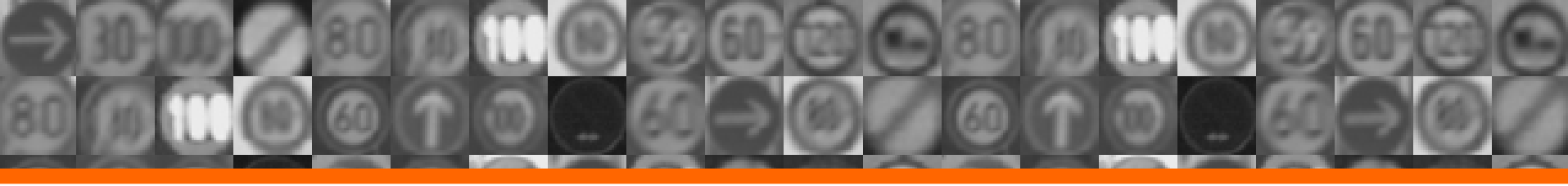
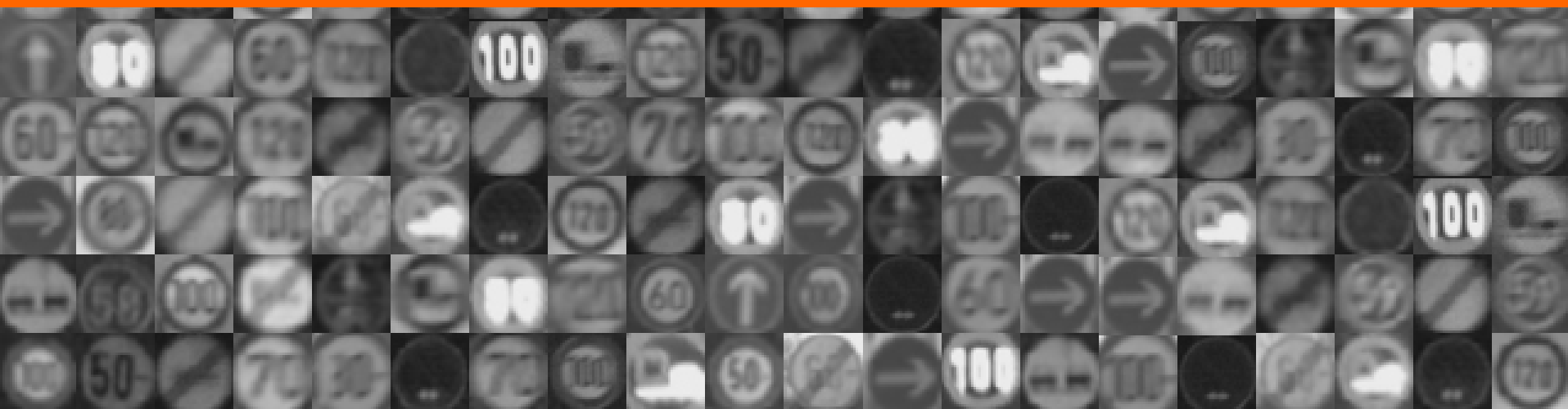


Multivariate Data Analysis and Machine Learning in High Energy Physics (IV)

Helge Voss (MPI–K, Heidelberg)

Graduierten-Kolleg , Freiburg, 11.5-15.5, 2009



Outline

- Summary of last lecture
- Boosted Decision Trees
- Rule Fitting

- Erratum: The correct “sigmoid function” formula is: $A(x)=\frac{1}{1+e^{x}}$: the sigmoid function

Recapitulatoin

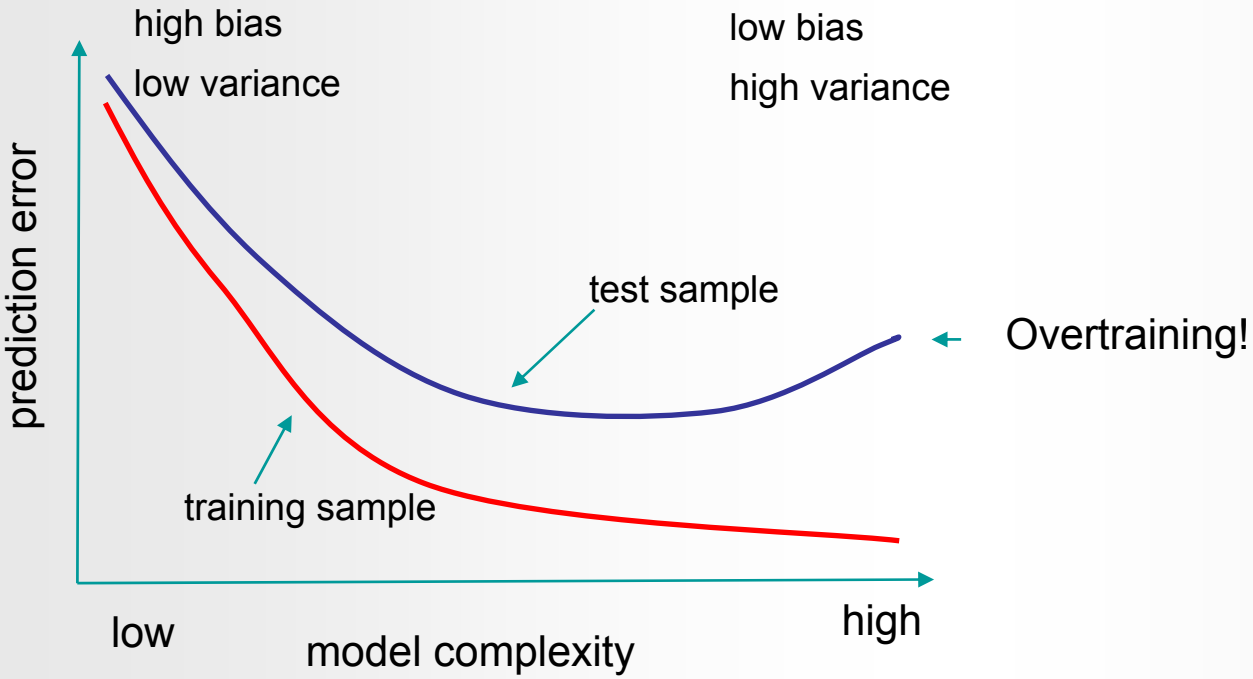
- There are two in principlal different approaches to build a MVA-classifier or regressor:
  - Trying to estimate directly P(x,S) or P(x,B) from the data and use Neyman-Pearson Lemma and use the Likelihood ratio to discriminate between S and B

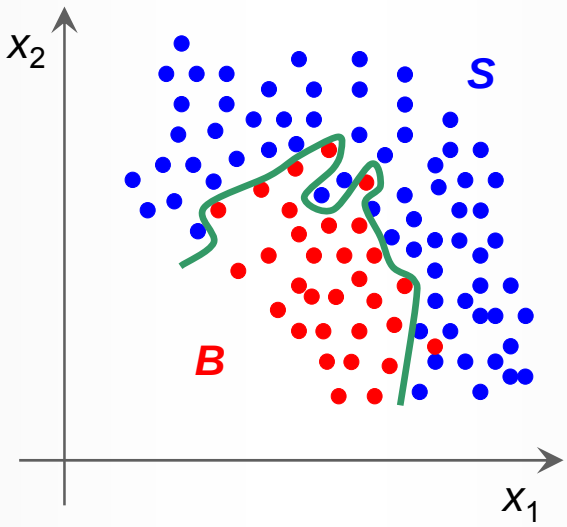
    → KNN, Likelihood (naïve Bayesian classifier)
  - Trying to find/fit the best hypersurface in the feature space of the training events that gives the desired separation between signal and background. The class of possible surfaces is described in a (general) model
    - Linear decision boundaries → e.g. Fisher discriminant
    - very general, piecewise defined non-linear model → e.g.Neural Network
    - general: some (linear) combination of "basis functions"

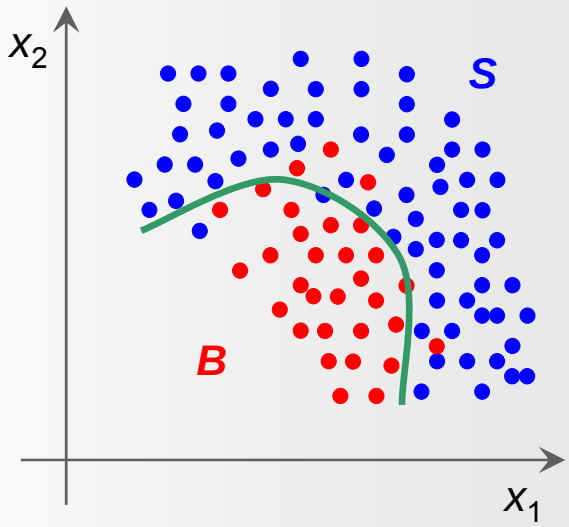
$$y(\vec{x}) = \sum_{i}^{M} \left(w_i h_i(\vec{x}) \right)$$

- Fit of the selection boundary or mapping function y(x) uses Loss function: L(y,y(x)) penalizes errors made in the prediction:
  - EPE(y(x)) = E(y – y(x))^2 squared error loss, typical "loss function" used in regression
    - by conditioning on "x" we can write:
  - EPE(y(x)) = E(|y-y(x)|) misclassification error, typical "loss function" for classification problems
- Overtraining: My model is too flexible, precisely fit that it mimics statistical fluctuations of the training data

Bias Variance Trade off







Cross Validation

- many (all) classifiers have tuning parameters "α" that need to be controlled against overtraining
  - number of training cycles, number of nodes (neural net)
  - smoothing parameter h (kernel density estimator)
  - ….
- the more free parameter a classifier has to adjust internally → more prone to overtraining
- more training data → better training results
- division of data set into "training" and "test" sample reduces the "training" data

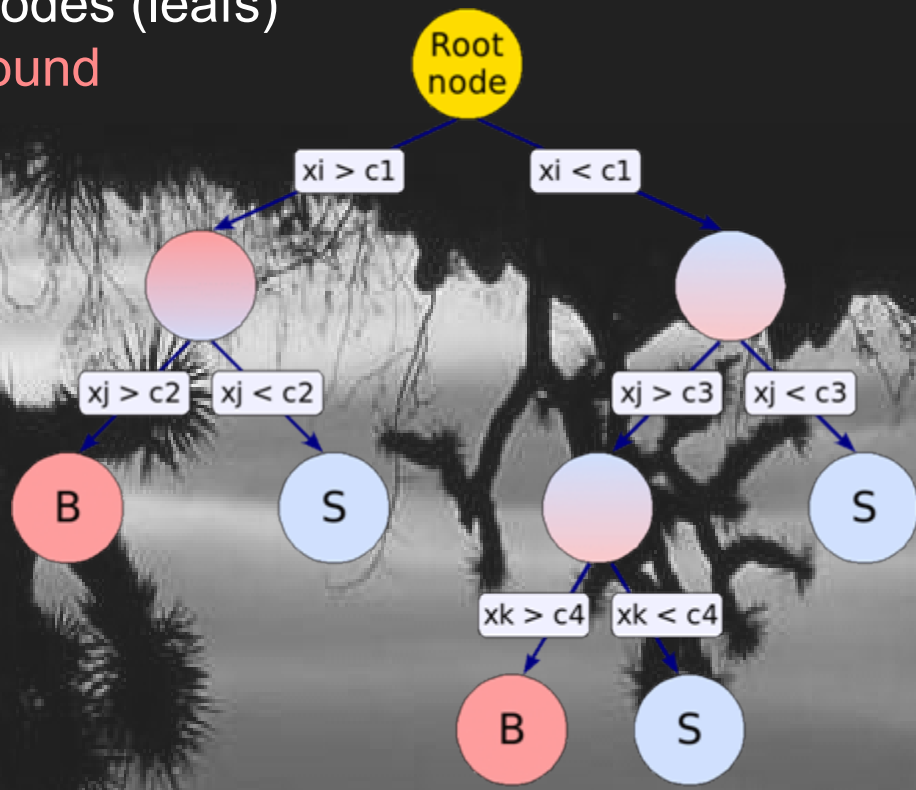
Cross Validation: divide the data sample into say 5 sub-sets

Train	Train	Train	Train	Test

- train 5 classifiers: $y_i(x,\alpha)$: i=1,..5,
- classifier $y_i(x,\alpha)$ is trained without the i-th sub sample
- calculate the test error: $$CV(\alpha) = \frac{1}{N_{events}} \sum_{k}^{events} L(y_i(x_k,\alpha)) \quad L: \text{loss function}$$
- choose tuning parameter a for which CV(α) is minimum and train the final classifier using all data

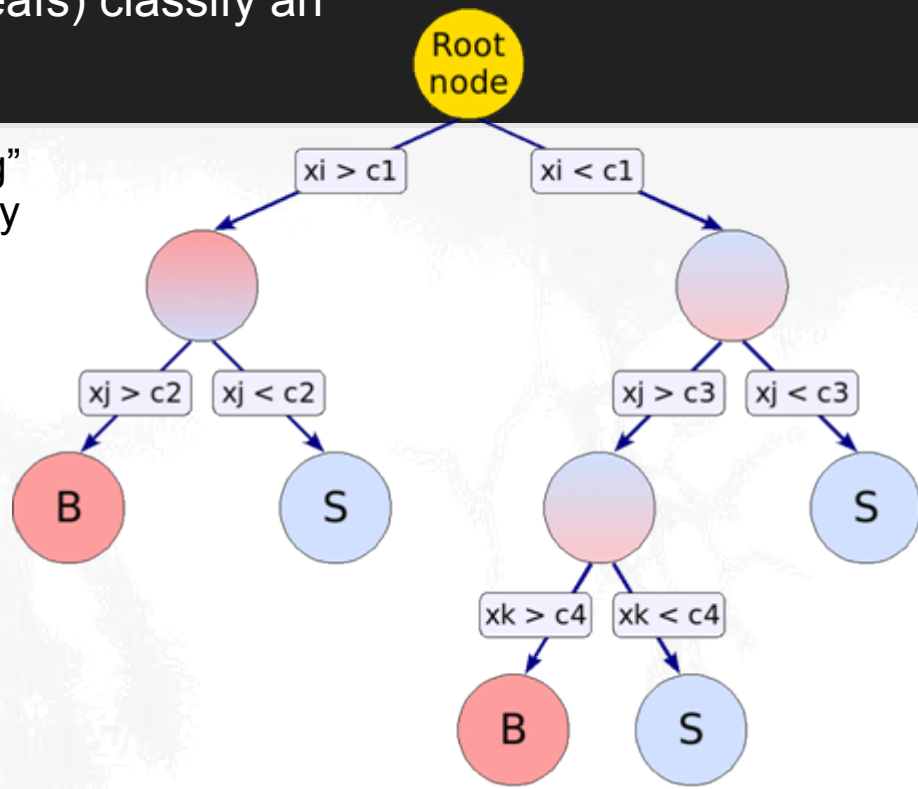
Boosted Decision Trees

- 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



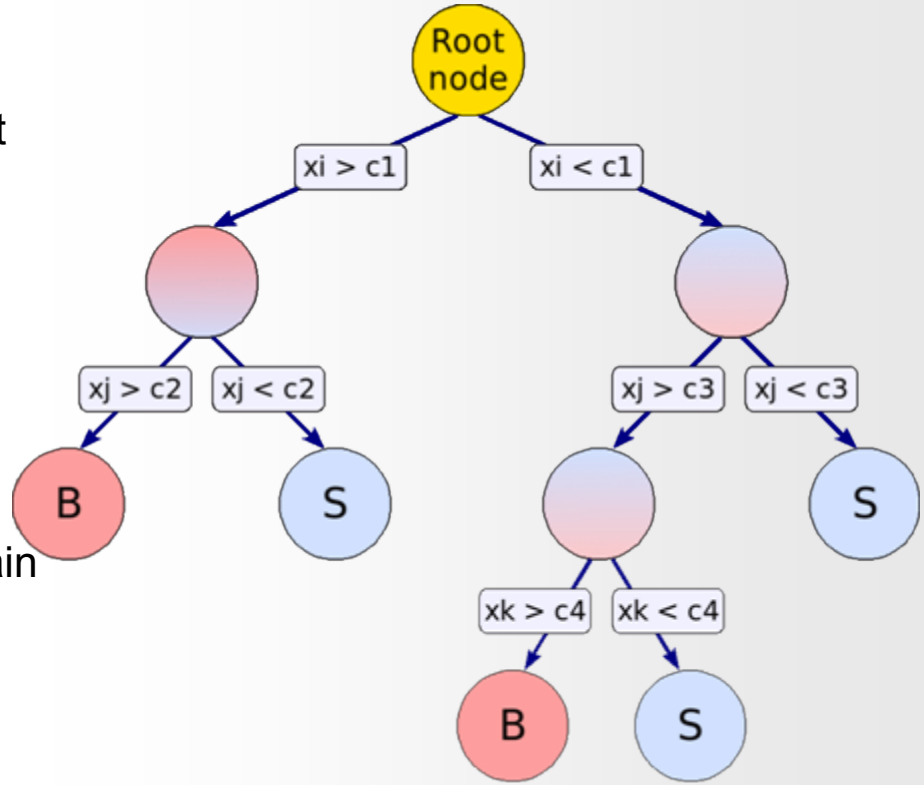
- used since a long time in general "data-mining" applications, less known in HEP (although very similar to "simple Cuts")
  - easy to interpret, visualised
  - independent of monotonous variable transformations, immune against outliers
  - weak variables are ignored (and don't (much) deteriorate performance)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

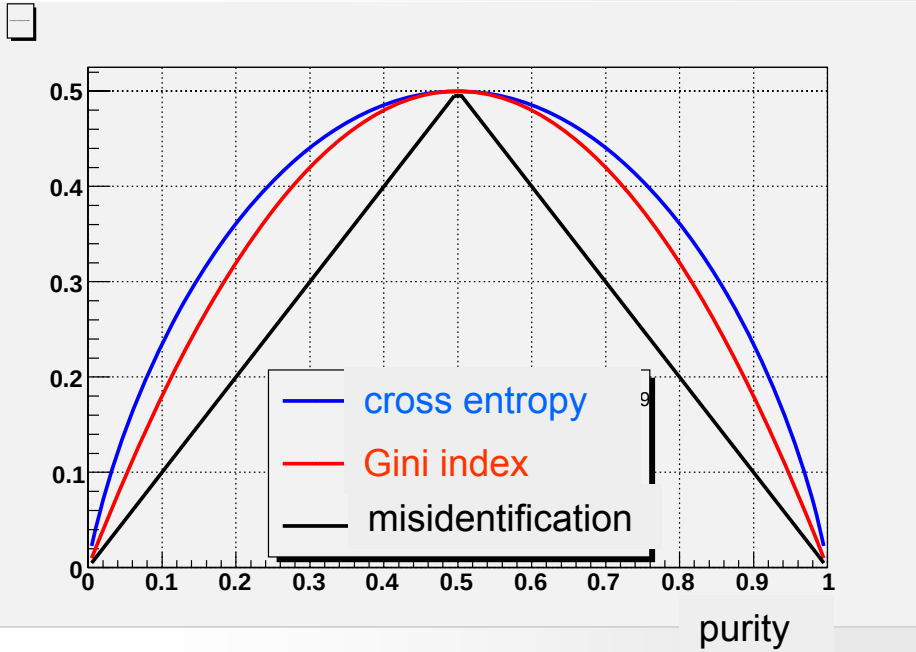
Separation Gain

- 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

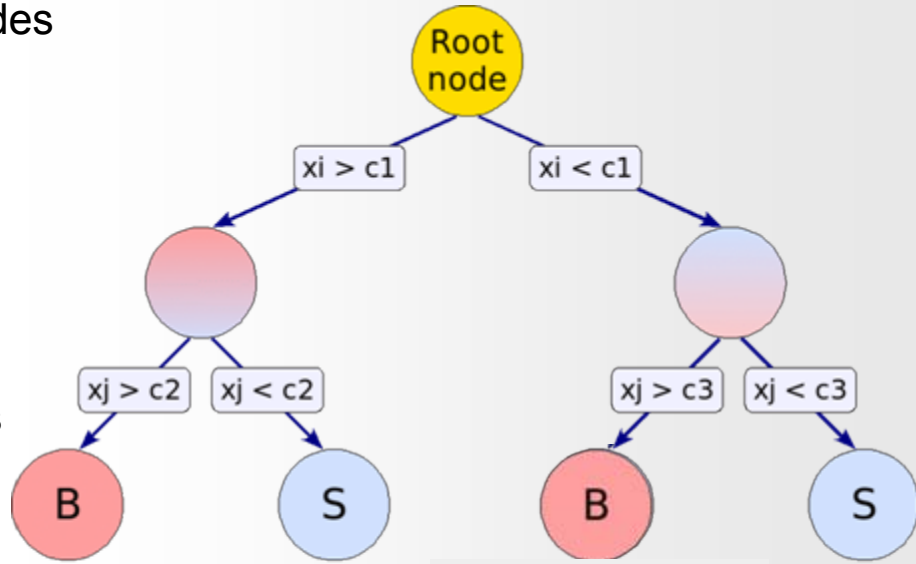
separation gain: e.g. $N_{Parent}*Gini_{Parent} - N_{left}*Gini_{LeftNode} - N_{right}*Gini_{RightNode}$

- 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

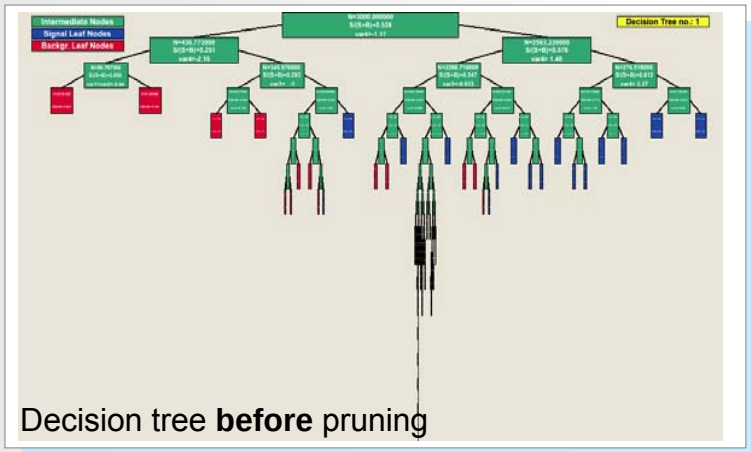
- e.g. Cost Complexity pruning:

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

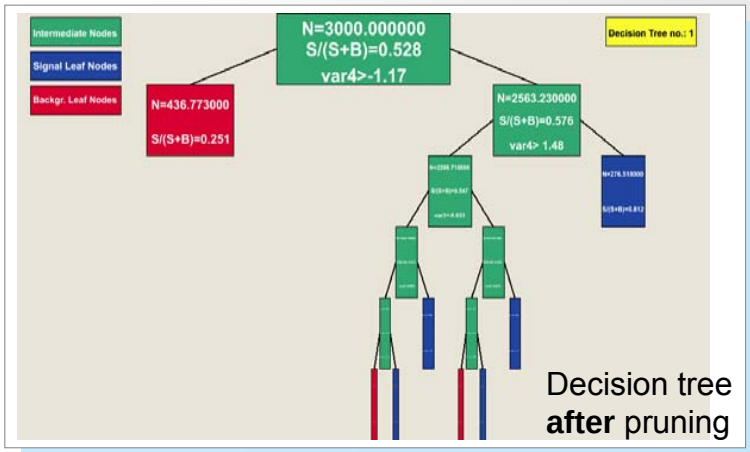
  - assign to every sub-tree, T C(T,α) :
  - find subtree T with minmal C(T,α) for given α
  - prune up to value of a that does not show overtraining in the test sample

Decision Tree Pruning

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



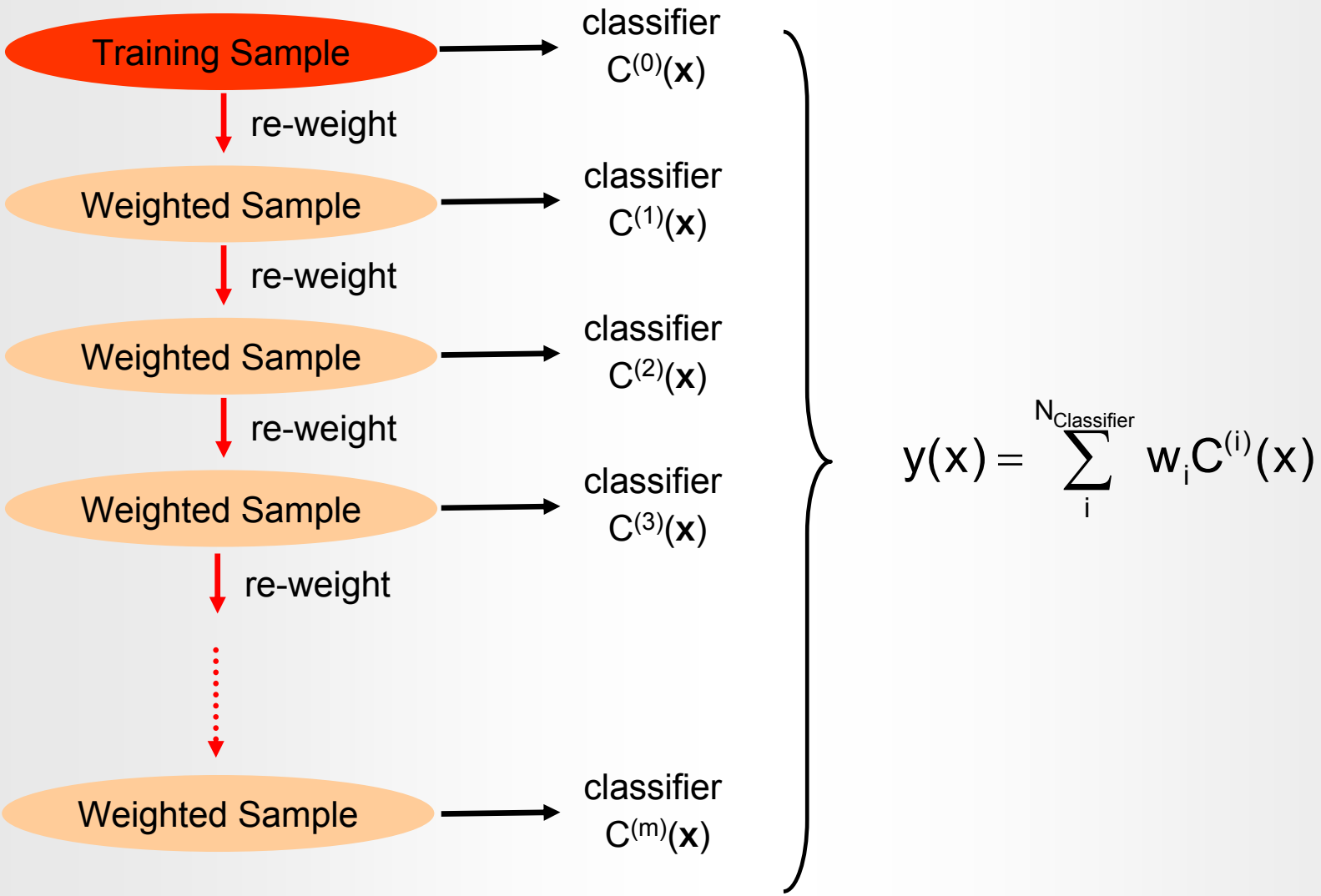
Decision tree **before** pruning



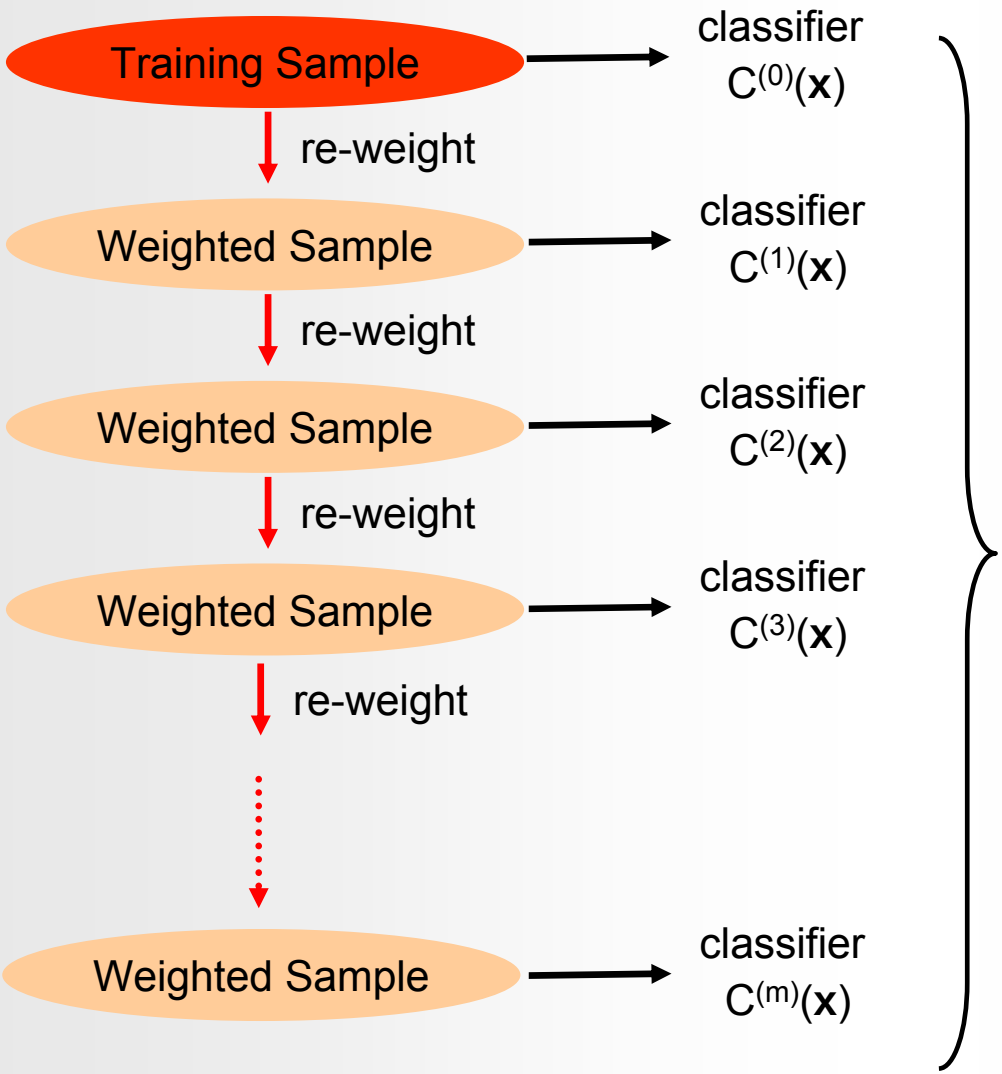
Decision tree **after** pruning

- 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)

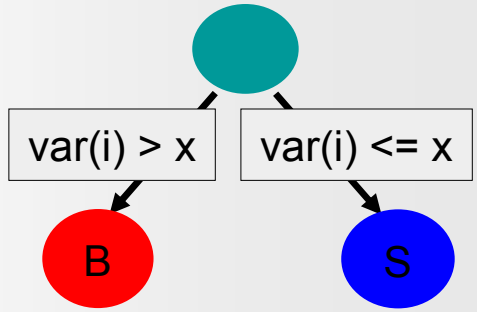
- Randomised Trees: **(Random Forest: trademark L.Breiman, A.Cutler)**
  - combine trees grown with:
    - random subsets of the training data only
    - consider at each node only a random subsets of variables for the split
    - NO Pruning!

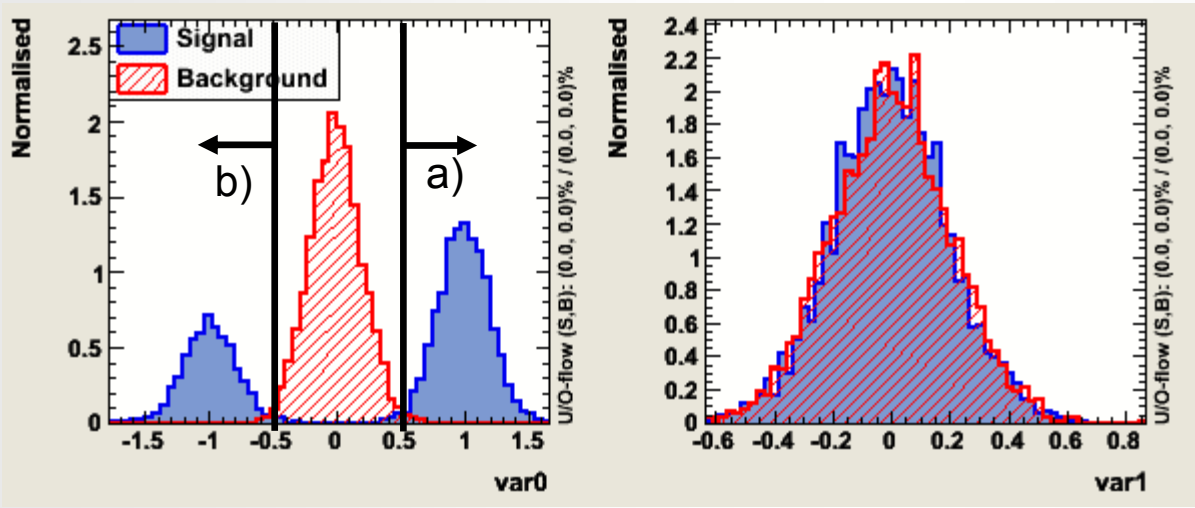
- These combined classifiers work surprisingly well, are very stable and almost perfect “out of the box” classifiers

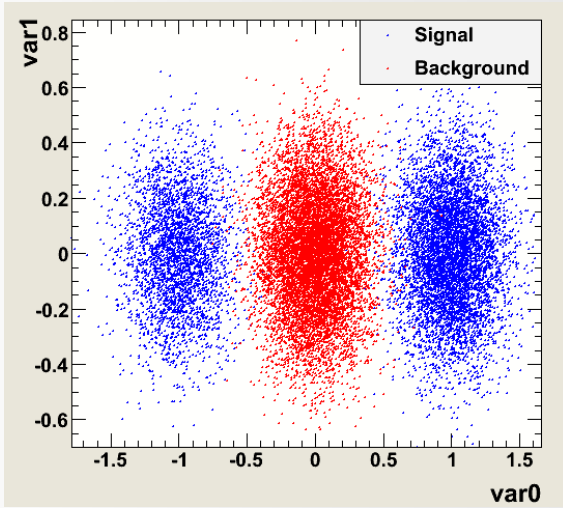
AdaBoost: A simple demonstration

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%

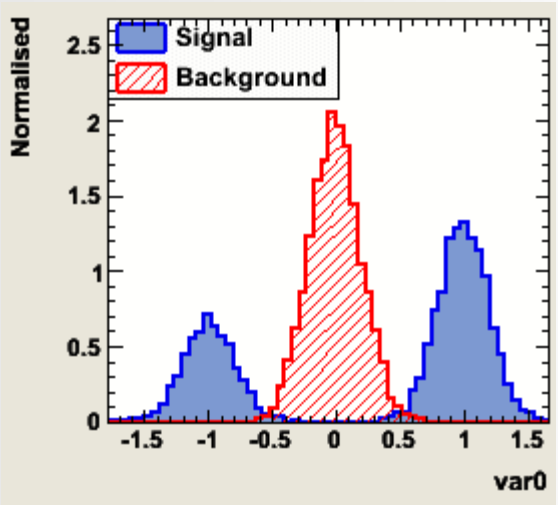
the training of a single decision tree stump will find “cut a)”

AdaBoost: A simple demonstration

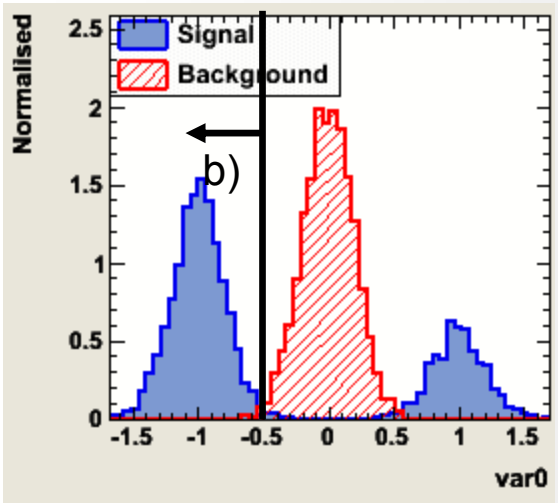
The first “tree”, choosing cut a) will give an error fraction: err = 0.165

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

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

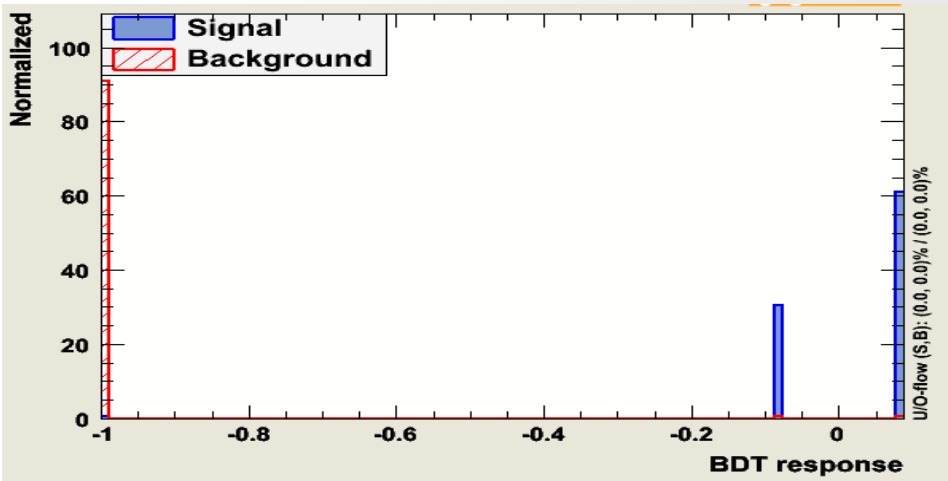


re-weight



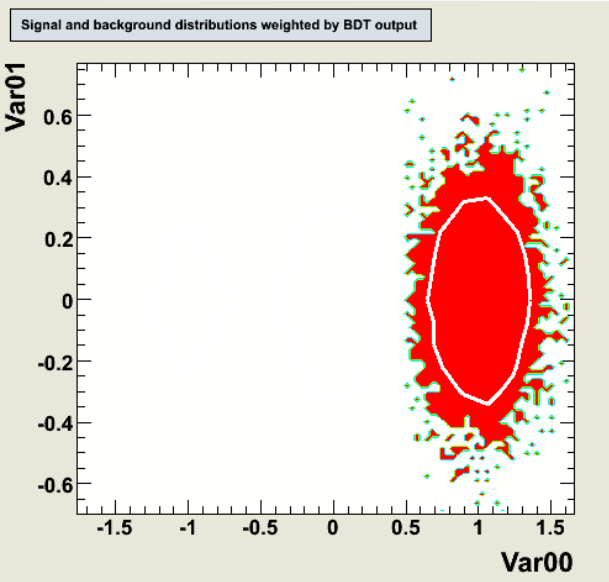
.. and hence will chose: “cut b)”: Var0 < -0.5

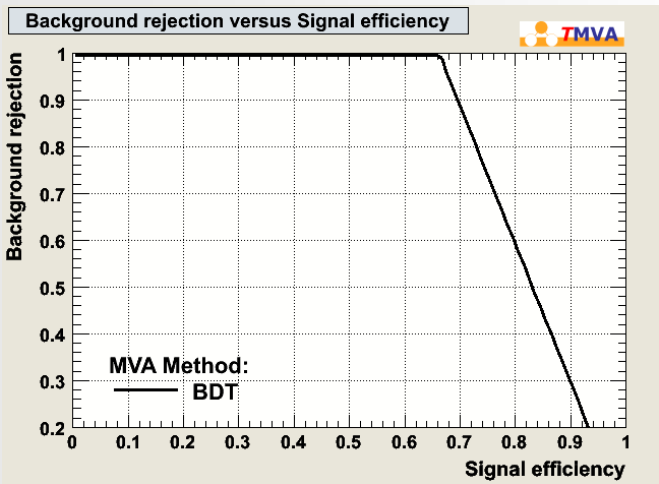
The combined classifier: Tree1 + Tree2
the (weighted) average of the response to a test event from both trees is able to separate signal from background as good as one would expect from the most powerful classifier



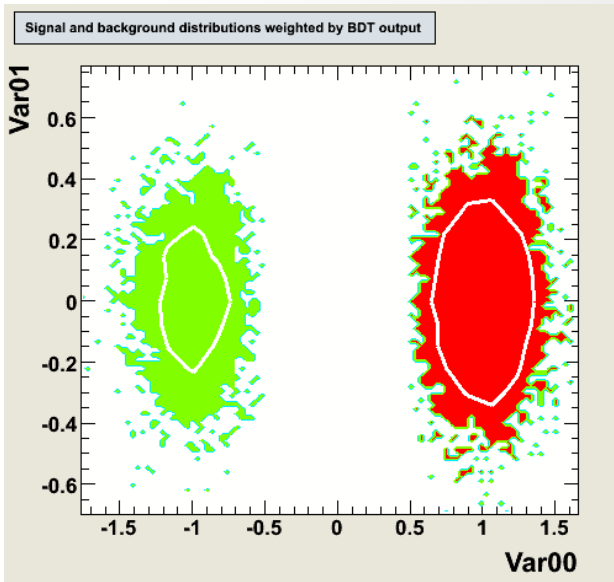
AdaBoost: A simple demonstration

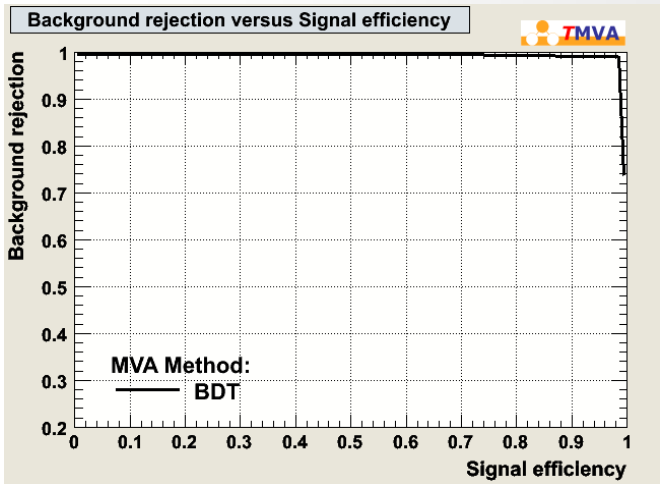
Only 1 tree “stump”





Only 2 tree “stumps” with AdaBoost





AdaBoost vs other Combined Classifiers

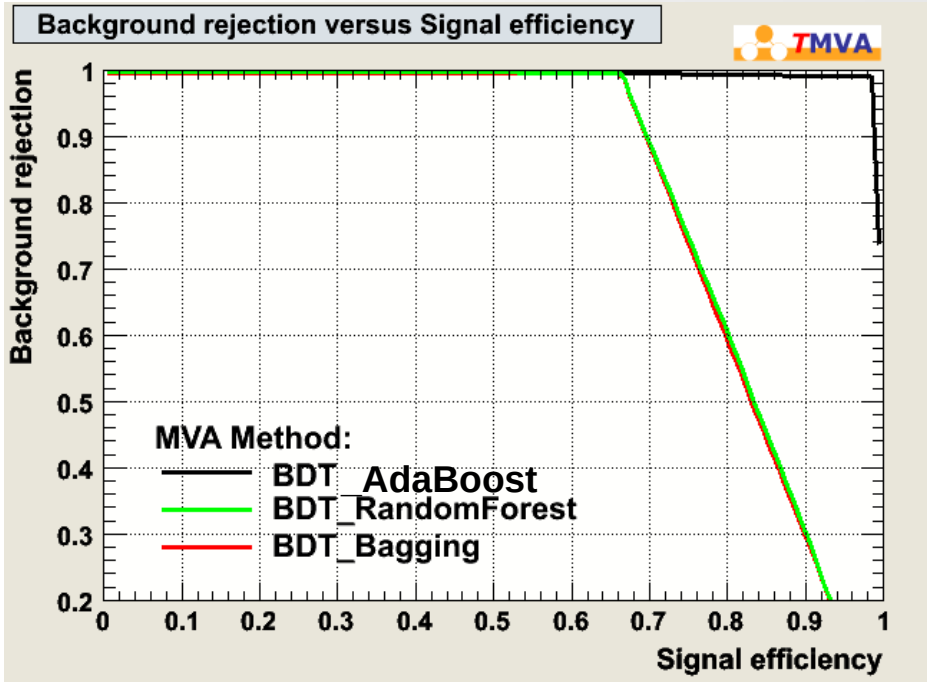
Sometimes people present “boosting” as nothing else then just “smearing” in order to make the Decision Trees more stable w.r.t statistical fluctuations in the training.

→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 !

Boosting at Work