

Multivariate statistical methods and data mining in particle physics

Lecture 4 (19 June, 2008)



Glen Cowan

RHUL Physics

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



Academic Training Lectures

CERN

16–19 June, 2008

Outline

Statement of the problem

Brief review of statistical formalism

Some general considerations

Multivariate classifiers:

Linear discriminant function

Neural networks

Naive Bayes classifier

Density estimation methods

Decision trees

Support Vector Machines

Thursday start



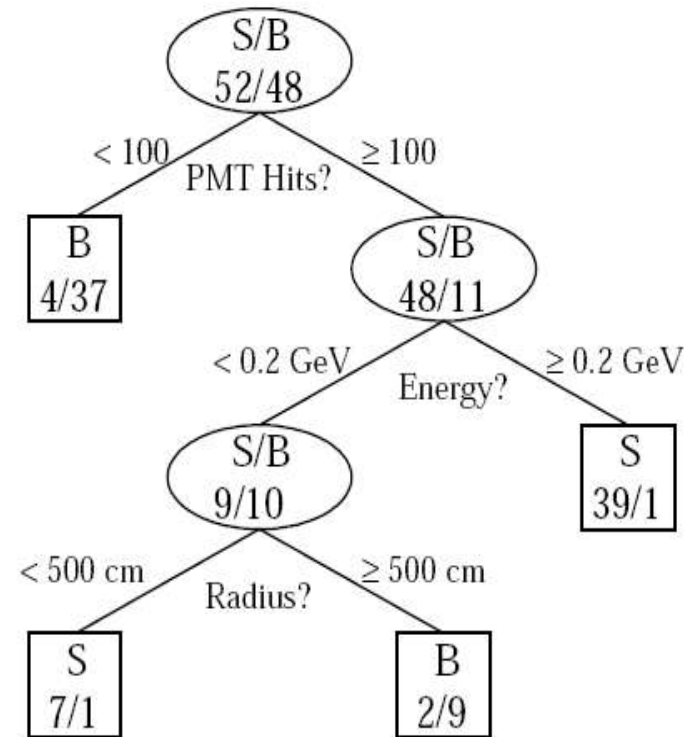
Decision trees

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 separation.

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

Classify resulting cut regions as signal or background based on the signal fraction above/below a specified value (e.g. $\frac{1}{2}$) in the terminal nodes (leaves).



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

Separation measure

Suppose the i th event has a weight w_i . The purity at a given node is

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

Measure of s/b separation in a node can be taken as e.g.

$$Gini = P(1 - P)$$

$$\text{Cross entropy} = -P \ln P - (1 - P) \ln(1 - P)$$

$$\text{Misclassification error} = 1 - \max(P, 1 - P)$$

These are maximum for $P = 0.5$ and zero for $P = 0$ or 1 .

Tend to give similar results in real examples; Gini index most common.

What variable to cut on

The variable is found that provides the greatest increase in the separation measure (e.g., Gini) in the two daughter nodes relative to the parent.

Continue splitting until the number of events in a node falls below a specified threshold (could be 1). Terminal nodes called **leaves**.

The 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**.

Define a discriminant function $f(\mathbf{x})$ as

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

Decision tree size

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.

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 tree stability

Decision trees tend to be very sensitive to statistical fluctuations in the training sample.

If e.g. two variables have similar separating power, then a small fluctuation in the training data could cause either of the two variables to be chosen to split the tree; all subsequent development of the tree is thereby affected.

Methods such as **boosting** can be used to stabilize the tree.

Boosting

Boosting is a general method of creating a set of classifiers which can be combined to achieve a new classifier that is more stable and has a smaller error than any individual one.

Often applied to decision trees but, can be applied to any classifier.

Suppose we have a training sample T consisting of N events with

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

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

w_1, \dots, w_N event weights

Now define a rule to create from this an ensemble of training samples T_1, T_2, \dots , derive a classifier from each and average them.

Boosting (2)

The framework of a general boosting algorithm can be written:

Initialize the first training sample T_1

Determine a classifier $f_1(\mathbf{x}, T_1)$ from T_1 and assign a weight α_1

Determine a new training sample T_2 from T_1

Iterate K times, so we have:

K training samples, T_1, \dots, T_K

K classifiers $f_1(\mathbf{x}, T_1), \dots, f_K(\mathbf{x}, T_K)$

K classifier weights, $\alpha_1, \dots, \alpha_K$

Form the final classifier as a function of those from the individual steps, usually as a weighted sum:

$$y(\mathbf{x}) = \sum_{k=1}^K \alpha_k f_k(\mathbf{x}, T_k)$$

Boosting (3)

The important point is that each classifier f_k depends on a modified version of the training sample with different event weights.

Trick is to create modifications that give smaller error rates than those of the preceding classifiers.

A number of ways exist for doing this – a successful one is **AdaBoost** (Freund and Schapire, 1997); short for *Adaptive Boosting*.

AdaBoost

First initialize the training sample T_1 using the original

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

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

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

with the weights equal and normalized such that

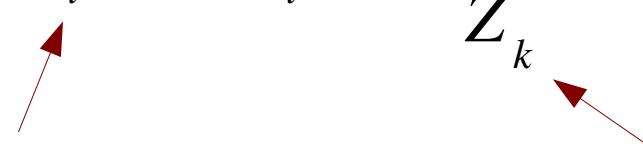
$$\sum_{i=1}^N w_i^{(1)} = 1$$

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

At each step k , associate a weight α_k with the classifier f_k .

Updating the event weights

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

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


i = event index

k = training sample index

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

Updating the event weights (2)

We can use the sign of $f_k(\mathbf{x})$ as a classifier. For an event with data \mathbf{x}_i ,

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

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

Therefore the product of the classifier value $f_k(\mathbf{x}_i)$ and class label y_i is

$y_i f_k(\mathbf{x}_i) > 0$ if the event is classified correctly

$y_i f_k(\mathbf{x}_i) < 0$ if the event is classified incorrectly

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

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

Error rate of the k th classifier

Therefore the classification error rate ε_k is

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

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

Assigning the classifier weight

Define the classifier $f_k(\mathbf{x})$ such that it minimizes the error rate ε_k .

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

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

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

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

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

AdaBoost error rate

So providing each classifier in the ensemble has $\epsilon_k > 1/2$, i.e., better than random guessing, then the error rate for the final classifier on the training data (not on unseen data) drops to zero.

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

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

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

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

Boosted decision tree example

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

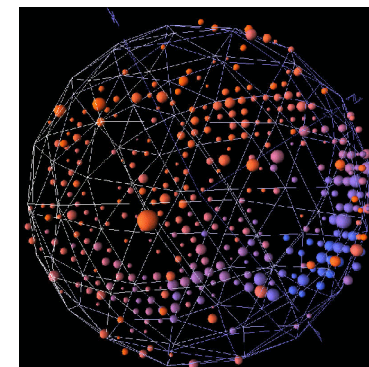
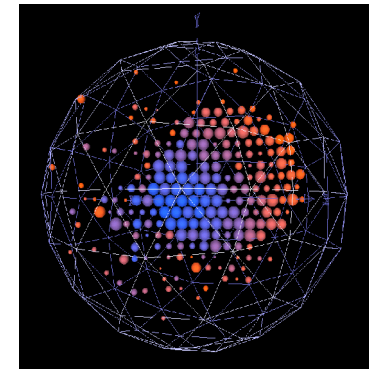
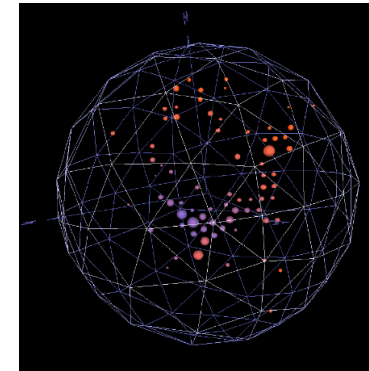
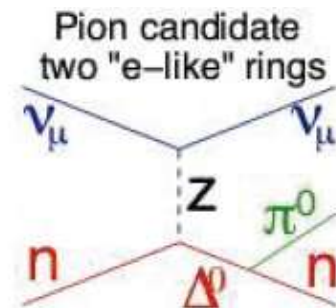
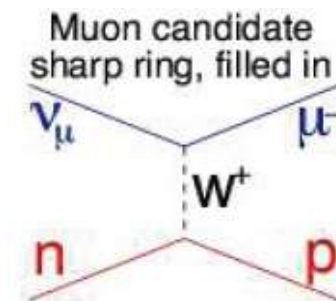
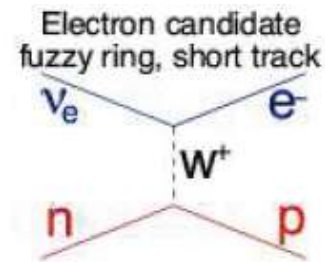
H.J. Yang, B.P. Roe, J. Zhu, “Studies of Boosted Decision Trees for MiniBooNE Particle Identification”, Physics/0508045, Nucl. Instrum. & Meth. A 555(2005) 370-385.

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.

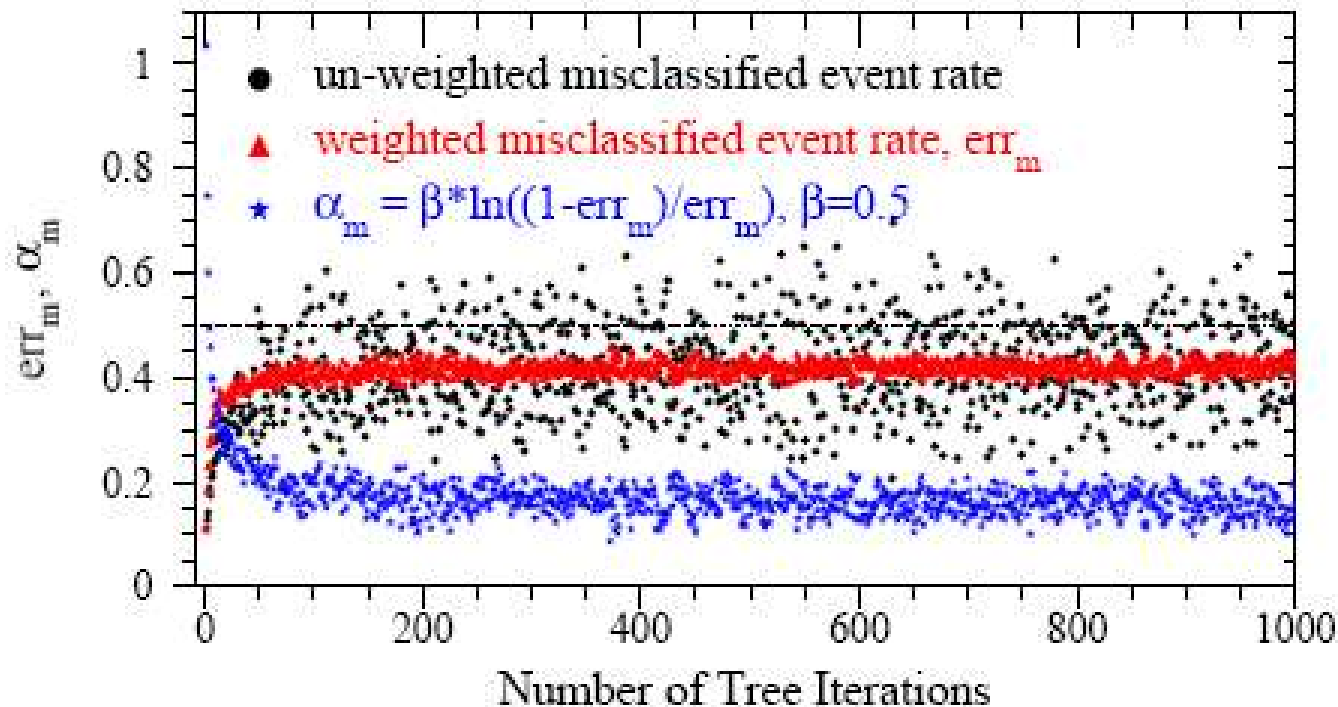
Large number (~200) input
variables measured for each
event.



H.J. Yang, MiniBooNE PID, DNP06

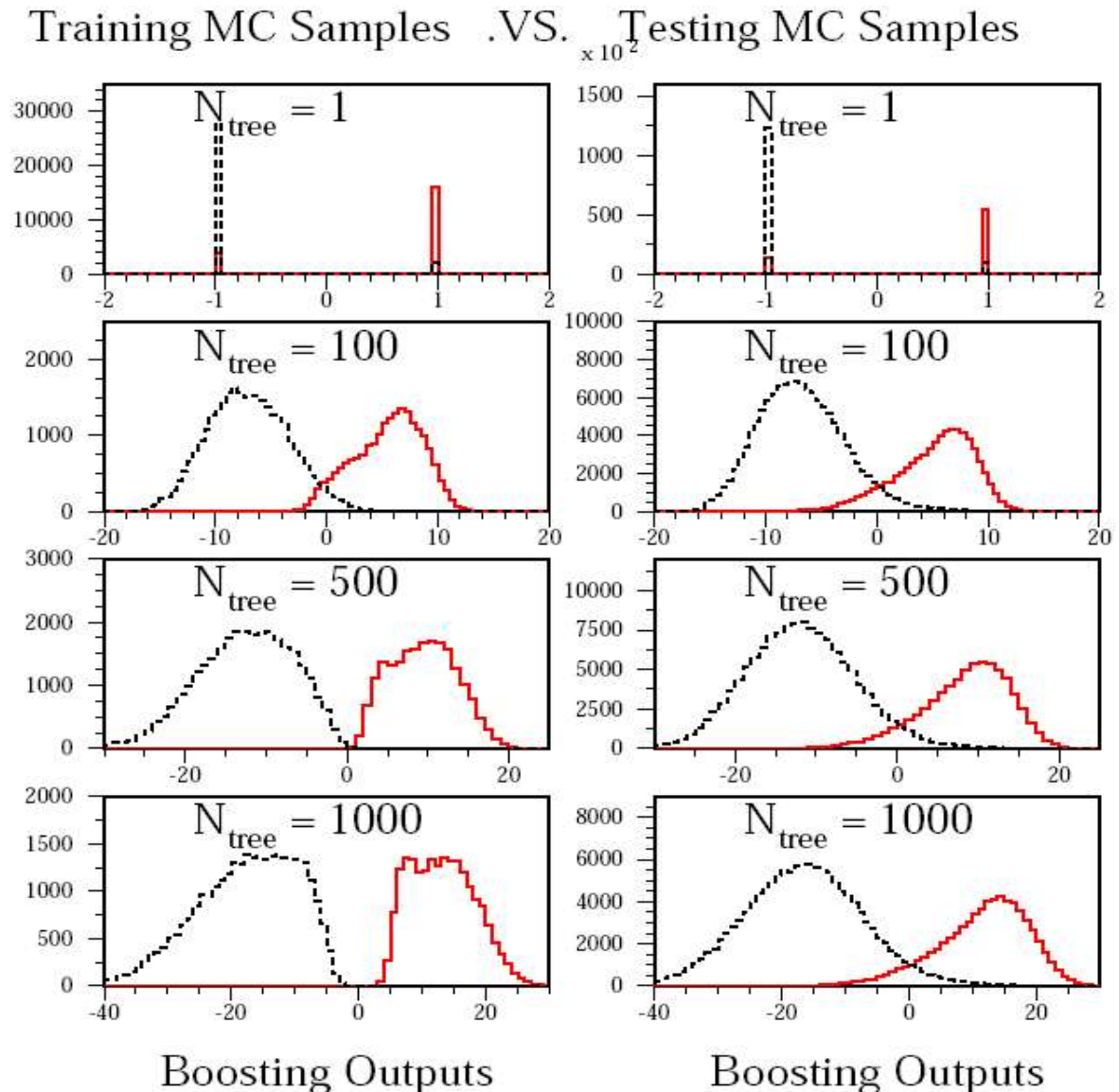
Boosted decision tree example

Each individual tree is relatively weak, with a misclassification error rate $\sim 0.4 - 0.45$



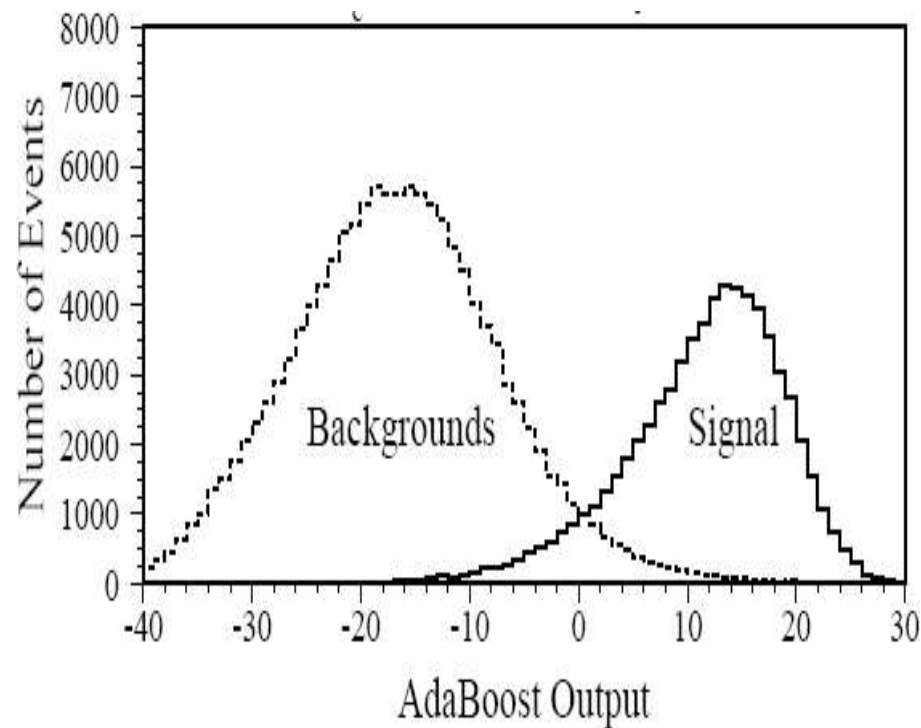
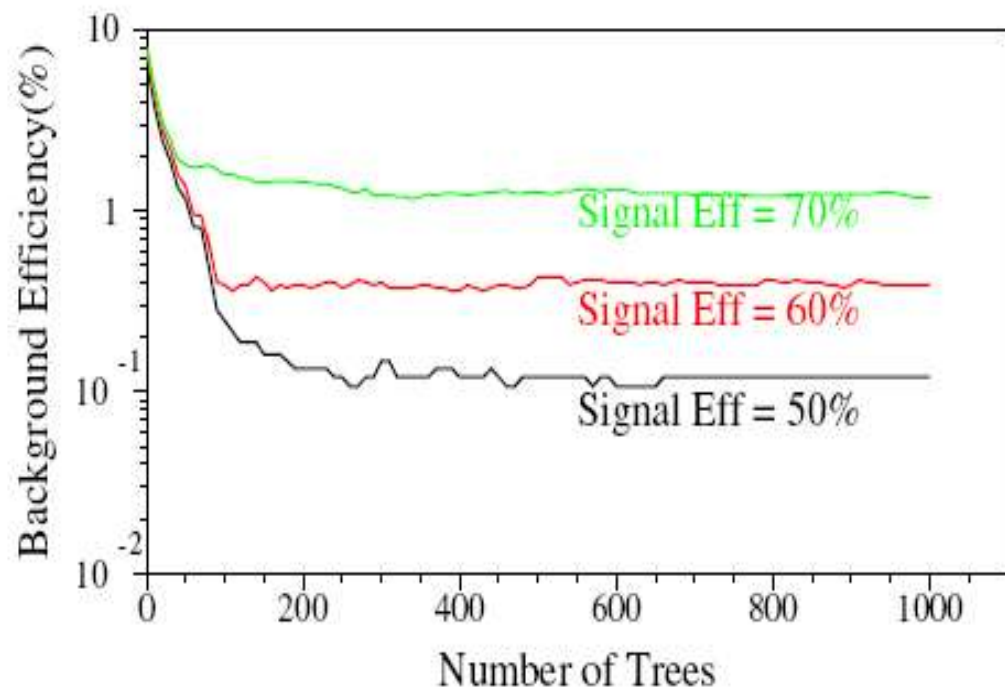
Monitoring overtraining

From MiniBooNE
example



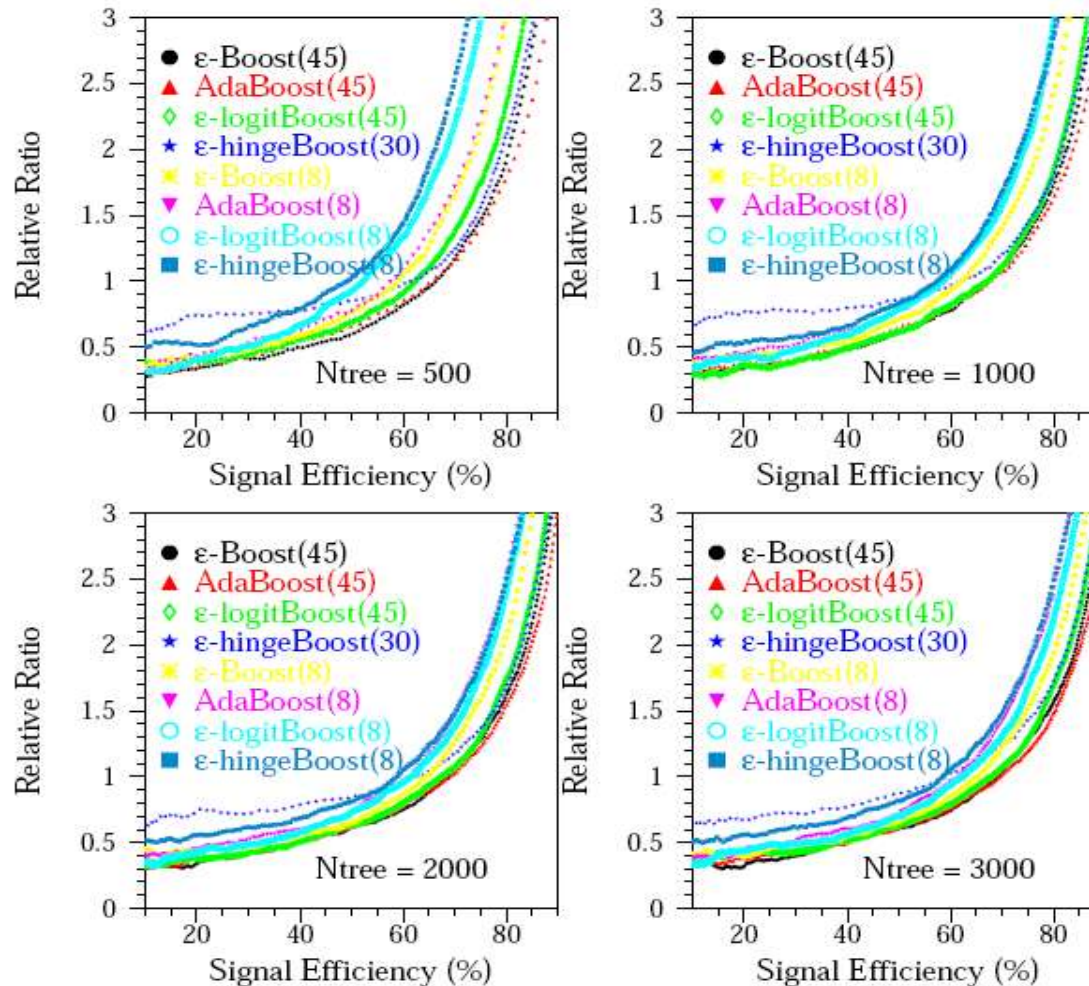
MiniBoone boosted decision tree

Here 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 tree weight.



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 work with an individual tree).

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

Support Vector Machines

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

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

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

Some references for kernel methods and SVMs:

The books mentioned 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

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

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

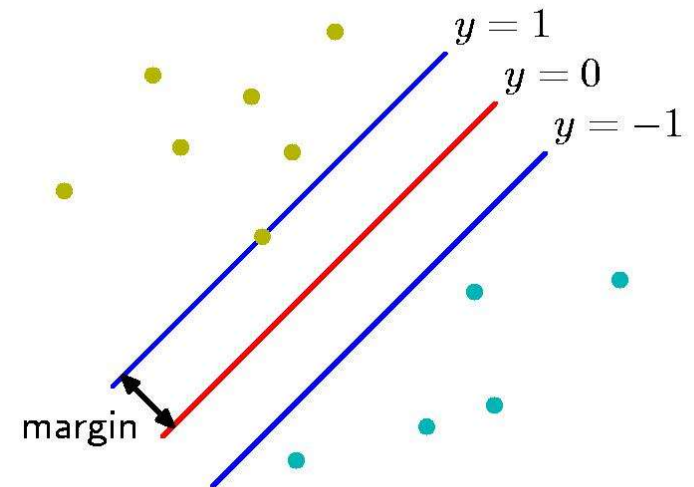
Suppose the classes can be separated by a hyperplane defined by a normal vector \mathbf{w} and scalar offset b (the “bias”). We have

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

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

or equivalently

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

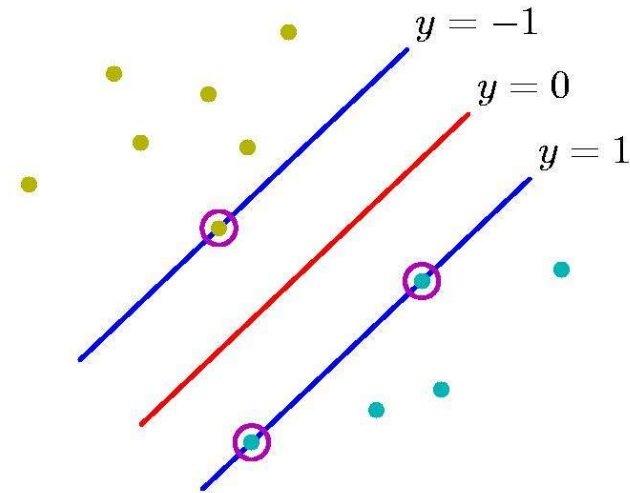


Bishop Ch. 7

Margin and support vectors

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

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



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

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

Linear SVM classifier

We can define the classifier using

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

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

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

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

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

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

Lagrangian formulation

This constrained minimization problem can be reformulated using a Lagrangian

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

positive Lagrange multipliers α_i

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

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

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

Dual formulation

The classifier function is thus

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

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

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

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

Nonseparable data

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

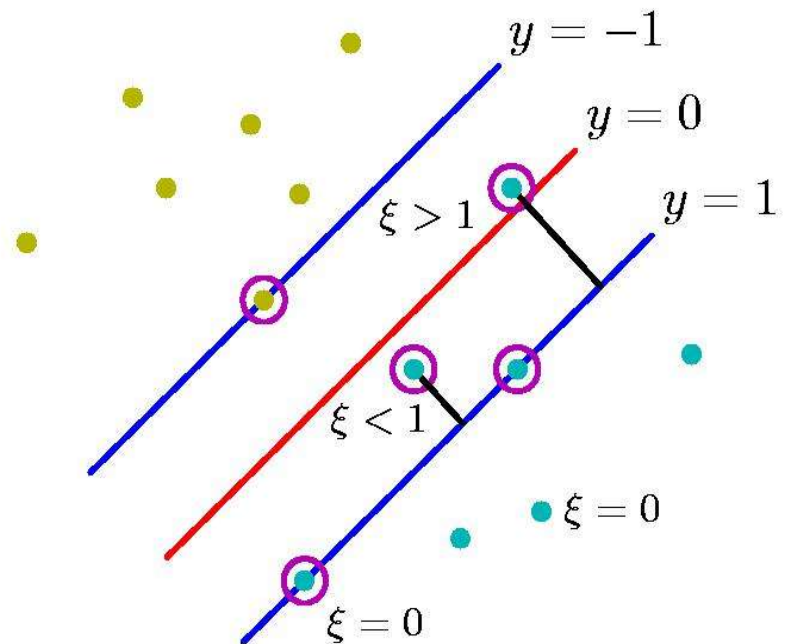
$$y_i(\mathbf{x}_i \cdot \mathbf{w} + b) + \xi_i - 1 \geq 0 \text{ with } \xi_i \geq 0 \text{ for all } i$$

Thus the training point \mathbf{x}_i is allowed to be up to a distance ξ_i on the wrong side of the 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 \mathbf{w} to be

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

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

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

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

Nonlinear SVM

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

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

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

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

Only dot products

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

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

so in the feature space this becomes

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

The Kernel trick

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

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

Functions having this property must satisfy Mercer's condition

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

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

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

Finding kernels

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

Frequently used kernels to construct classifiers are e.g.

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

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

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

Using an SVM

To use an SVM the user must as a minimum choose

- a kernel function (e.g. Gaussian)

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

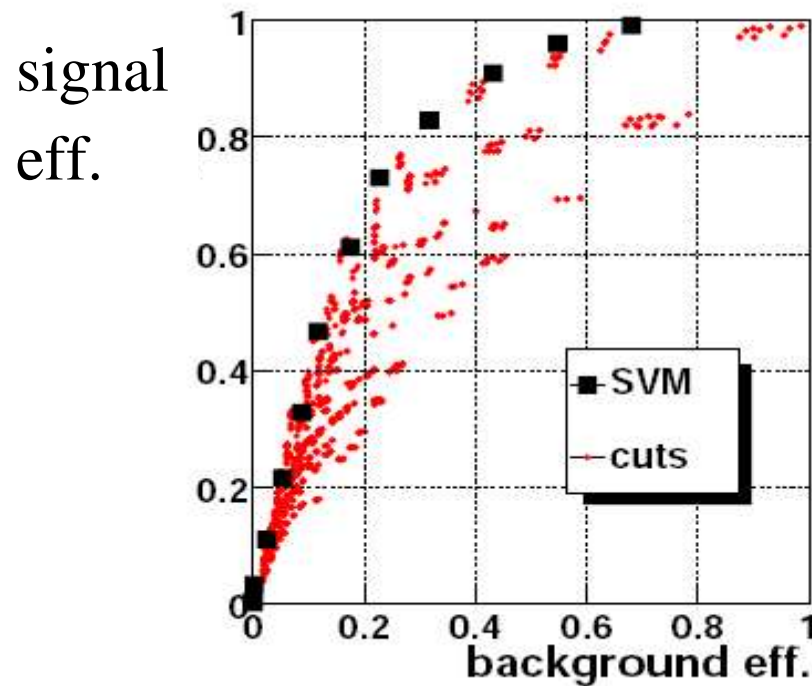
- the cost parameter C (plays role of regularization parameter)

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

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

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 analysis (A. Vaiciulis, contribution to PHYSTAT02).



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: $\hat{\sigma}_s = (n - b)/\epsilon_s L$

Now need to estimate systematic error...

If e.g. training (MC) data \neq 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.

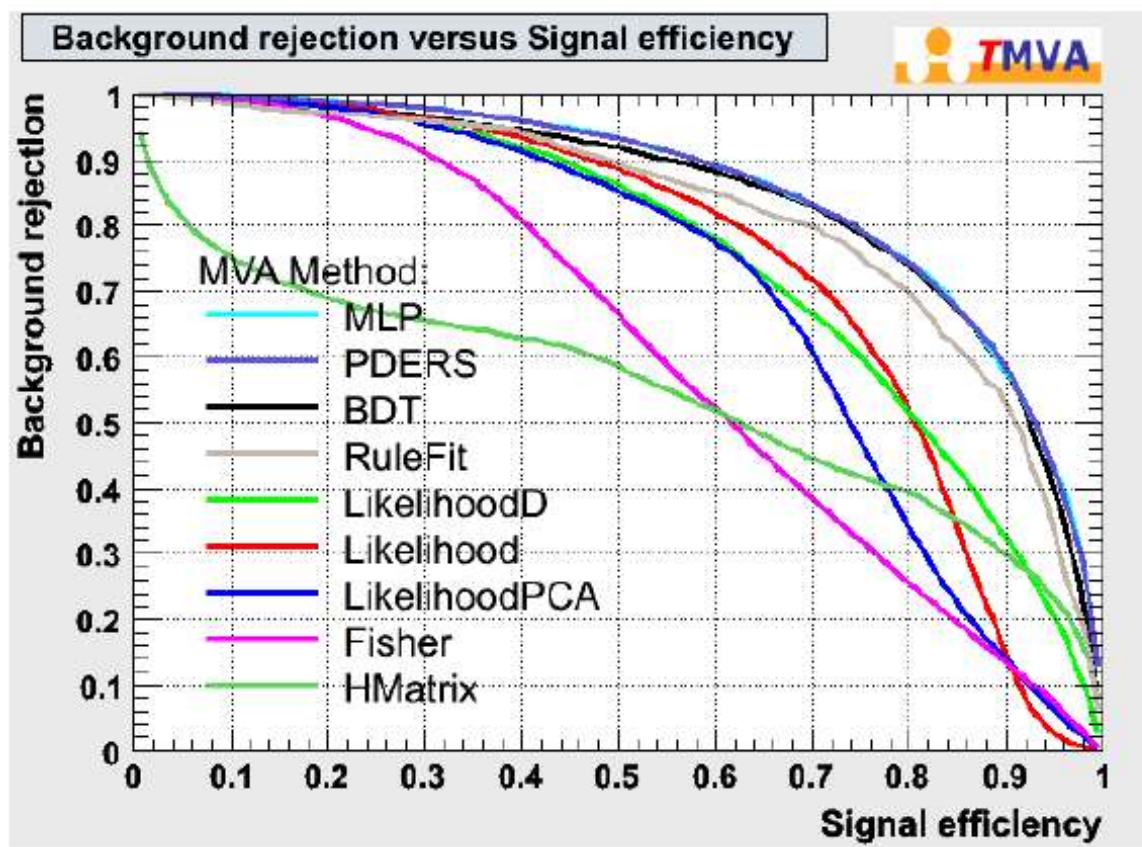
Look at control samples to test joint distributions of inputs.

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.)

Comparing multivariate methods (TMVA)



Choose the best one!

Lecture 4 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.

Fortunately tools to investigate these methods are now widely available.

Extra slides

Imperfect pdf estimation

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

If we use poor estimates to construct the test variable

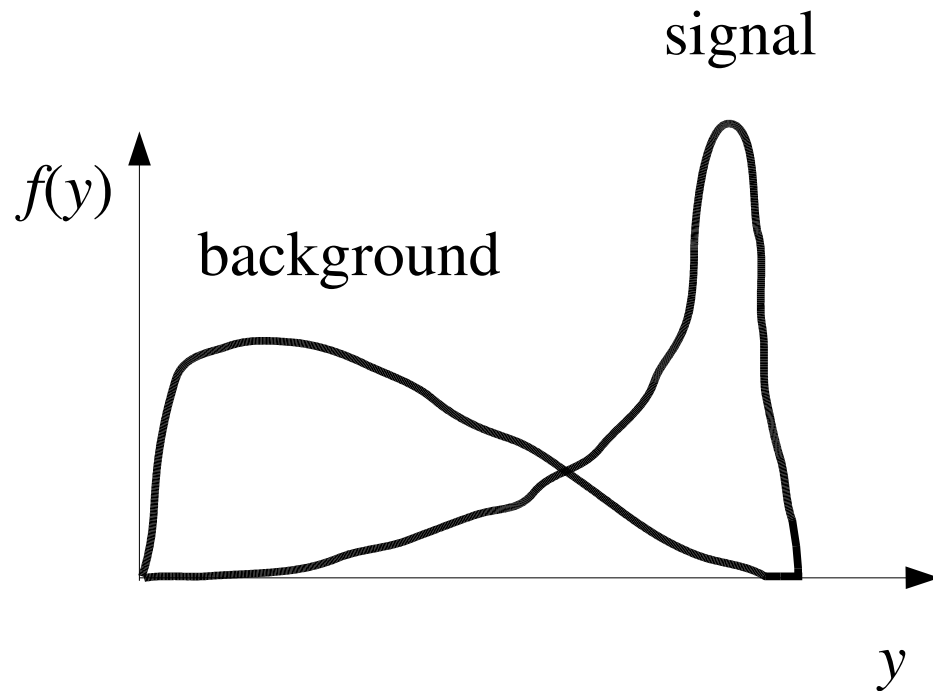
$$y(\vec{x}) = \frac{\hat{p}(\vec{x}|H_0)}{\hat{p}(\vec{x}|H_1)}$$

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

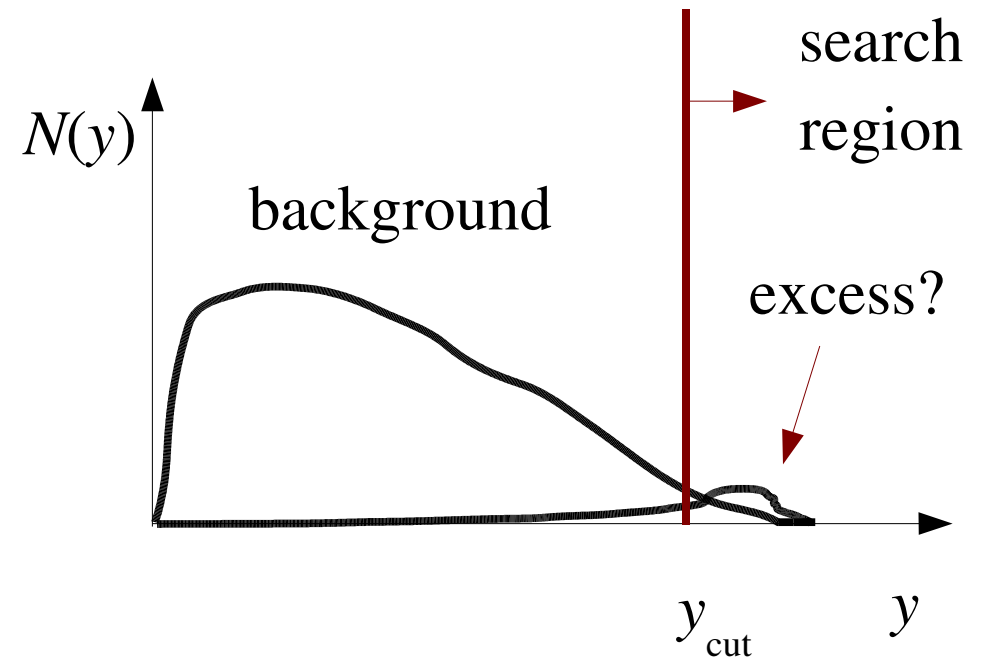
But can this cause us e.g. to make a false discovery?

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 $y(\mathbf{x})$.

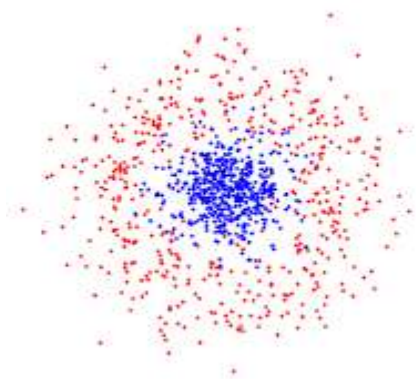
But we do not blindly rely on the MC model for the training data for background; we need to test it by comparing to real data in control samples where no signal is expected.

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

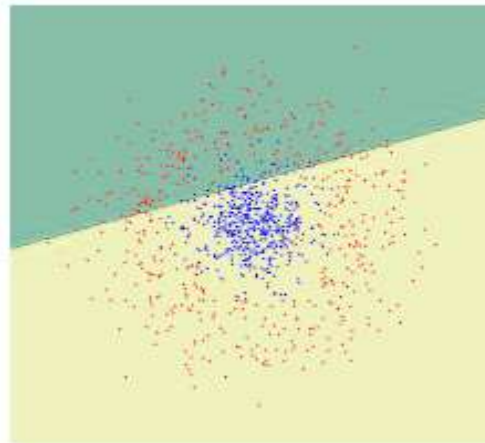
AdaBoost study with linear classifier

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

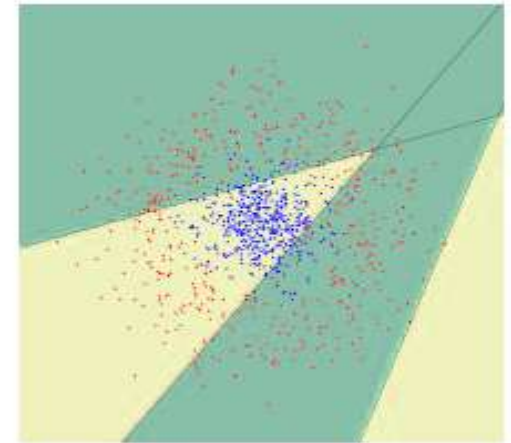
Start with a problem for which a linear classifier is weak:



$t = 1$



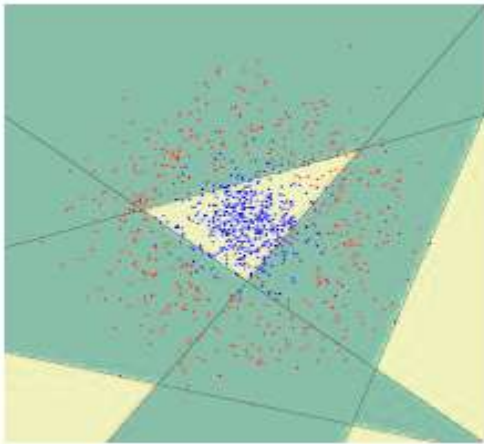
$t = 3$



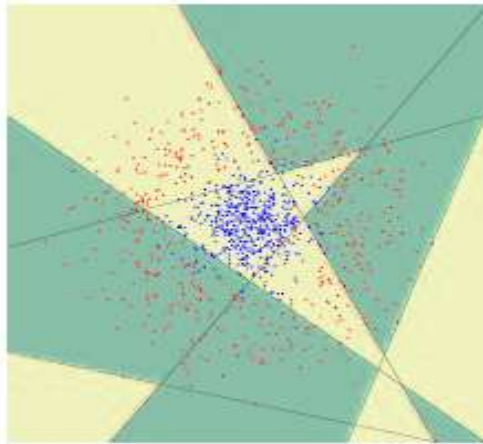
AdaBoost study with linear classifier

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

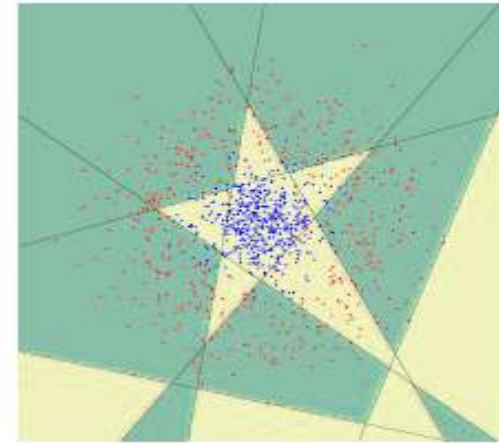
$t = 5$



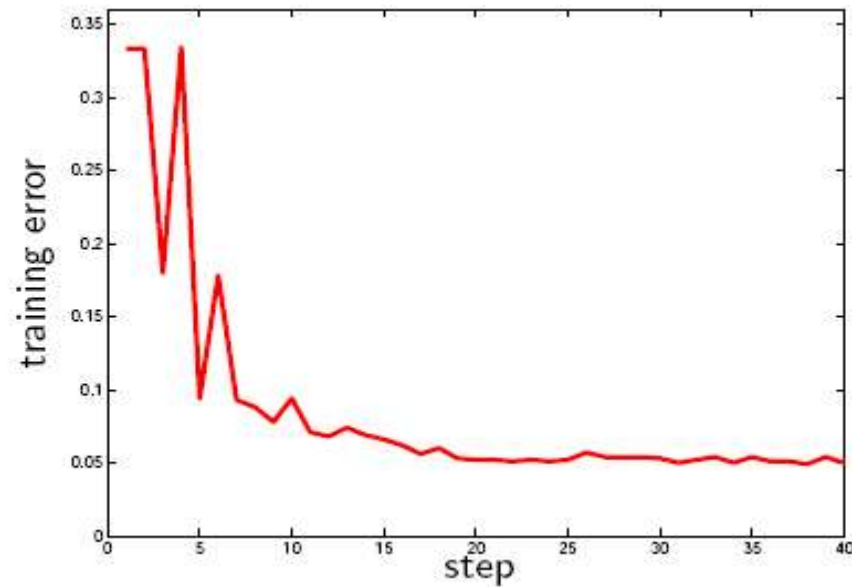
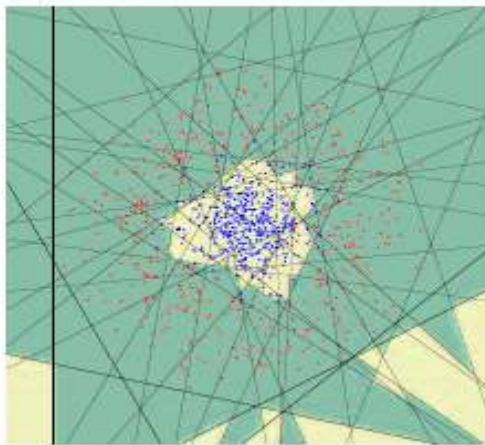
$t = 6$



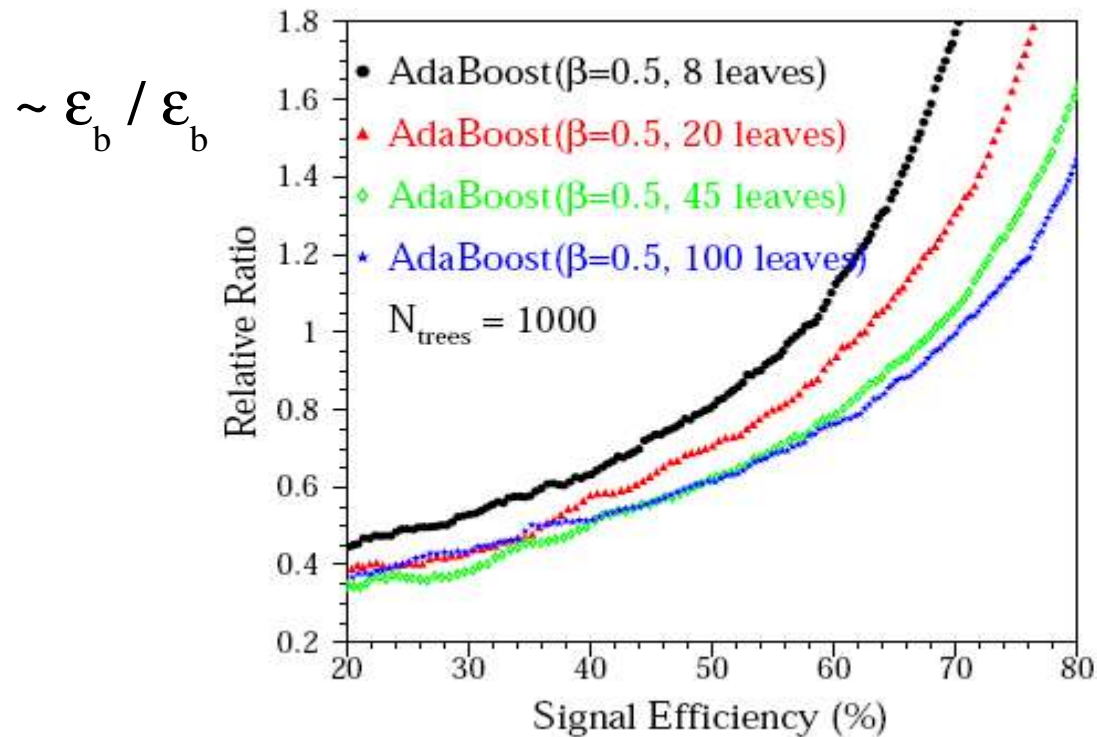
$t = 7$



$t = 40$



MiniBooNE Decision tree performance



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