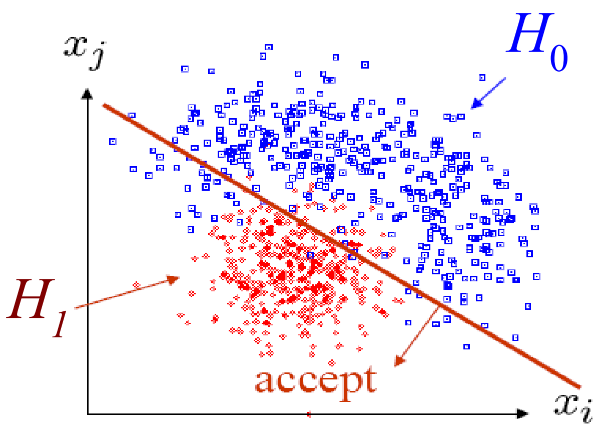
Finding an optimal decision boundary

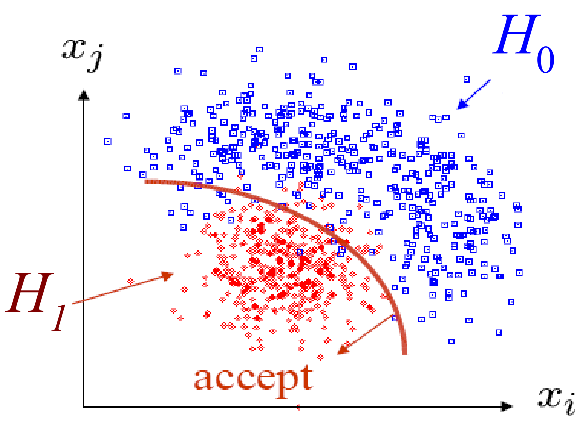
In particle physics usually start by making simple "cuts":

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



Maybe later try some other type of decision boundary:





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 for multivariate analysis

`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

Test statistics

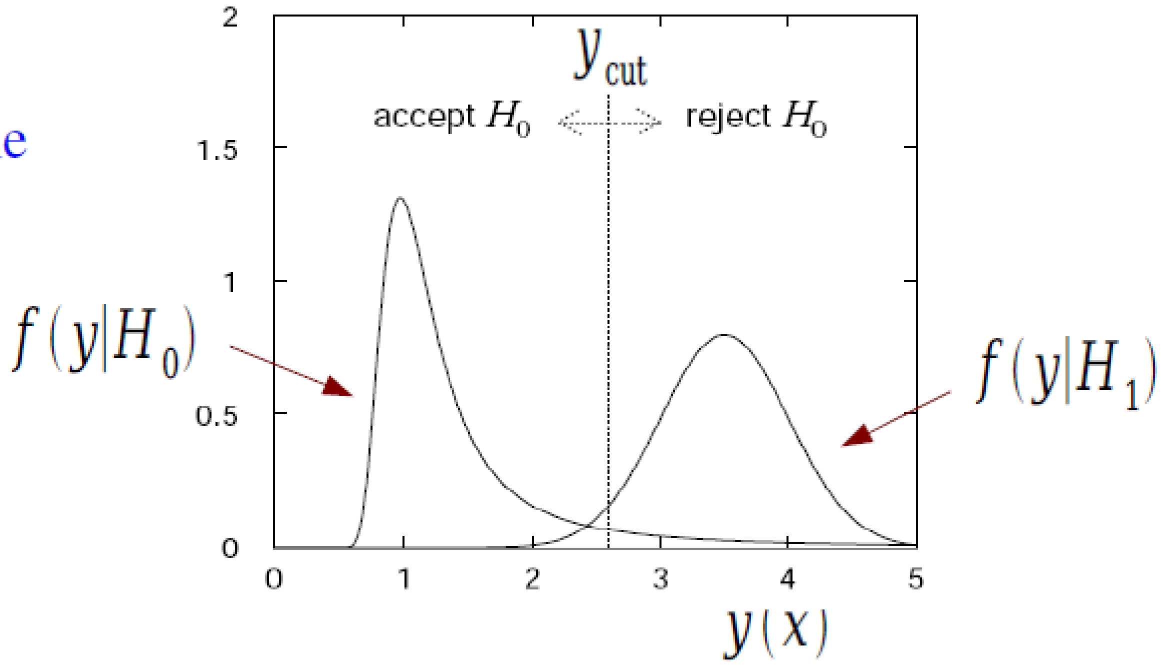
The decision boundary is a surface in the *n*-dimensional space of input variables, e.g., $y(\vec{x}) = \text{const}$.

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)



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.

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

$$\text{generate} \quad \vec{x} \sim p(\vec{x}|\mathrm{s}) \longrightarrow \vec{x}_1, \ldots, \vec{x}_{N_s}$$

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

"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.

Some “standard” multivariate methods

Place cuts on individual variables

Simple, intuitive, in general not optimal

Linear discriminant (e.g. Fisher)

Simple, optimal if the event types are Gaussian distributed with equal covariance, otherwise not optimal.

Probability Density Estimation based methods

Try to estimate $p(\mathbf{x}|\mathrm{s})$, $p(\mathbf{x}|\mathrm{b})$ then use $y(\vec{x}) = \hat{p}(\mathbf{x}|\mathrm{s})/\hat{p}(\mathbf{x}|\mathrm{b})$.

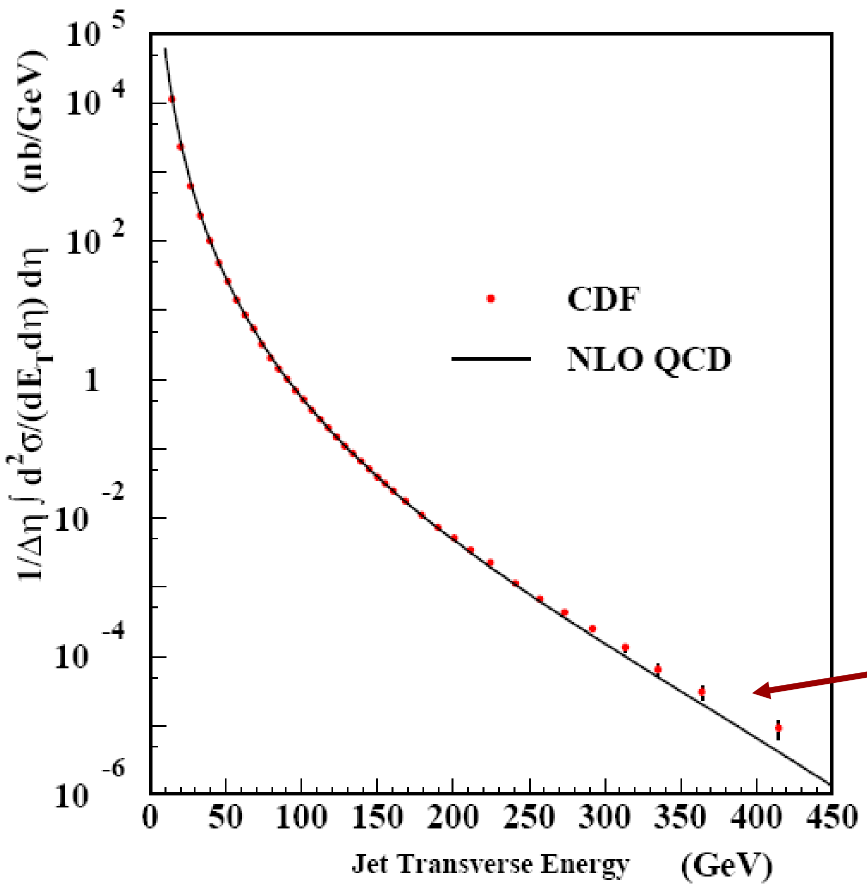
In principle best, difficult to estimate $p(\mathbf{x})$ for high dimension.

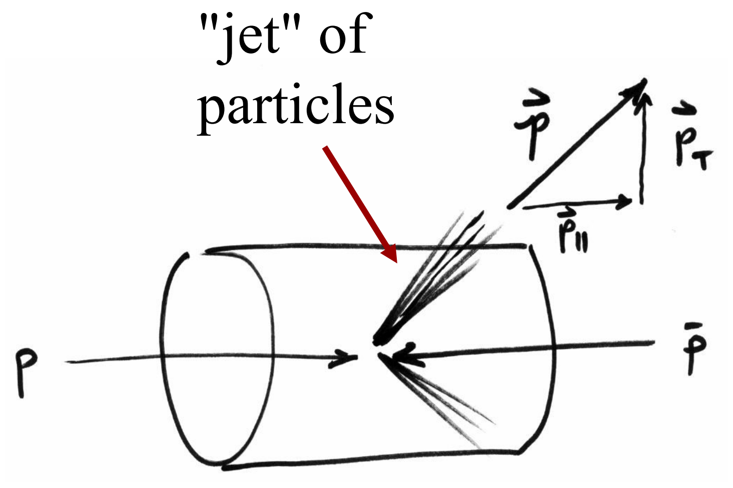
Neural networks

Can produce arbitrary decision boundary (in principle optimal), but can be difficult to train, result non-intuitive.

Example of a "cut-based" study

In the 1990s, the CDF experiment at Fermilab (Chicago) measured the number of hadron jets produced in proton-antiproton collisions as a function of their momentum perpendicular to the beam direction:

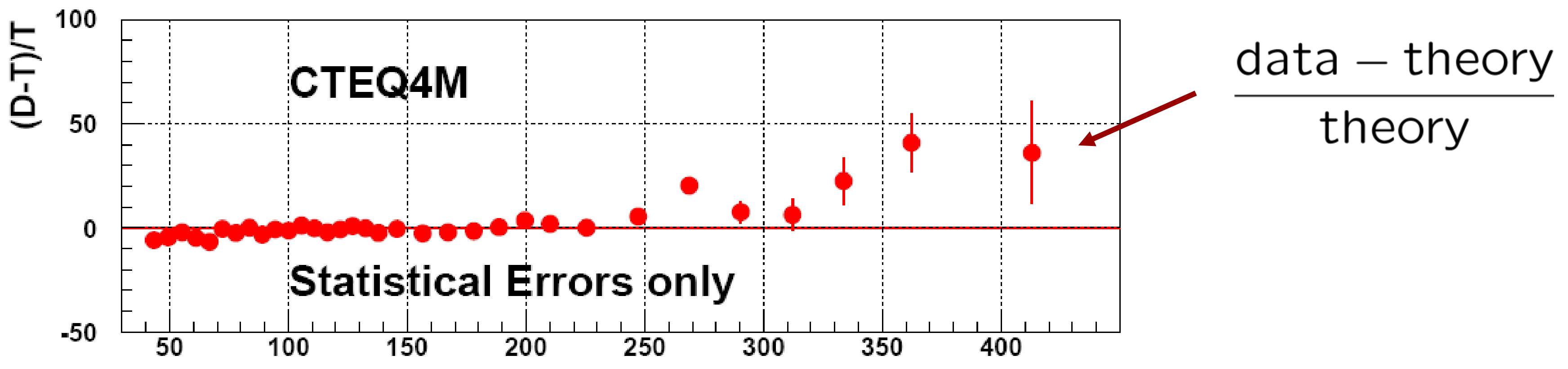




Prediction low relative to data for very high transverse momentum.

High p_T jets = quark substructure?

Although the data agree remarkably well with the Standard Model (QCD) prediction overall, the excess at high p_T appears significant:



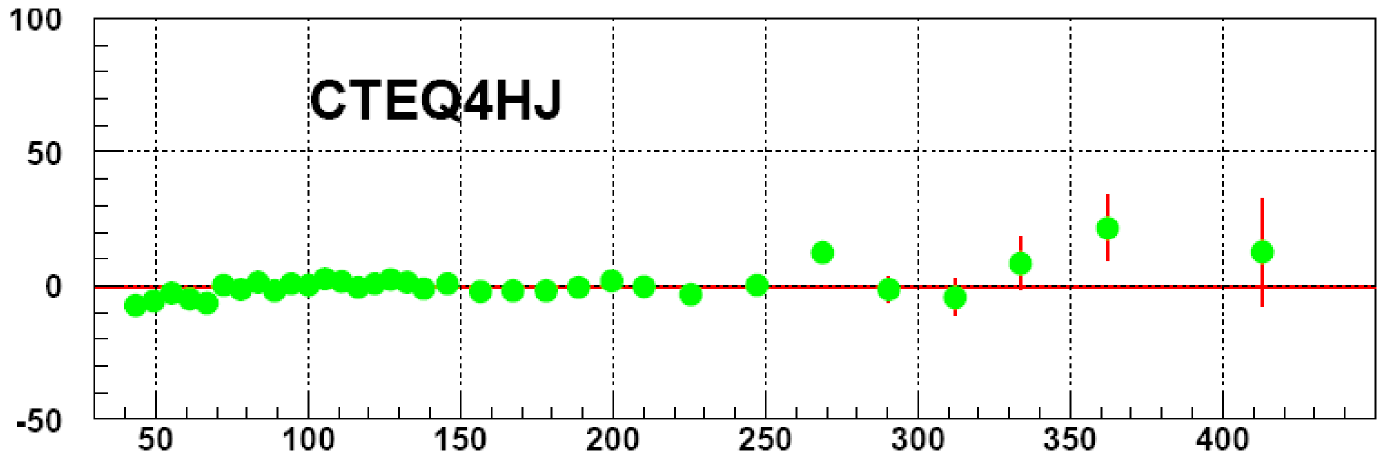
The fact that the variable is "understandable" leads directly to a plausible explanation for the discrepancy, namely, that quarks could possess an internal substructure.

Would not have been the case if the variable plotted was a complicated combination of many inputs.

High p_T jets from parton model uncertainty

Furthermore the physical understanding of the variable led one to a more plausible explanation, namely, an uncertain modeling of the quark (and gluon) momentum distributions inside the proton.

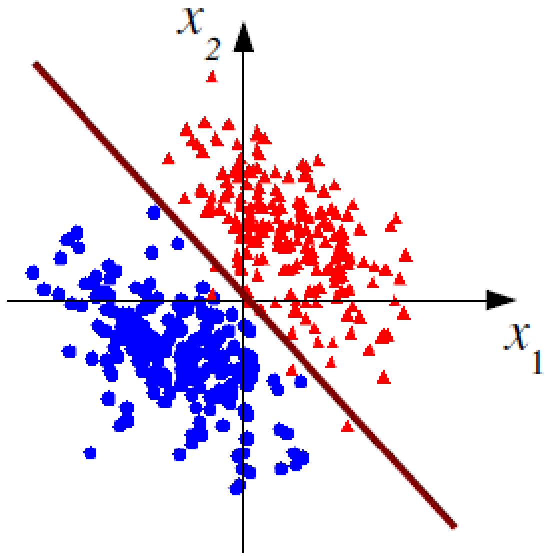
When model adjusted, discrepancy largely disappears:

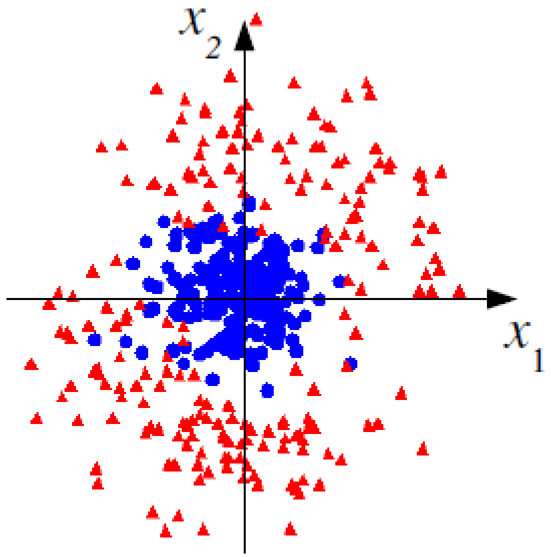


Can be regarded as a "success" of the cut-based approach. Physical understanding of output variable led to solution of apparent discrepancy.

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.

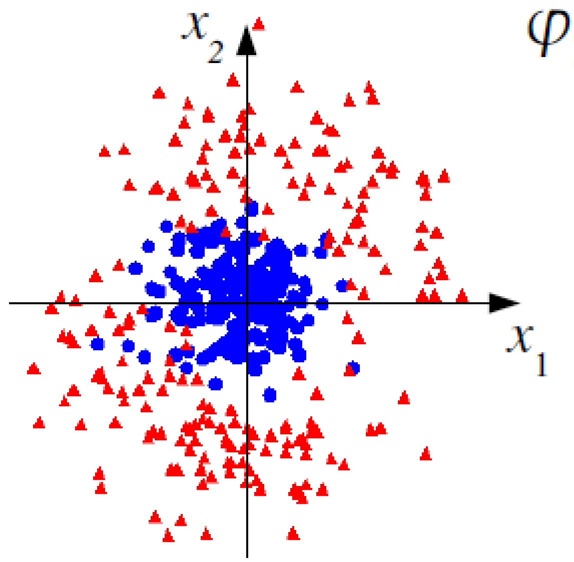
Nonlinear transformation of inputs

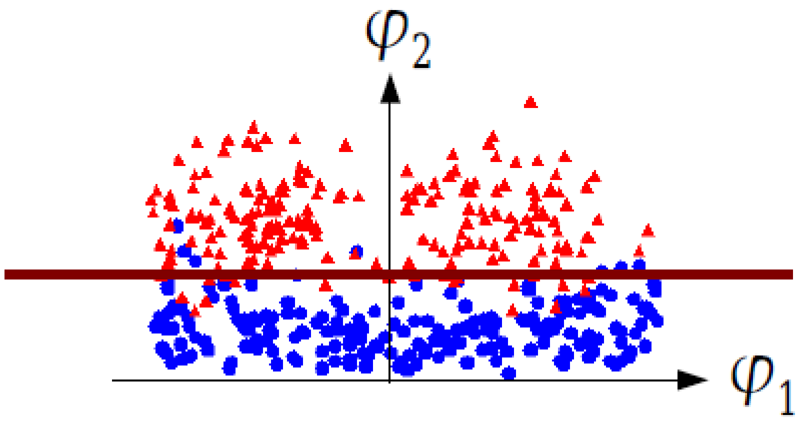
We can try to find a transformation, $x_1, \ldots, x_n \to \varphi_1(\vec{x}), \ldots, \varphi_m(\vec{x})$ so that the transformed "feature space" variables can be separated better by a linear boundary:

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

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

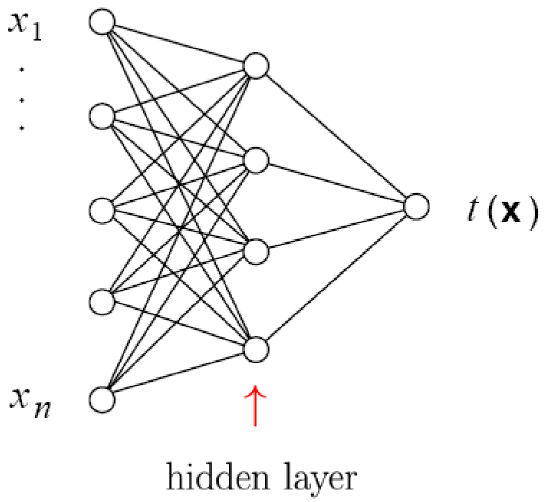
Here, guess fixed basis functions (no free parameters)





Neural networks in particle physics

For many years, the only "advanced" classifier used in particle physics.

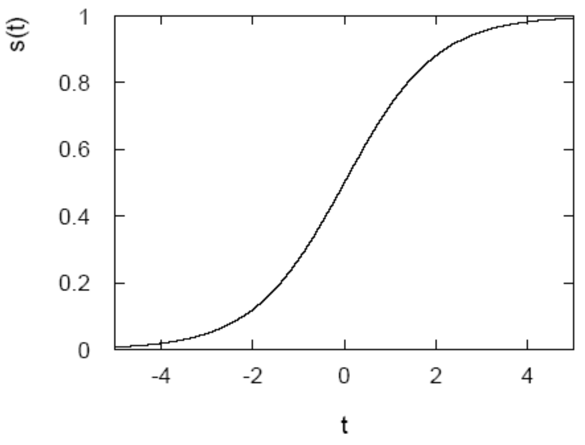


$$h_i(\vec{x}) = s\left(w_{i0} + \sum_{j=1}^{n} w_{ij} x_j\right) ,$$

$$t(\vec{x}) = s\left(a_0 + \sum_{i=1}^{n} a_i h_i(\vec{x})\right) .$$

Usually use single hidden layer,
logistic sigmoid activation function:

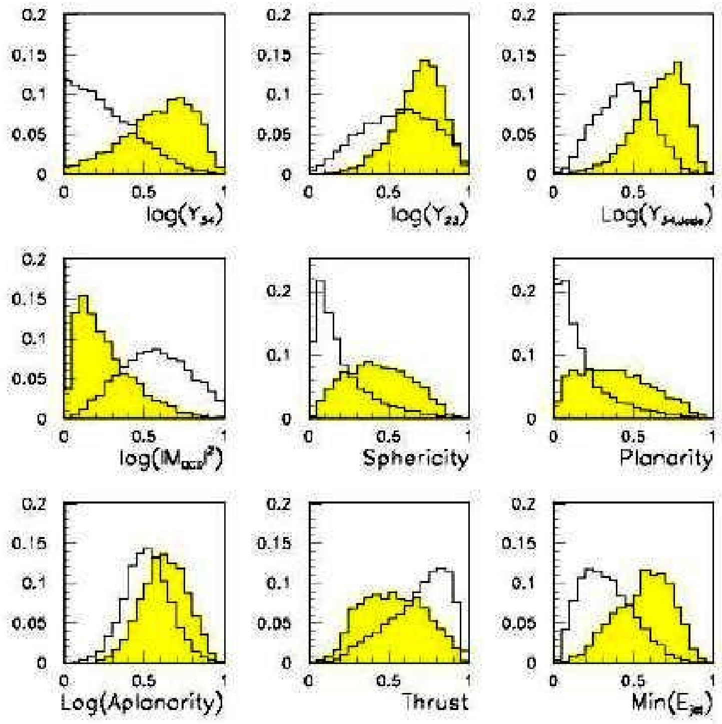
$$s(u) = (1 + e^{-u})^{-1}$$



Neural network example from LEP II

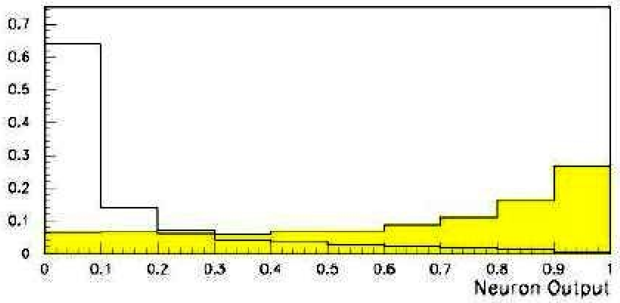
Signal: $e^+e^- \to W^+W^-$ (often 4 well separated hadron jets)

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)

Some issues with neural networks

In the example with WW events, goal was to select these events so as to study properties of the W boson.

> Needed to avoid using input variables correlated to the properties we eventually wanted to study (not trivial).

In principle a single hidden layer with an sufficiently large number of nodes can approximate arbitrarily well the optimal test variable (likelihood ratio).

> Usually start with relatively small number of nodes and increase until misclassification rate on validation data sample ceases to decrease.

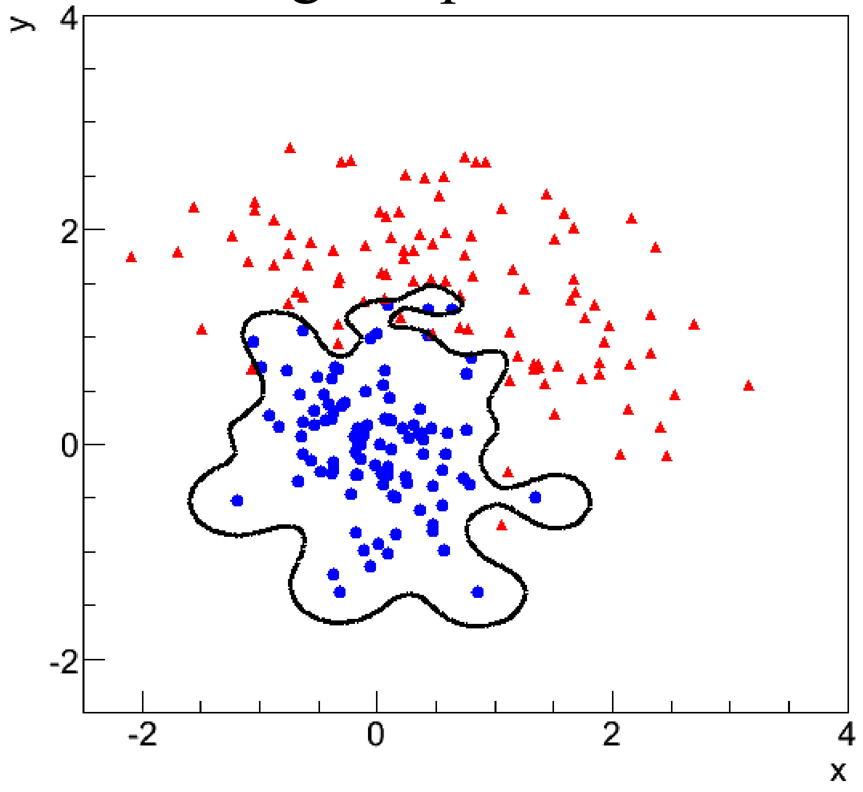
Often MC training data is cheap -- problems with getting stuck in local minima, overtraining, etc., less important than concerns of systematic differences between the training data and Nature, and concerns about the ease of interpretation of the output.

Overtraining

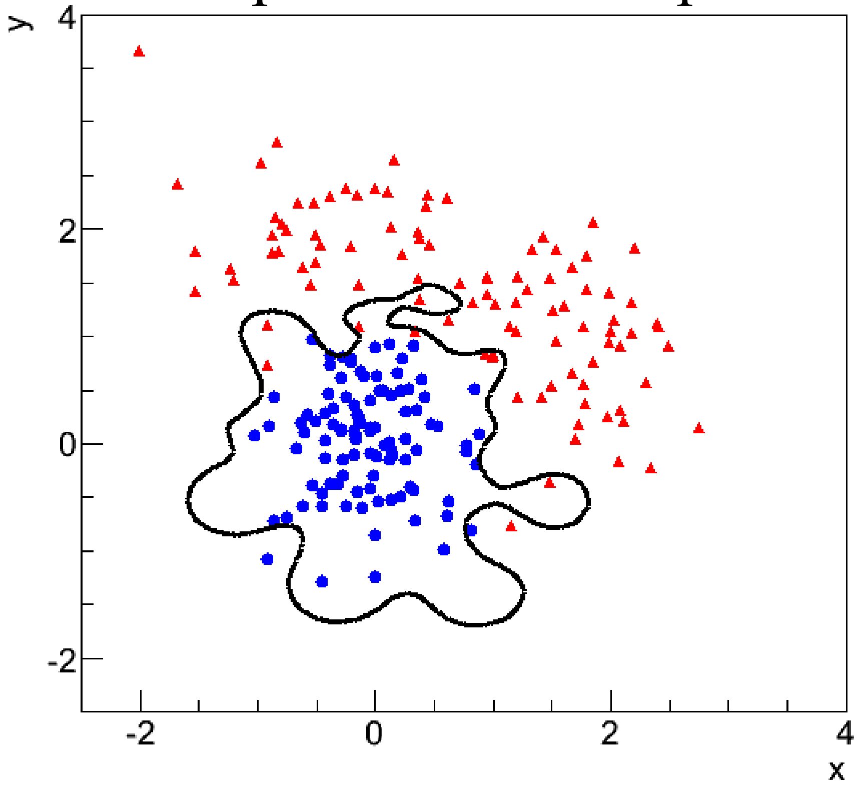
If decision boundary is too flexible it will conform too closely to the training points → overtraining.

Monitor by applying classifier to independent test sample.

training sample

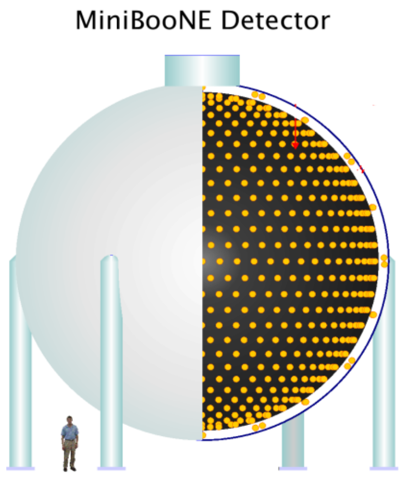


independent test sample

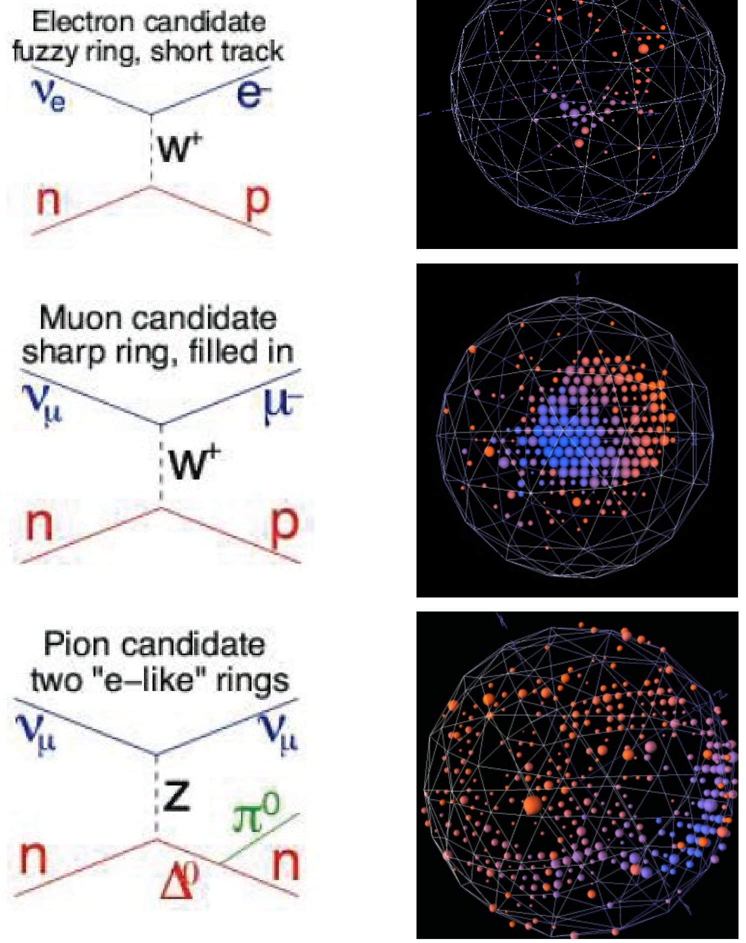


Particle i.d. in MiniBooNE

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:

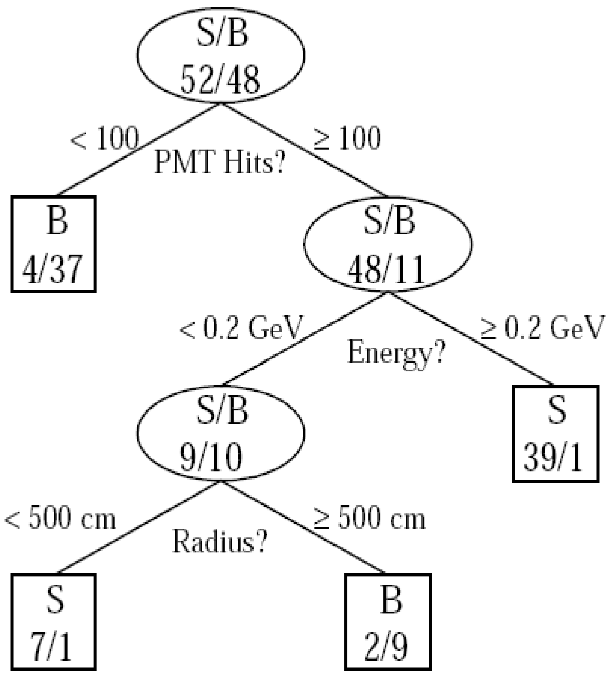
$$P = \frac{\sum_{\text{signal}} w_i}{\sum_{\text{signal}} w_i + \sum_{\text{background}} w_i}$$

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 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

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

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.

AdaBoost

A successful boosting algorithm is AdaBoost (Freund & Schapire, 1997).

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

Train the classifier $f_1(\boldsymbol{x})$ (e.g. a decision tree) using the weights $\boldsymbol{w}^{(1)}$

so as to minimize the classification error rate,

$$\varepsilon_1 = \sum_{i=1}^{N} w_i^{(1)} I(y_i f_1(\mathbf{x}_i) \leqslant 0) \,,$$

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

Updating the event weights (AdaBoost)

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

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

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

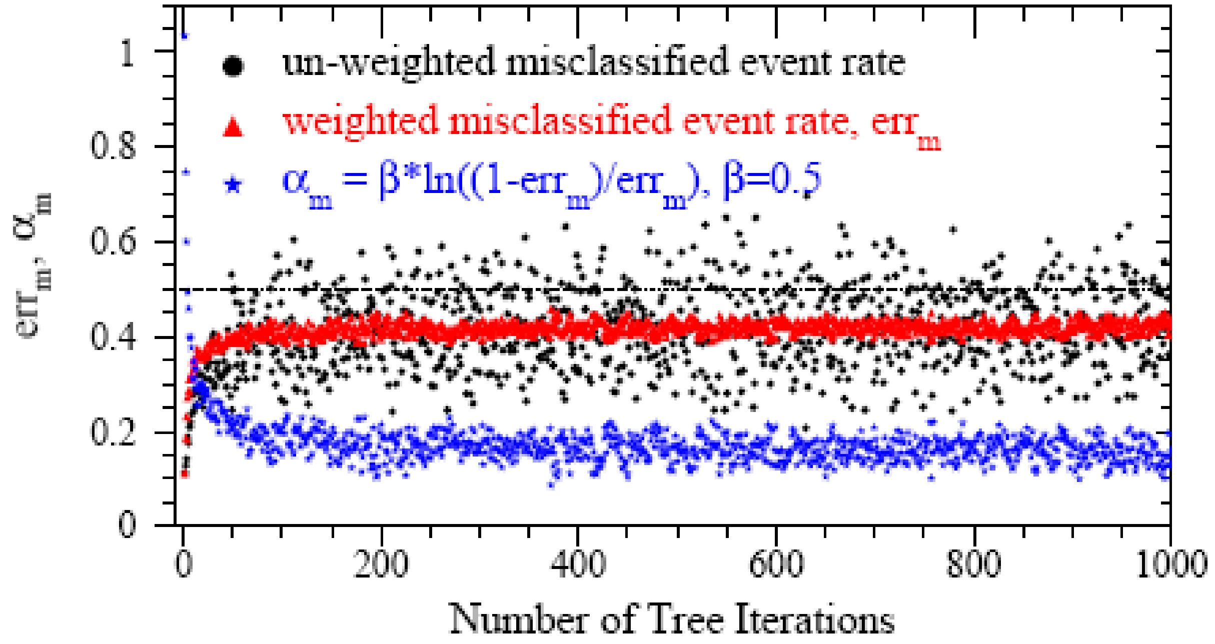
Normalize so that $\sum_i w_i^{(k+1)} = 1$

Iterate *K* times, final classifier is $y(\mathbf{x}) = \sum_{k=1}^{K} \alpha_k f_k(\mathbf{x}, T_k)$

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

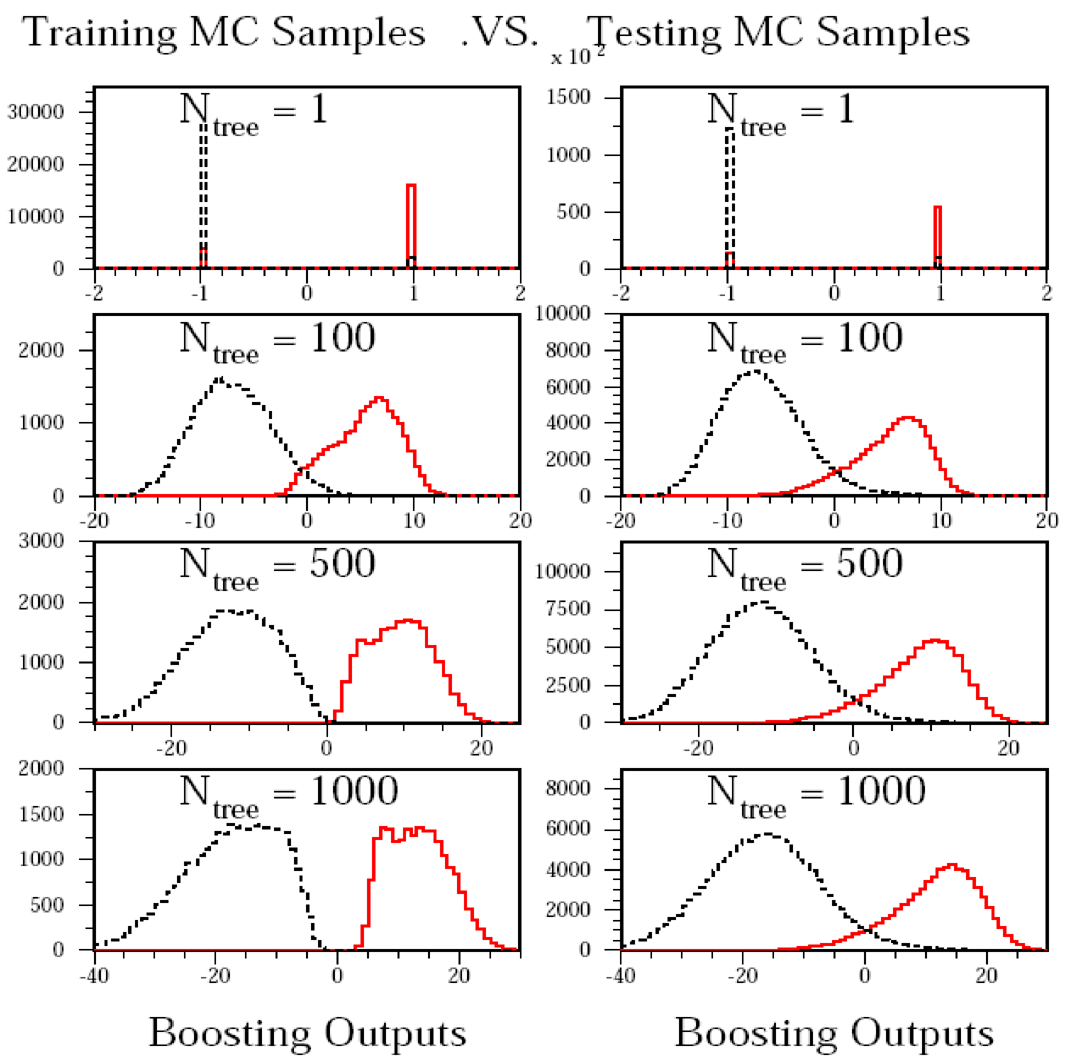


B. Roe et al., NIM 543 (2005) 577

Monitoring overtraining

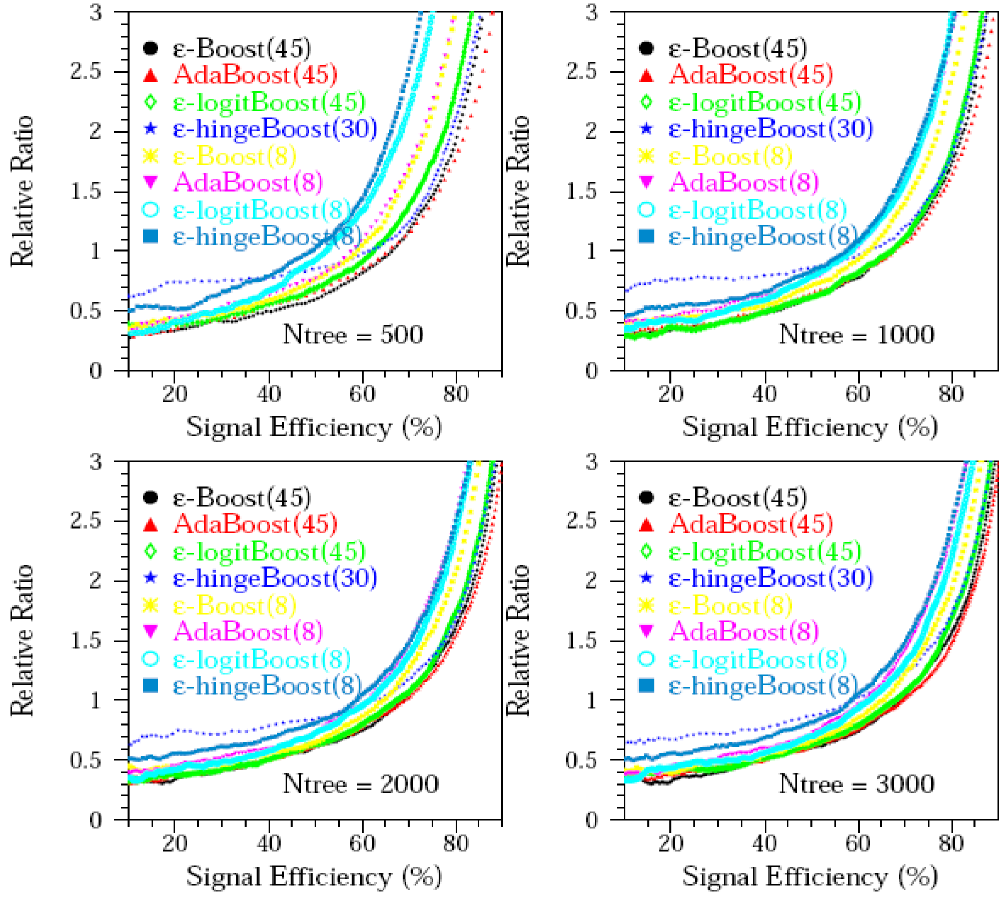
From MiniBooNE example:

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...).

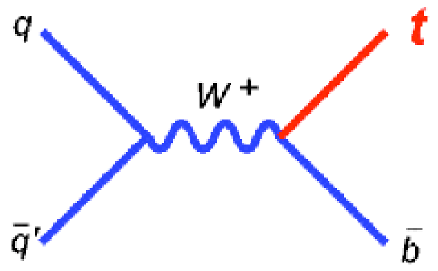
If a tree has only a few leaves it is easy to visualize (but rarely use only a single tree).

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

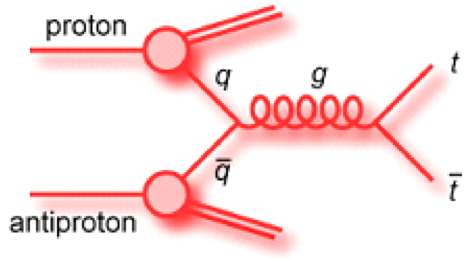
Other ways of combining weaker classifiers: Bagging (Boostrap-Aggregating), generates the ensemble of classifiers by random sampling with replacement from the full training sample.

Single top quark production (CDF/D0)

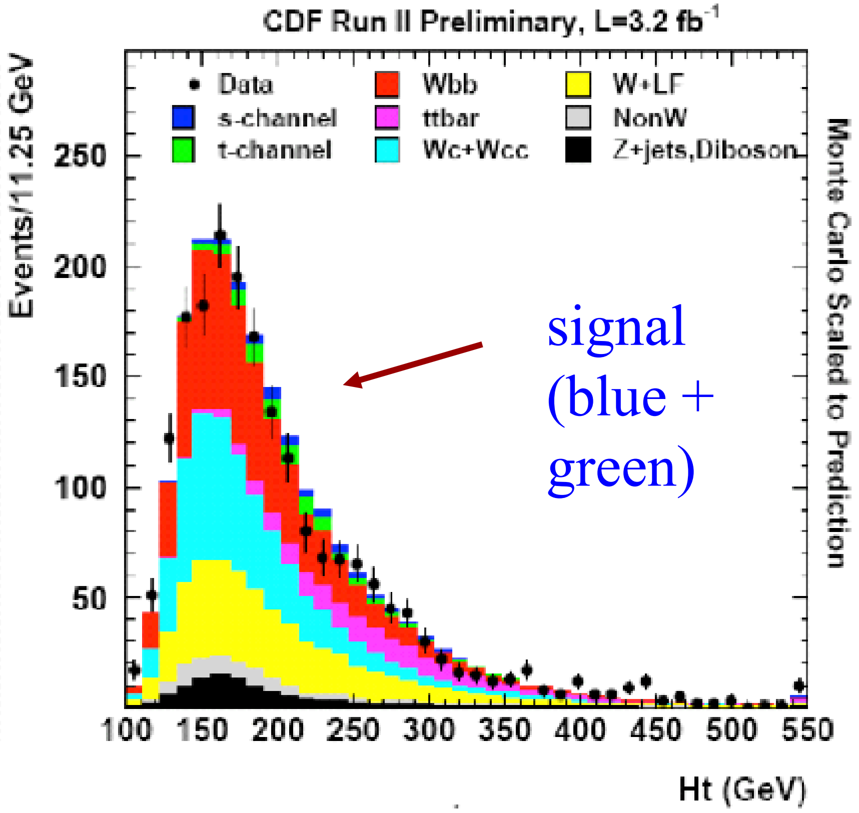
Top quark discovered in pairs, but SM predicts single top production.



Pair-produced tops are now a background process.

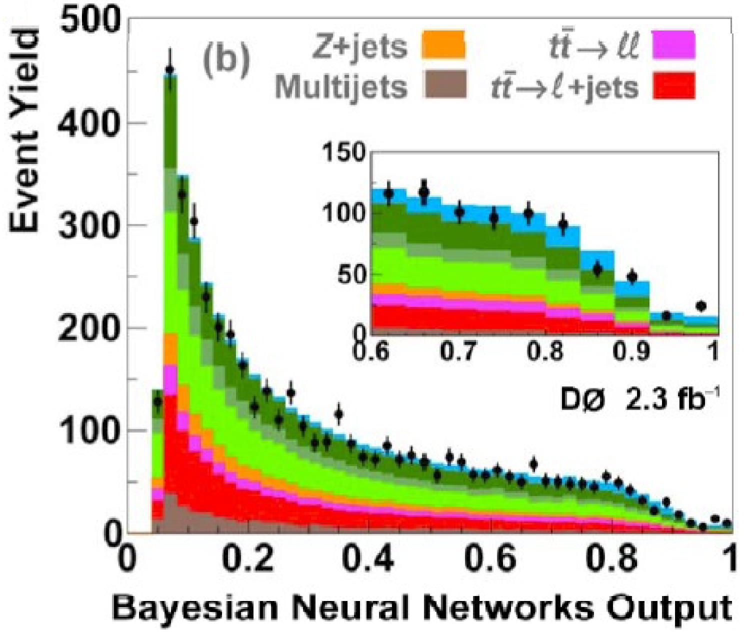


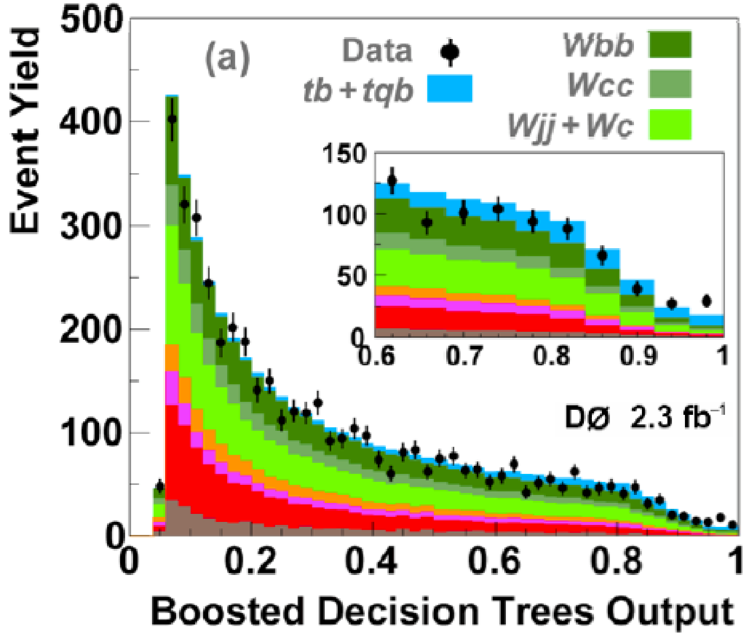
Use many inputs based on jet properties, particle i.d., ...



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.

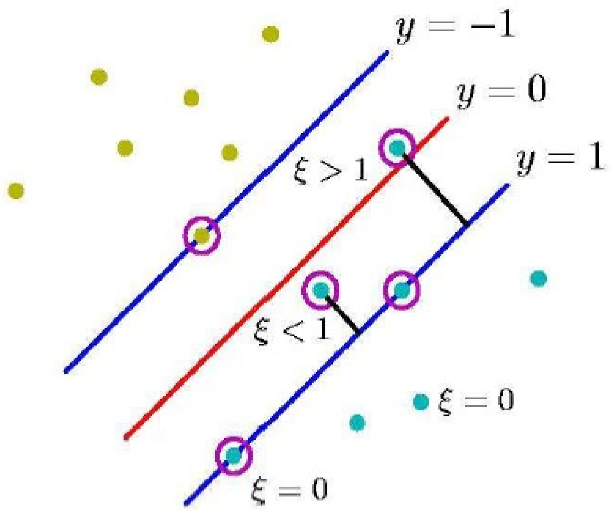
Support Vector Machines

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 in `www.pp.rhul.ac.uk/~cowan/mainz_lectures.html`
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)

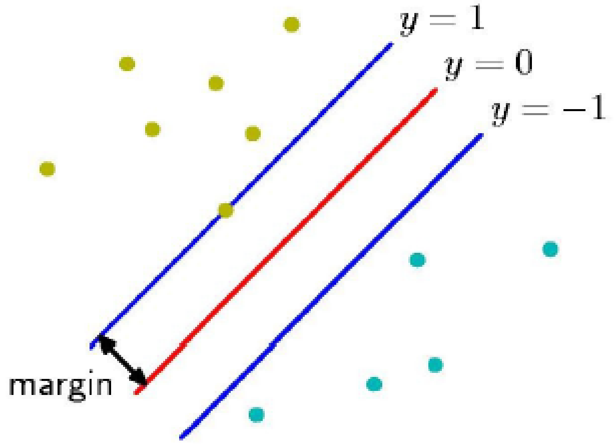
Suppose the classes can be separated by a hyperplane defined by a normal vector $\boldsymbol{w}$ and scalar offset b (the "bias"). We have

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

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

or equivalently

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

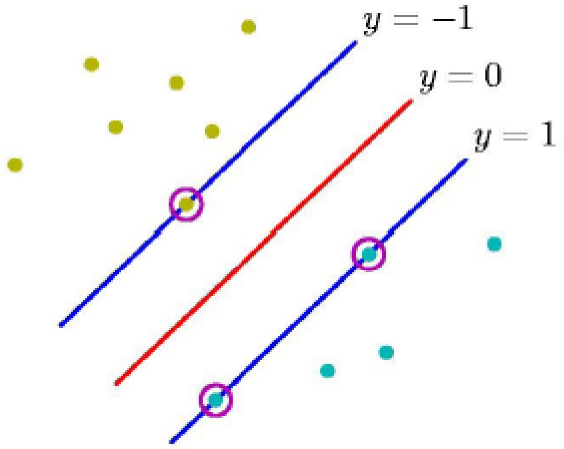


Bishop Ch. 7

Margin and support vectors

The distance between the hyperplanes defined by $y(\boldsymbol{x}) = \boldsymbol{x}\cdot\boldsymbol{w} + b = +1$ 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

$$f(\mathbf{x}) = \mathrm{sign}(\mathbf{x} \cdot \mathbf{w} + b)$$

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

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

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

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

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

Lagrangian formulation

This constrained minimization problem can be reformulated using a Lagrangian

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

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

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x_i}$$

(sum only contains support vectors)

Dual formulation

The classifier function is thus

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

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

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

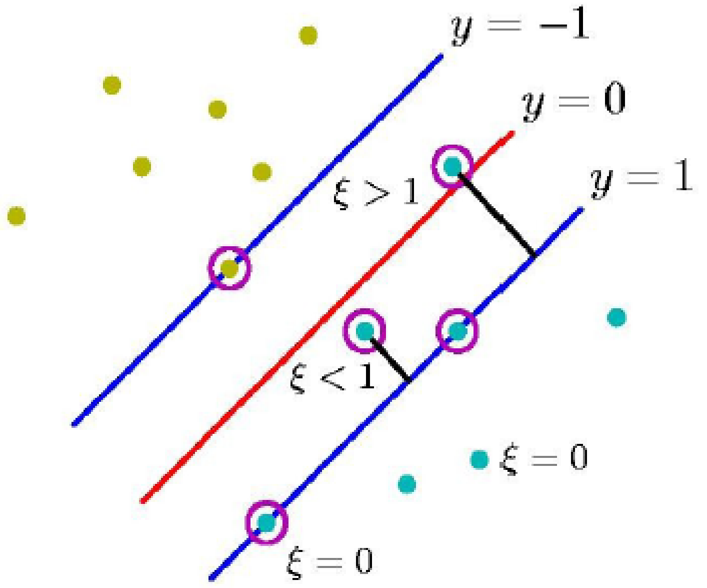
So this means that both the classifier function and the Lagrangian only involve dot products of vectors in the input variable space.

Nonseparable data

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

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

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



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

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

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

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

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

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

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

so in the feature space this becomes

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

The Kernel trick

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

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

Functions having this property must satisfy Mercer's condition

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

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

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

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

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

$$K(\mathbf{x},\mathbf{x}')=\tanh(\kappa(\mathbf{x}\cdot\mathbf{x}')+\theta)$$ 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)
a cost parameter C (plays role of regularization parameter)

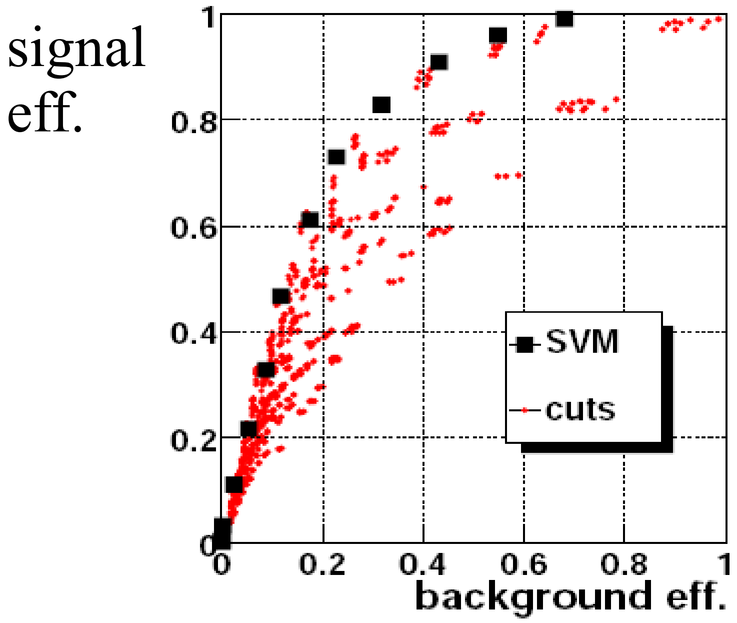
The training is relatively straightforward because, in contrast to neural networks, the function to be minimized has a single global minimum.

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

The advantages/disadvantages and rationale behind the choices above is not always clear to the particle physicist -- help needed here.

SVM in particle physics

SVMs are very popular in the Machine Learning community but have yet to find wide application in HEP. Here is an early example from a CDF top quark anlaysis (A. Vaiciulis, contribution to PHYSTAT02).



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.

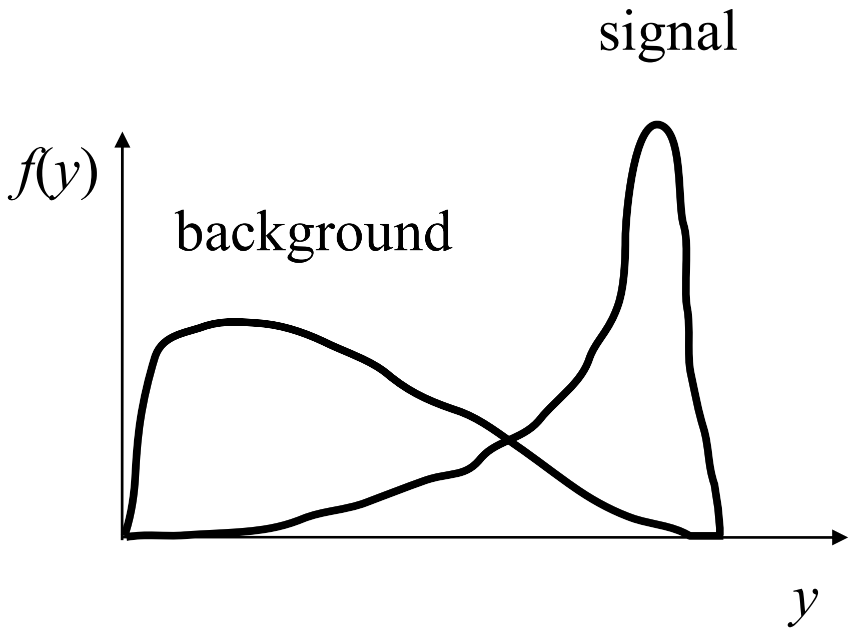
Quotes I like

"Keep it simple.
As simple as possible.
Not any simpler."
– A. Einstein

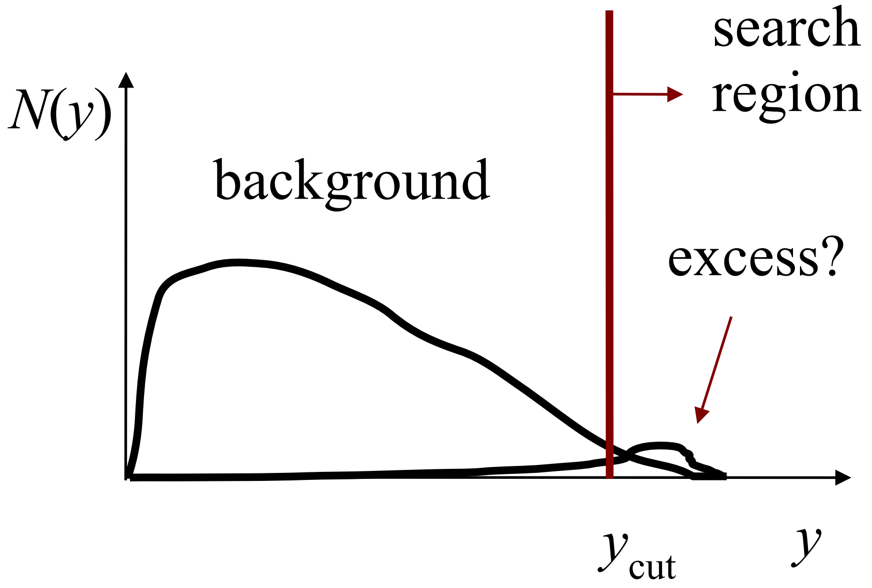
"If you believe in something
you don't understand, you suffer,..."
– Stevie Wonder

Extra slides

Using classifier output for discovery



Normalized to unity



Normalized to expected number of events

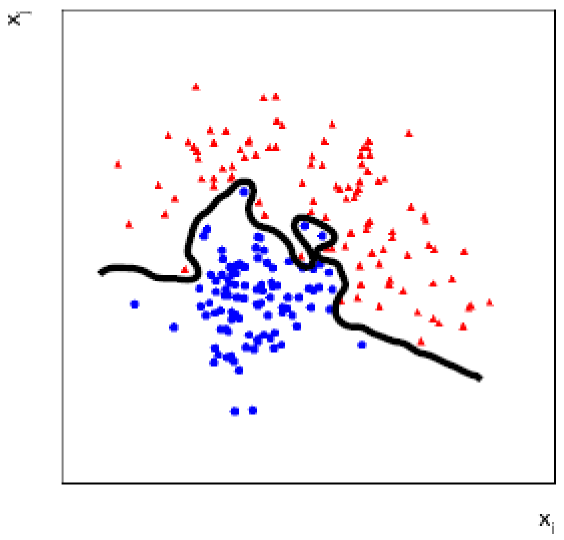
Discovery = number of events found in search region incompatible with background-only hypothesis.

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

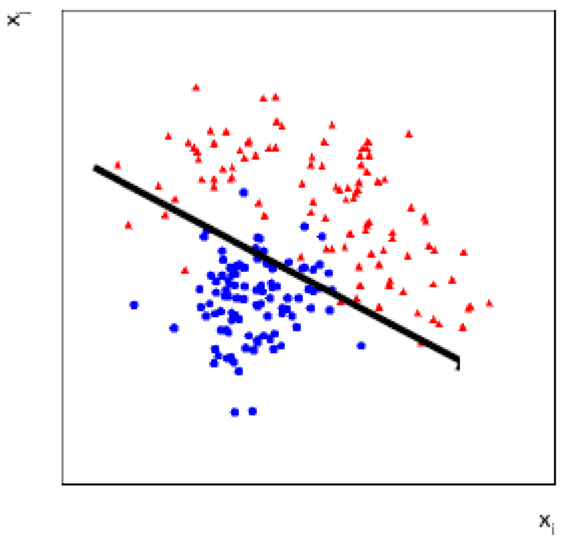
Decision boundary flexibility

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.

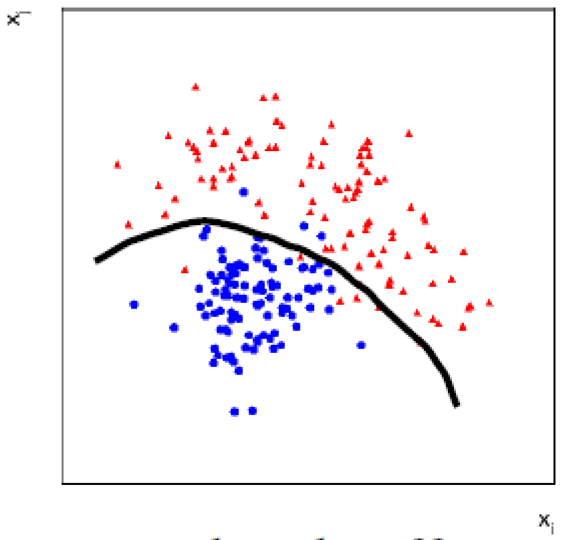
Goal is to determine the boundary using a finite amount of training data so as to best separate between the event types for an unseen data sample.



overtraining



boundary too rigid



good trade-off