Topics in Statistical Data Analysis for HEP
Lecture 2: Multivariate Methods

CERN-JINR European School

of High Energy Physics

Bautzen, 14–27 June 2009

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

Outline

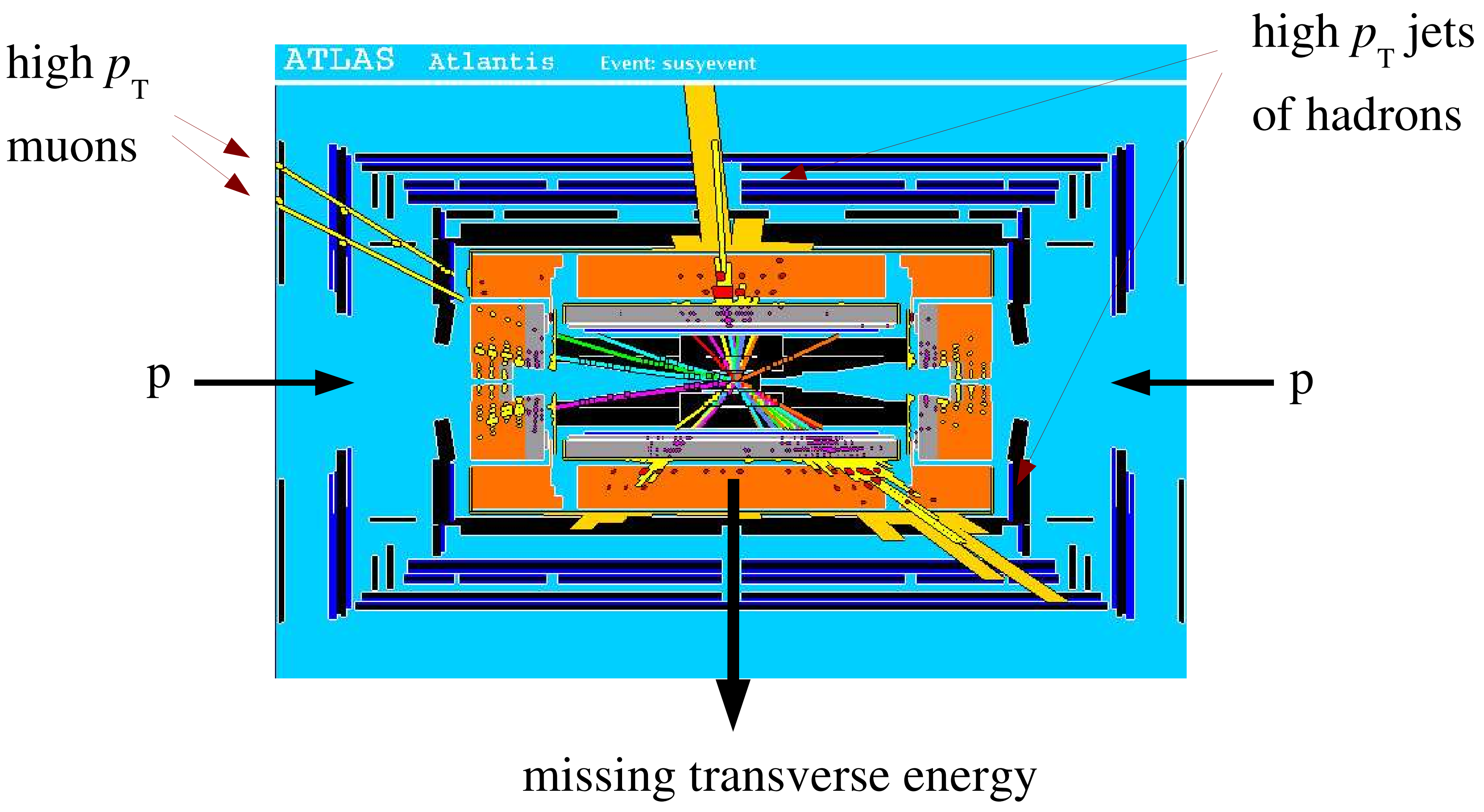
Lecture #1: An introduction to Bayesian statistical methods

- Role of probability in data analysis (Frequentist, Bayesian)
- A simple fitting problem : Frequentist vs. Bayesian solution
- Bayesian computation, Markov Chain Monte Carlo
- Setting limits / making a discovery

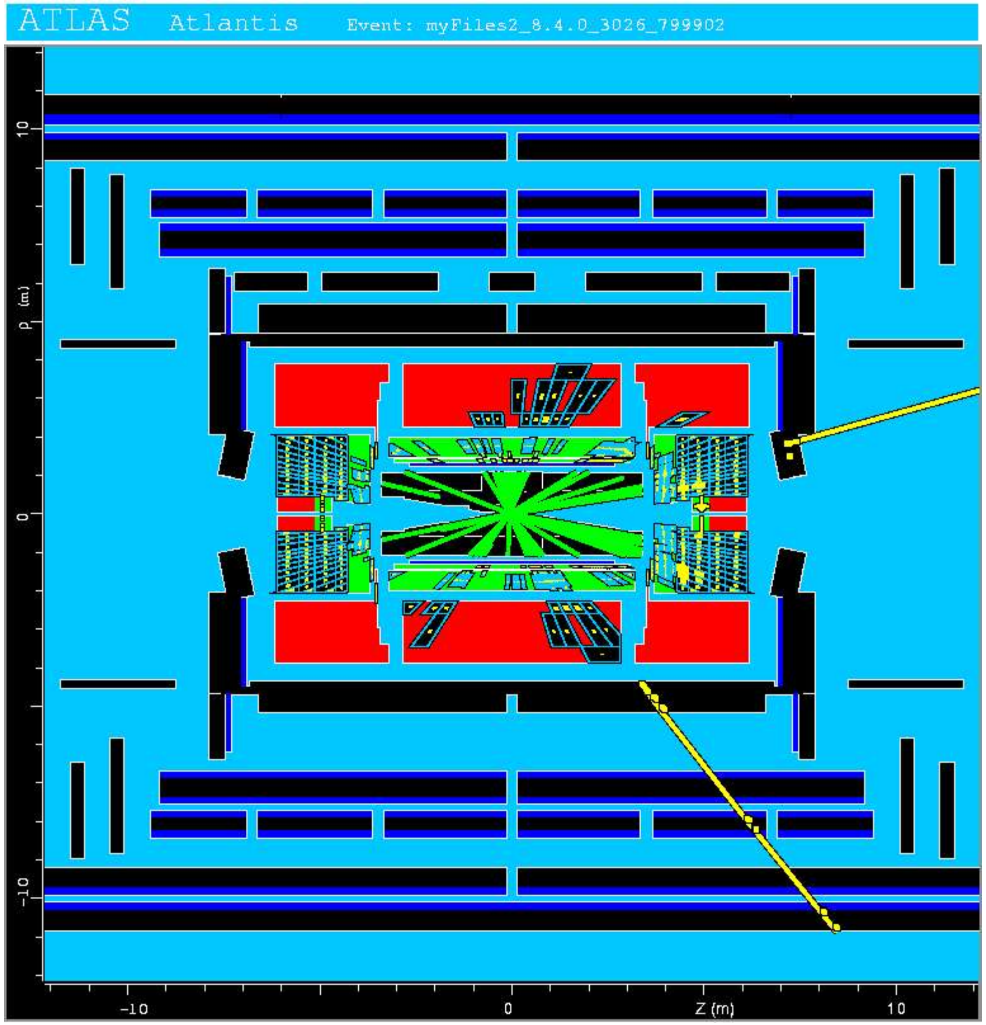
Lecture #2: Multivariate methods for HEP

- Event selection as a statistical test
- Neyman-Pearson lemma and likelihood ratio test
- Some multivariate classifiers:
  - Boosted Decision Trees
  - Support Vector Machines

A simulated SUSY event in ATLAS



Background events



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

→ can easily mimic a SUSY event.

LHC data

At LHC, ~10^9 pp collision events per second, mostly uninteresting

do quick sifting, record ~200 events/sec

single event ~ 1 Mbyte

1 "year" ≈ 10^7 s, 10^{16} pp collisions / year

2×10^9 events recorded / year (~2 Pbyte / year)

For new/rare processes, rates at LHC can be vanishingly small

e.g. Higgs bosons detectable per year could be ~10^3

→ 'needle in a haystack'

For Standard Model and (many) non-SM processes we can generate simulated data with Monte Carlo programs (including simulation of the detector).

A simulated event

PYTHIA Monte Carlo

pp → gluino-gluino

Event listing (summary)

I	particle/jet	KS	KF	orig	p_x	p_y	p_z	E	m
1	!p+!	21	2212	0	0.000	0.000	7000.000	7000.000	0.938
2	!p+!	21	2212	0	0.000	0.000	-7000.000	7000.000	0.938
3	!g!	21	21	1	0.863	-0.323	1739.862	1739.862	
4	!ubar!	21	-2	2	-0.621	-0.163	-777.415	777.415	
5	!g!	21	21	3	-2.427	5.486	1487.857	1487.869	
6	!g!	21	21	4	-62.910	63.357	-463.274	471.799	
7	!~g!	21	1000021	0	314.363	544.843	498.897	979.192	
8	!~g!	21	1000021	0	-379.700	-476.000	525.686	980.477	
9	!~chi_1-!	21	-1000024	7	130.058	112.247	129.860	263.141	
10	!sbar!	21	-3	7	259.400	187.468	83.100	330.664	
11	!c!	21	4	7	-79.403	242.409	283.026	381.016	
12	!~chi_20!	21	1000023	8	-326.241	-80.971	113.712	385.931	
13	!b!	21	5	8	-51.841	-294.077	389.853	491.098	
14	!bbar!	21	-5	8	-0.597	-99.577	21.299	101.944	
15	!~chi_10!	21	1000022	9	103.352	81.316	83.457	175.000	
16	!s!	21	3	9	5.451	38.374	52.302	65.100	
17	!cbar!	21	-4	9	20.839	-7.250	-5.938	22.899	
18	!~chi_10!	21	1000022	12	-136.266	-72.961	53.246	181.914	
19	!nu_mu!	21	14	12	-78.263	-24.757	21.719	84.910	
20	!nu_mubar!	21	-14	12	-107.801	16.901	38.226	115.620	
21	gamma	1	22	4	2.636	1.357	0.125	2.967	
22	(~chi_1-)	11	-1000024	9	129.643	112.440	129.820	262.999	
23	(~chi_20)	11	1000023	12	-322.330	-80.817	113.191	382.444	
24	~chi_10	1	1000022	15	97.944	77.819	80.917	169.004	
25	~chi_10	1	1000022	18	-136.266	-72.961	53.246	181.914	
26	nu_mu	1	14	19	-78.263	-24.757	21.719	84.910	
27	nu_mubar	1	-14	20	-107.801	16.901	38.226	115.620	
28	(Delta++)	11	2224	2	0.222	0.012	-2734.287	2734.287	

397	pi+	1	211	209	0.006	0.398	-308.296	308.297	0.140
398	gamma	1	22	211	0.407	0.087	-1695.458	1695.458	0.000
399	gamma	1	22	211	0.113	-0.029	-314.822	314.822	0.000
400	(pi0)	11	111	212	0.021	0.122	-103.709	103.709	0.135
401	(pi0)	11	111	212	0.084	-0.068	-94.276	94.276	0.135
402	(pi0)	11	111	212	0.267	-0.052	-144.673	144.674	0.135
403	gamma	1	22	215	-1.581	2.473	3.306	4.421	0.000
404	gamma	1	22	215	-1.494	2.143	3.051	4.016	0.000
405	pi-	1	-211	216	0.007	0.738	4.015	4.085	0.140
406	pi+	1	211	216	-0.024	0.293	0.486	0.585	0.140
407	K+	1	321	218	4.382	-1.412	-1.799	4.968	0.494
408	pi-	1	-211	218	1.183	-0.894	-0.176	1.500	0.140
409	(pi0)	11	111	218	0.955	-0.459	-0.590	1.221	0.135
410	(pi0)	11	111	218	2.349	-1.105	-1.181	2.855	0.135
411	(Kbar0)	11	-311	219	1.441	-0.247	-0.472	1.615	0.498
412	pi-	1	-211	219	2.232	-0.400	-0.249	2.285	0.140
413	K+	1	321	220	1.380	-0.652	-0.361	1.644	0.494
414	(pi0)	11	111	220	1.078	-0.265	0.175	1.132	0.135
415	(K_S0)	11	310	222	1.841	0.111	0.894	2.109	0.498
416	K+	1	321	223	0.307	0.107	0.252	0.642	0.494
417	pi-	1	-211	223	0.266	0.316	-0.201	0.480	0.140
418	nbar0	1	-2112	226	1.335	1.641	2.078	3.111	0.940
419	(pi0)	11	111	226	0.899	1.046	1.311	1.908	0.135
420	pi+	1	211	227	0.217	1.407	1.356	1.971	0.140
421	(pi0)	11	111	227	1.207	2.336	2.767	3.820	0.135
422	n0	1	2112	228	3.475	5.324	5.702	8.592	0.940
423	pi-	1	-211	228	1.856	2.606	2.808	4.259	0.140
424	gamma	1	22	229	-0.012	0.247	0.421	0.489	0.000
425	gamma	1	22	229	0.025	0.034	0.009	0.043	0.000
426	pi+	1	211	230	2.718	5.229	6.403	8.703	0.140
427	(pi0)	11	111	230	4.109	6.747	7.597	10.961	0.135
428	pi-	1	-211	231	0.551	1.233	1.945	2.372	0.140
429	(pi0)	11	111	231	0.645	1.141	0.922	1.608	0.135
430	gamma	1	22	232	-0.383	1.169	1.208	1.724	0.000
431	gamma	1	22	232	-0.201	0.070	0.060	0.221	0.000

•

•

•

Multivariate event selection

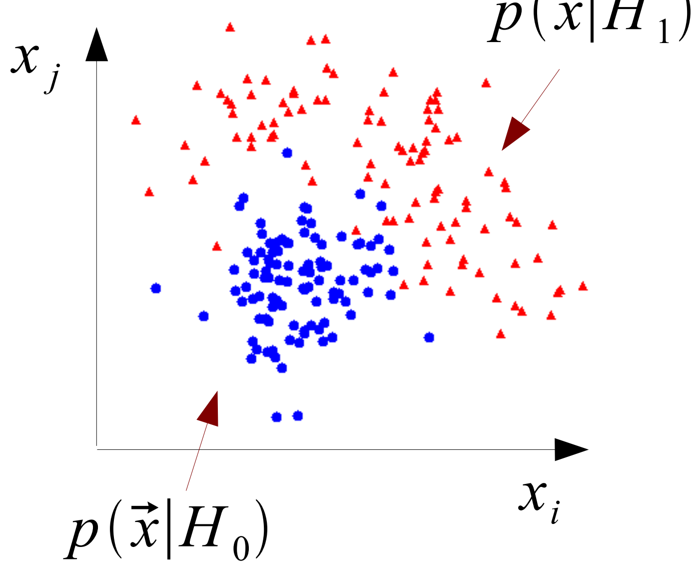
Suppose for each event we measure a set of numbers $\vec{x} = (x_1, \ldots, x_n)$

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 $pp \to t\bar{t}$, $pp \to \tilde{g}\tilde{g}, \ldots$



E.g. hypotheses (class labels) H_0, H_1, ...

Often simply "signal", "background"

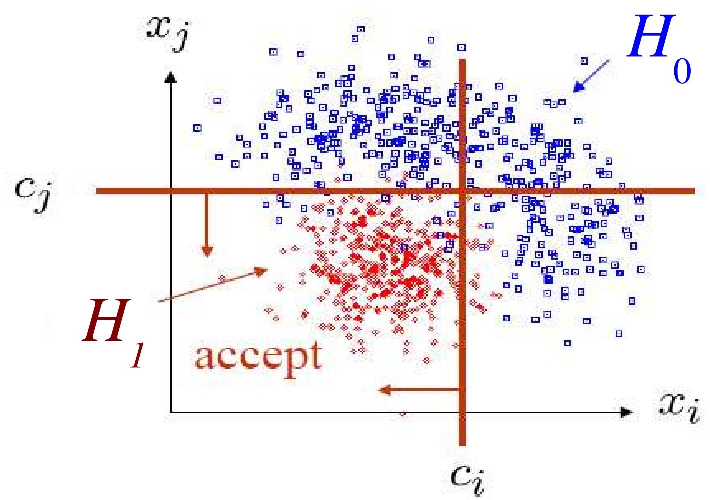
We want to separate (classify) the event types in a way that exploits the information carried in many variables.

Finding an optimal decision boundary

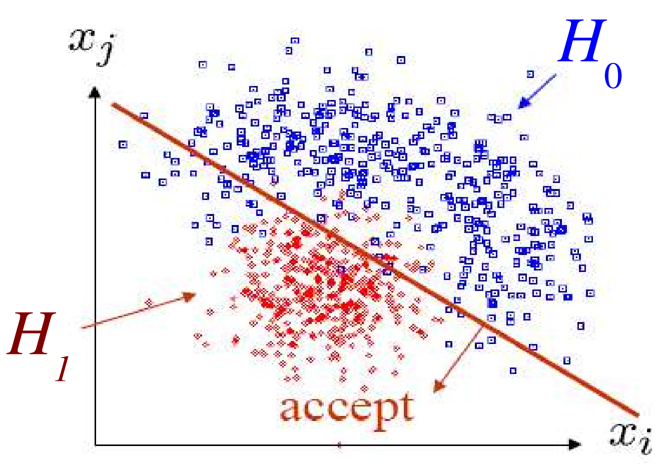
Maybe select events with "cuts":

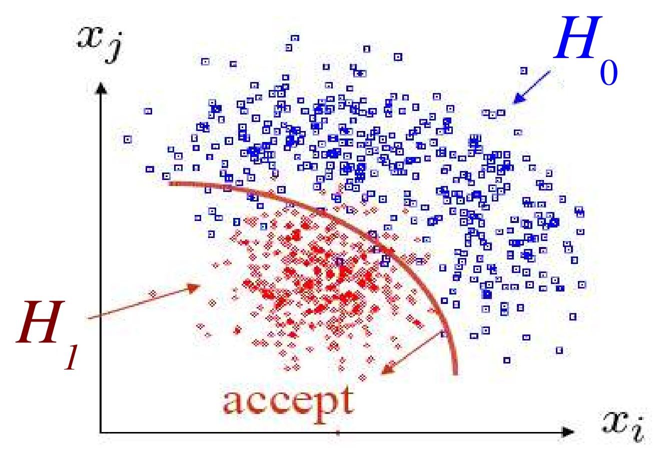
$x_i < c_i$

$x_j < c_j$



Or maybe use some other type of decision boundary:





Goal of multivariate analysis is to do this in an "optimal" way.

Test statistics

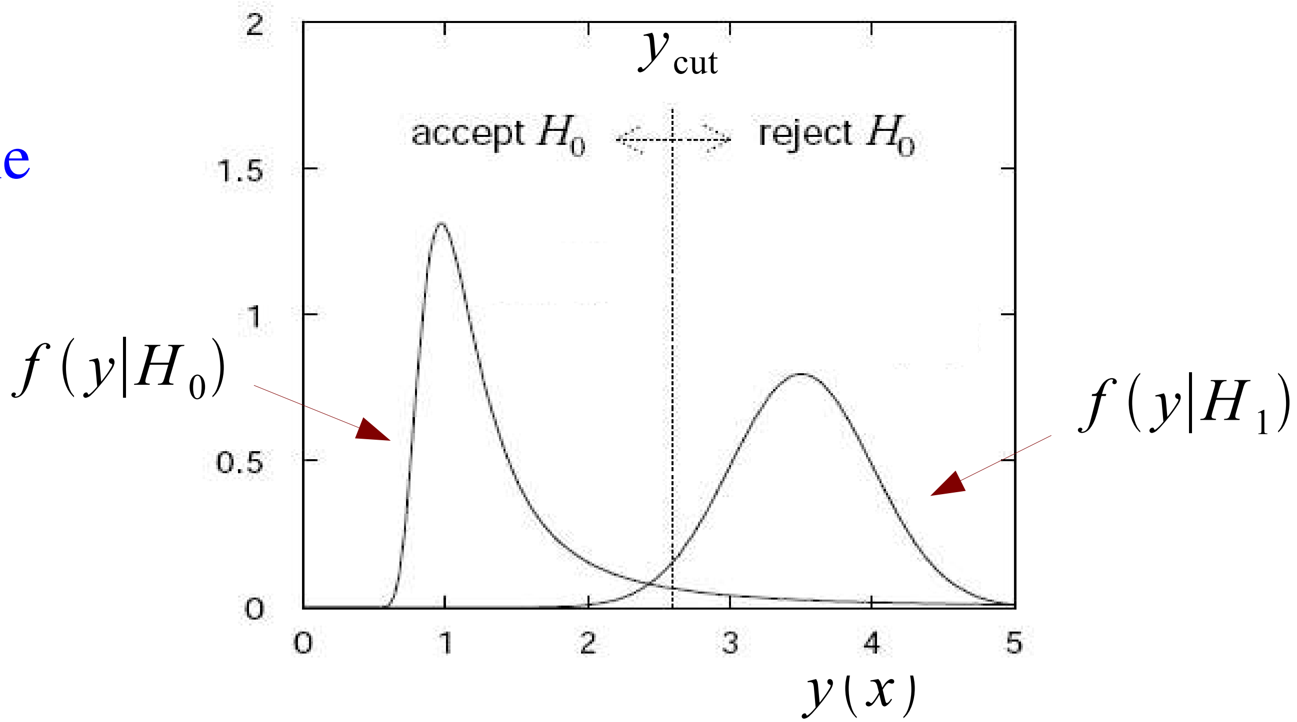
The decision boundary is a surface in the *n*-dimensional space of input variables, e.g., $y(\vec{x}) = \text{const}$.

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:

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 → prohibitively large number of cells to populate, can't generate enough training data.

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.,

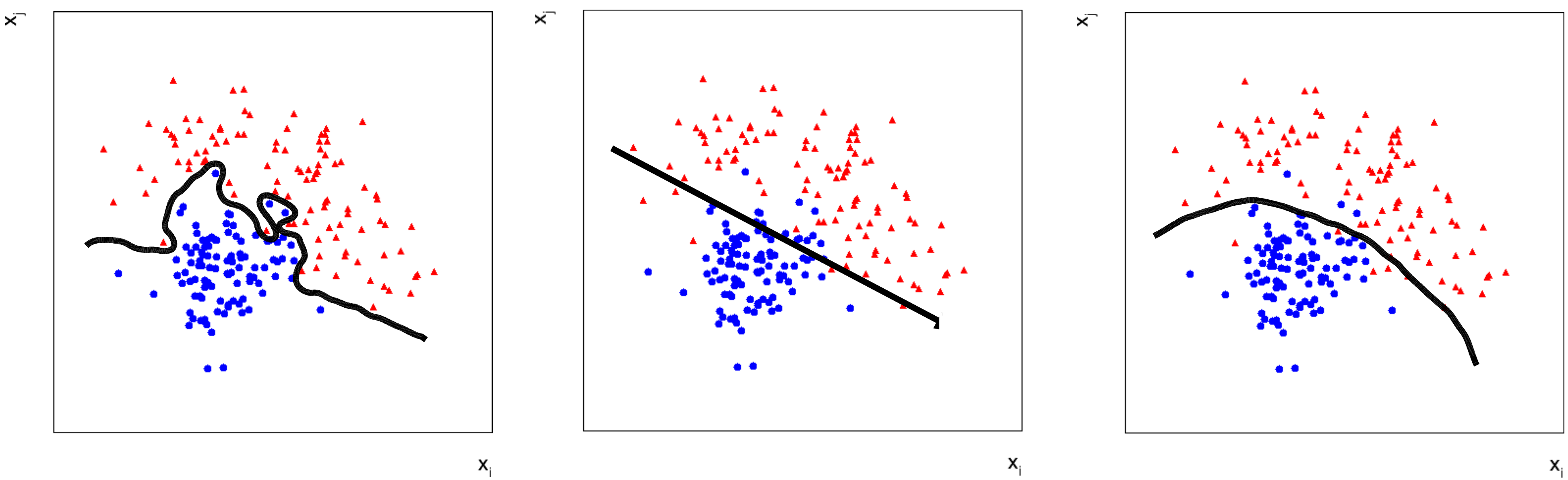
Event selection for further study

Searches for new event types

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

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

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

Neural networks

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

Decision trees

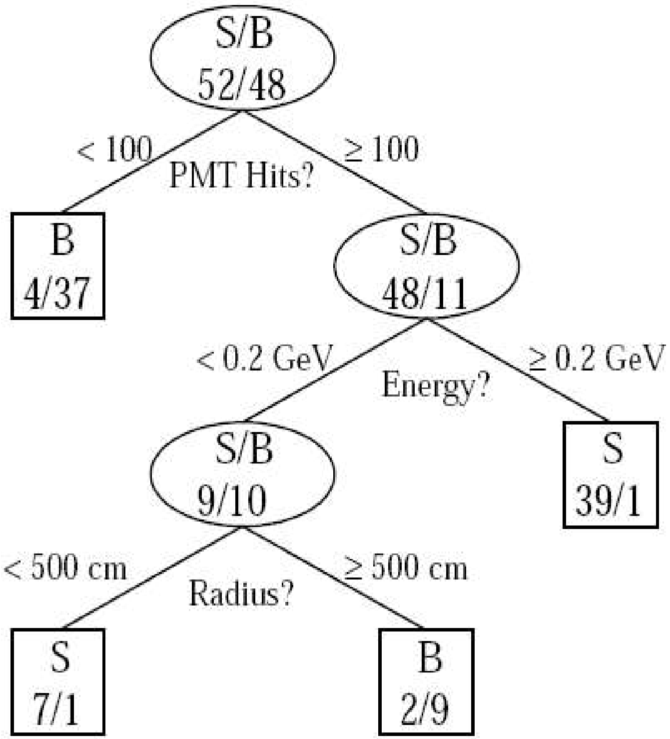
In a decision tree repeated cuts are made on a single variable until some stop criterion is reached.

The decision as to which variable is used is based on best achieved improvement in signal purity:

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

where w_i. is the weight of the *i*th event.

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



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

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

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

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,

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

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

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

i = event index

k = training sample index

Normalize so that

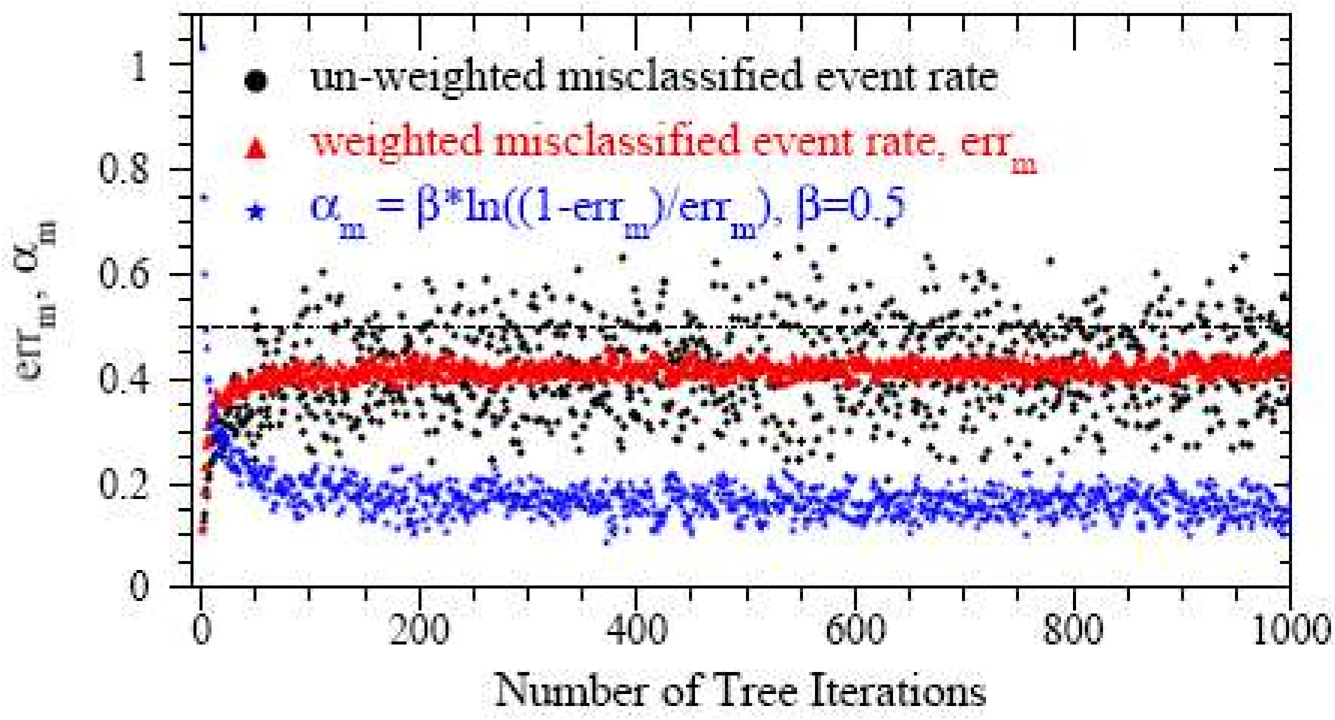
$$\sum_i w_i^{(k+1)} = 1$$

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

BDT example from MiniBooNE

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

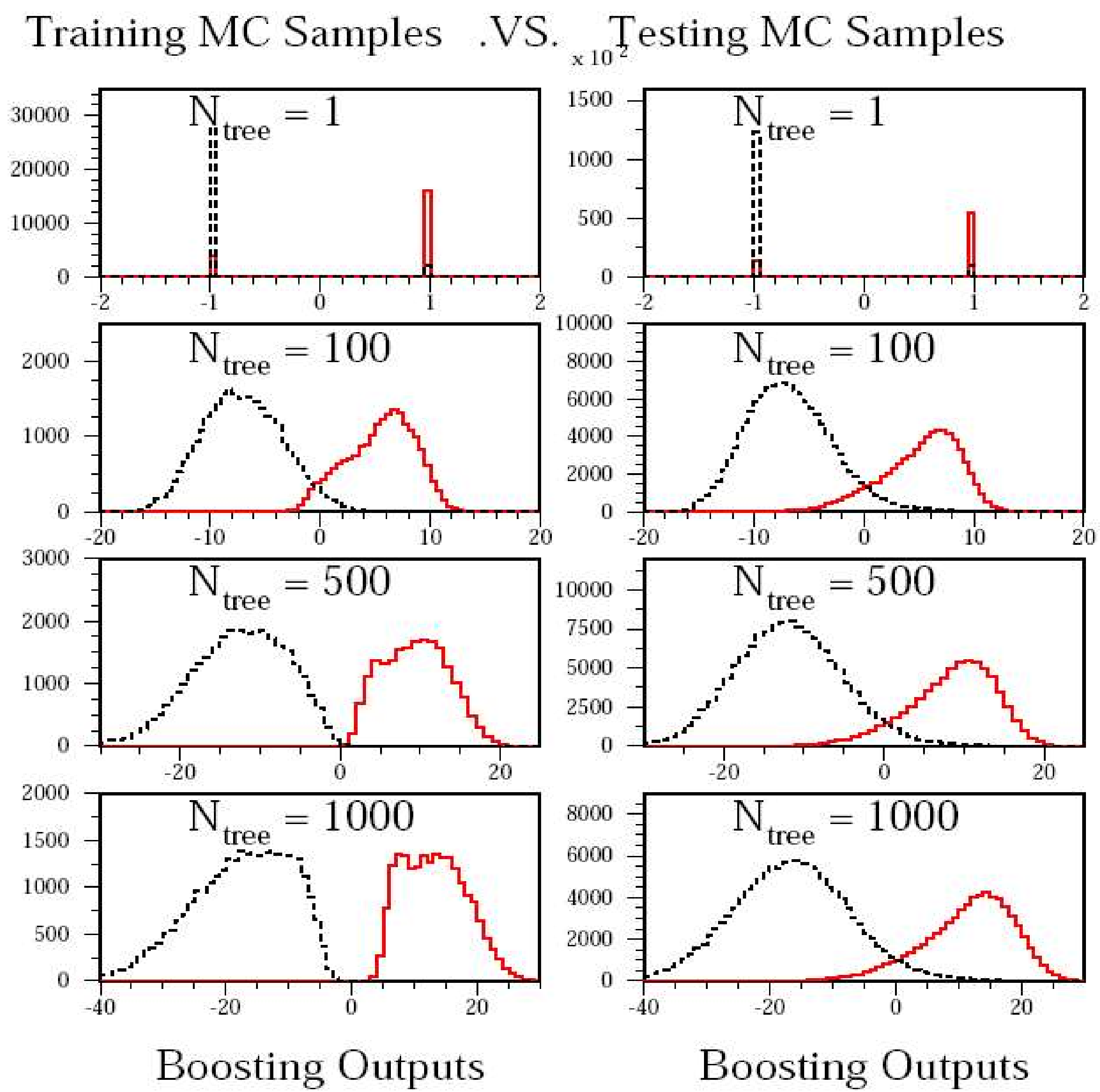
Each individual tree is relatively weak, with a misclassification error rate ~ 0.4 – 0.45



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

Monitoring overtraining

From MiniBooNE example



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

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)

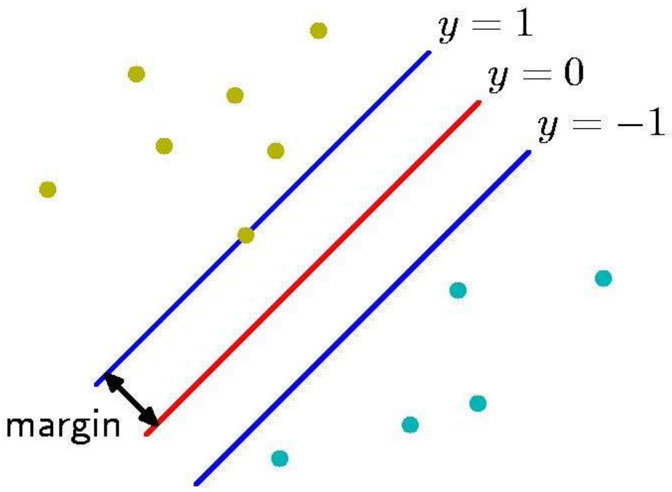
Suppose the classes can be separated by a hyperplane defined by a normal vector $\boldsymbol{w}$ and scalar offset b (the "bias"). We have

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

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

or equivalently

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

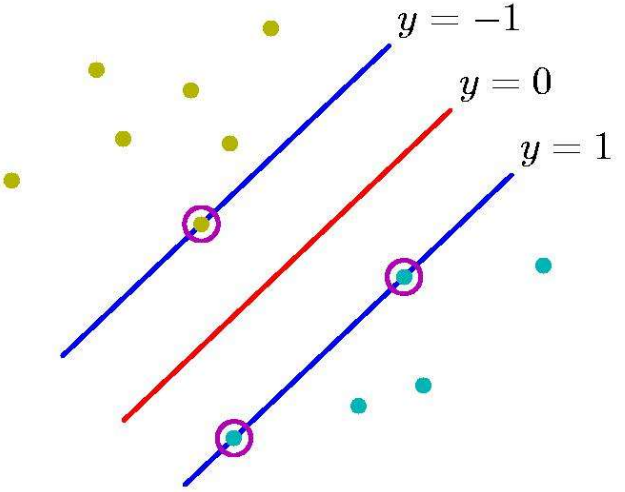


Bishop Ch. 7

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

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

Lagrangian formulation

This constrained minimization problem can be reformulated using a Lagrangian

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

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

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

(sum only contains support vectors)

Dual formulation

The classifier function is thus

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

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

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

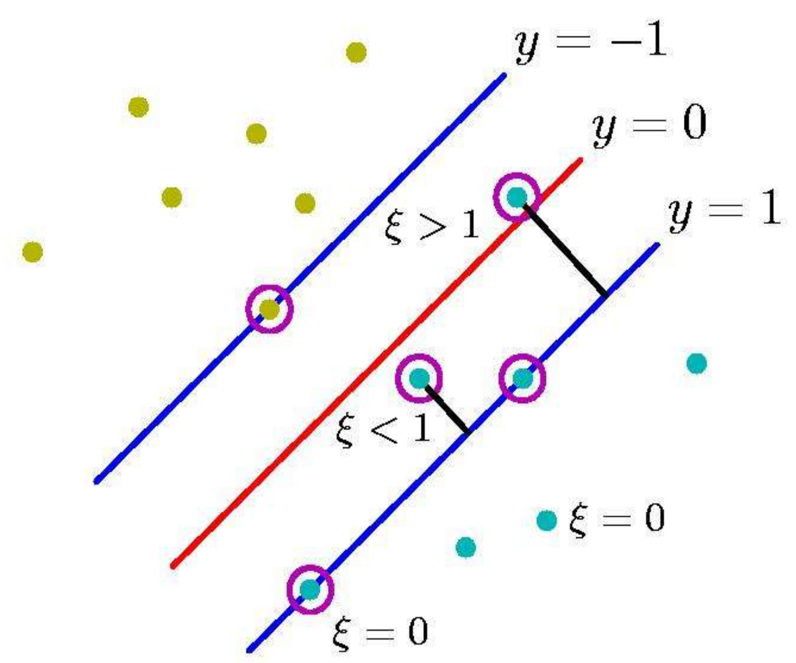
So this means that both the classifier function and the Lagrangian only involve dot products of vectors in the input variable space.

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

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

so in the feature space this becomes

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

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.

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

the cost parameter C (plays role of regularization parameter)

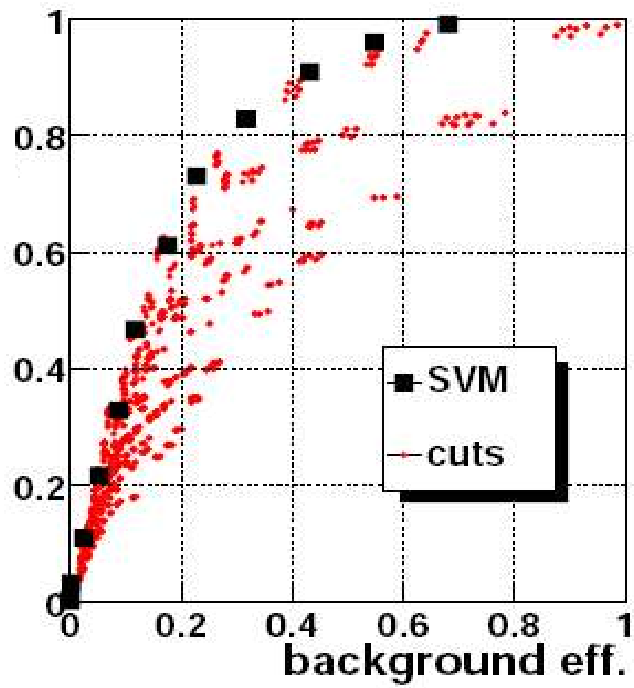
The training is relatively straightforward because, in contrast to neural networks, the function to be minimized has a single global minimum.

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

SVM in HEP

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).

signal eff.



Multivariate analysis discussion

For all methods, need to check:

Sensitivity to statistically unimportant variables (best to drop those that don't provide discrimination);

Level of smoothness in decision boundary (sensitivity to over-training)

Given the test variable, next step is e.g., select n events and estimate a cross section of signal: $\hat{\sigma}_s = (n - b)/\varepsilon_s L$

Now need to estimate systematic error...

If e.g. training (MC) data ≠ Nature, test variable is not optimal, but not necessarily biased.

But our estimates of background b and efficiencies would then be biased if based on MC. (True also for 'simple cuts'.)

Multivariate analysis discussion (2)

But in a cut-based analysis it may be easier to avoid regions where untested features of MC are strongly influencing the decision boundary.

Look at control samples to test joint distributions of inputs.

Try to estimate backgrounds directly from the data (sidebands).

The purpose of the statistical test is often to select objects for further study and then measure their properties.

Need to avoid input variables that are correlated with the properties of the selected objects that you want to study. (Not always easy; correlations may be poorly known.)

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

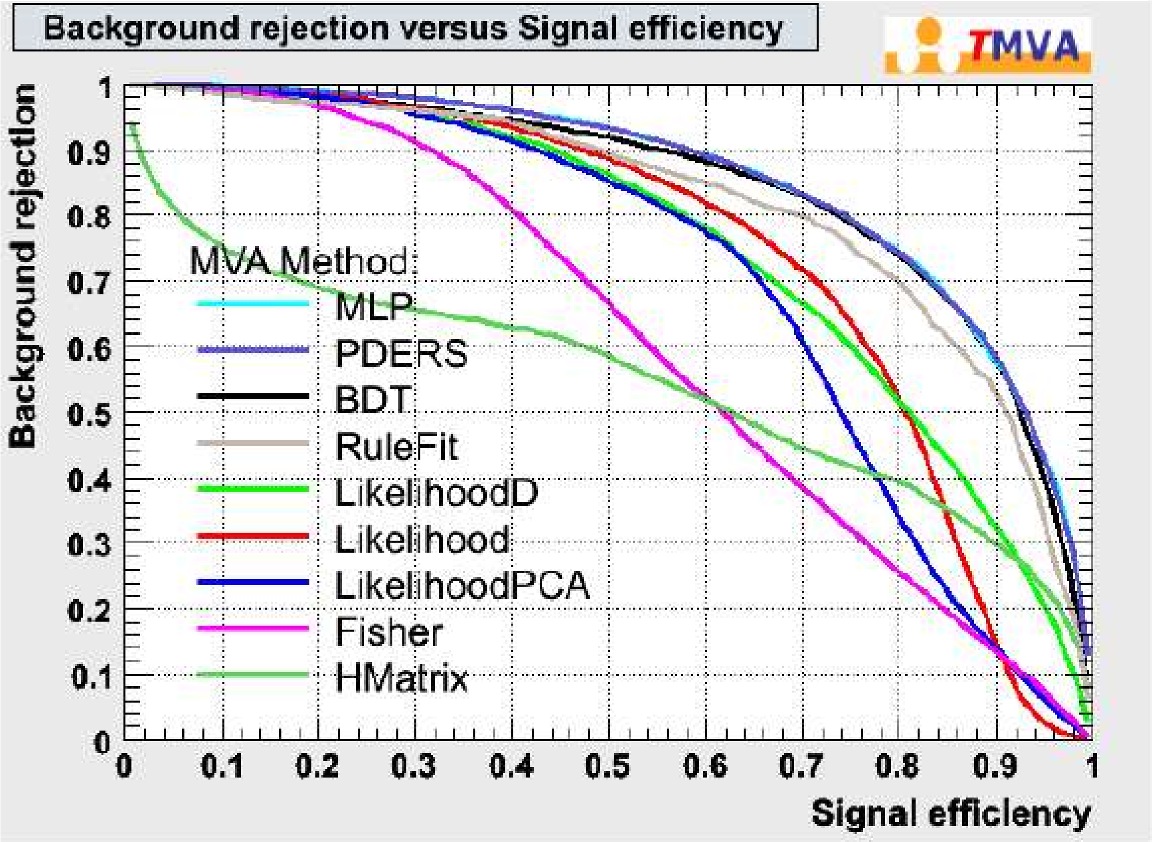
`StatPatternRecognition`, I. Narsky, physics/0507143

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

Comparing multivariate methods (TMVA)



Choose the best one!

Summary

Boosted Decision Trees and Support Vector Machines are two examples of relatively modern developments in Machine Learning that are only recently attracting attention in HEP.

There are now many multivariate methods on the market and it is difficult to make general statements about performance; this is often very specific to the problem.

Expect advanced multivariate methods to have a major impact in areas where one struggles for statistical significance, not in precision measurements.

A simpler (e.g. “cut-based”) analysis may be considered more robust, but e.g. a 5σ signal from an SVM supported by 4σ from cuts may win.

Fortunately tools to investigate these methods are now widely available.

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

Boosted decision tree example

First use of boosted decision trees in HEP was for particle identification for the MiniBoone neutrino oscillation experiment.

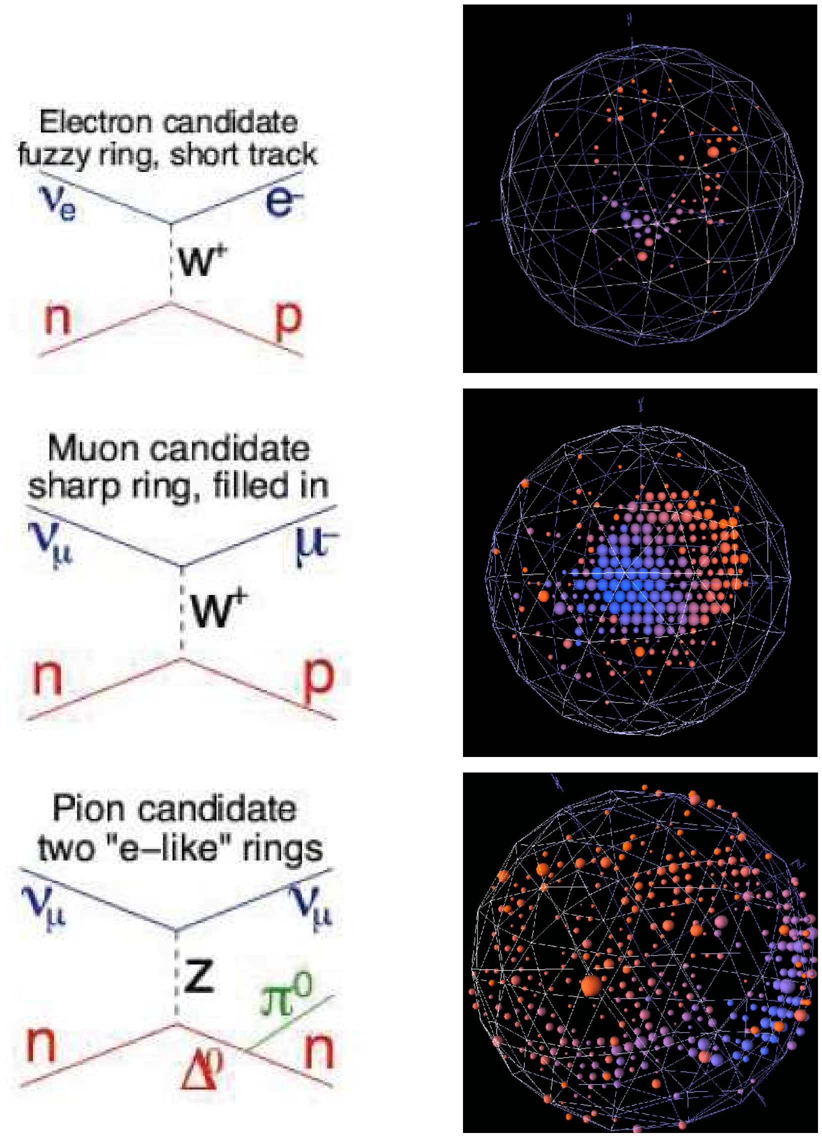
H.J.Yang, B.P. Roe, J. Zhu, "Studies of Boosted Decision Trees for MiniBooNE Particle Identification", Physics/0508045, Nucl. Instum. & Meth. A 555(2005) 370-385.

B.P. Roe, H.J. Yang, J. Zhu, Y. Liu, I. Stancu, G. McGregor, "Boosted decision trees as an alternative to artificial neural networks for particle identification", physics/0408124, NIMA 543 (2005) 577-584.

Particle i.d. in MiniBooNE

Search for ν_μ to ν_e oscillations required particle i.d. using information from Cherenkov detector.

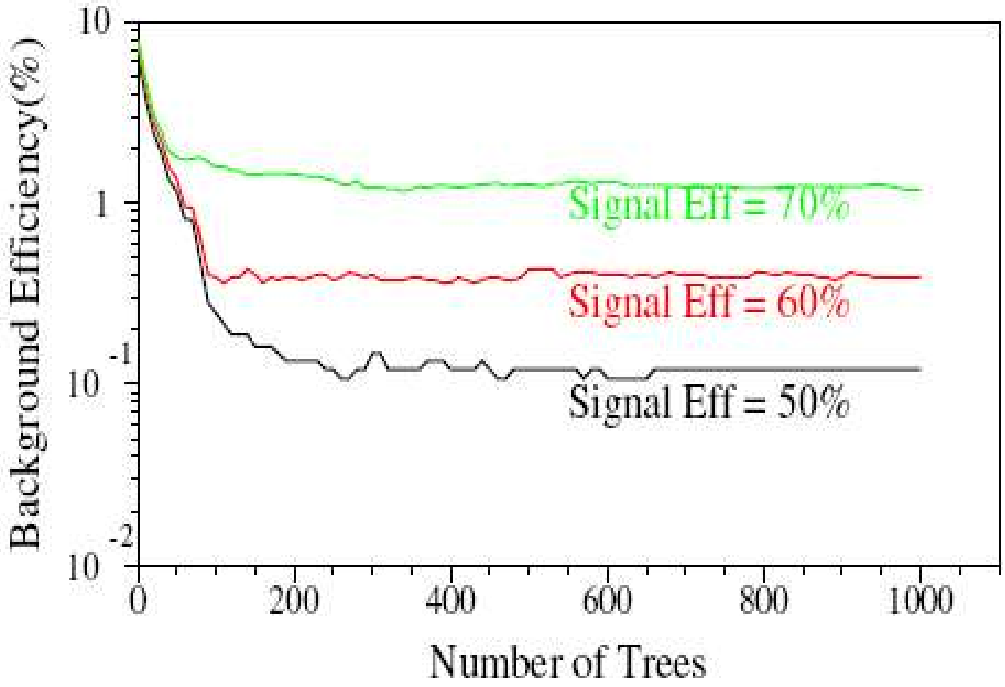
Large number (~200) input variables measured for each event.

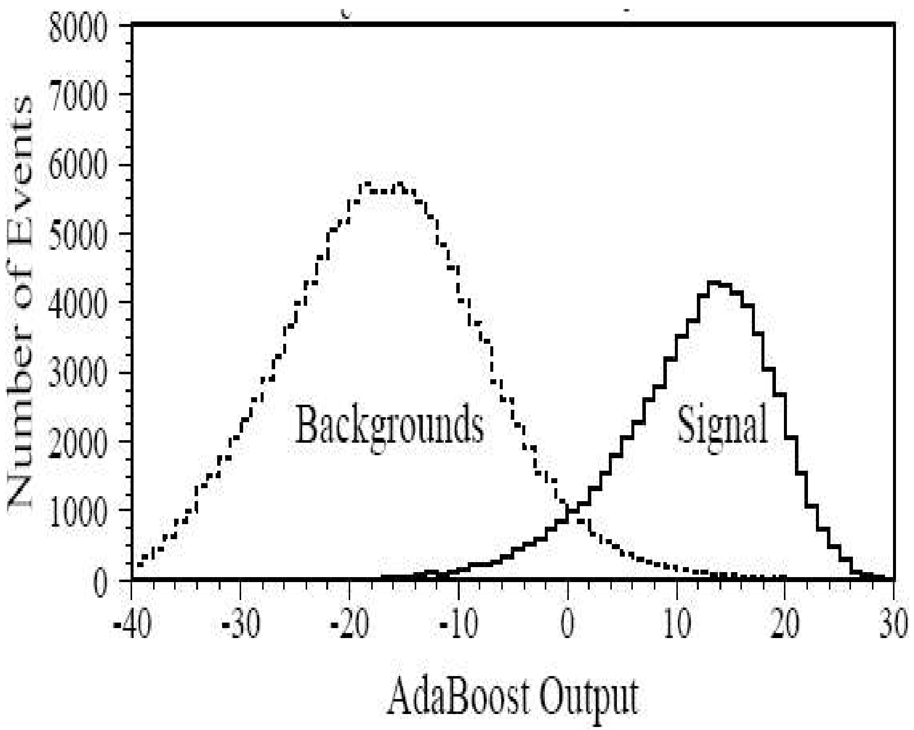


H.J. Yang, MiniBooNE PID, DNP06

MiniBoone boosted decision tree

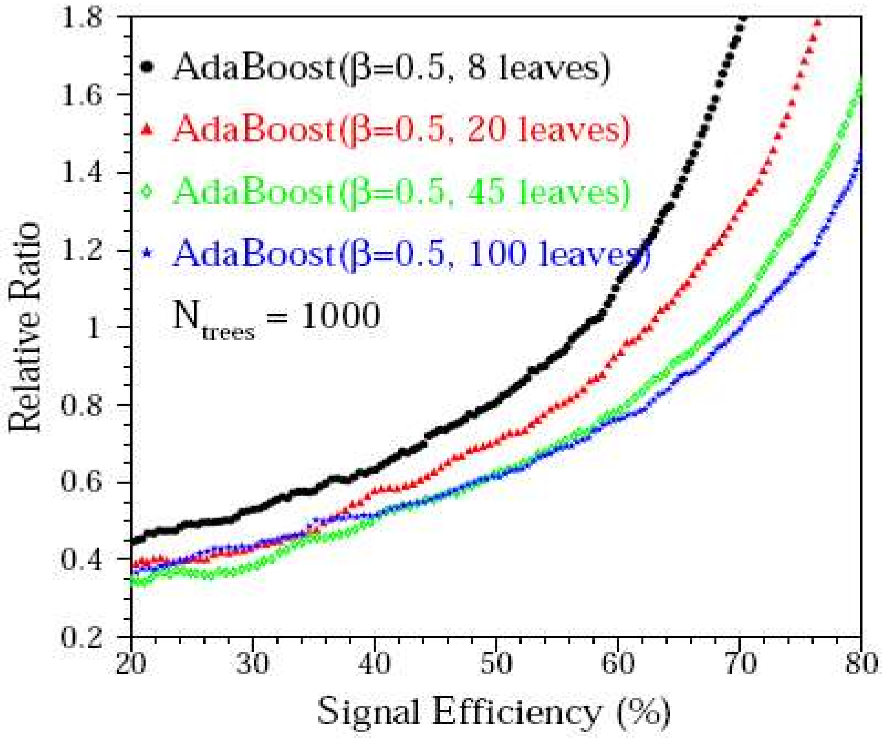
Here performance stable after a few hundred trees





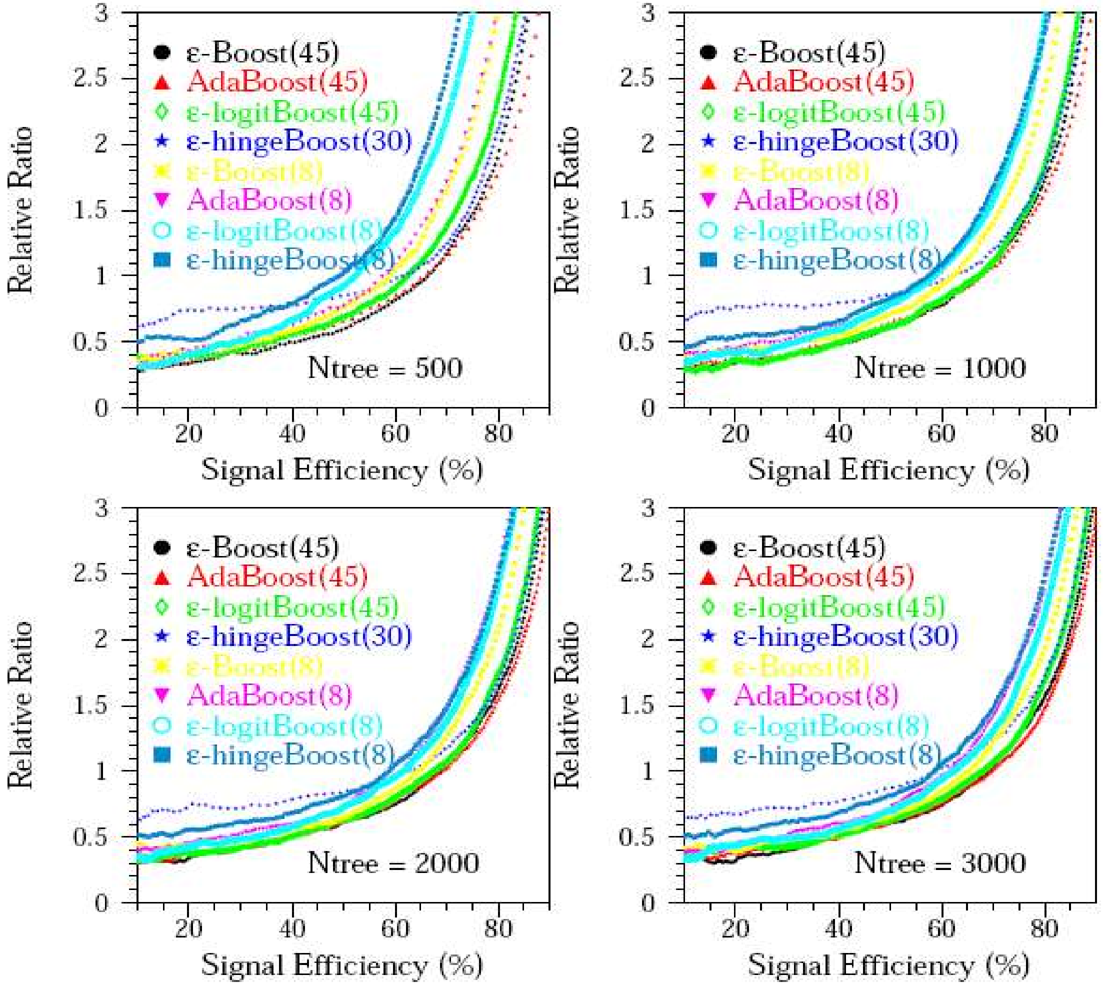
MiniBooNE Decision tree performance

$\sim \varepsilon_b / \varepsilon_b$



Comparison of boosting algorithms

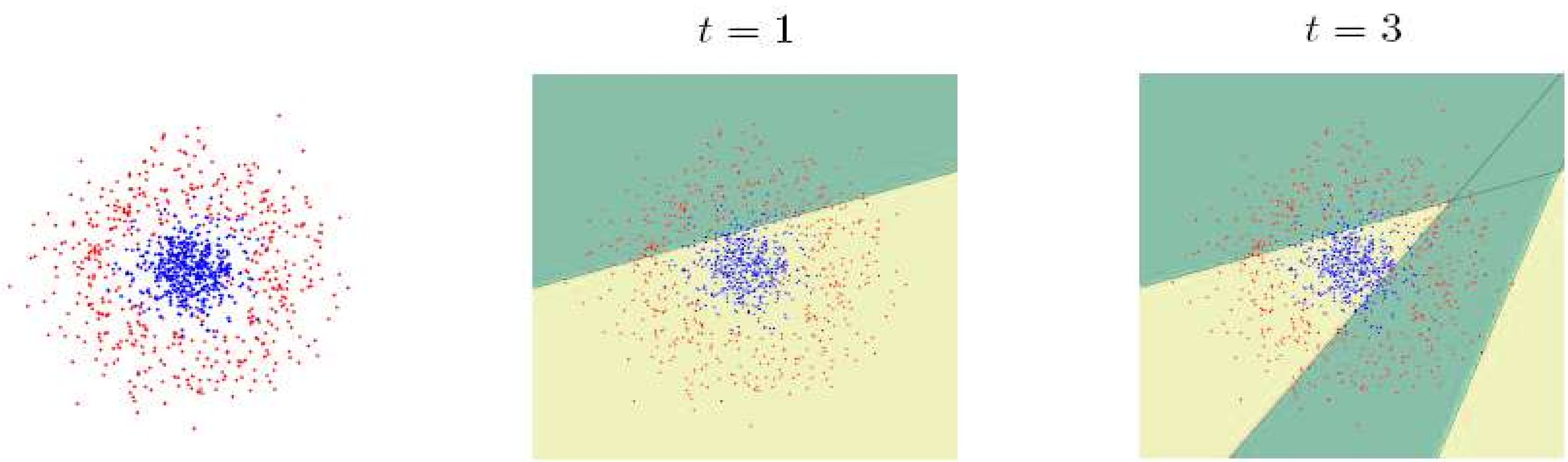
A number of boosting algorithms on the market; differ in the update rule for the tree weight.



AdaBoost study with linear classifier

J. Sochman, J. Matas, **`cmp.felk.cvut.cz`**

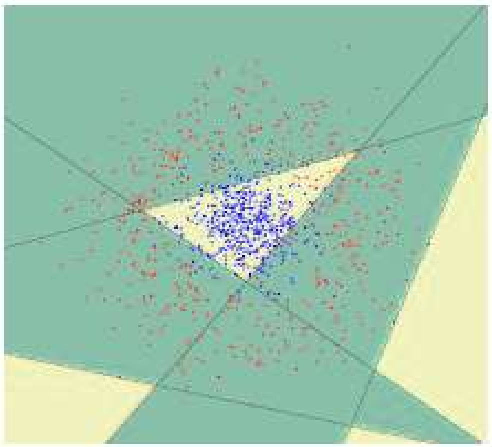
Start with a problem for which a linear classifier is weak:



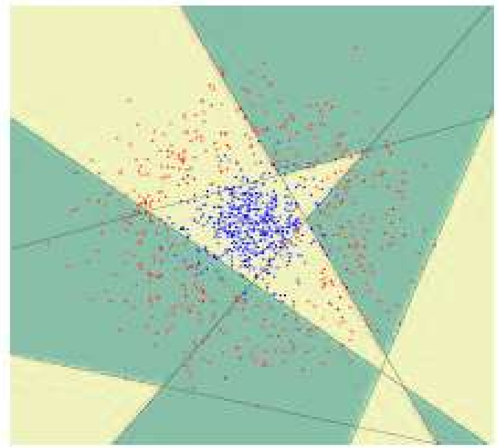
AdaBoost study with linear classifier

J. Sochman, J. Matas, **`cmp.felk.cvut.cz`**

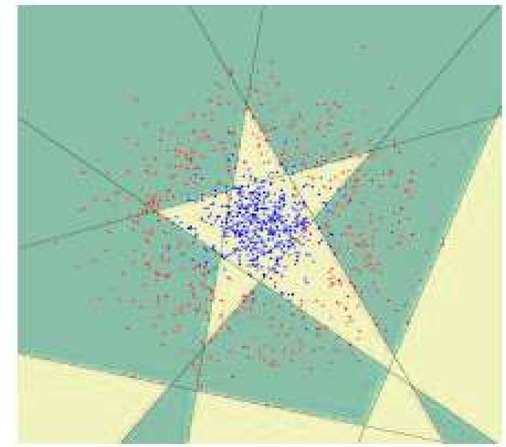
$t = 5$



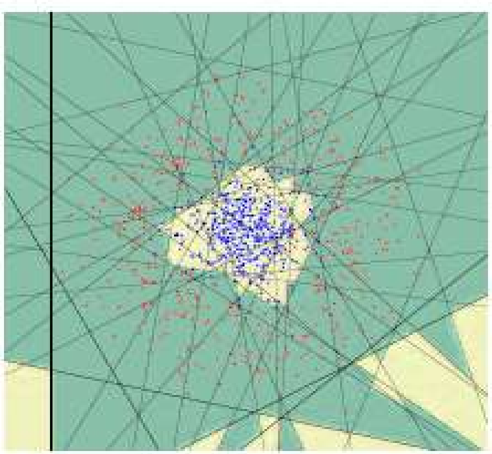
$t = 6$

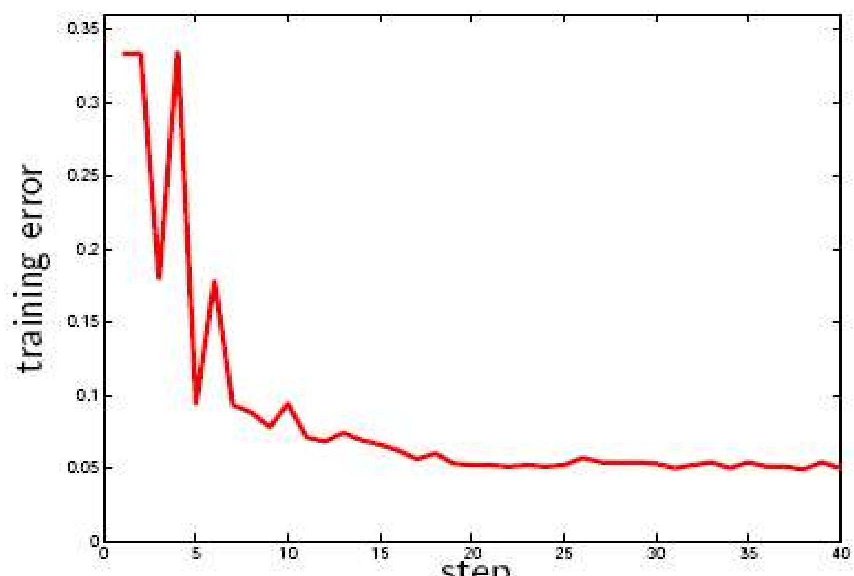


$t = 7$



$t = 40$





Imperfect pdf estimation

What if the approximation we use (e.g., parametric form, assumption of variable independence, etc.) to estimate $p(\boldsymbol{x})$ is wrong?

If we use poor estimates to construct the test variable

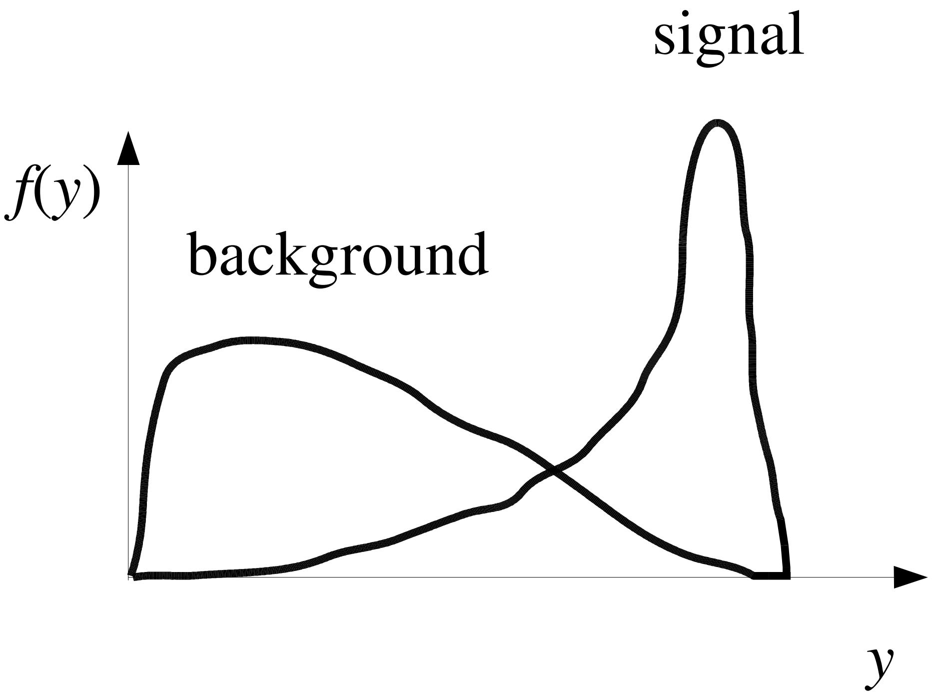
$$y(\vec{x}) = \frac{\hat{p}(\vec{x}|H_0)}{\hat{p}(\vec{x}|H_1)}$$

then the discrimination between the event classes will not be optimal.

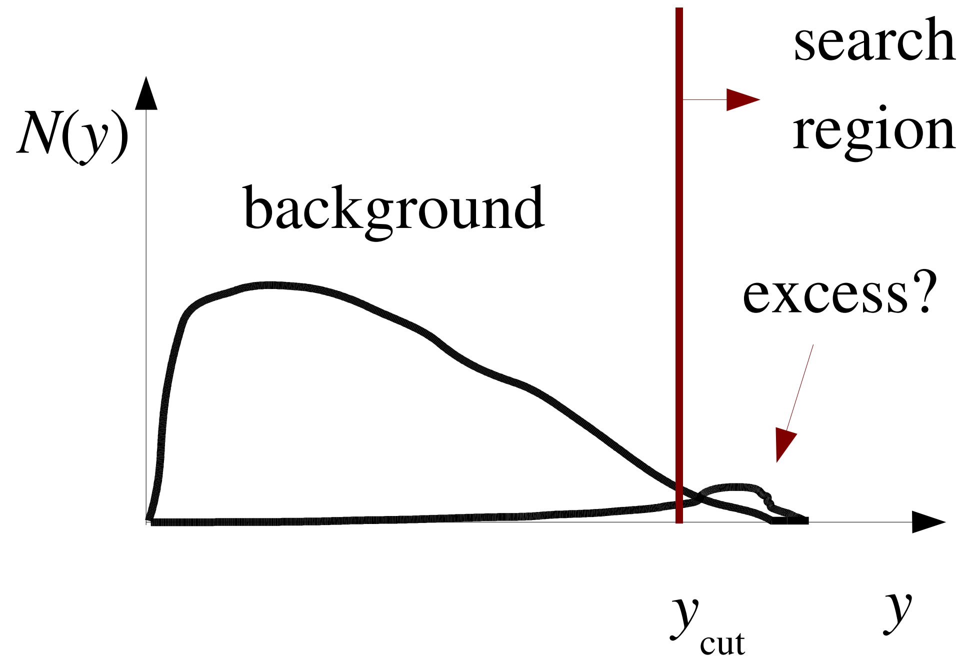
But can this cause us e.g. to make a false discovery?

Even if the estimate of $p(x)$ used in the discriminating variable are imperfect, this will not affect the accuracy of the distributions $f(y|H_0)$, $f(y|H_1)$; this only depends on the reliability of the training data.

Using the 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. Maximize the probability of this happening by setting y_{cut} for maximum $s/\sqrt{b}$ (roughly true).

Controlling false discovery

So for a reliable discovery what we depend on is an accurate estimate of the expected number of background events, and this accuracy only depends on the quality of the training data; works for any function $y(\boldsymbol{x})$.

But we do not blindly rely on the MC model for the training data for background; we need to test it by comparing to real data in control samples where no signal is expected.

The ability to perform these tests will depend on on the complexity of the analysis methods.