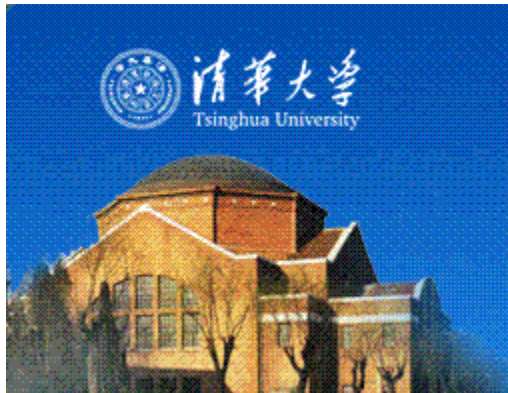


Statistical Methods in Particle Physics

Day 3: Multivariate Methods (II)



清华大学高能物理研究中心

2010年4月12—16日



Glen Cowan

Physics Department

Royal Holloway, University of London

`g.cowan@rhul.ac.uk`

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

Outline of lectures

Day #1: Introduction

Review of probability and Monte Carlo

Review of statistics: parameter estimation

Day #2: Multivariate methods (I)

Event selection as a statistical test

Cut-based, linear discriminant, neural networks

→ Day #3: **Multivariate methods (II)**

More multivariate classifiers: BDT, SVM ,...

Day #4: Significance tests for discovery and limits

Including systematics using profile likelihood

Day #5: Bayesian methods

Bayesian parameter estimation and model selection

Day #3: outline

Nonparametric probability density estimation methods

histograms

kernel density estimation,

K nearest neighbour

Boosted Decision Trees

Support Vector Machines

Review of the problem

Suppose we have events from Monte Carlo of two types, signal and background (each instance of \mathbf{x} is multivariate):

$$\begin{array}{l} \text{generate } \vec{x} \sim p(\vec{x}|\mathbf{s}) \longrightarrow X_1^{\vec{}}, \dots, X_{N_s}^{\vec{}} \\ \text{generate } \vec{x} \sim p(\vec{x}|\mathbf{b}) \longrightarrow X_1^{\vec{}}, \dots, X_{N_b}^{\vec{}} \end{array}$$

Goal is to use these training events to adjust the parameters of a test statistic (“classifier”) $y(\mathbf{x})$, that we can then use on real data to distinguish between the two types.

Yesterday we saw linear classifiers, neural networks and started talking about probability density estimation methods, where we find the classifier from the ratio of estimated pdfs.

Parametric density estimation

If we have a parametric function for one or both of the densities,

$$p(\vec{x}; \theta_1, \dots, \theta_m)$$


then we can estimate the parameters using the training data with e.g. the method of maximum likelihood, i.e., choose the parameter estimates $\hat{\theta}_1, \dots, \hat{\theta}_m$ to be the values that maximize the **likelihood function**:

$$L(\theta_1, \dots, \theta_m) = \prod_{i=1}^N p(\vec{x}_i; \theta_1, \dots, \theta_m)$$

Finally simply take

$$\hat{p}(\vec{x}) = p(\vec{x}; \hat{\theta}_1, \dots, \hat{\theta}_m)$$

Function evaluation generally fast,
storage requirements low.



Product over all training events
(assumes events statistically
independent, not strictly true if
we have multiple candidate
“events” per collision event).

Parametric density estimation (2)

The number of parameters in a reasonable model is usually much smaller than the corresponding number of degrees of freedom in nonparametric methods, so a parametric estimate of the pdf will have higher statistical accuracy for a given amount of training data. Trade off:

few parameters: model not flexible and may not describe data, but parameters accurately determined.

many parameters: model flexible enough to describe the true pdf, but parameter estimates have large statistical errors.

Even if a full parametric model is not available, $p(x)$ may (approximately) factorize into a parametric part for a subset of the variables:

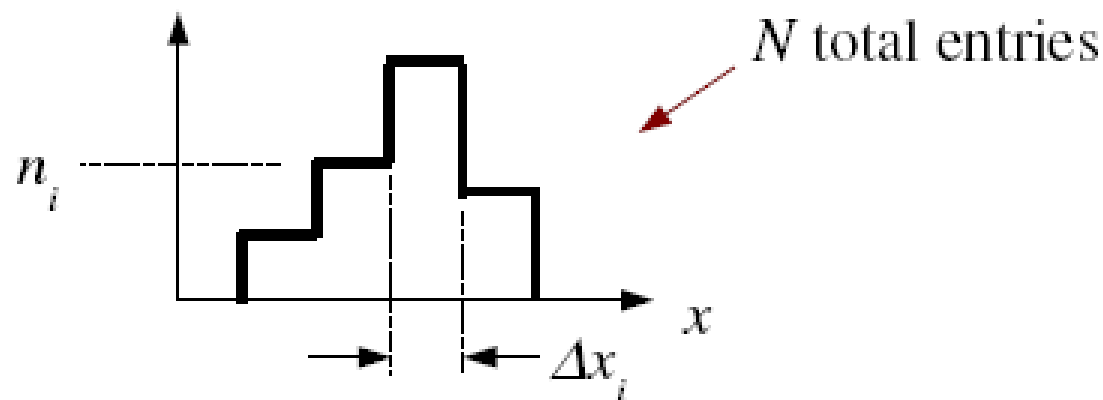
$$p(x_1, \dots, x_n) = p(x_1, \dots, x_s; \theta_1, \dots, \theta_m) q(x_{s+1}, \dots, x_n)$$

so we can mix parametric and non-parametric methods.

Histograms

Start by considering one-dimensional case, goal is to estimate pdf $p(x)$ of continuous r.v. x .

Simplest non-parametric estimate of $p(x)$ is a histogram:



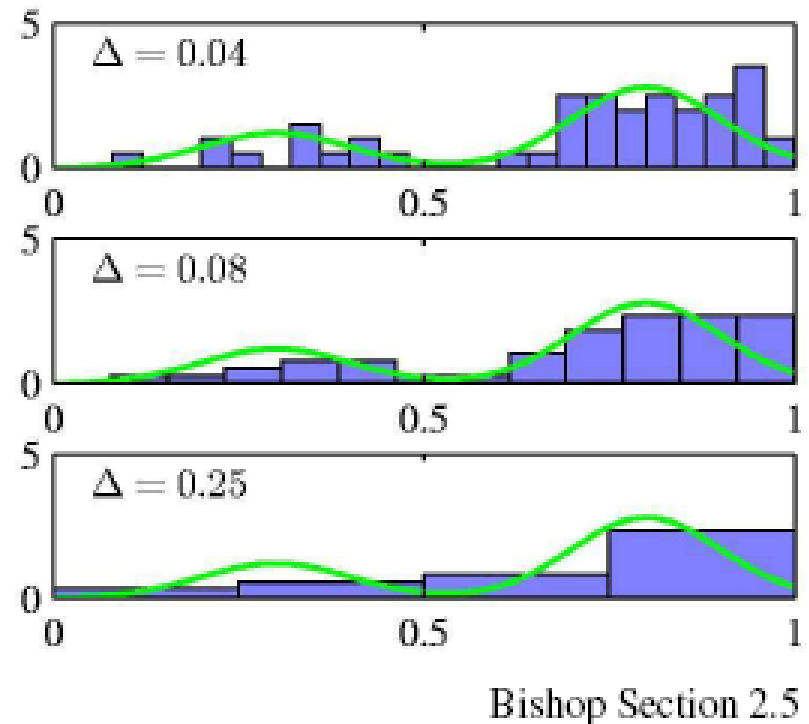
$$\hat{p}(x) = \frac{n_i}{N \Delta x_i} \text{ for } x \text{ in bin } i$$

Histograms (2)

Small bin width: estimate is very spiky, structure not really part of underlying distribution.

Medium bin width: best

Large bin width: too smooth and thus fails to capture e.g. bimodal character of parent distribution



Histograms (3)

Advantages: once histogram computed, the data can be discarded.

Disadvantage: discontinuities at bin edges, scaling with dimensionality.

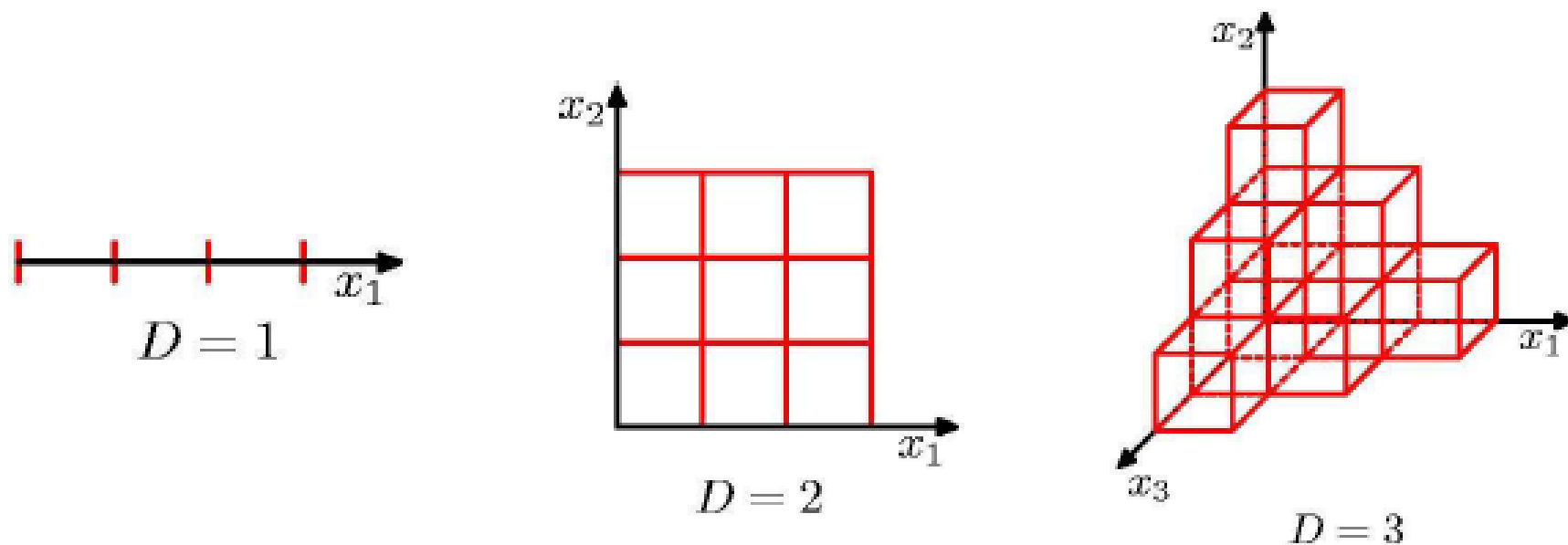
In general we can do much better than histograms, but they still show important features common to many methods:

To estimate pdf at $\mathbf{x}, = (x_1, \dots, x_D)$ we should count the number of events in some **local neighbourhood near \mathbf{x}** (requires definition of “local”, i.e., a distance measure, e.g., Euclidian).

The bin width Δx plays the role of a **smoothing parameter** defining the size of the local neighbourhood. If it is too large, local structure is washed out; too small, and the estimate is subject to statistical fluctuations.

The curse of dimensionality

The difficulty in determining the density in a high-dimensional histogram is an example of the “curse of dimensionality” (Bellman, 1961).



The number of cells in a D -dimensional histogram grows exponentially.

Counting events in a local volume

Consider a small volume V centred about $\mathbf{x} = (x_1, \dots, x_D)$.

This is in contrast to the histogram where the bin edges were fixed.

Suppose from N total events we find K in V .

$$\text{Take as estimate for } p(\mathbf{x}) \quad \hat{p}(\mathbf{x}) = \frac{K}{N V}$$

Two approaches:

Fix V and determine K from the data

Fix K and determine V from the data

Kernels

E.g. take V to be hypercube centered at the \mathbf{x} where we want $p(\mathbf{x})$.

Define $k(\mathbf{u}) = 1$ for $|u_i| \leq 1/2$ and 0 otherwise, $i = 1, \dots, D$

i.e., the function is nonzero inside a unit hypercube centred about \mathbf{x} and zero outside.

$k(\mathbf{u})$ is an example of a **kernel function** (here called a Parzen window).

Kernel density estimators

\mathbf{x} where we
want estimate

\mathbf{x} of i th
training event

Estimate $p(\mathbf{x})$ using:

$$\hat{p}(\mathbf{x}) = \frac{1}{N h^D} \sum_{i=1}^N k\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right)$$

side of hypercube

where we used $V = h^D$ for the volume of the hypercube.

Thus the estimate at \mathbf{x} is the obtained from the sum of N hypercubes,
one centred about each of the data points \mathbf{x}_i

This is an example of a **kernel density estimator** (KDE or Parzen estimator).

Gaussian KDE

The Parzen window KDE has discontinuities at the edges of the hypercubes; we can avoid these with a smoother kernel function e.g., Gaussian:

$$\hat{p}(x) = \frac{1}{N} \sum_{i=1}^N \frac{1}{(2\pi h^2)^{1/2}} \exp\left(\frac{-\|\vec{x} - \vec{x}_i\|^2}{2h^2}\right)$$

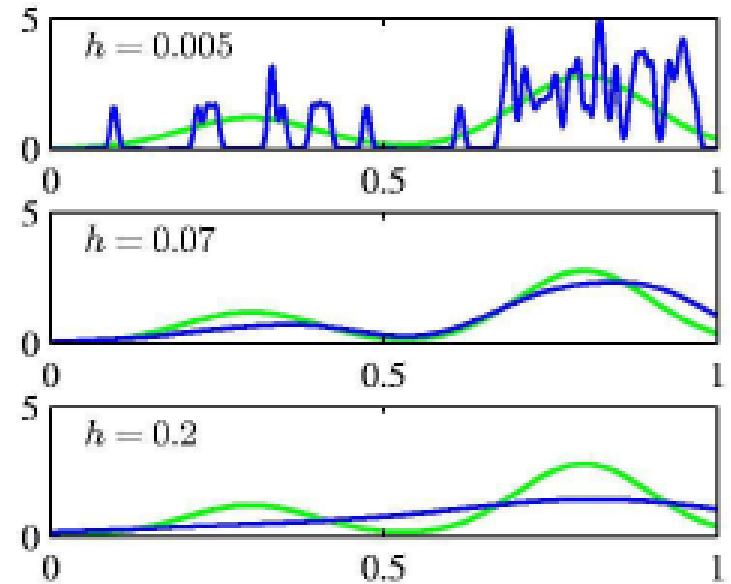
That is, to estimate $p(x)$:

Place a Gaussian of standard deviation h centred about each training data point;

At a given x , add up the contribution from all the Gaussians and divide by N .

Gaussian KDE, choice of h

The Gaussian KDE shows the same basic issues as did the histogram:



Bishop Section 2.5

KDE – general

We can choose any kernel function $k(\mathbf{u})$ as long as it satisfies

$$k(\mathbf{u}) \geq 0,$$

$$\int k(\mathbf{u}) d\mathbf{u} = 1$$

Advantage of KDE: essentially no training!

To get $p(\mathbf{x})$ simply compute the required sum of terms.

Disadvantages: A single function evaluation of $p(\mathbf{x})$ requires carrying out the sum over all events, and the entire data set must be stored.

Expectation value of $\hat{p}(\vec{x})$

To see role of kernel, compute expectation value of $\hat{p}(\vec{x})$

$$\begin{aligned} E[\hat{p}(\mathbf{x})] &= \frac{1}{N h^D} \sum_{i=1}^N E\left[k\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right)\right] = \frac{1}{h^D} E\left[k\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right)\right] \\ &= \frac{1}{h^D} \int k\left(\frac{\mathbf{x}-\mathbf{x}'}{h}\right) p(\mathbf{x}') d\mathbf{x}' \end{aligned}$$

Expectation value of the estimator $\hat{p}(\mathbf{x})$ is the convolution of the true density $p(\mathbf{x})$ with the kernel function.

For $h \rightarrow 0$ the kernel becomes a delta function, and $E[\hat{p}(\mathbf{x})]$ approaches the true density (zero bias). But for finite N the variance $\mathcal{V}[\hat{p}(\mathbf{x})]$ becomes infinite.

Choice of h using mean squared error

Suppose we knew the true $p(\mathbf{x})$ (or had a reference standard). We can compute the Mean Squared Error (MSE) of our estimator:

$$\begin{aligned}\text{MSE}[\hat{p}(\mathbf{x})] &= E[(\hat{p}(\mathbf{x}) - p(\mathbf{x}))^2] \\ &= \underbrace{(E[\hat{p}(\mathbf{x}) - p(\mathbf{x})])^2}_{\text{bias squared}} + \underbrace{E[(\hat{p}(\mathbf{x}) - p(\mathbf{x}))^2]}_{\text{variance}}\end{aligned}$$

We could e.g. choose h so that it minimizes the integral of the MSE over \mathbf{x} (or maybe in some region of interest):

$$\int \text{MSE}[\hat{p}(\mathbf{x})] d\mathbf{x}$$

If both the kernel and $p(\mathbf{x})$ are a Gaussian, this gives $h = \left(\frac{4}{3}\right)^{1/5} \sigma N^{-1/5}$

Adaptive KDE

In the simplest form of KDE the smoothing parameter h is a constant.

In regions high density (lots of events) we want small h so as to not wash out structure.

In regions of low density, small h would lead to statistical fluctuations in the estimate (structure not present in parent distribution).

So we may want to allow the size of the local neighbourhood over which we average to **vary** depending on the local density.

In sample point adaptive KDE the bandwidth becomes a function of $p(\mathbf{x})$:

$$h = \frac{h_0}{\sqrt{p(\mathbf{x})}}$$

KDE boundary issues

Some components of \mathbf{x} may have finite limits. But if we use e.g. a Gaussian kernel, then some of the probability “spills out” of the allowed range.

The probability inside the range is therefore underestimated.

One solution is to renormalize the kernel so that its integral inside the allowed range is equal to unity.

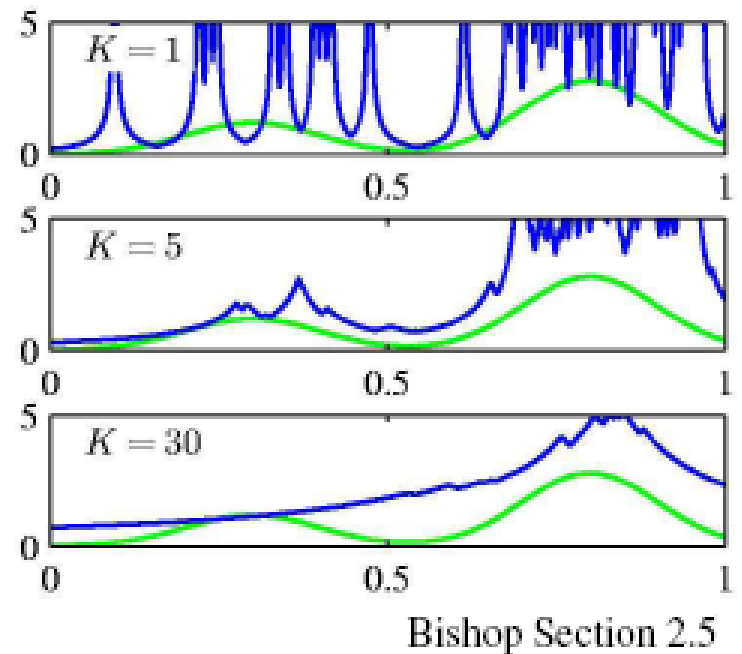
Another option is to “mirror” the distribution about the boundary. The events from outside spilling in compensate those spilling out.

K -nearest neighbour method

Instead of fixing V , consider a fixed number of events K and find the appropriate V such that it just contains K events.

The density estimate is then simply $\hat{p}(\mathbf{x}) = \frac{K}{NV}$

K plays role of smoothing parameter.
Large K means lower statistical error in the estimate of the density,
e.g., $K=100$ gives 10% accuracy.
But large K means you need a bigger volume, estimate is less local.



Estimate of $p(\mathbf{x})$ is not a true pdf – integral over entire space diverges.

K -nearest neighbour method

Example from TMVA manual – here the algorithm is used directly as a classifier. The event type is assigned based on majority vote within the volume.

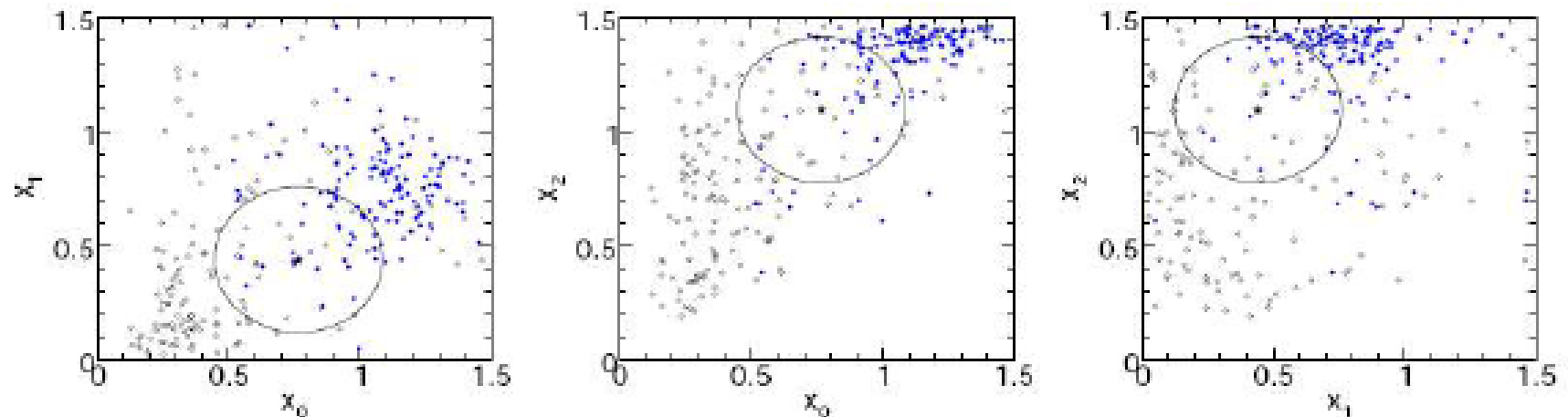


Figure 11: Example for the k -nearest neighbour algorithm in a three-dimensional space (i.e., for three discriminating input variables). The three plots are projections upon the two-dimensional coordinate planes. The full (open) circles are the signal (background) events. The k -NN algorithm searches for 20 nearest points in the *nearest neighborhood* (circle) of the query event, shown as a star. The nearest neighborhood counts 13 signal and 7 background points so that query event may be classified as a signal candidate.

Feature normalization

K -NN algorithms rely on a metric in the input variable space to define the volume.

If there is a great difference in the ranges spanned by some of the variables, then they are implicitly given different weights.

Typically scale the input variables so as to give approximately equal distances between relevant features. Try e.g.

Linear scaling in unit range: $x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \quad (0 \leq x' \leq 1)$

Standardization: $x' = \frac{x - \mu}{\sigma} \quad (\text{zero mean, unit variance})$

Curse of dimensionality for K -NN

Suppose our data are uniformly distributed in a D -dimensional unit cube.

We want a “small” volume to capture a fraction $r = K/N$ of the events.

Make a hypercube local neighbourhood with side e , volume e^D

Its volume fraction is $r = e^D$

The side of an edge of the small volume is $e = r^{1/D}$

For e.g. $r=0.001$ and $D = 30$ the side of the “neighborhood” is 0.8, almost the entire range of the input.

(cf. Hastie, Tibshirani and Friedman p 23.)

Boosted decision trees

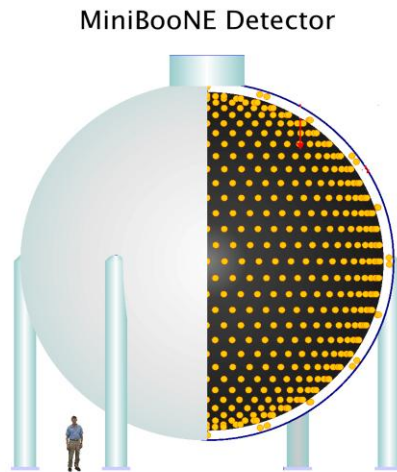
First use of boosted decision trees in HEP was for particle identification for the MiniBoone neutrino oscillation experiment.

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.

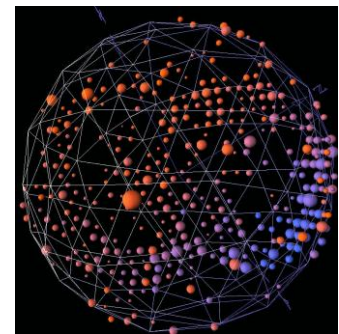
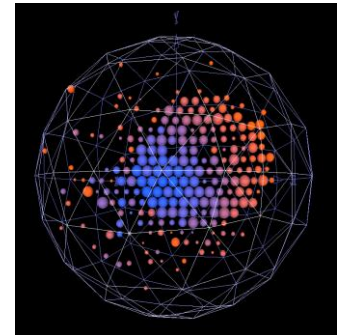
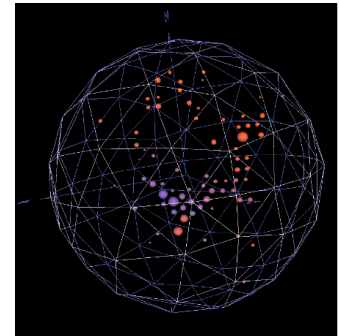
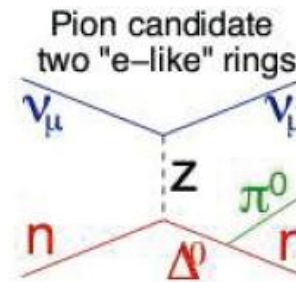
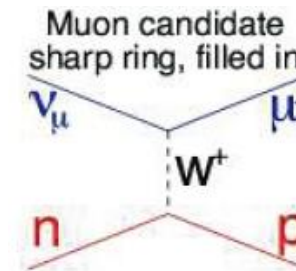
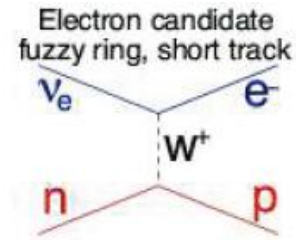
B.P. Roe, H.J. Yang, J. Zhu, Y. Liu, I. Stancu, G. McGregor, “Boosted decision trees as an alternative to artificial neural networks for particle identification”, *physics/0408124*, *NIMA* 543 (2005) 577-584.

Particle i.d. in MiniBooNE

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:

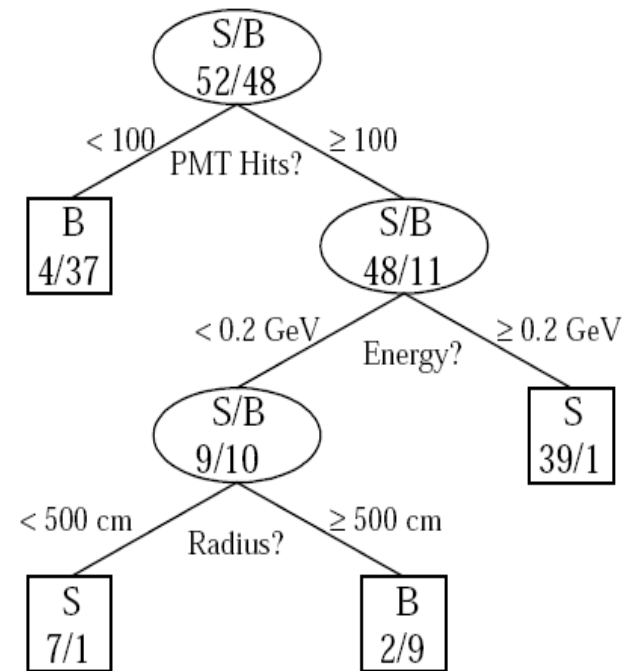
$$P = \frac{\sum_{\text{signal}} w_i}{\sum_{\text{signal}} w_i + \sum_{\text{background}} w_i}$$

where w_i is the weight of the i th 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

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

$\mathbf{x}_1, \dots, \mathbf{x}_N$	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.

AdaBoost

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

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

Train the classifier $f_1(\mathbf{x})$ (e.g. a decision tree) using the weights $w^{(1)}$ so as to minimize the classification error rate,

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

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

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

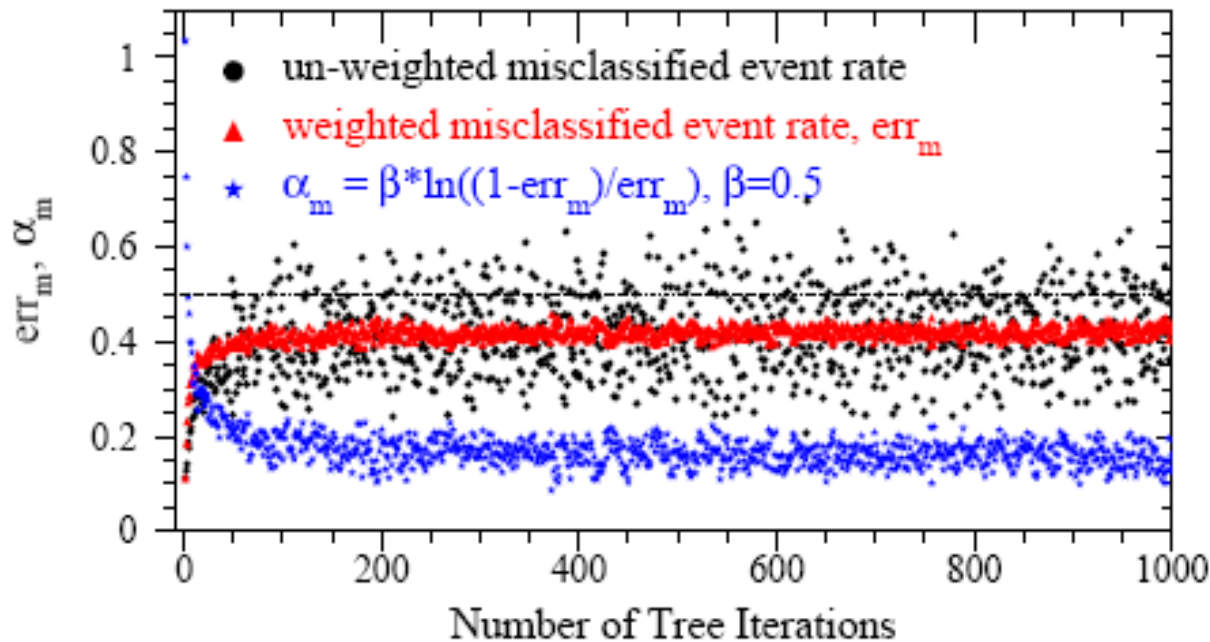
$i = \text{event index}$ $k = \text{training sample index}$ Normalize so that $\sum_i w_i^{(k+1)} = 1$

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 $\sim 0.4 - 0.45$

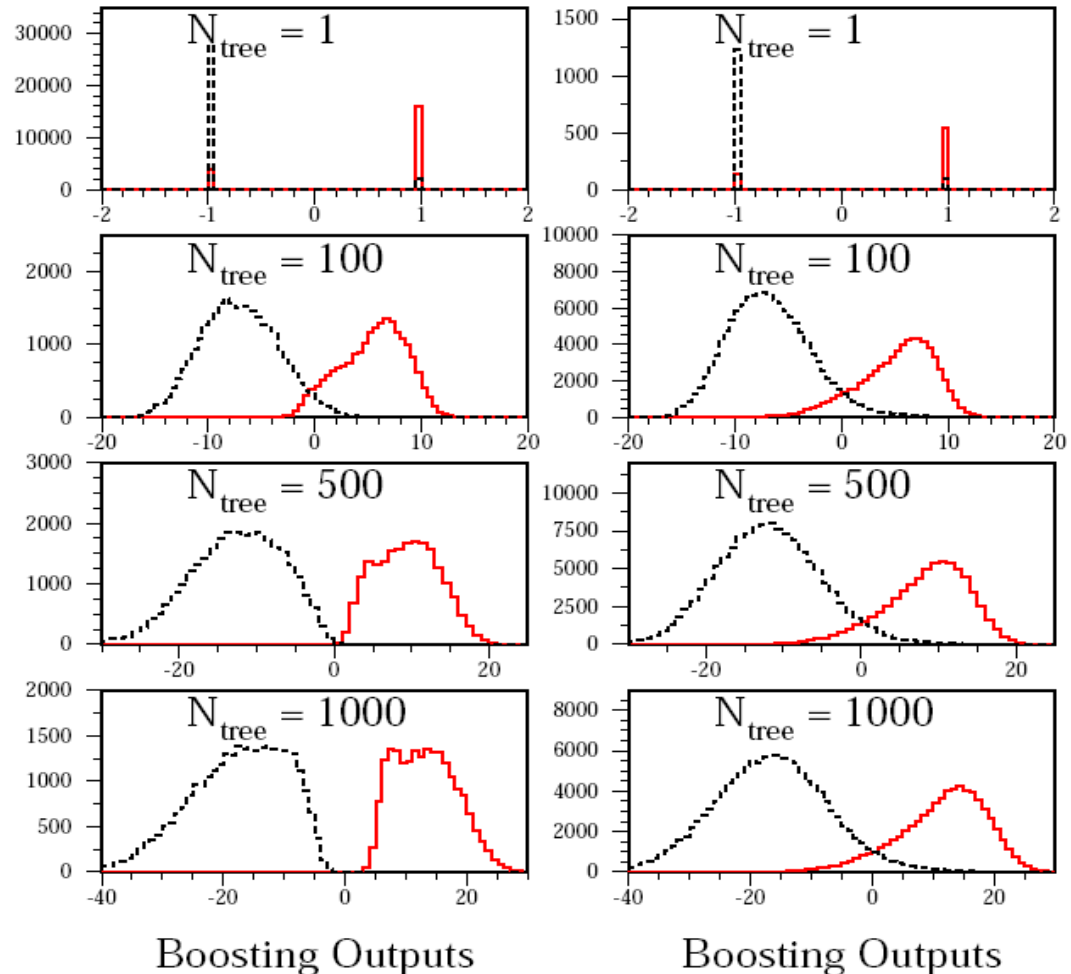


B. Roe et al., NIM 543 (2005) 577

Monitoring overtraining

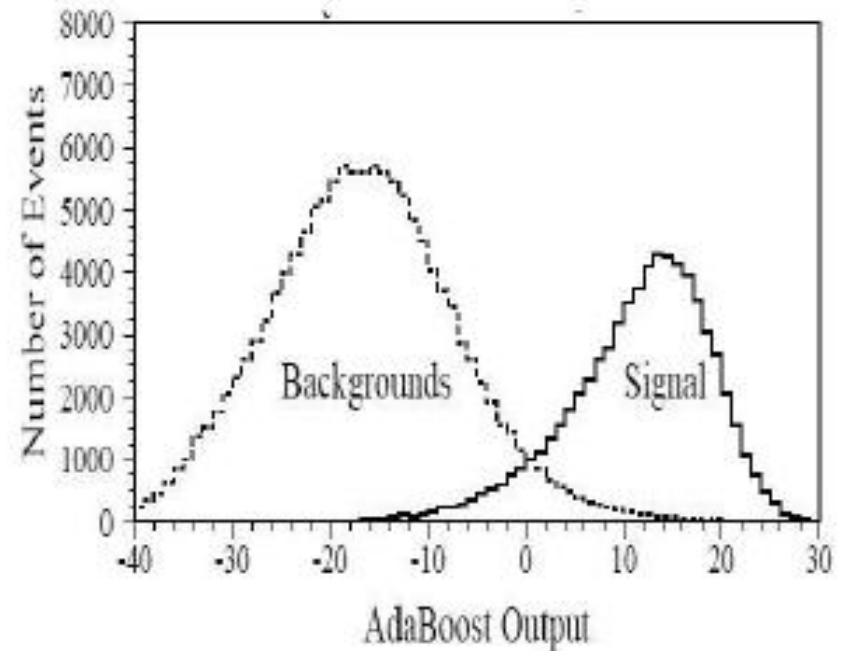
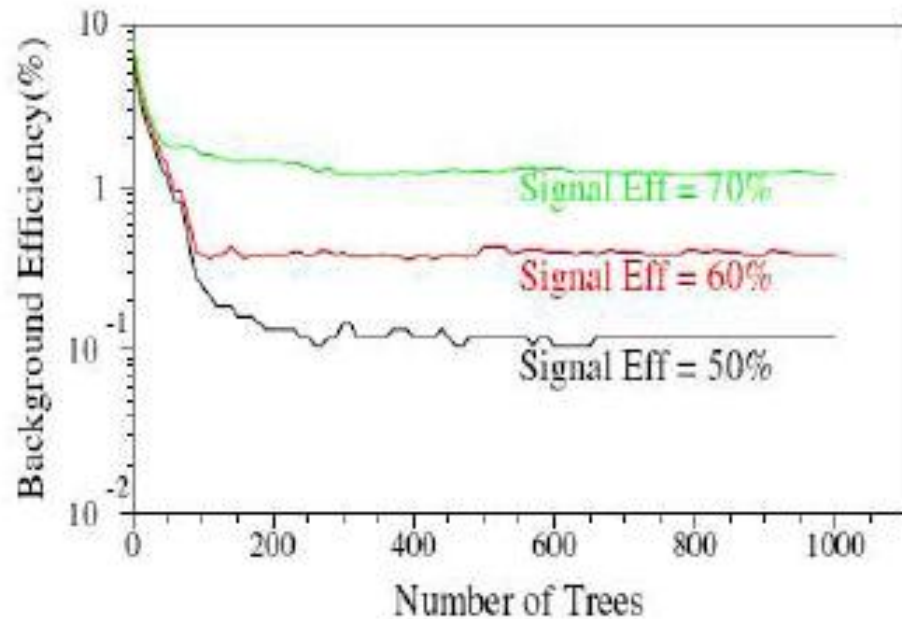
From MiniBooNE
example:
Performance stable
after a few hundred
trees.

Training MC Samples .VS. Testing MC Samples $\times 10^2$



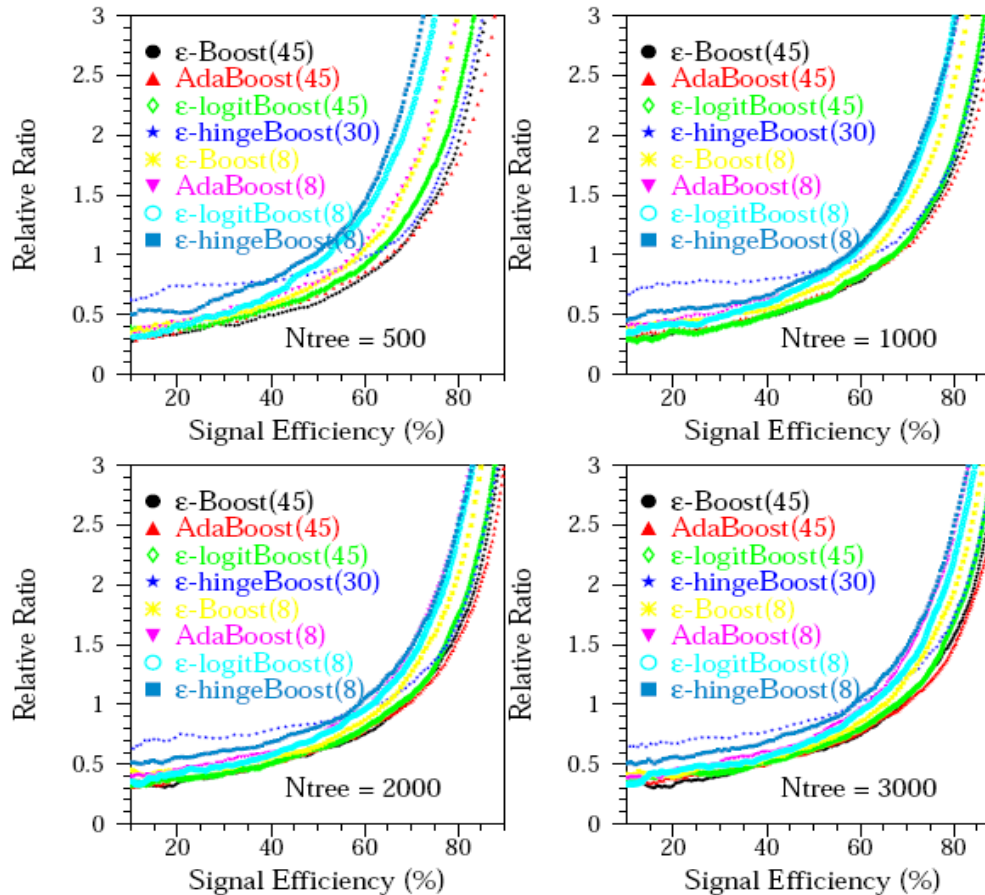
Monitoring overtraining (2)

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 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...).

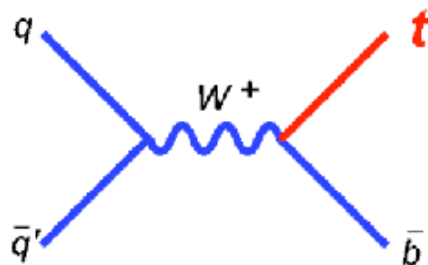
If a tree has only a few leaves it is easy to visualize (but rarely use only a single tree).

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

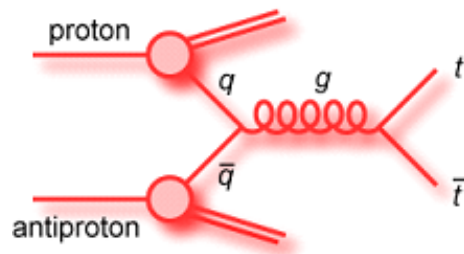
Other ways of combining weaker classifiers: Bagging (Bootstrap-Aggregating), generates the ensemble of classifiers by random sampling with replacement from the full training sample.

Single top quark production (CDF/D0)

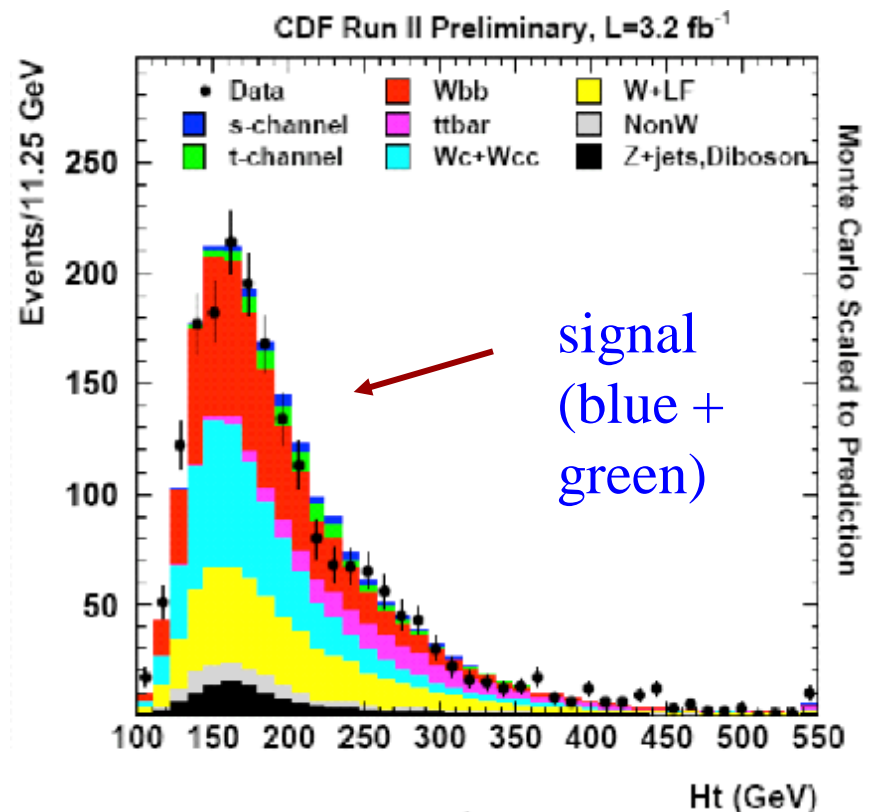
Top quark discovered in pairs, but SM predicts single top production.



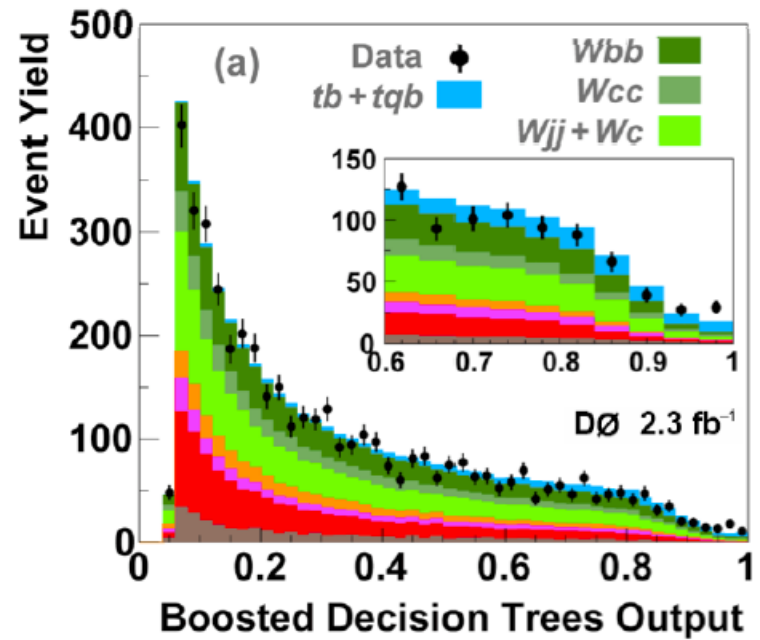
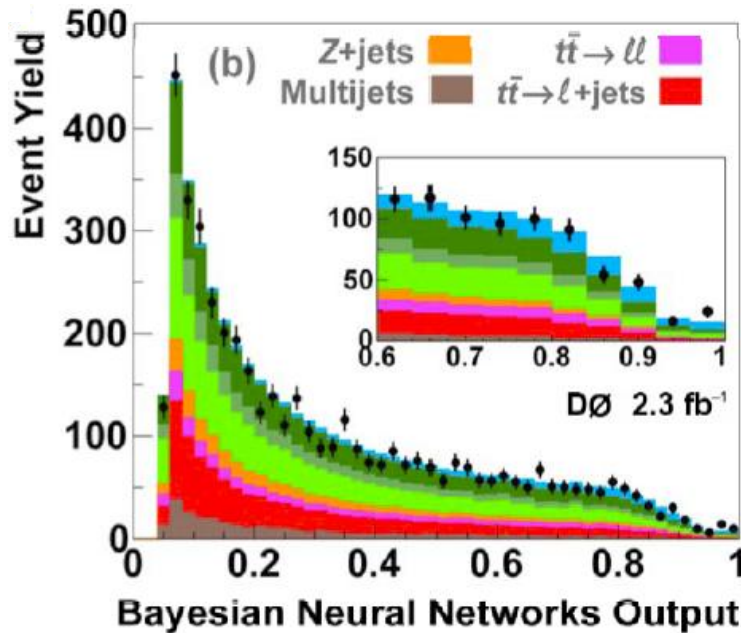
Pair-produced tops are now a background process.



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



Different classifiers for single top



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

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

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

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

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

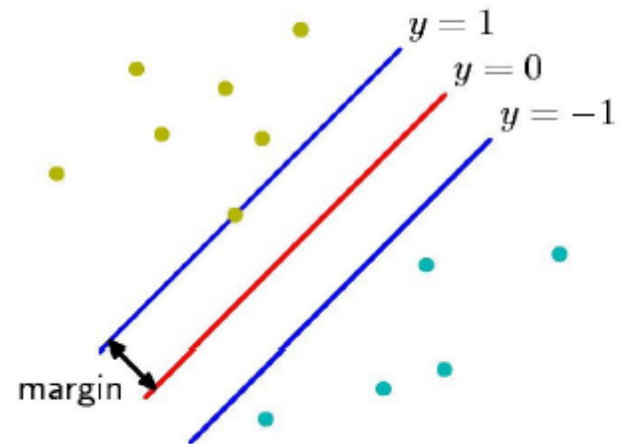
Suppose the classes can be separated by a hyperplane defined by a normal vector \mathbf{w} and scalar offset b (the “bias”). We have

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

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

or equivalently

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

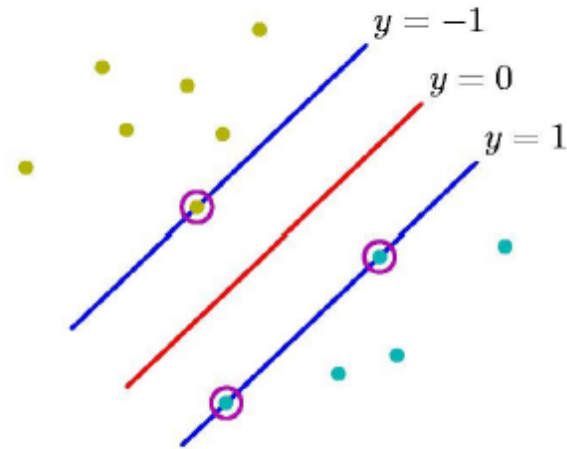


Bishop Ch. 7

Margin and support vectors

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

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

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

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

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 :

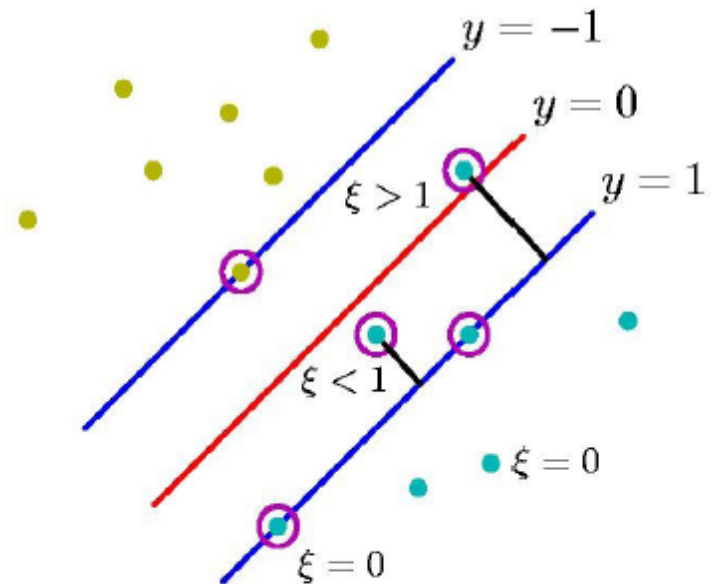
$$y_i(\mathbf{x}_i \cdot \mathbf{w} + b) + \xi_i - 1 \geq 0 \text{ with } \xi_i \geq 0 \text{ for all } i$$

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

$$\vec{\phi}(\mathbf{x}) = (\phi_1(\mathbf{x}), \dots, \phi_m(\mathbf{x}))$$

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

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

so in the feature space this becomes

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

The Kernel trick

How do the dot products help? It turns 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

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

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)

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

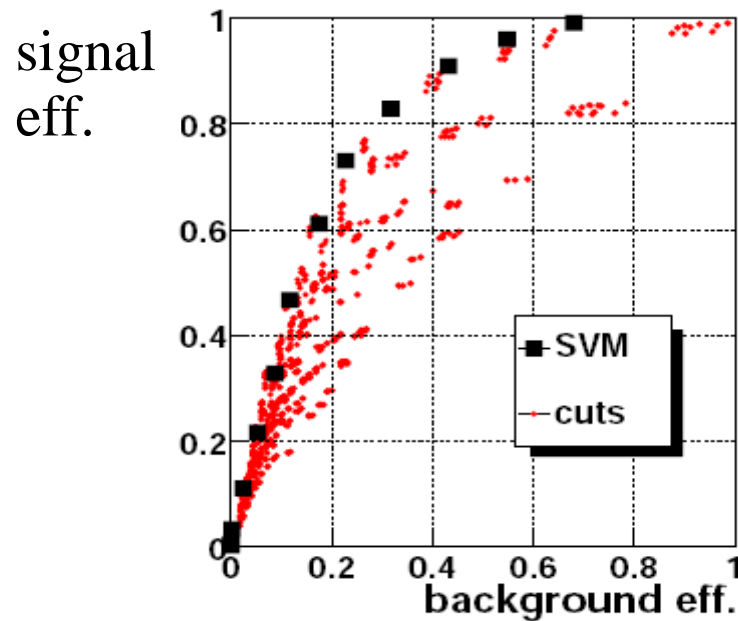
The training is relatively straightforward because, in contrast to neural networks, the function to be minimized has a single global minimum.

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 analysis (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

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

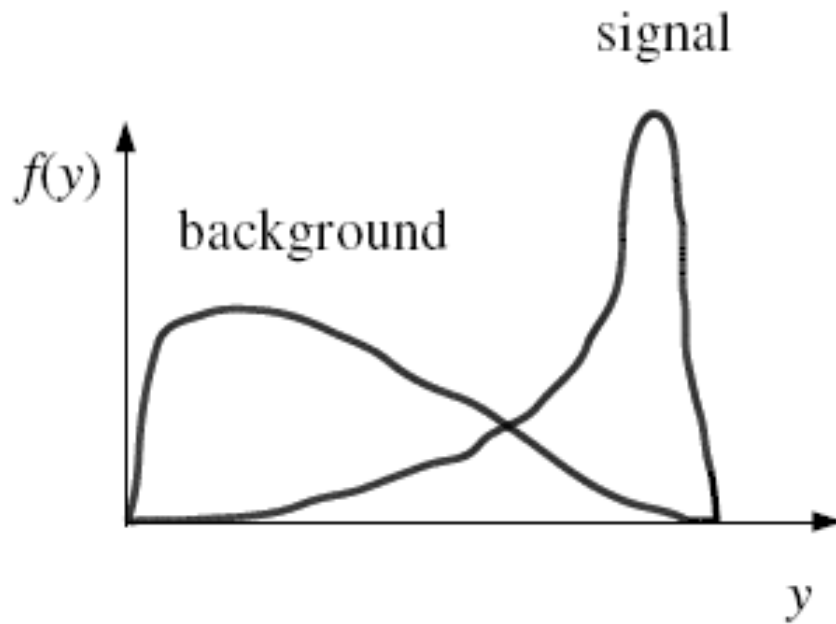
$$y(\vec{x}) = \frac{\hat{p}(\vec{x}|H_0)}{\hat{p}(\vec{x}|H_1)}$$

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

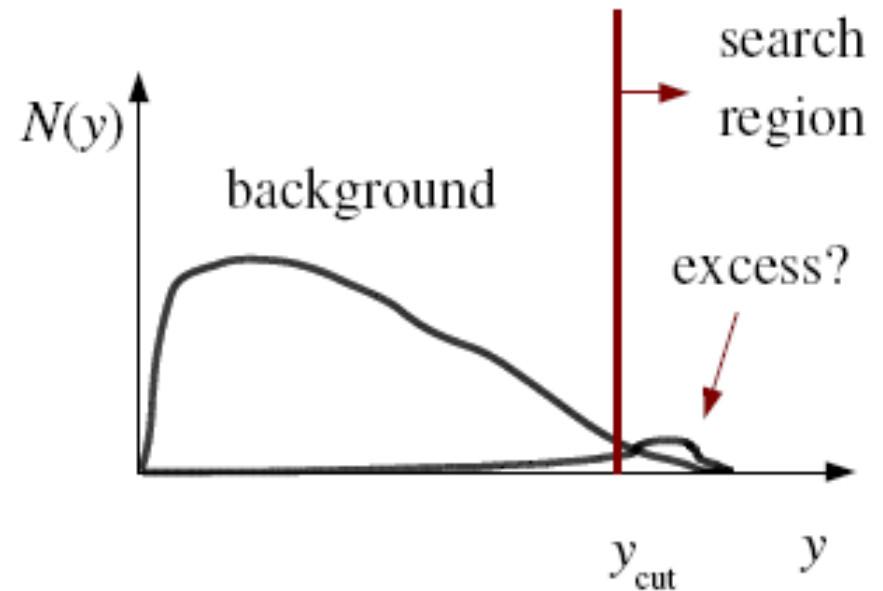
But can this cause us e.g. to make a false discovery?

Even if the estimate of $p(\mathbf{x})$ 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 $y(\mathbf{x})$.

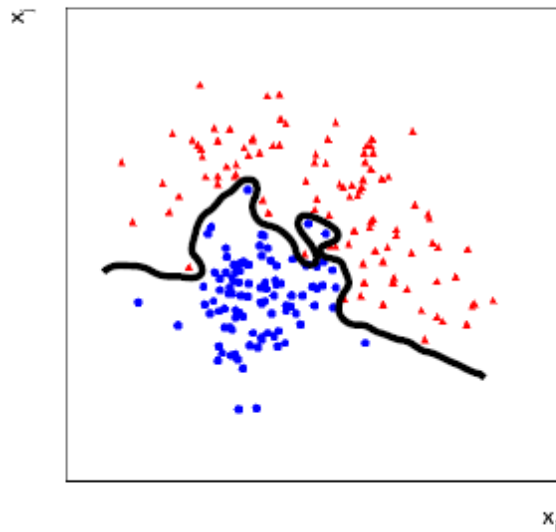
But we do not blindly rely on the MC model for the training data for background; we need to test it by comparing to real data in control samples where no signal is expected.

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

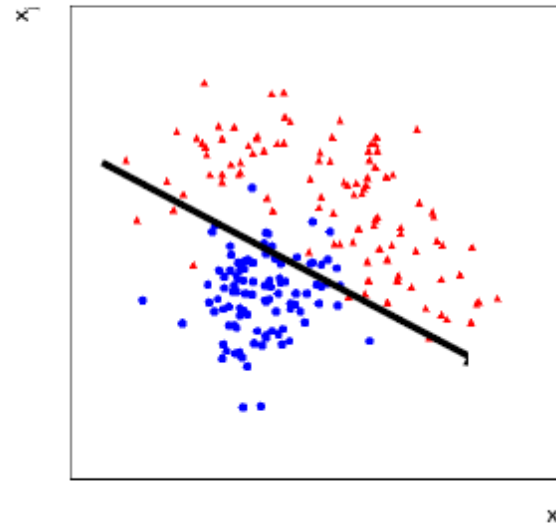
Decision boundary flexibility

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.

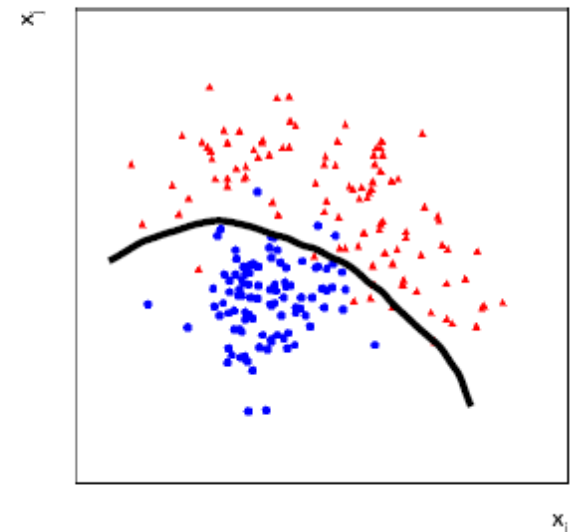
Goal is to determine the boundary using a finite amount of training data so as to best separate between the event types for an unseen data sample.



overtraining



boundary too rigid



good trade-off