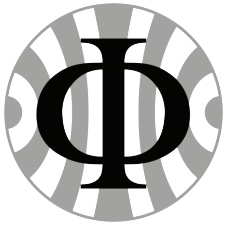
Statistical Methods in Particle Physics / WS 13

Lecture XI

Multivariate Methods

Niklaus Berger

Physics Institute, University of Heidelberg



Part XII:

Multivariate Methods

Classification problems

- Start with a large data sample
  (millions or billions of collisions or decays per second)
- Want to look at a rare or very rare process
  (a few Higgses per day, a few μ→eee decays per year)
- Need to pump up the signal-to-background ratio
  (at good signal efficiency)
- Start with the trigger
  (only record interesting events - a few hundred per second)
- Perform event selection on recorded data
  (topic for today)

12.1. Ideal case: PDFs completely known

If the signal and background pdf are both known:

Neyman-Pearson Lemma:

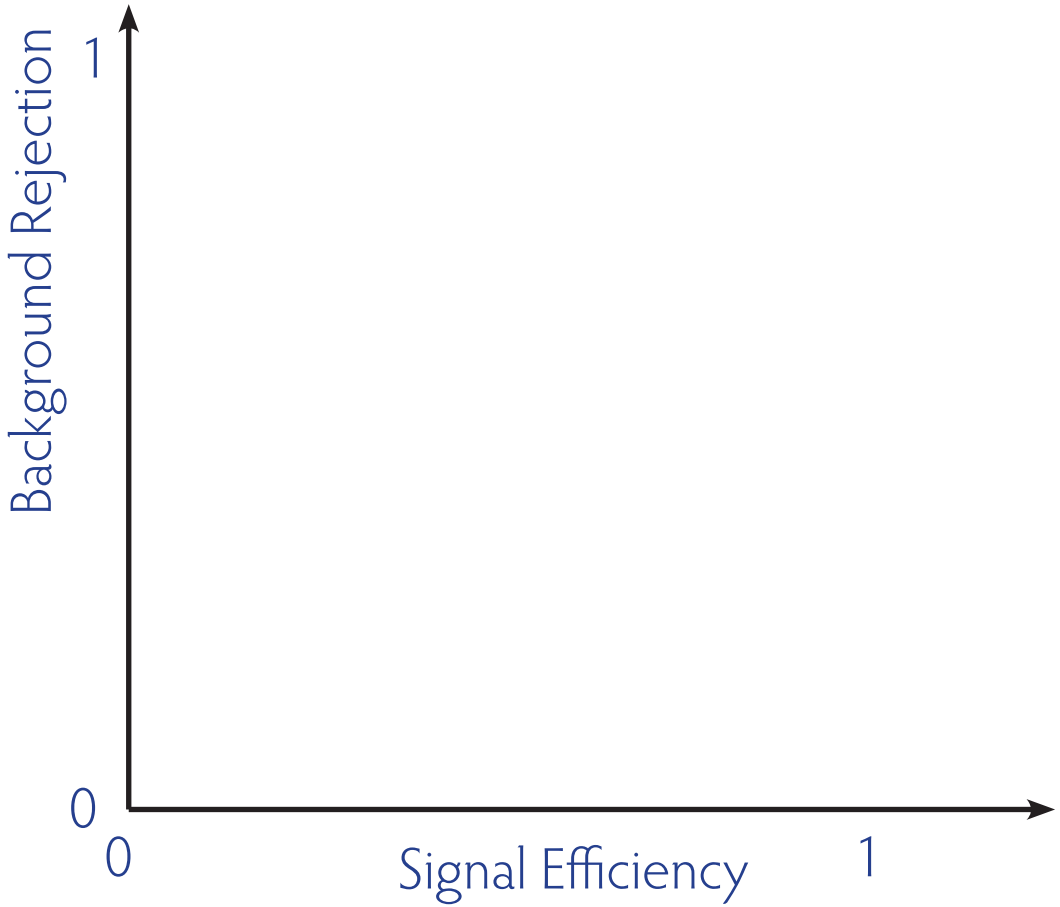
Likelihood ratio: $y(x) = P(x | S) / P(x | B)$

is the best possible selection criterion

How well we can select is given by the overlap of the PDFs

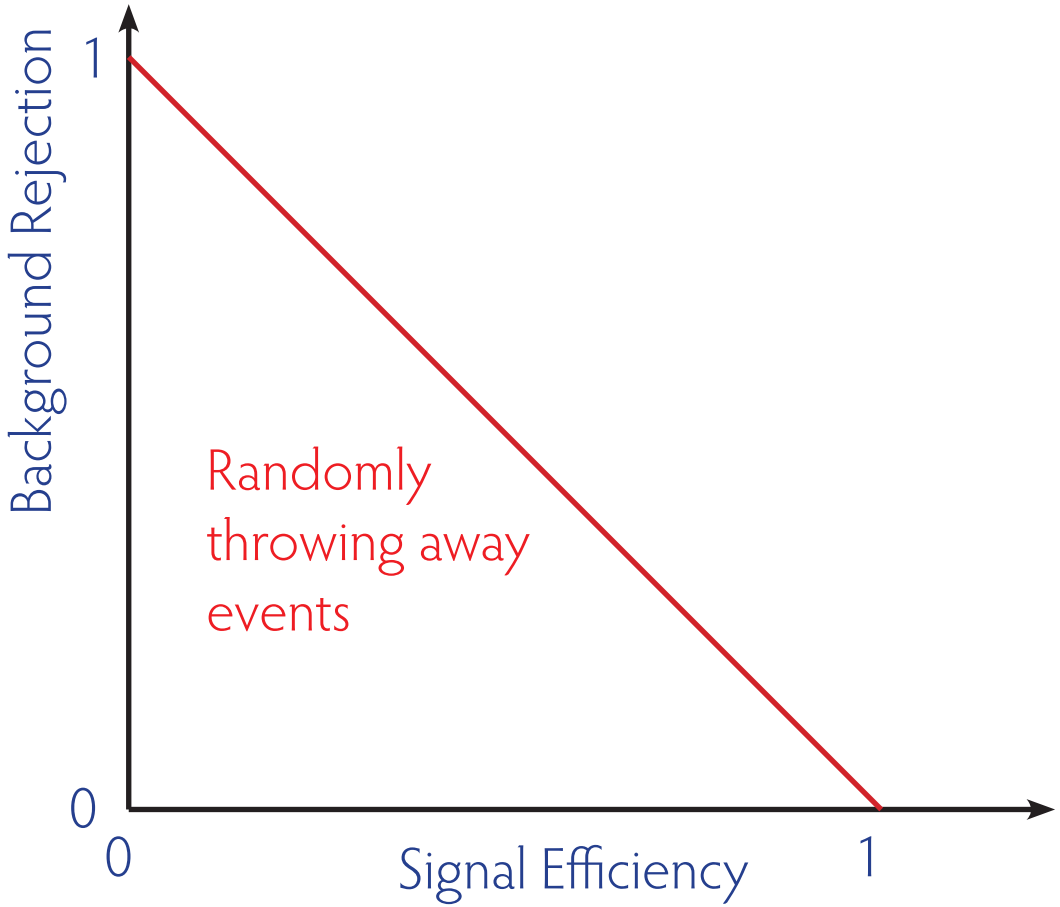
12.2. Goodness of selection: ROC Curves

- Receiver Operating Characteristics - originally from signal transmission in electrical engineering



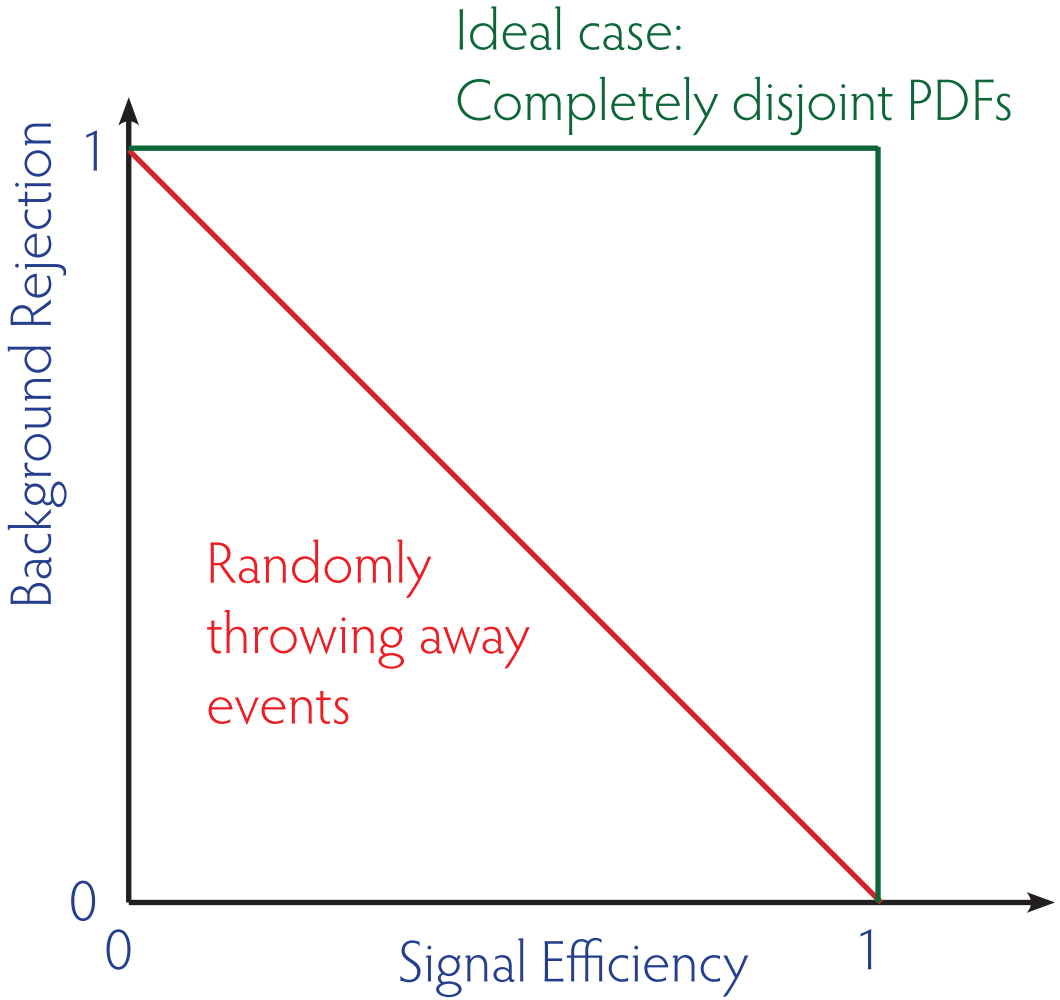
ROC Curves

- Receiver Operating Characteristics - originally from signal transmission in electrical engineering



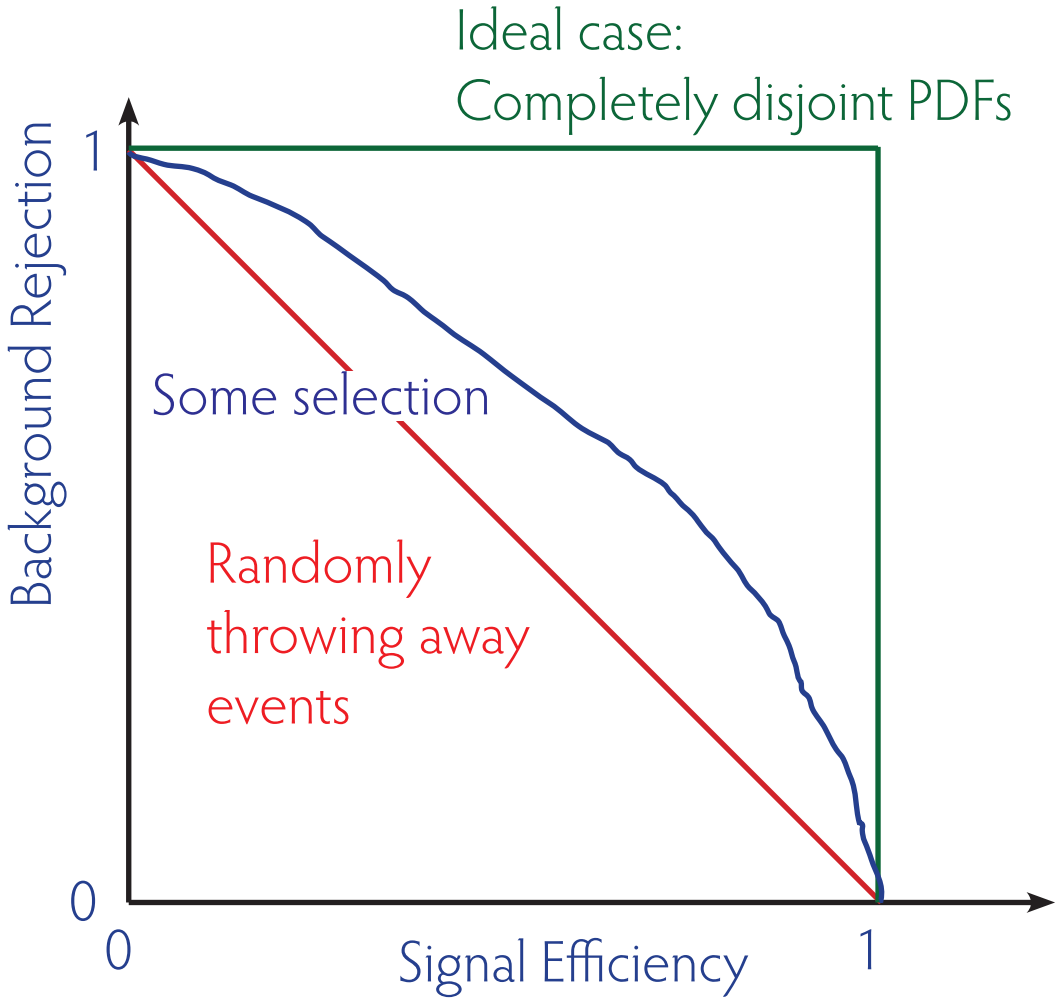
ROC Curves

- Receiver Operating Characteristics - originally from signal transmission in electrical engineering



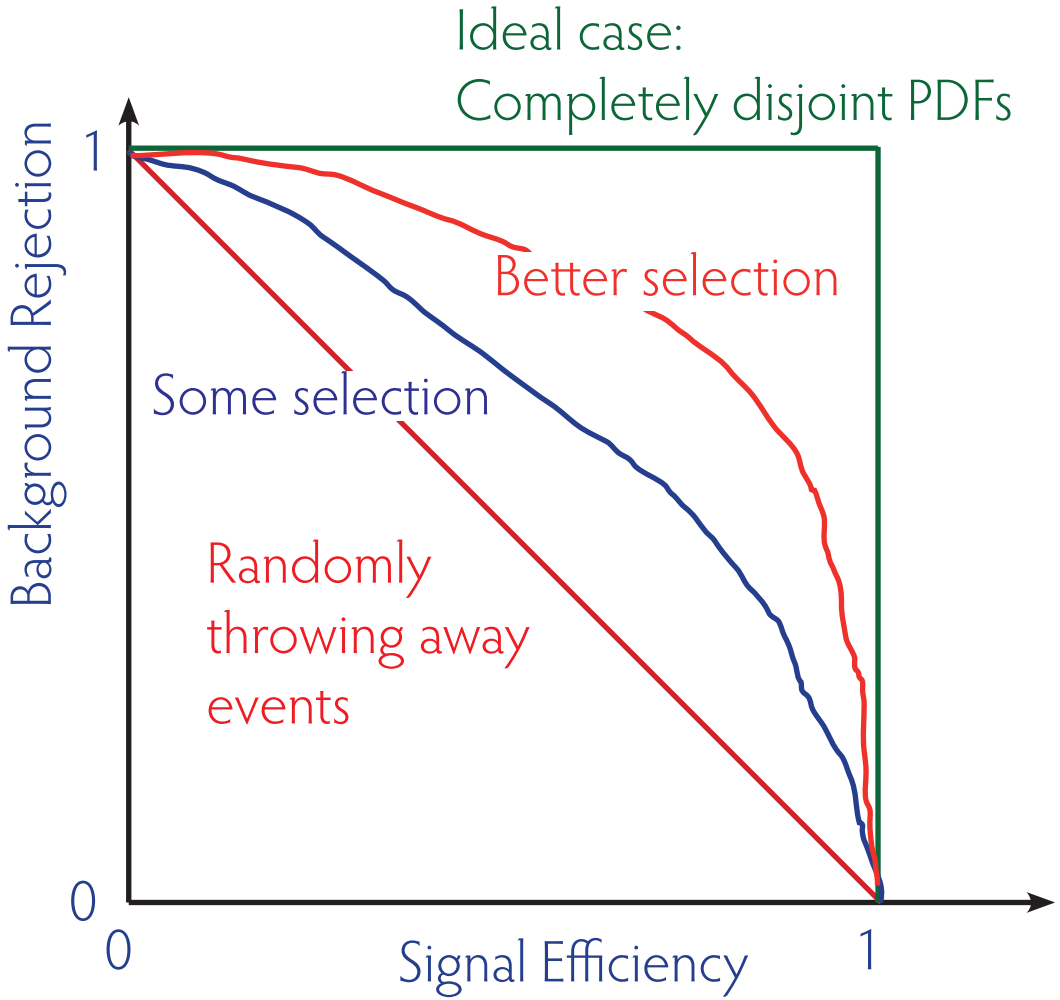
ROC Curves

- Receiver Operating Characteristics - originally from signal transmission in electrical engineering



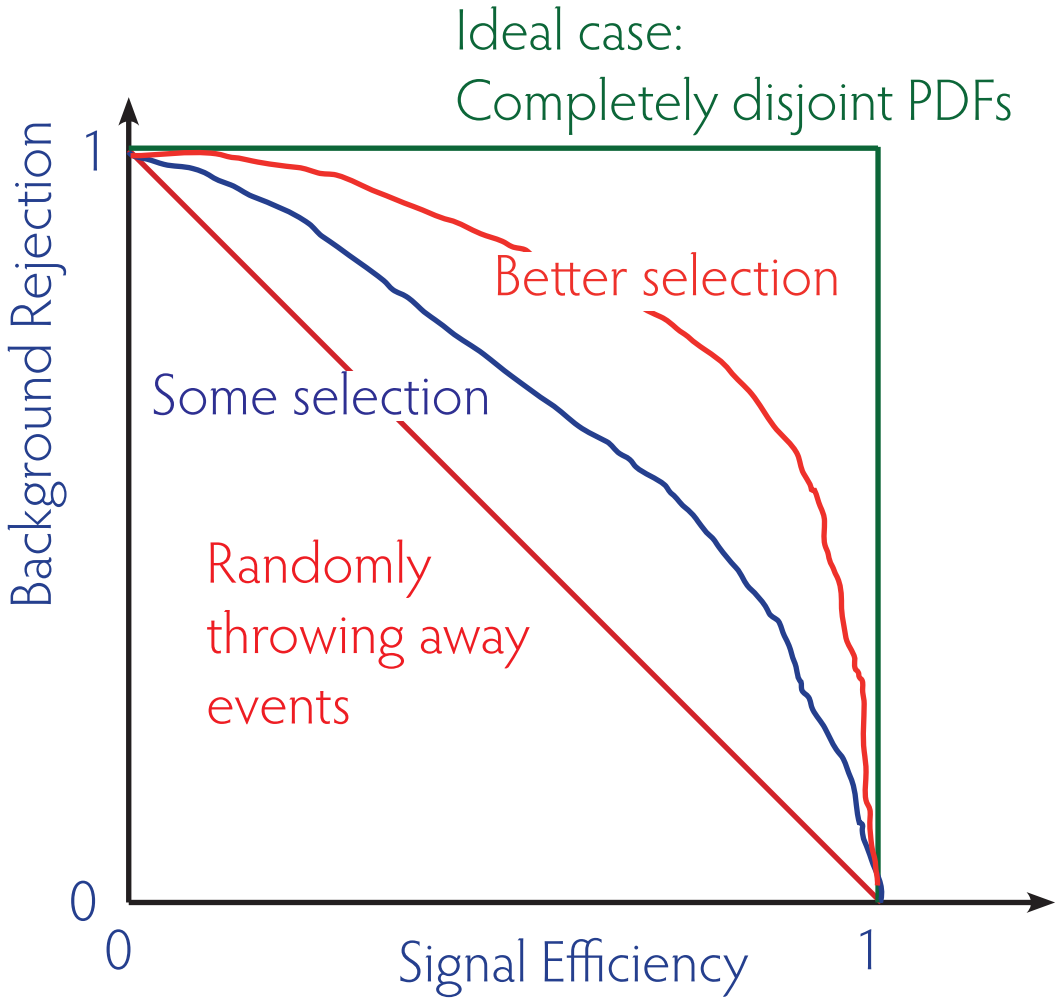
ROC Curves

- Receiver Operating Characteristics - originally from signal transmission in electrical engineering



ROC Curves

- Receiver Operating Characteristics - originally from signal transmission in electrical engineering



- How far you can go to the upper right is limited by Neyman-Pearson
- Rest of this lecture: Find good selections if PDFs are not known

12.3. Cut based selections

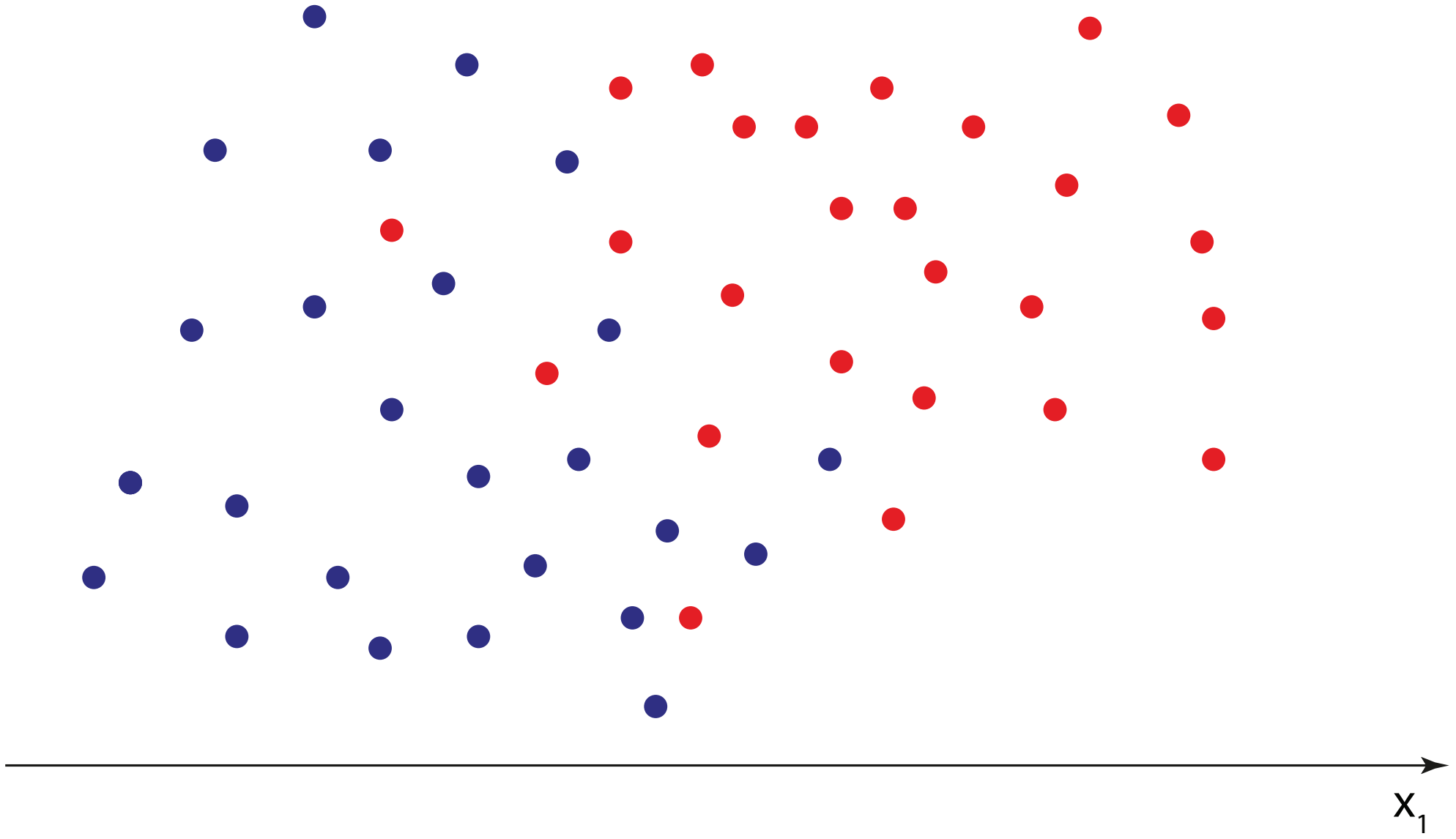
Choose sensible variables and throw away events outside of certain boundaries

How to choose cuts:

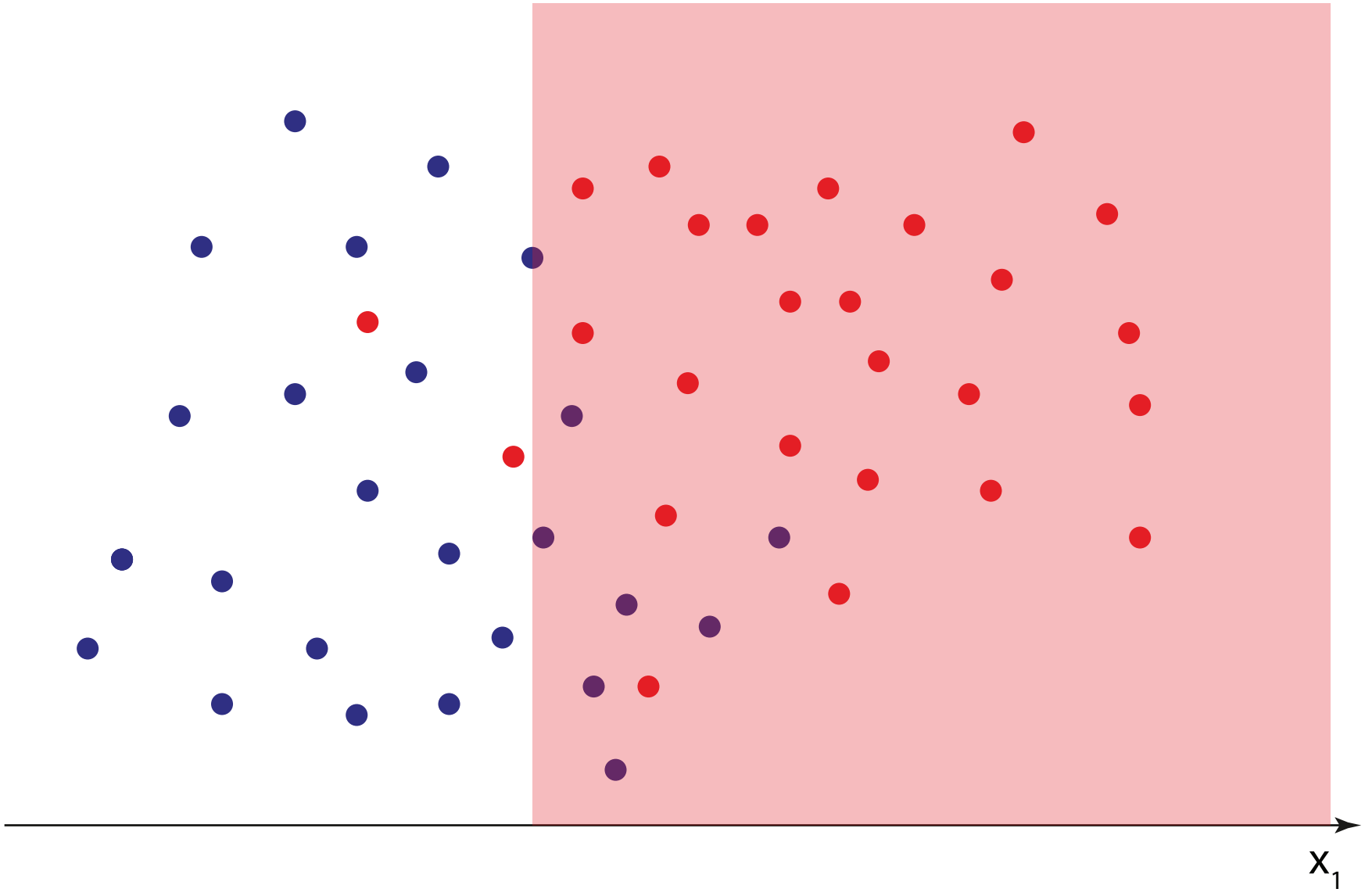
- Physically sensible cuts (e.g. three sigma around the π^0 mass)
- From looking at signal and background MC
- From looking at signal MC and backgrounds from (signal free) control regions

- NOT by choosing cuts such that the signal peak in the data looks nice!

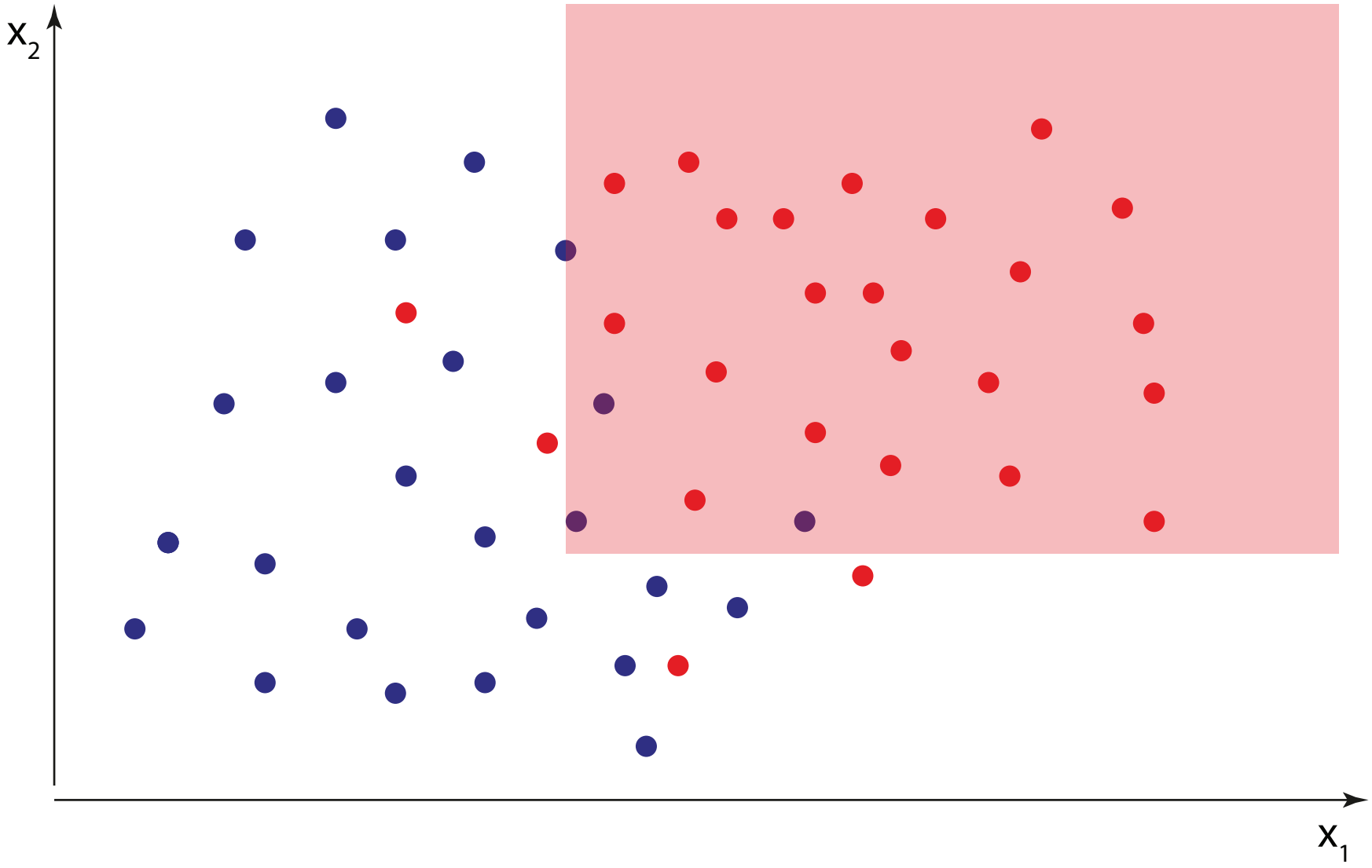
Cut based selections



Cut based selections



Cut based selections



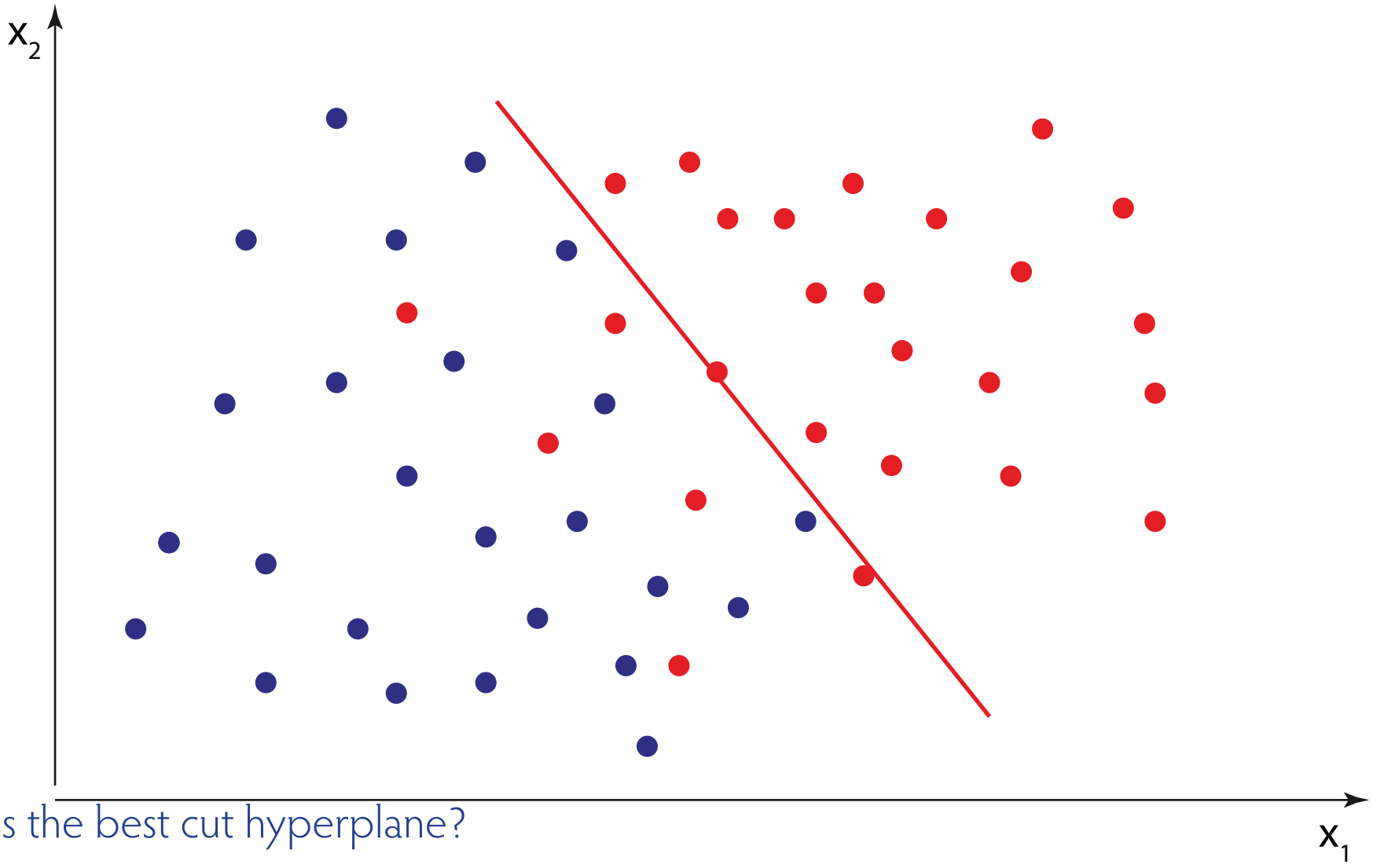
More than cut based selections

Of course, cut areas need not be (hyper-) rectangles

In high dimensionality, hard to find good cuts by eye

- Can have cuts at angles with axes:
  - Fisher discriminant
- Can have nonlinear cut surfaces:
  - Estimate PDF from training sample:
    - Kernel density estimators
  - Try to find best boundary by machine learning:
    - Neural networks
    - Boosted decision trees
    - (and many more)
- For all these: Need a training sample independent of the data

12.4. Fisher discriminant



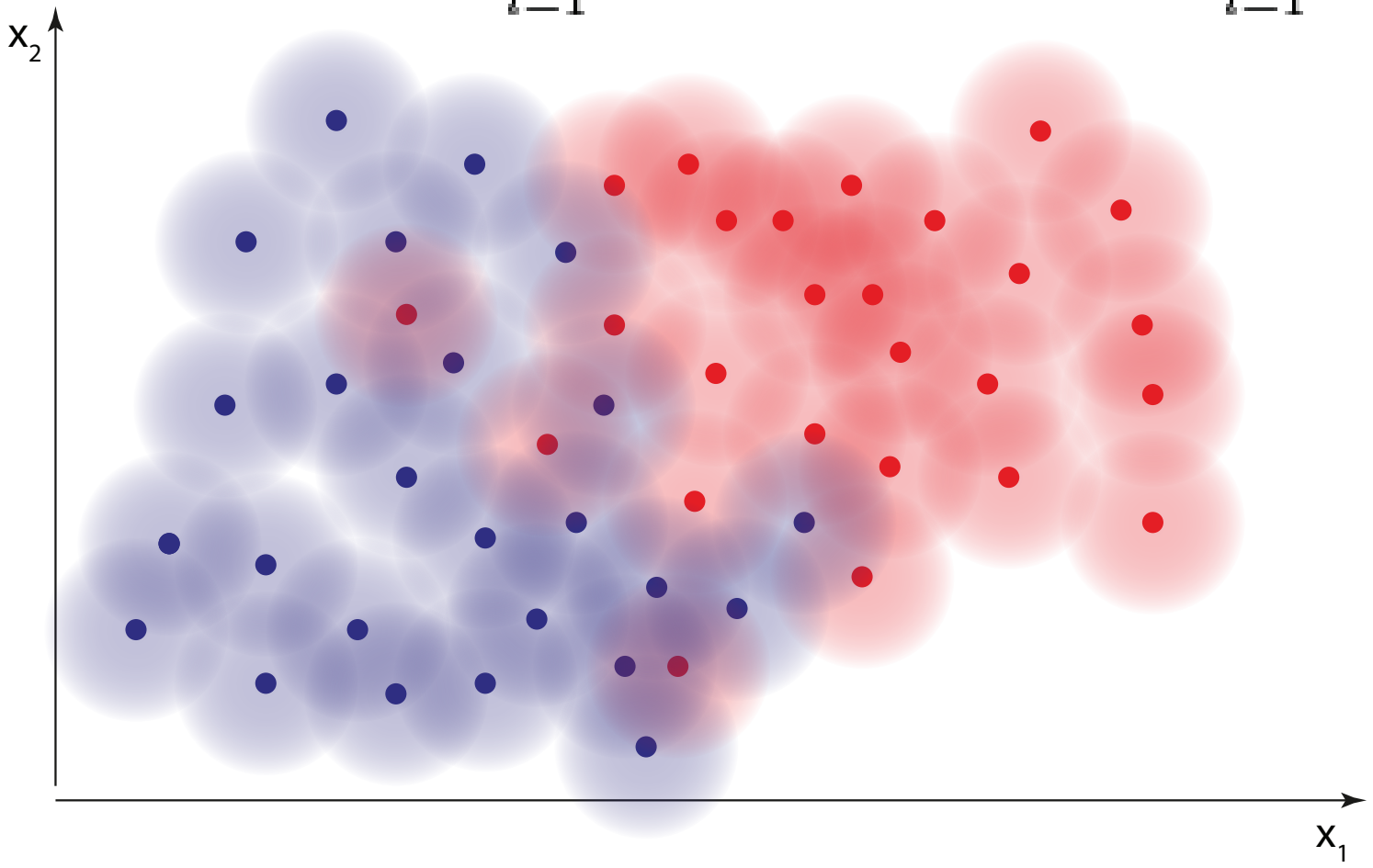
What is the best cut hyperplane?

Equivalent: Onto which line do we have to project, to get the best separation?

12.5. Kernel density estimators

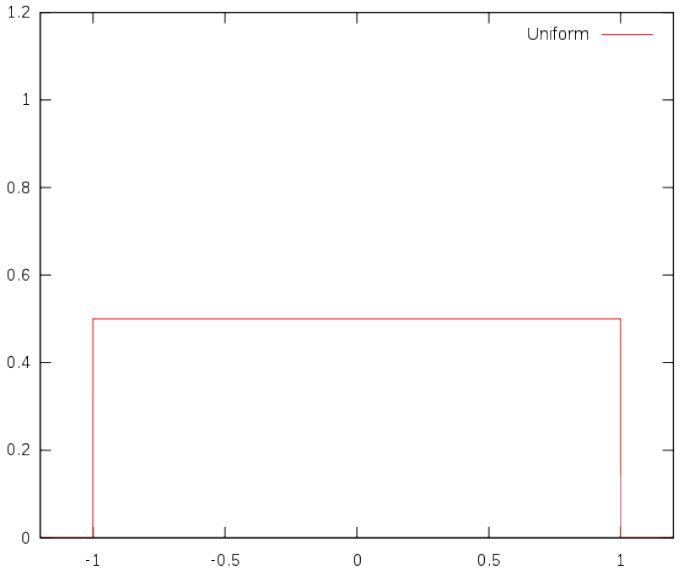
Idea: Smear training data set to get an approximation to the PDFs

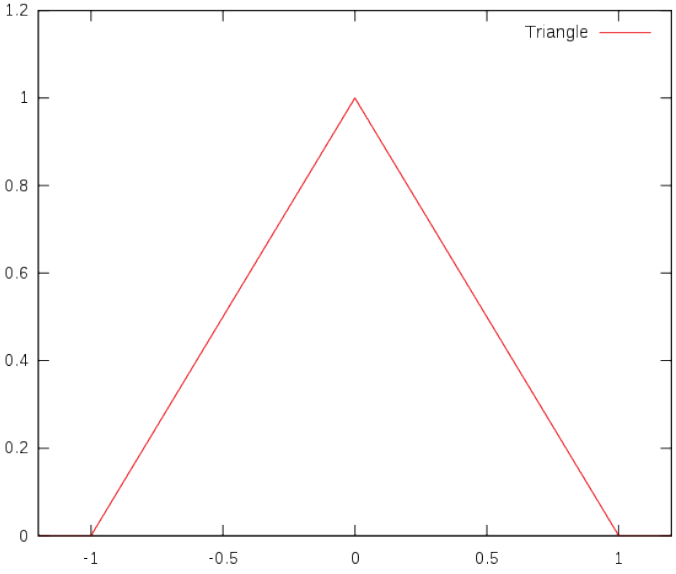
$$\hat{f}_h(x) = \frac{1}{n}\sum_{i=1}^{n} K_h(x - x_i) \quad = \frac{1}{nh}\sum_{i=1}^{n} K\Big(\frac{x - x_i}{h}\Big),$$

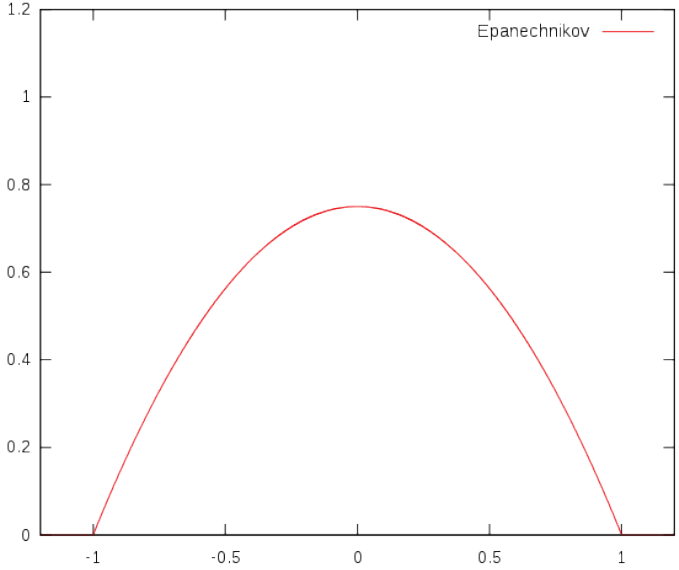


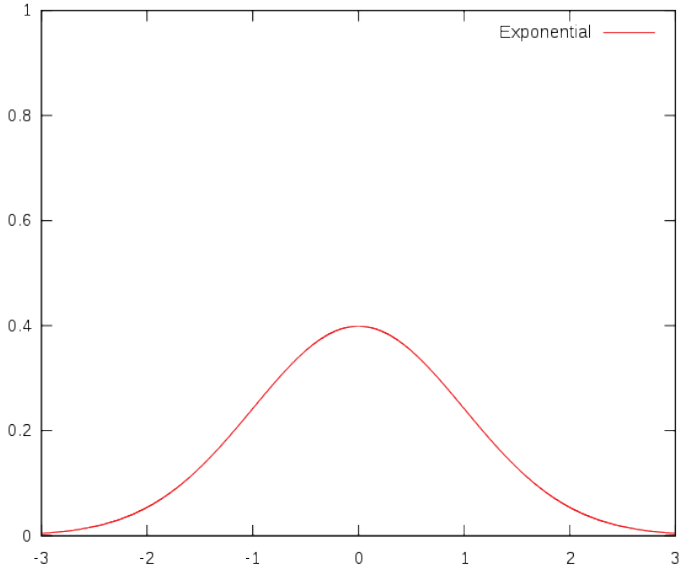
Kernel density estimators

- What shape (Kernel) to use for smearing?
  - Rectangle with uniform distribution
    (simple, but discontinuous at the edges)
  - Triangular
    (somewhat better)
  - Gaussian
    (slow, as it never goes to zero)
  - Epanechnikov
    (nice, but no closed form)







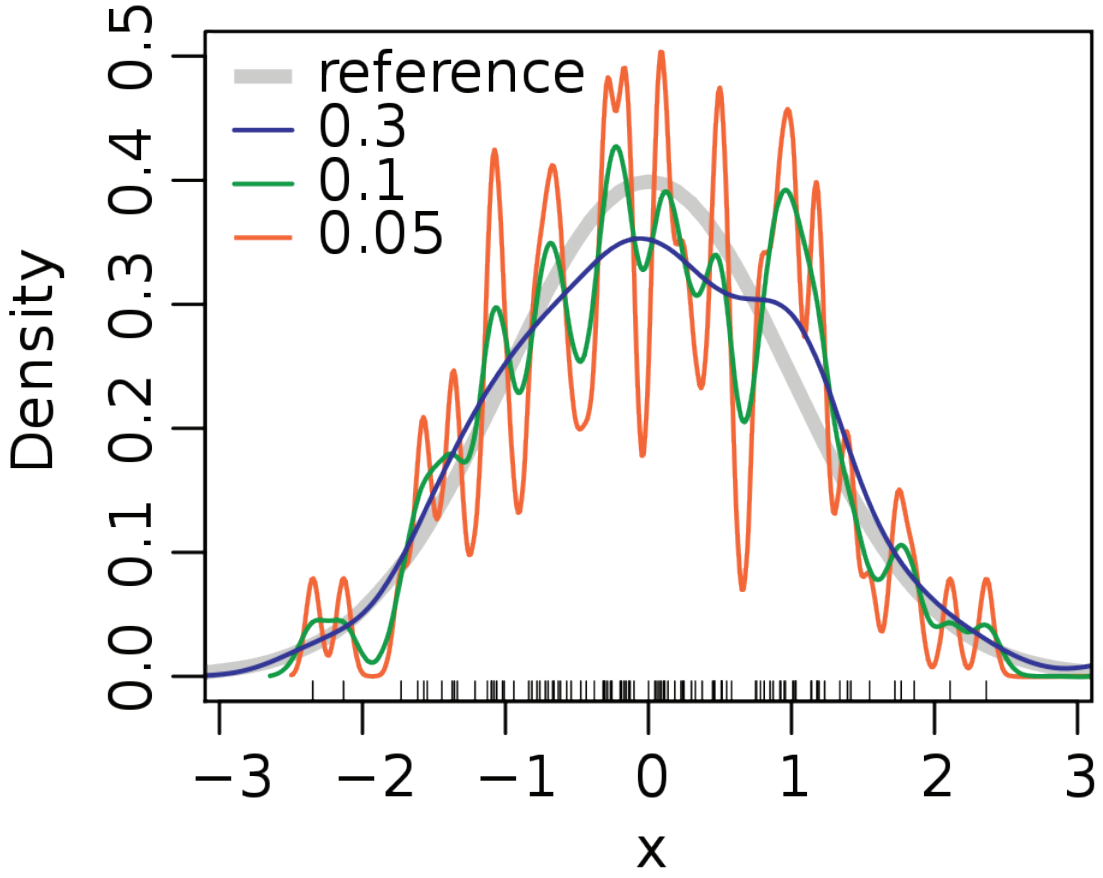


Kernel density estimators

- How big is the Kernel? Size of the smoothing parameter h:
  - Too small: non-smooth distribution, overtraining
  - Too big: might miss features of the PDF

- Curse of dimensionality:
  For D large, there is often
  no "close" training point

  To fill the phase space, h has to
  be of the same order as the phase
  space edge length...



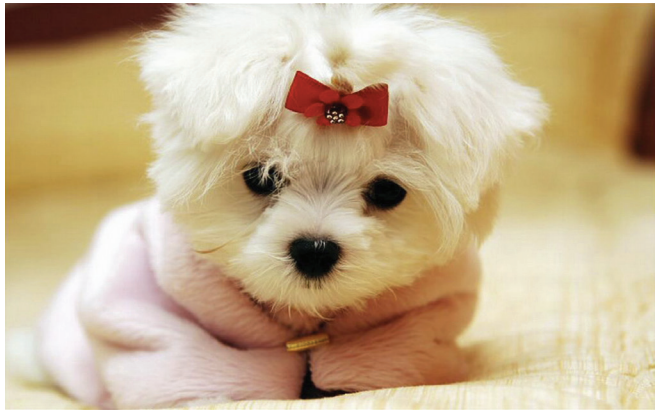
12.5. Pattern recognition and machine learning

- Humans are extremely good at recognizing patterns

Pattern recognition and machine learning

- Humans are extremely good at recognizing patterns
  even small kids can correctly classify animals as dogs



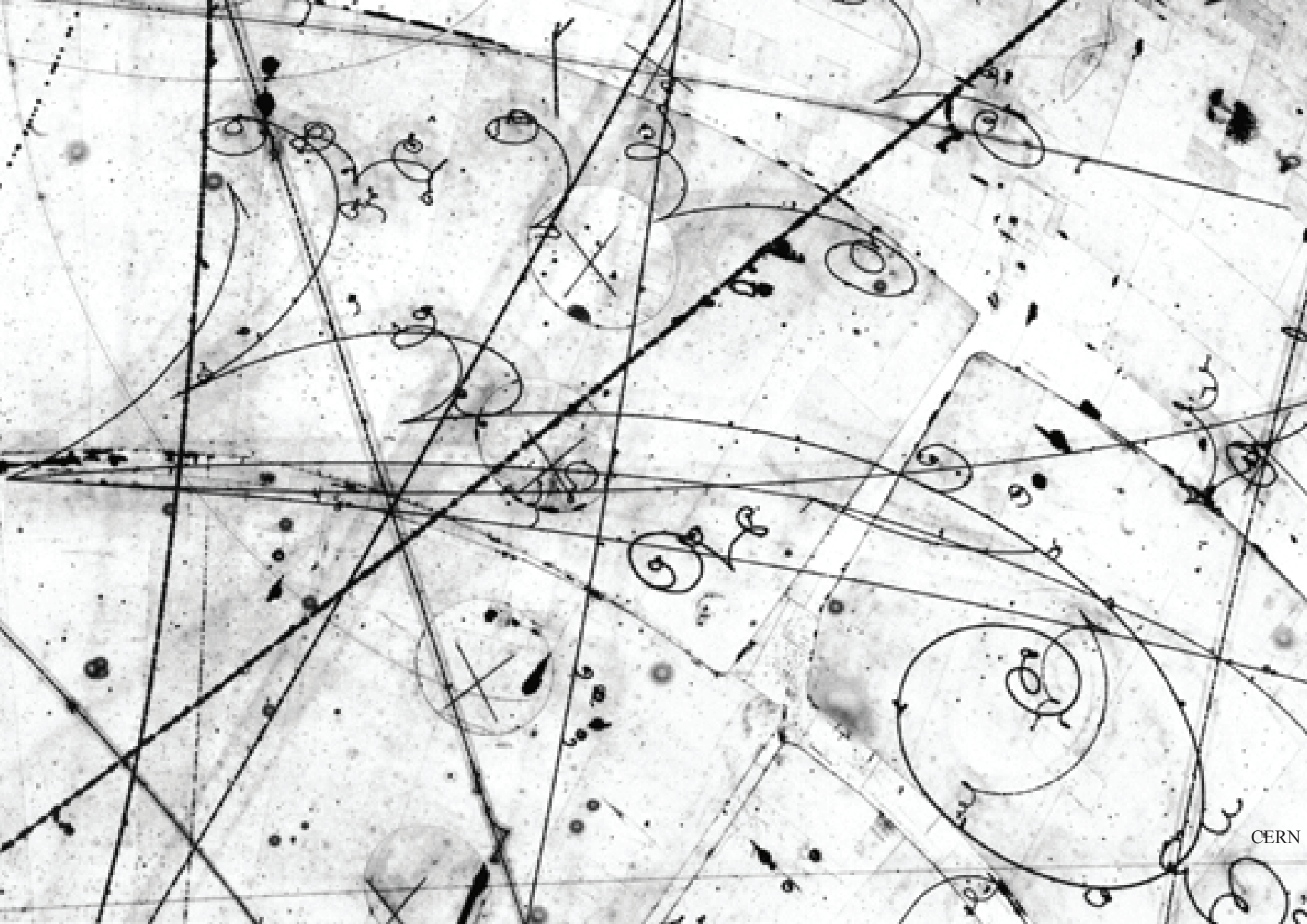









Digression: Using humans for pattern recognition


CERN

CERN

CLASSIFY STORY SCIENCE GALAXY ZOO DISCUSS PROFILE LANGUAGE

Classify

UKIDSS Invert

Examples Restart

SHAPE

Is the galaxy simply smooth and rounded, with no sign of a disk?

Smooth

Features or disk

Star or artifact

Pattern recognition and machine learning

- Unfortunately, human pattern recognition goes only to (projections to) three dimensions...
- Despite decades of efforts, computers are fairly bad at this (think of speech recognition)
- They however do not mind about high dimensionality

If you have many variables, each with a little separation power (but not enough for a cut) use machine learning for multivariate method

- Train a black box
- Many inputs, single output: the classifier

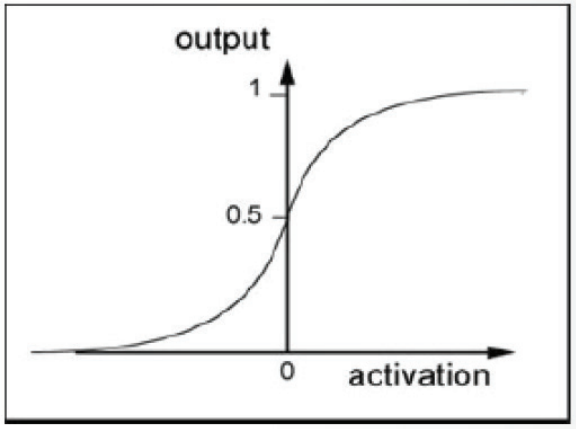
- Generally have to make sure our decision boundary is wiggly enough to capture features of PDF but should not reflect fluctuations in the training sample (overtraining)

12.6. Neural Networks

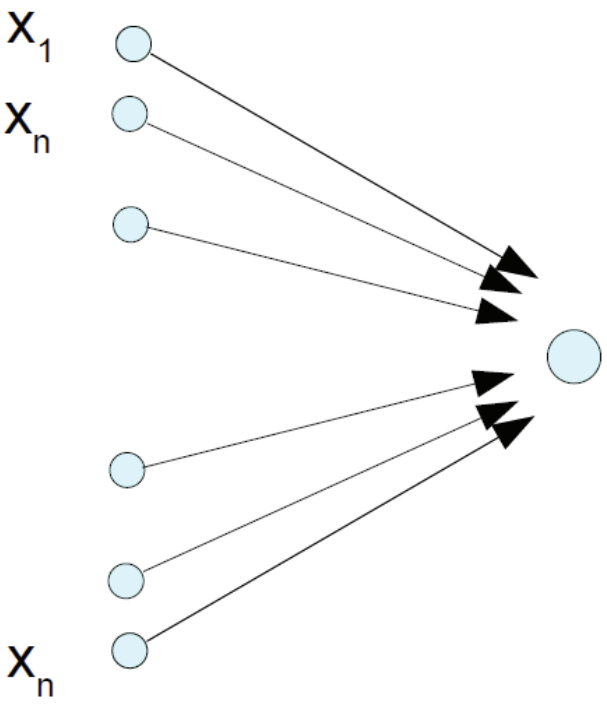
(Crude) attempt to model a brain

- Nodes: Model neurons
  Nonlinear response to weighted sum of inputs (usually sigmoid)
- Inputs: Model axons and synapses
  Have a weight (which is what is trained)
- We usually use networks without feedback - feed-forward network

$$A(x)=\frac{1}{1-e^{x}}$$: the sigmoid function



Single layer perceptron



t(x)

Output node

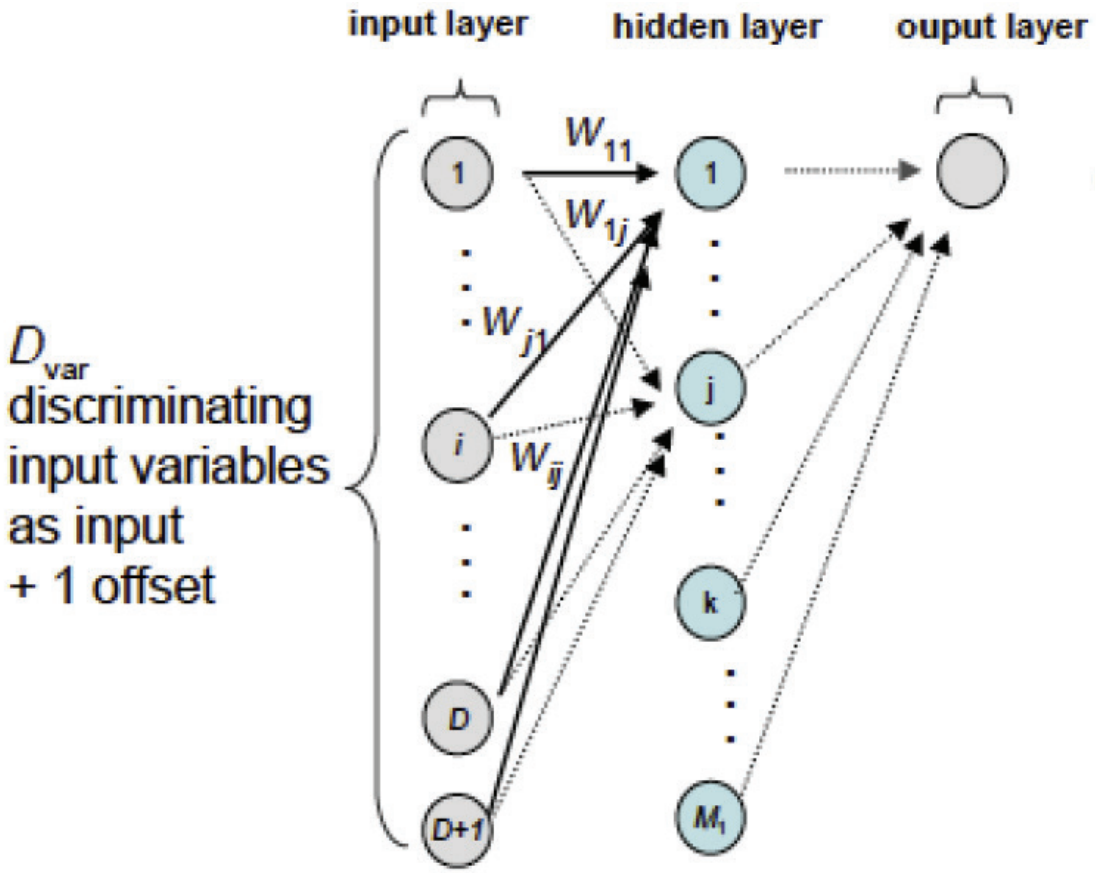
$$t(\vec{x}) = A\left(a_0 + \sum_{i=1}^{n} a_i x_i\right)$$

Where:

- a_0: threshold
- A(x): activation function

NODES = input layer

Double layer perceptron



$$t(\vec{x}) = \sum_{i}^{M} w_{0i}\, A\left(w_{i0} + \sum_{j=1}^{n} w_{ij}\, x_j\right)$$

$t(\vec{x})$ is

- A linear combination of
  - non-linear functions of
    - linear combination of
      - the input data

Double layer perceptron

- Weierstrass theorem: Any nonlinear function of the inputs can be approximated arbitrarily well if there are enough hidden nodes
- Not much is know whether it is better to have a single hidden layer with lots of nodes or fewer nodes in more layers
- In practice, two hidden layers seem to work better...

Neural network training

Use training events to adjust the weights such that:

- **t(x) → 0 for background events**
- **t(x) → 1 for signal events**

How do we adjust?

Minimize loss function:

$$L(w) = \sum_{i}^{\text{events}} (t(x_i) - t(C))^2$$

Predicted event type

True event type

where:

$$t(C) = \begin{cases} 1 & \text{for } C = \text{signal} \\ 0 & \text{for } C = \text{bkgr} \end{cases}$$

t(x) is a very "wiggly" function with many local minima. A global overall fit in the many parameters is possible but not the most efficient method to train neutral networks ...

Neural network training

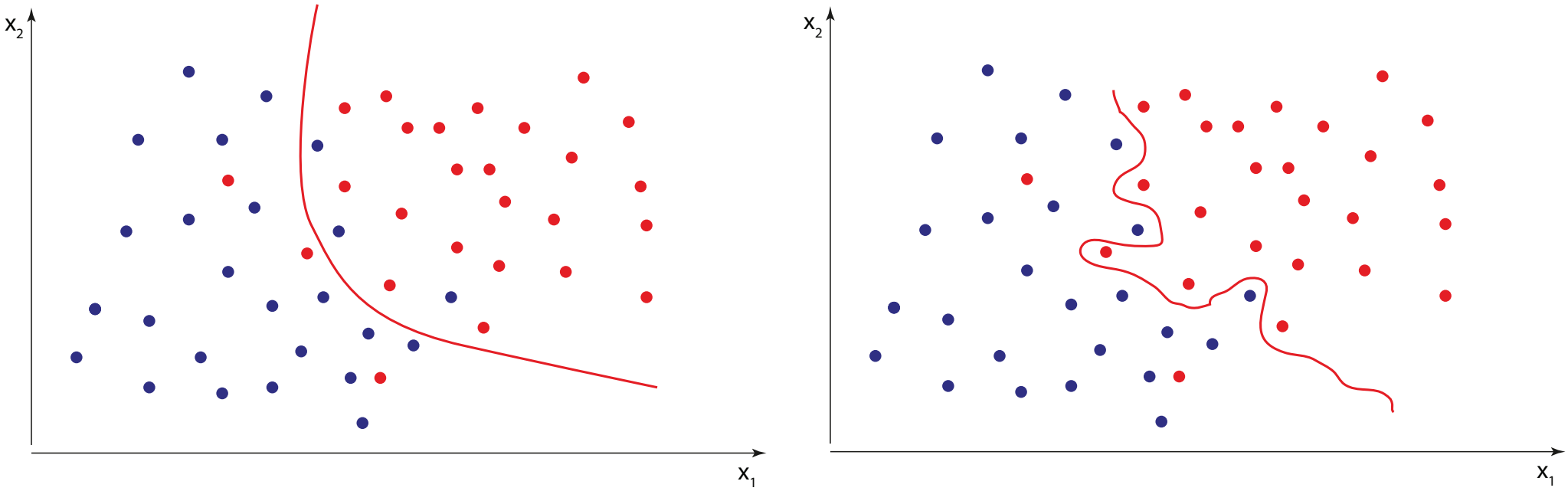
Use smarter methods instead of a global overall fit in the many parameters:

- Back propagation: learn from experience, gradually adjust your perception to match reality
- Online learning: learn event by event and not only at the end of your life from the entire experience

- Start with random weights
- Adjust weights in each step a bit, in the direction of the steepest descent of the loss function
- Training is repeated n times over the whole data sample: HOW OFTEN??

NOTE: for online learning, the training events should be mixed randomly, otherwise you first steer in a wrong direction from which it is afterward hard to get out again !!

Overtraining

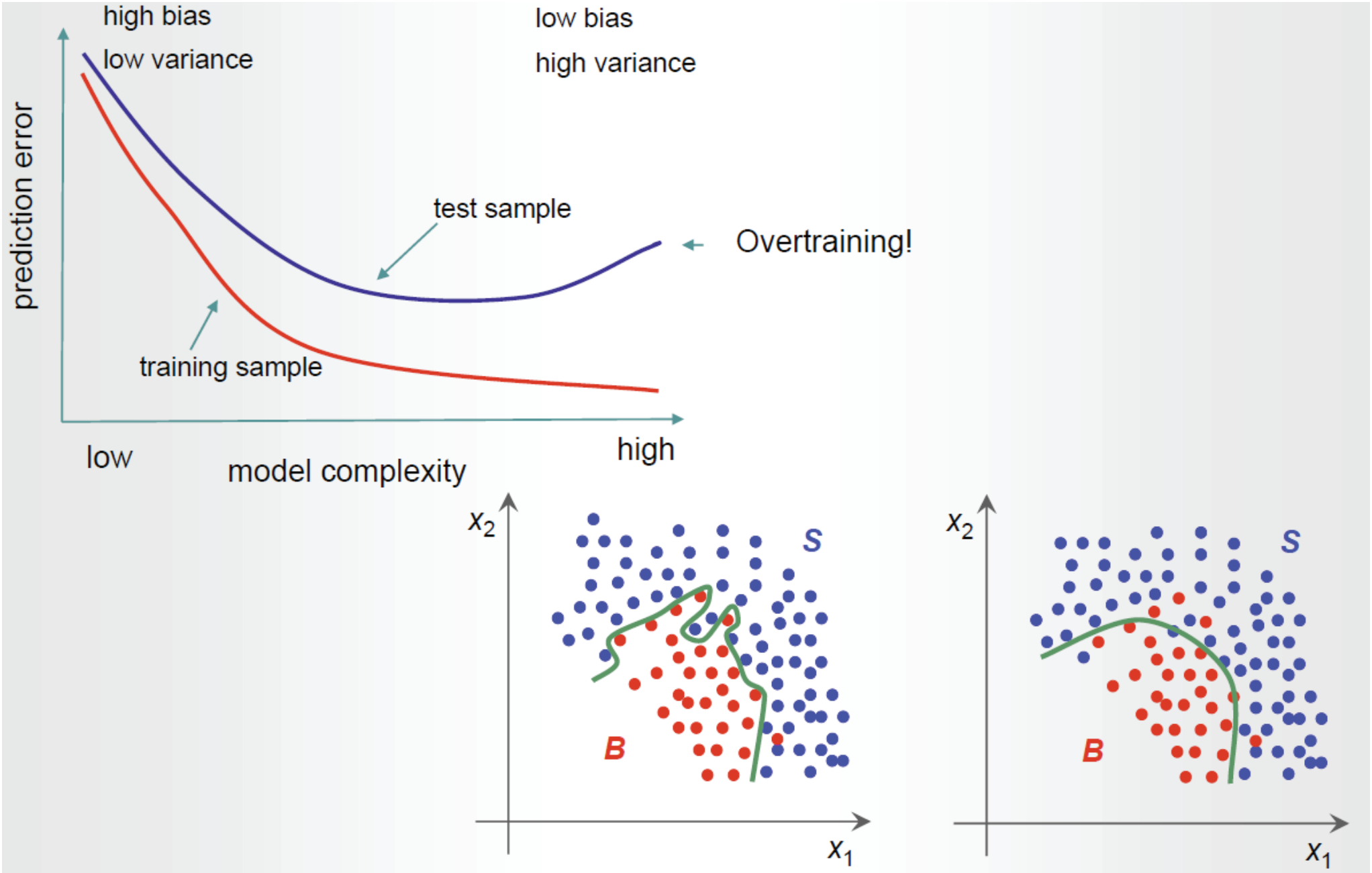


Always reserve a part of your "training" data for testing the classifier

More training on the same data will always improve the classifier on those data

If results start getting worse on the test sample, stop training

Overtraining



12.7. Boosted decision trees

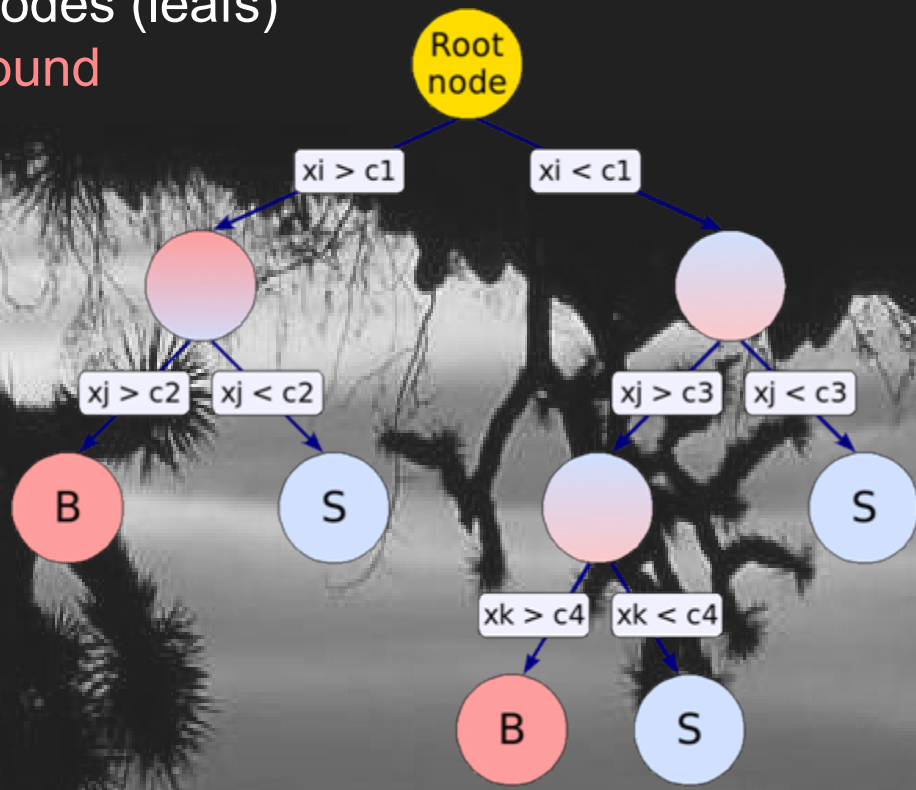
Idea:

- Slice phase space sequentially into little (hyper-) cubes that are either signal or background-like (decision tree)
- Repeat many times with different/modified training data

Following slides from Helge Voss

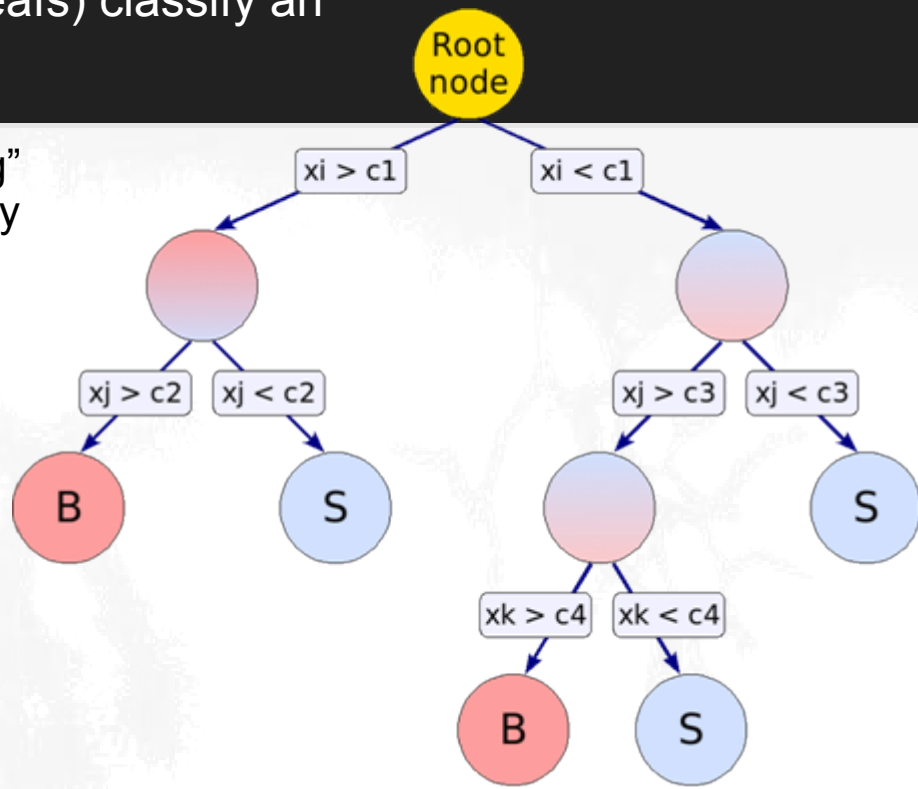
Boosted Decision Trees

- Decision Tree: Sequential application of cuts splits the data into nodes, where the final nodes (leafs) classify an event as signal or background



Boosted Decision Trees

- Decision Tree: Sequential application of cuts splits the data into nodes, where the final nodes (leafs) classify an event as signal or background



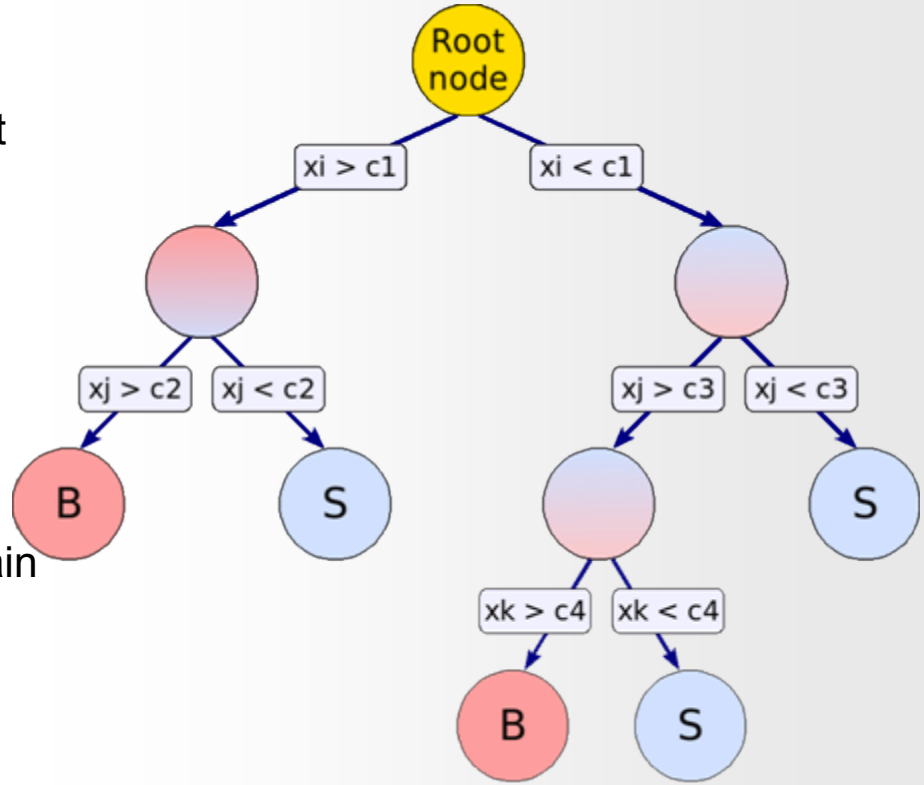
- used since a long time in general "data-mining" applications, less known in HEP (although very similar to "simple Cuts")
  - easy to interpret, visualised
  - independent of monotonous variable transformations, immune against outliers
  - weak variables are ignored (and don't (much) deteriorate performance)
- Disadvatage → very sensitive to statistical fluctuations in training data

- Boosted Decision Trees (1996): combine a whole forest of Decision Trees, derived from the same sample, e.g. using different event weights.
  - overcomes the stability problem

→ became popular in HEP since MiniBooNE, B.Roe et.a., NIM 543(2005)

Growing a Decision Tree

- start with training sample at the root node
- split training sample at node into two, using a cut in the variable that gives best separation gain
- continue splitting until:
  - minimal #events per node
  - maximum number of nodes
  - maximum depth specified
  - a split doesn't give a minimum separation gain
- leaf-nodes classify S,B according to the majority of events or give a S/B probability



- Why no multiple branches (splits) per node ?
  - → Fragments data too quickly; also: multiple splits per node = series of binary node splits
- What about multivariate splits?
  - → Time consuming
  - → other methods more adapted for such correlatios

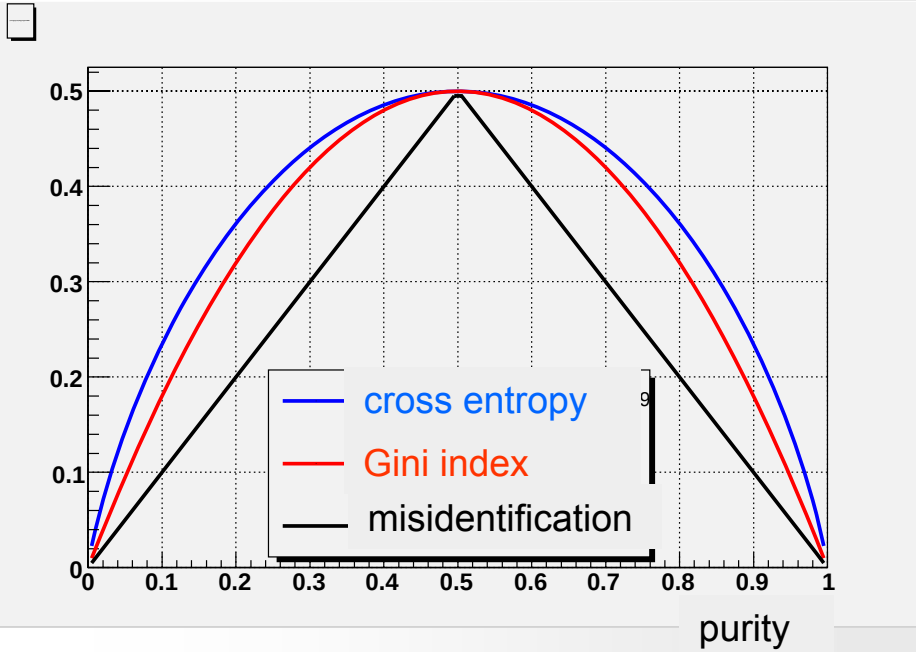
Separation Gain

- What do we mean by “best separation gain”?
- define a measure on how mixed S and B in a node are:
  - Gini-index: (Corrado Gini 1912, typically used to measure income inequality)

    p (1-p) : p=purity
  - Cross Entropy:

    -(plnp + (1-p)ln(1-p))
  - Misidentification:

    1-max(p,1-p)



- difference in the various indices are small, most commonly used: Gini-index

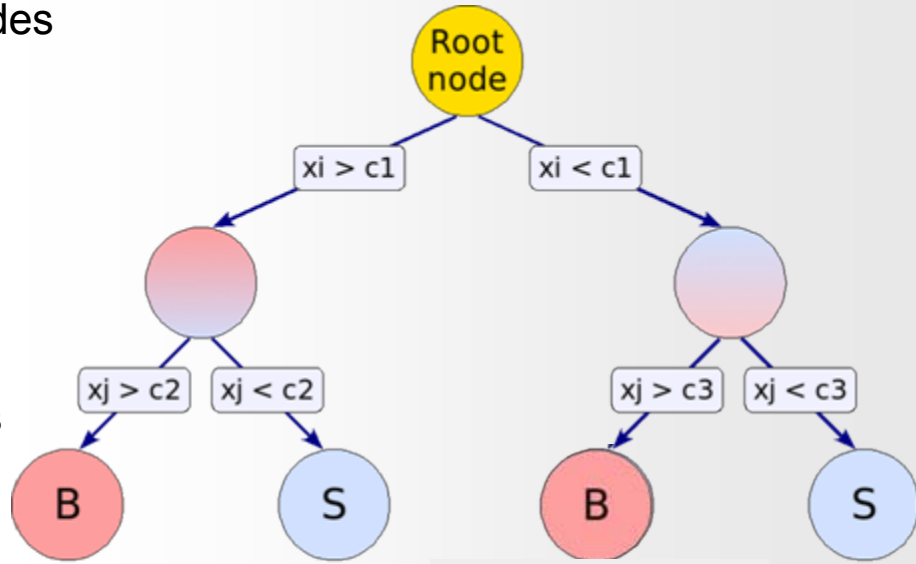
separation gain: e.g. $N_{Parent}*Gini_{Parent} - N_{left}*Gini_{LeftNode} - N_{right}*Gini_{RightNode}$

- Choose amongst all possible variables and cut values the one that maximised the this.

Decision Tree Pruning

- One can continue node splitting until all leaf nodes are basically pure (using the training sample)

  → obviously: that's overtraining



- Two possibilities:
  - stop growing earlier

    generally not a good idea, useless splits might open up subsequent usefull splits
  - grow tree to the end and "cut back", nodes that seem statistically dominated:

    → pruning

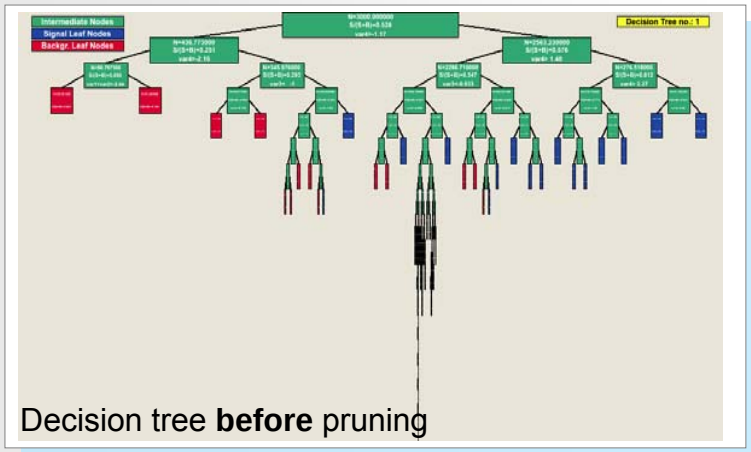
- e.g. Cost Complexity pruning:

$$C(T,\alpha) = \underbrace{\sum_{\text{leafs of T}} \sum_{\text{events in leaf}} |\, y(x) - y(C)\,|}_{\text{Loss function}} + \underbrace{\alpha N_{\text{leaf nodes}}}_{\text{regularisaion/ cost parameter}}$$

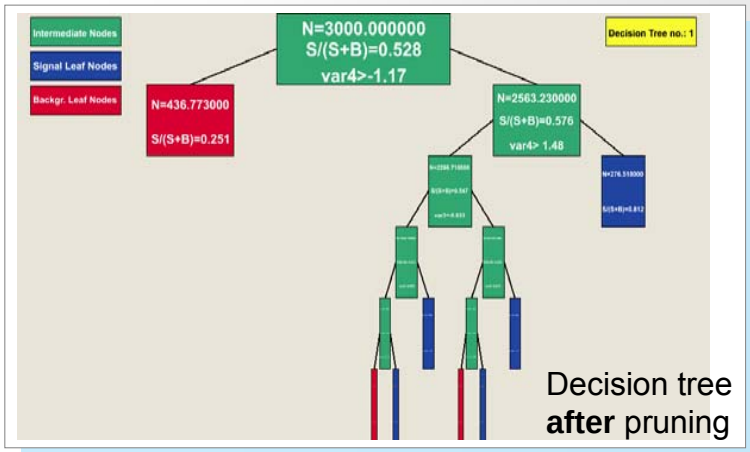
  - assign to every sub-tree, T C(T,α) :
  - find subtree T with minmal C(T,α) for given α
  - prune up to value of a that does not show overtraining in the test sample

Decision Tree Pruning

- "Real life" example of an optimally pruned Decision Tree:



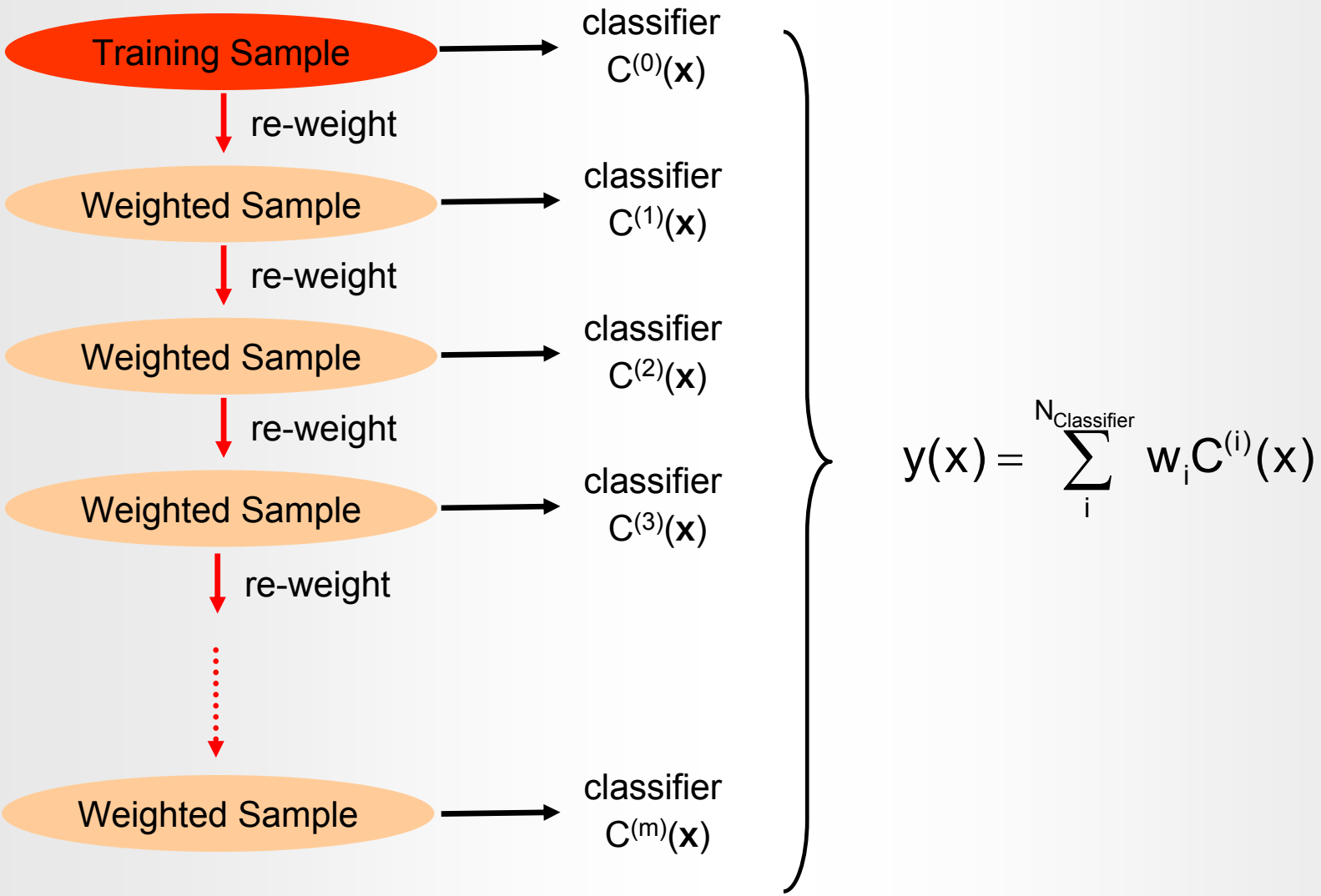
Decision tree **before** pruning



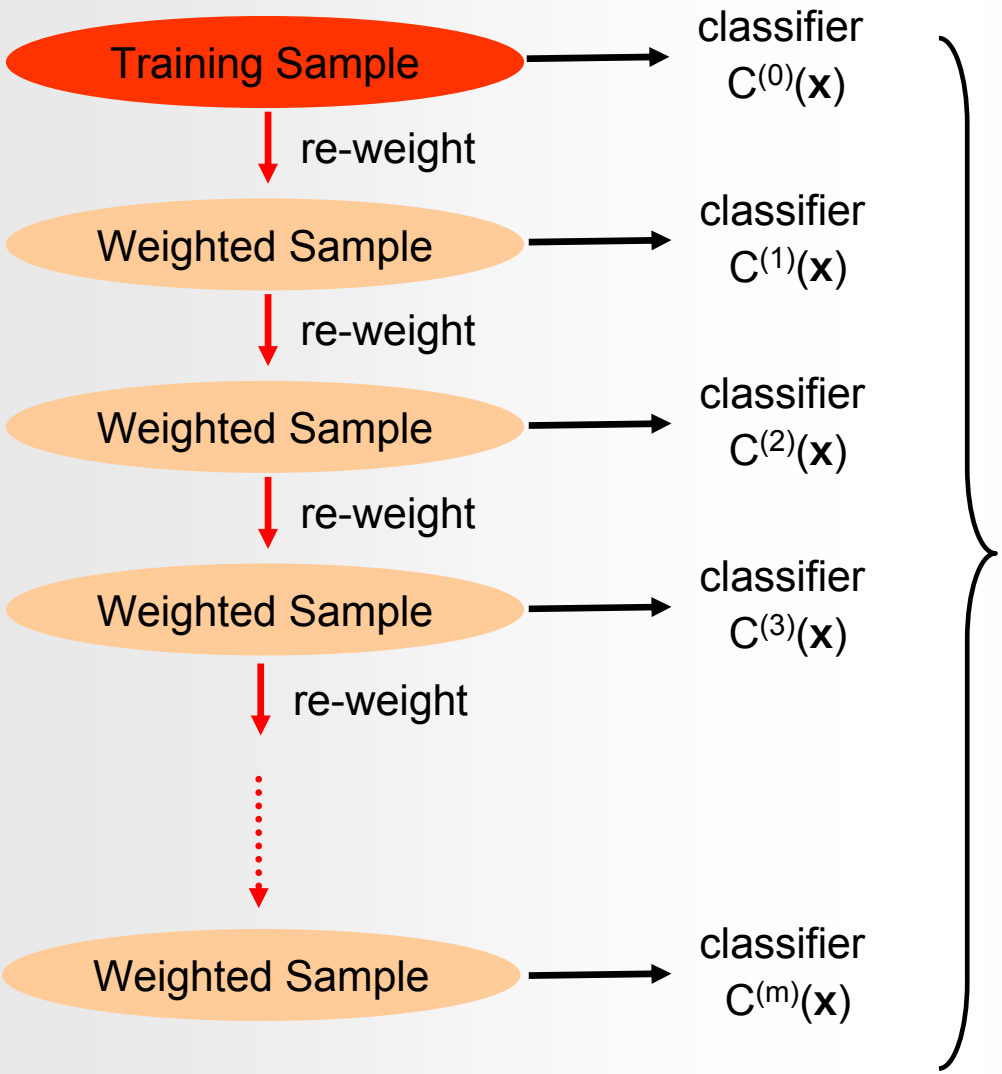
Decision tree **after** pruning

- Pruning algorithms are developed and applied on individual trees
  - optimally pruned single trees are not necessarily optimal in a forest !

Boosting


Training Sample
re-weight
Weighted Sample
re-weight
Weighted Sample
re-weight
Weighted Sample
re-weight
Weighted Sample
classifier
$C^{(0)}(\mathbf{x})$
classifier
$C^{(1)}(\mathbf{x})$
classifier
$C^{(2)}(\mathbf{x})$
classifier
$C^{(3)}(\mathbf{x})$
classifier
$C^{(m)}(\mathbf{x})$
$y(x) = \sum_{i}^{N_{Classifier}} w_i C^{(i)}(x)$

Adaptive Boosting (AdaBoost)



- AdaBoost re-weights events misclassified by previous classifier by:

$$\frac{1-f_{err}}{f_{err}} \quad \text{with:}$$

$$f_{err} = \frac{\text{misclassified events}}{\text{all events}}$$

- AdaBoost weights the classifiers also using the error rate of the individual classifier according to:

$$y(x) = \sum_{i}^{N_{Classifier}} \log\left(\frac{1-f_{err}^{(i)}}{f_{err}^{(i)}}\right) C^{(i)}(x)$$

Bagging and Randomised Trees

other classifier combinations:

- Bagging:
  - combine trees grown from “bootstrap” samples

  (i.e re-sample training data with replacement)

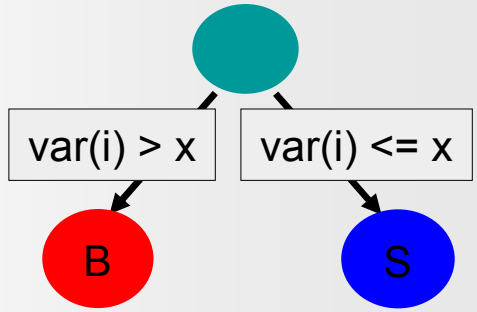
- Randomised Trees: **(Random Forest: trademark L.Breiman, A.Cutler)**
  - combine trees grown with:
    - random subsets of the training data only
    - consider at each node only a random subsets of variables for the split
    - NO Pruning!

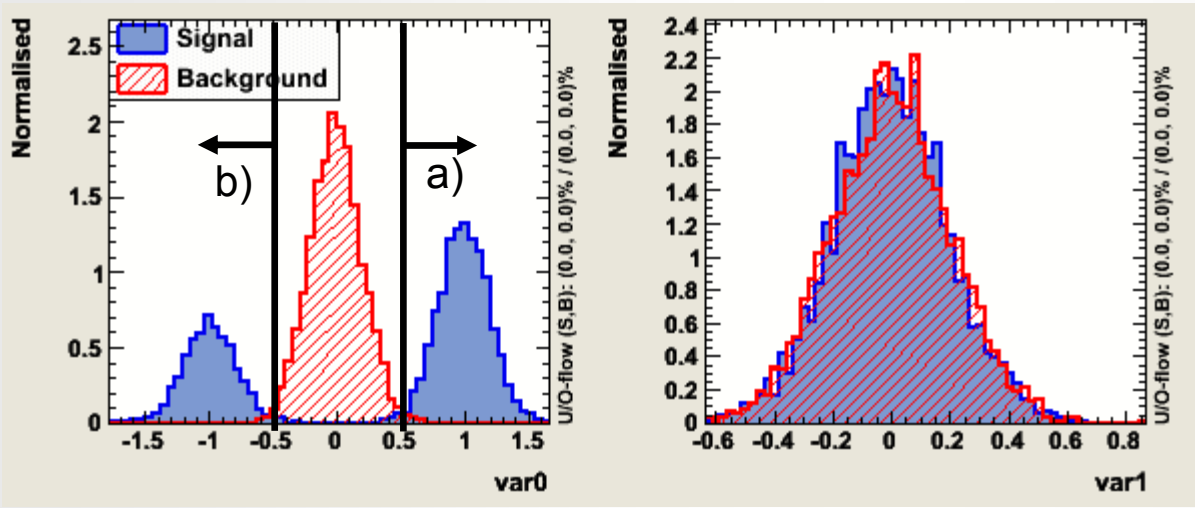
- These combined classifiers work surprisingly well, are very stable and almost perfect “out of the box” classifiers

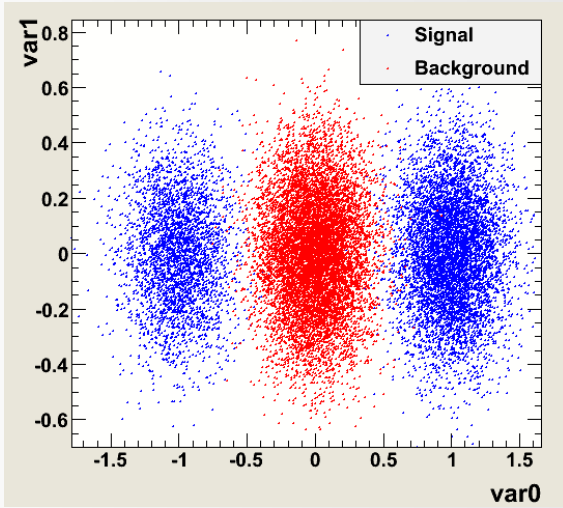
AdaBoost: A simple demonstration

The example: (somewhat artificial…but nice for demonstration) :

- Data file with three “bumps”
- Weak classifier (i.e. one single simple “cut” ↔ decision tree stumps)







Two reasonable cuts: a) Var0 > 0.5 → ε_{signal}=66% $\varepsilon_{bkg} \approx 0\%$ misclassified events in total 16.5%

or

b) Var0 < -0.5 → ε_{signal}=33% $\varepsilon_{bkg} \approx 0\%$ misclassified events in total 33%

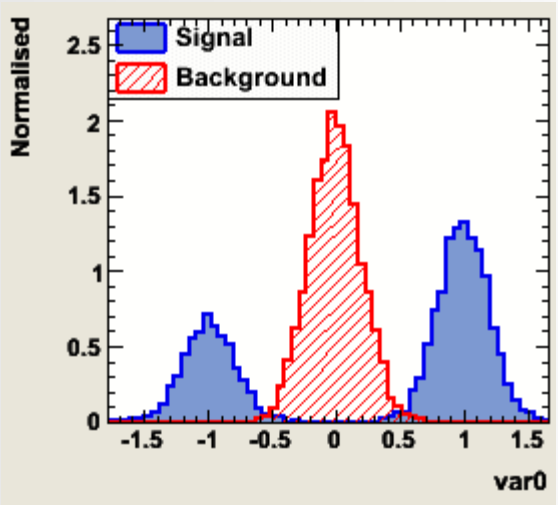
the training of a single decision tree stump will find “cut a)”

AdaBoost: A simple demonstration

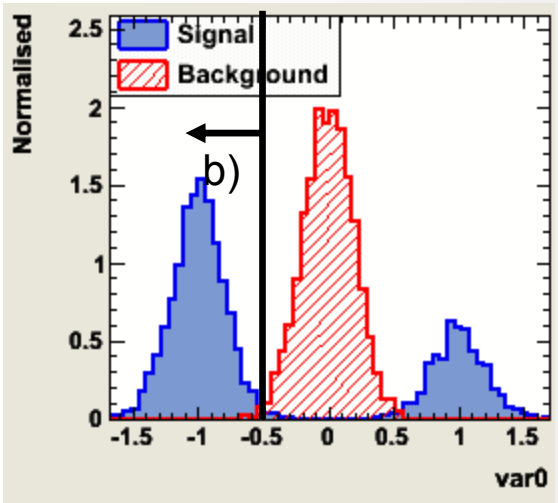
The first “tree”, choosing cut a) will give an error fraction: err = 0.165

➔ before building the next “tree”: weight wrong classified training events by (1-err/err)) ≈ 5

➔ the next “tree” sees essentially the following data sample:

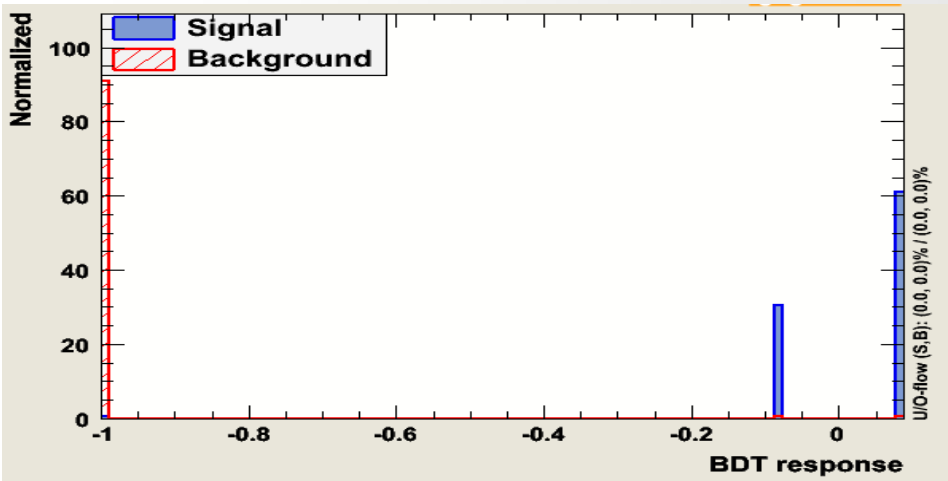


re-weight



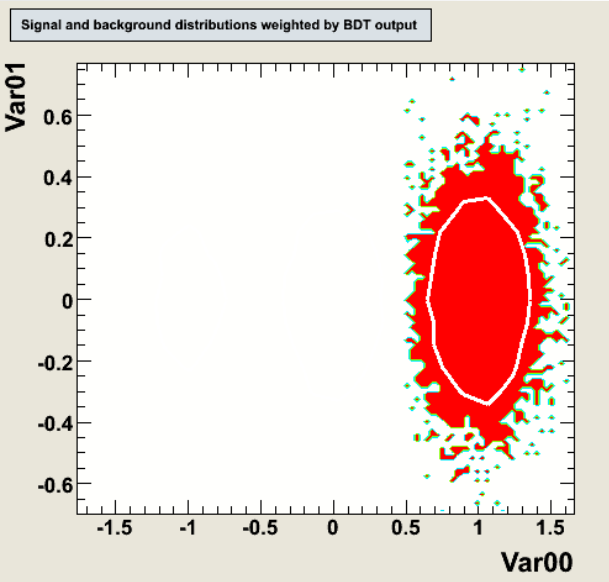
.. and hence will chose: “cut b)”: Var0 < -0.5

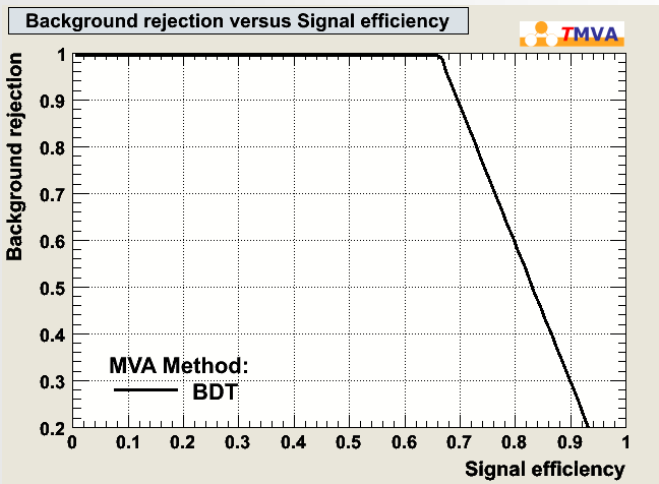
The combined classifier: Tree1 + Tree2
the (weighted) average of the response to a test event from both trees is able to separate signal from background as good as one would expect from the most powerful classifier



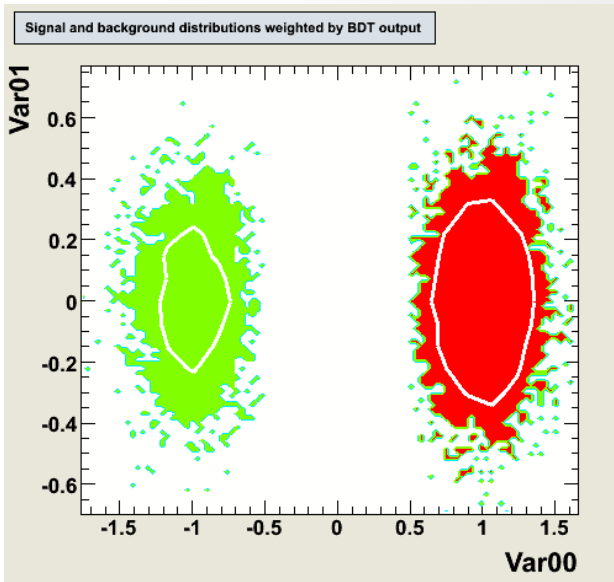
AdaBoost: A simple demonstration

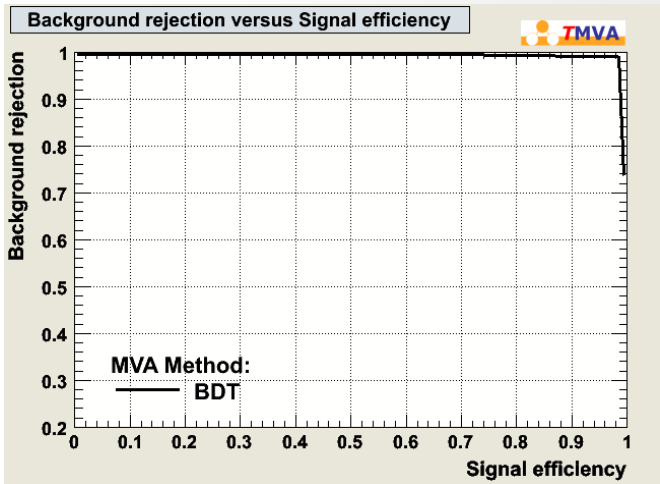
Only 1 tree “stump”





Only 2 tree “stumps” with AdaBoost





AdaBoost vs other Combined Classifiers

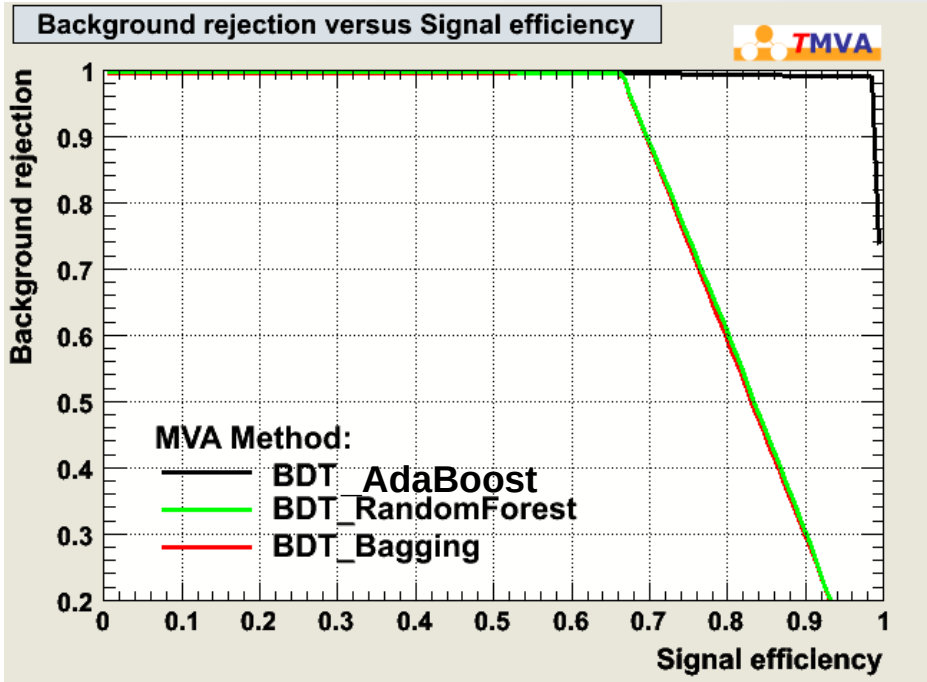
Sometimes people present “boosting” as nothing else then just “smearing” in order to make the Decision Trees more stable w.r.t statistical fluctuations in the training.

→clever “boosting” however can do more, than for example:

- Random Forests
- Bagging

as in this case, pure statistical fluctuations are not enough to enhance the 2nd peak sufficiently

however: a “fully grown decision tree” is much more than a “weak classifier”
→ “stabilization” aspect is more important



Surprisingly: Often using smaller trees (weaker classifiers) in AdaBoost and other clever boosting algorithms (i.e. gradient boost) seems to give overall significantly better performance !

12.8. TMVA

- As multivariate classifiers are black boxes anyway, use existing package
- In ROOT, this is TMVA: tmva.sourceforge.net

Many methods available

- Rectangular cut optimisation
- Projective likelihood estimation (PDE approach)
- Multidimensional probability density estimation (PDE - range-search approach and PDE-Foam)
- Multidimensional k-nearest neighbour method
- Linear discriminant analysis (H-Matrix, Fisher and linear (LD) discriminants)
- Function discriminant analysis (FDA)
- Artificial neural networks (three different MLP implementations)
- Boosted/Bagged decision trees
- Predictive learning via rule ensembles (RuleFit)
- Support Vector Machine (SVM)