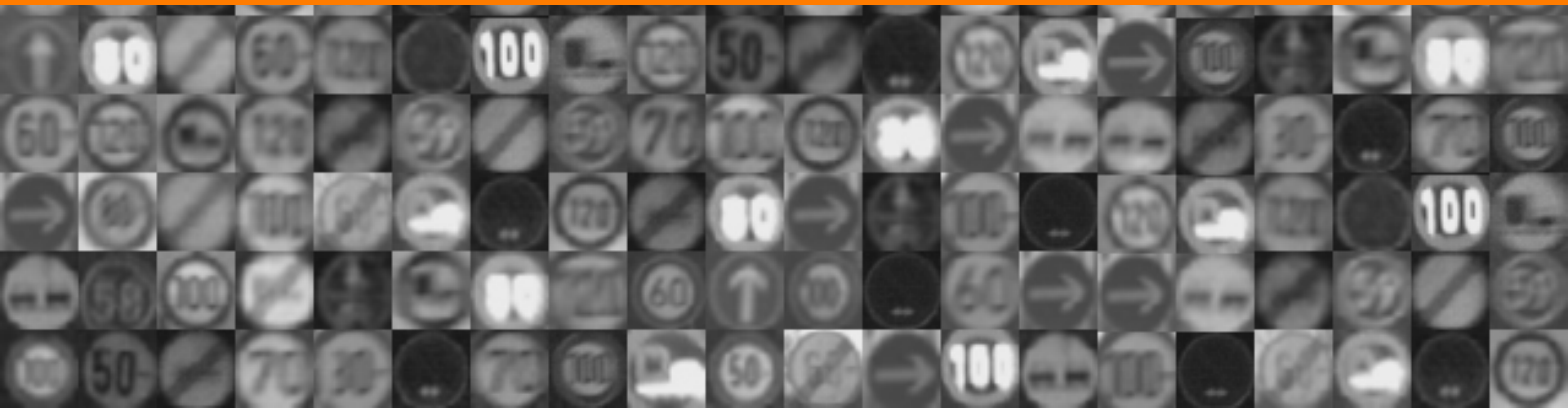
Decision Trees and Boosting

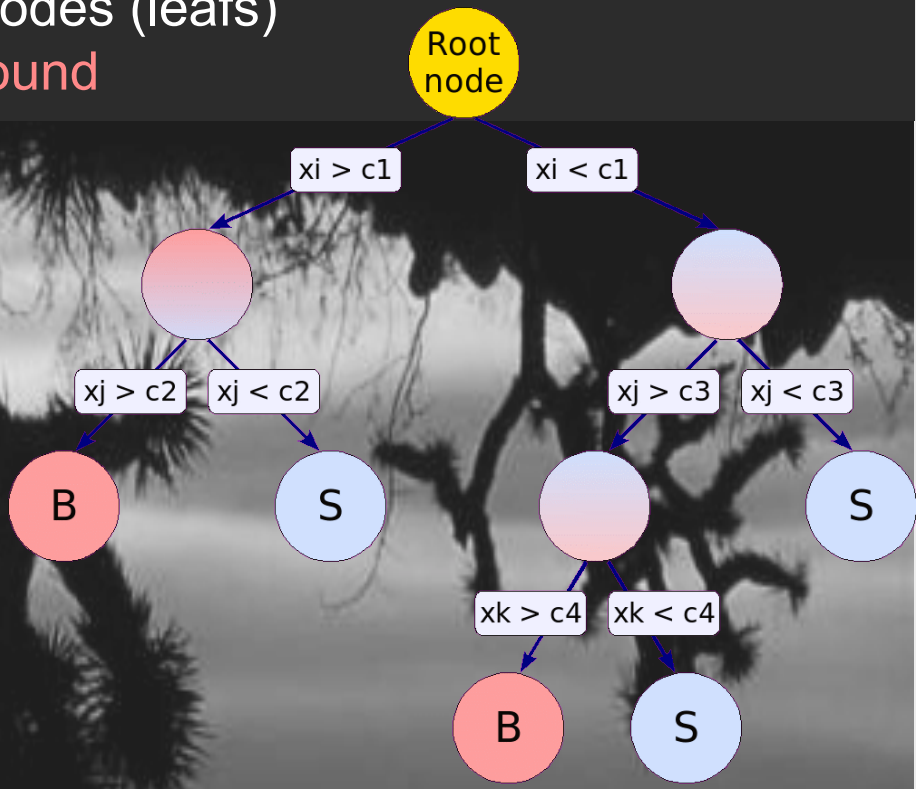
Helge Voss (MPI–K, Heidelberg)

TMVA Workshop , CERN, 21 Jan 2011



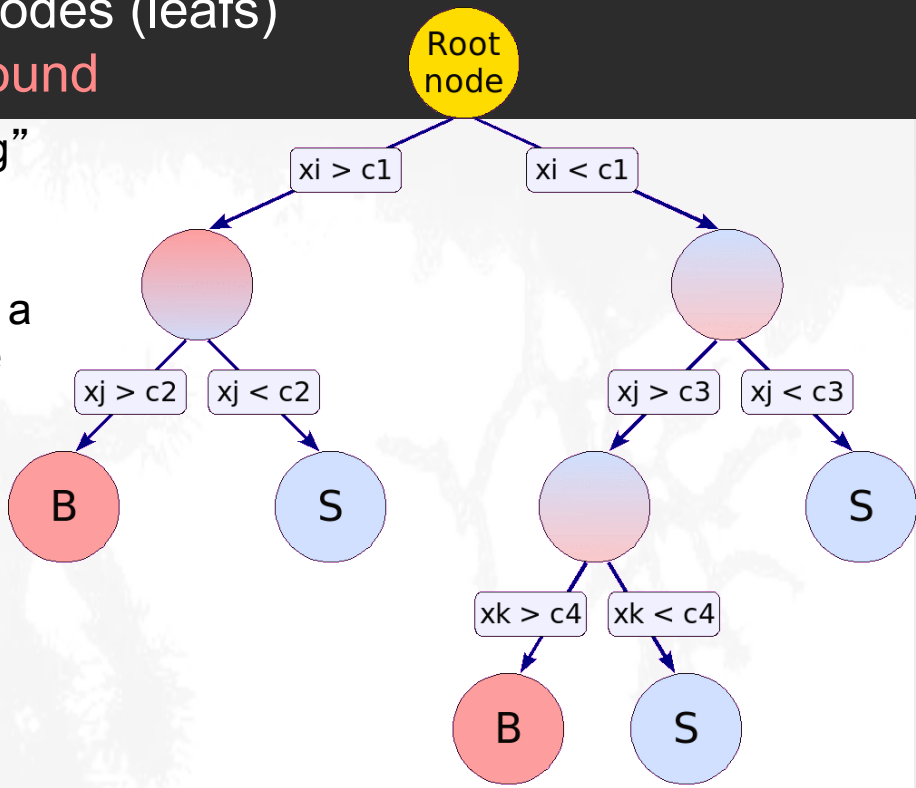
Boosted Decision Trees

- 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



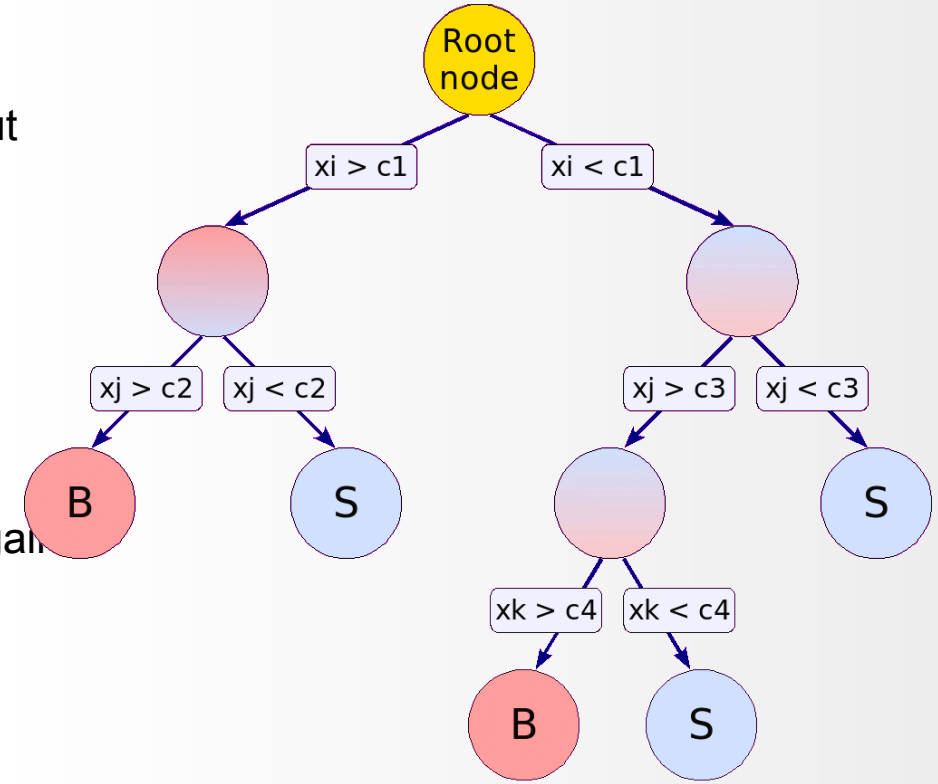
- used since a long time in general "data-mining" applications, less known in (High Energy) Physics
  - similar to "simple Cuts": each leaf node is a set of cuts. → many boxes in phase space attributed either to signal or backgr.
  - independent of monotonous variable transformations, immune against outliers
  - weak variables are ignored (and don't (much) deteriorate performance)
- Disadvantage → 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 correlations

  → we'll see later that for "boosted" DTs weak (dull) classifiers are often better, anyway

Separation Gain

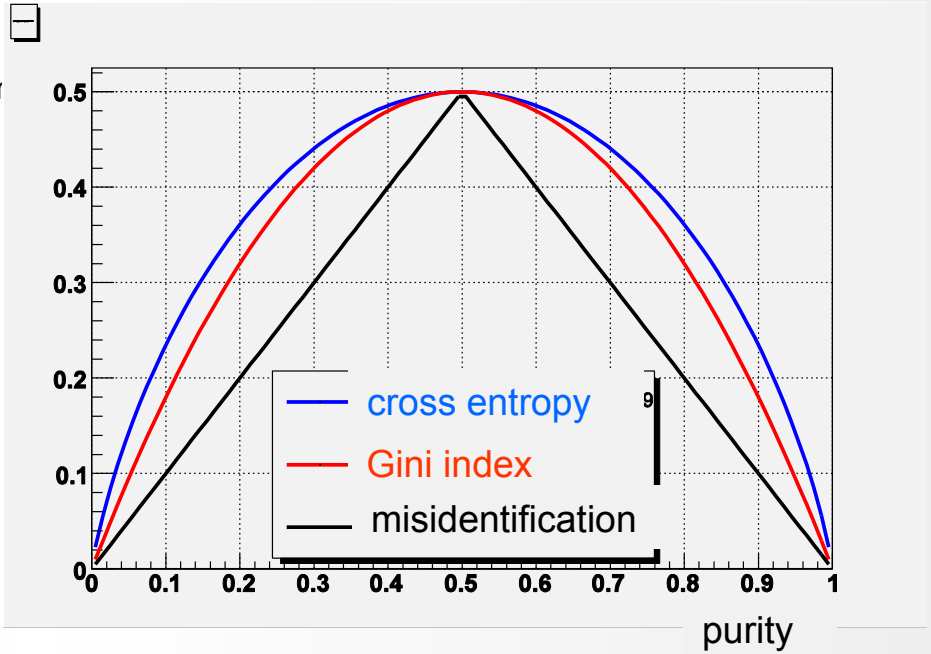
- What do we mean by "best separation gain"?

- define a measure on how mixed S and B are in a node:
  - MisClassification:

    1-max(p,1-p)
  - Gini-index: (Corrado Gini 1912, typically used to r

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

    -(plnp + (1-p)ln(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}*$Gini_{RightNode}$

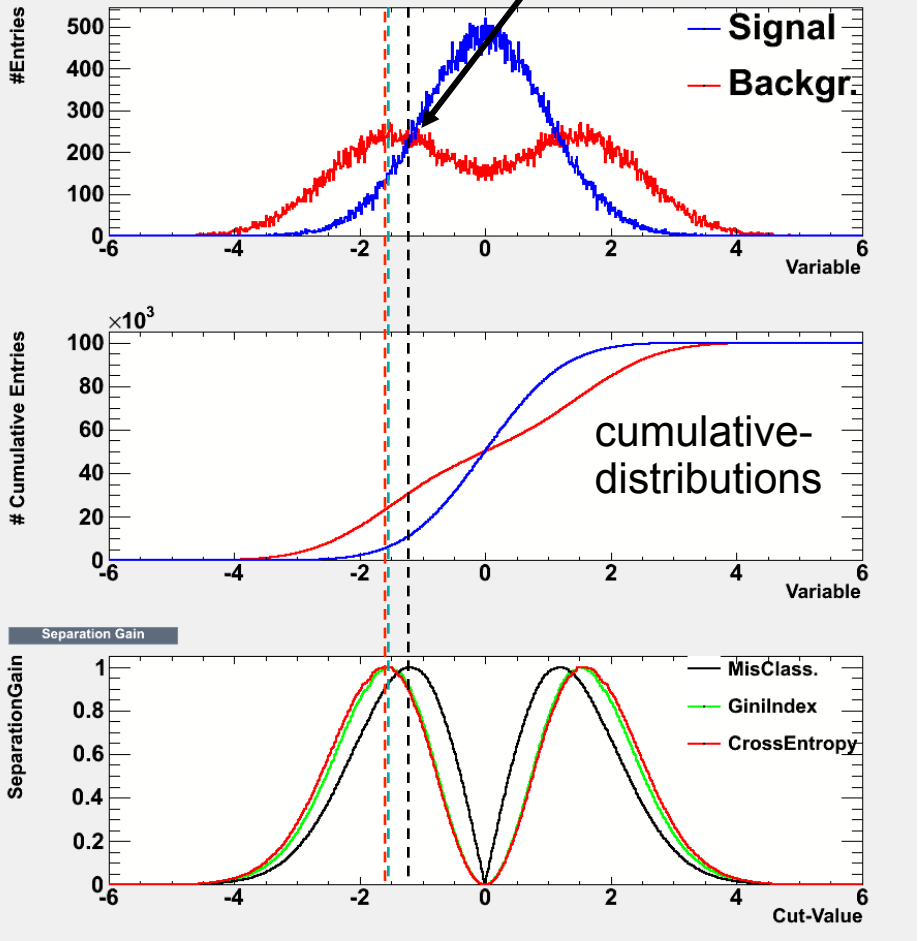
- Consider *all* variables and *all* possible cut values

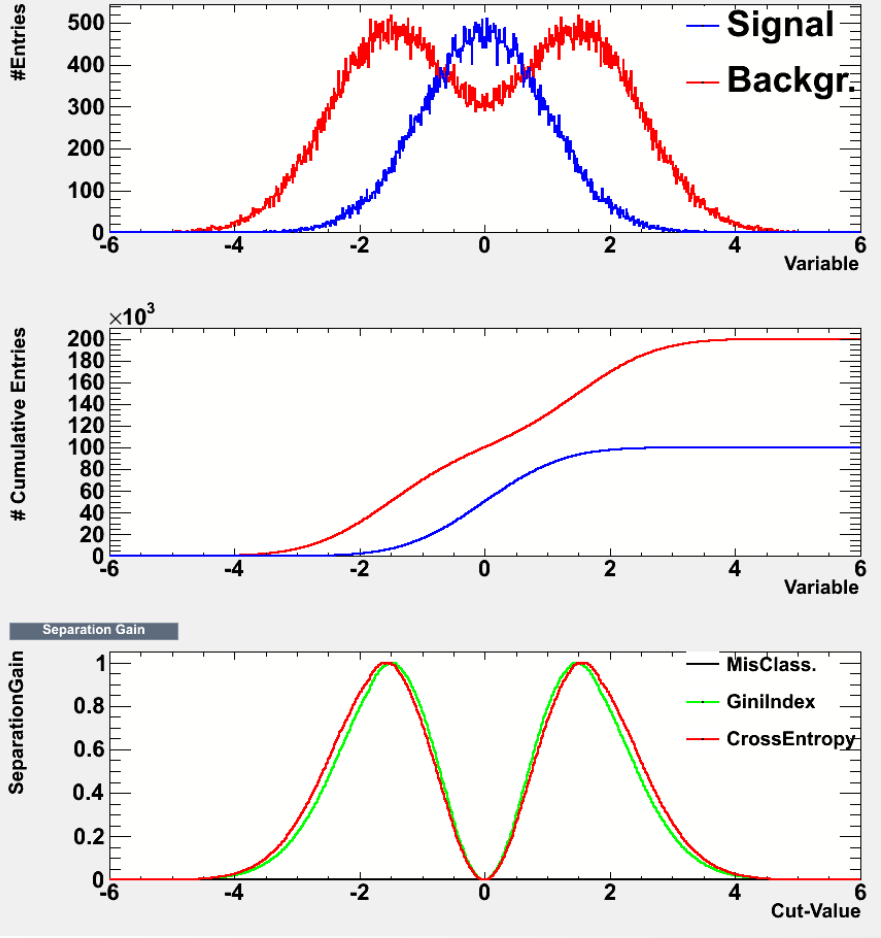
  → select variable and cut that maximises the separation gain.

Separation Gain

MisClassificationError → sort of the classical way to choose cut

BUT:





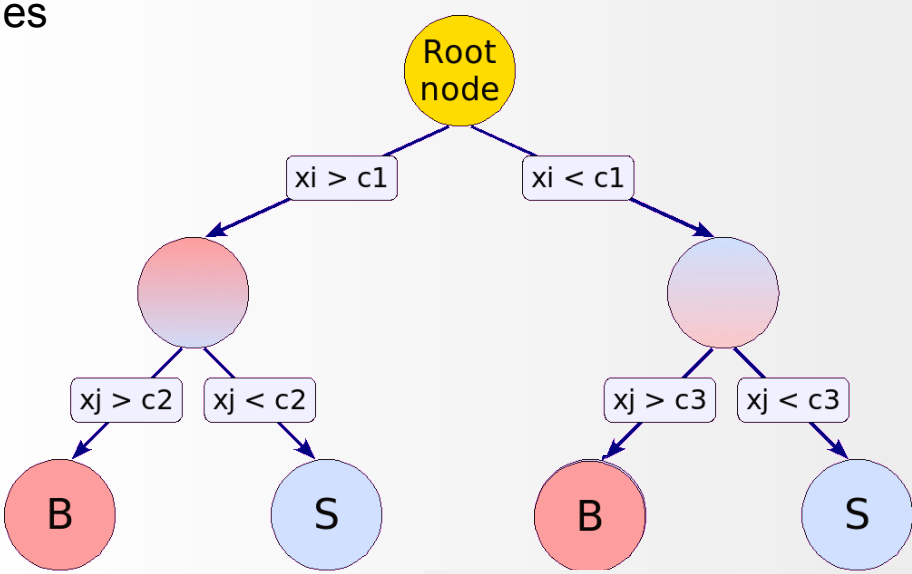
There are cases where the simple "misclassificaton" does not have any optimium at all!

other example: S=400,B=400 → (S=300,B=100) (S=100,B=300) or (S=200,B=0) (S=200,B=400)

→ equal in terms of misclassification error, but GiniIndex/Entropy favour the latter

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, even useless splits might open up subsequent useful splits
  - grow tree to the end and "cut back", nodes that seem statistically dominated:

    → pruning

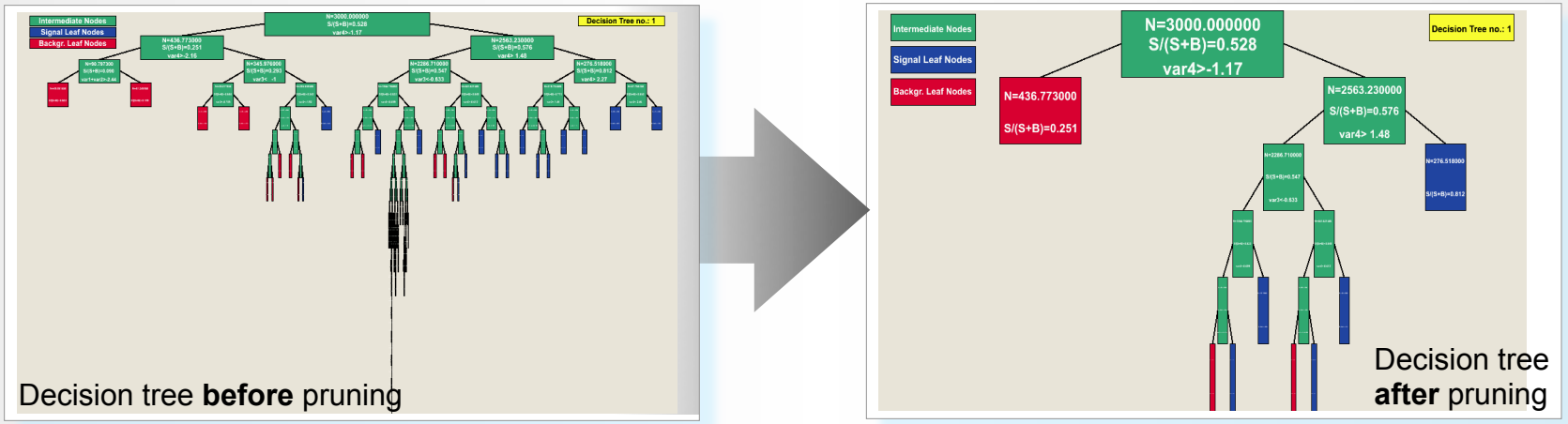
- e.g. Cost Complexity pruning:

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

  - assign to every sub-tree, T $C(T,\alpha)$:
  - find subtree T with minmal $C(T,\alpha)$ for given α
    - use subsequent weakest link pruning
- which cost parameter α ?
  - large enough to avoid overtraining
  - tuning parameter or "cross validation" (still to come in TMVA hopefully soon…)

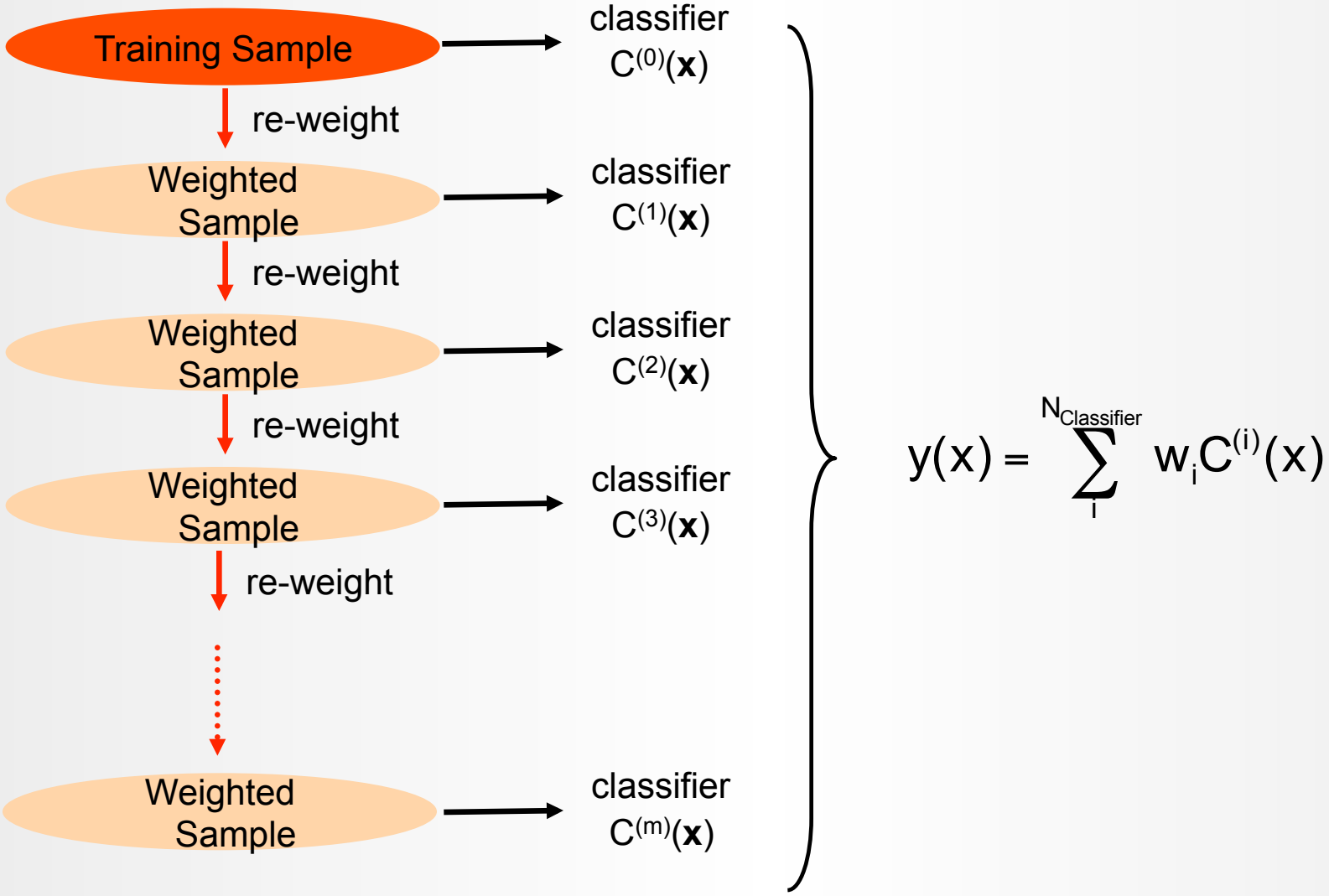
Decision Tree Pruning

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

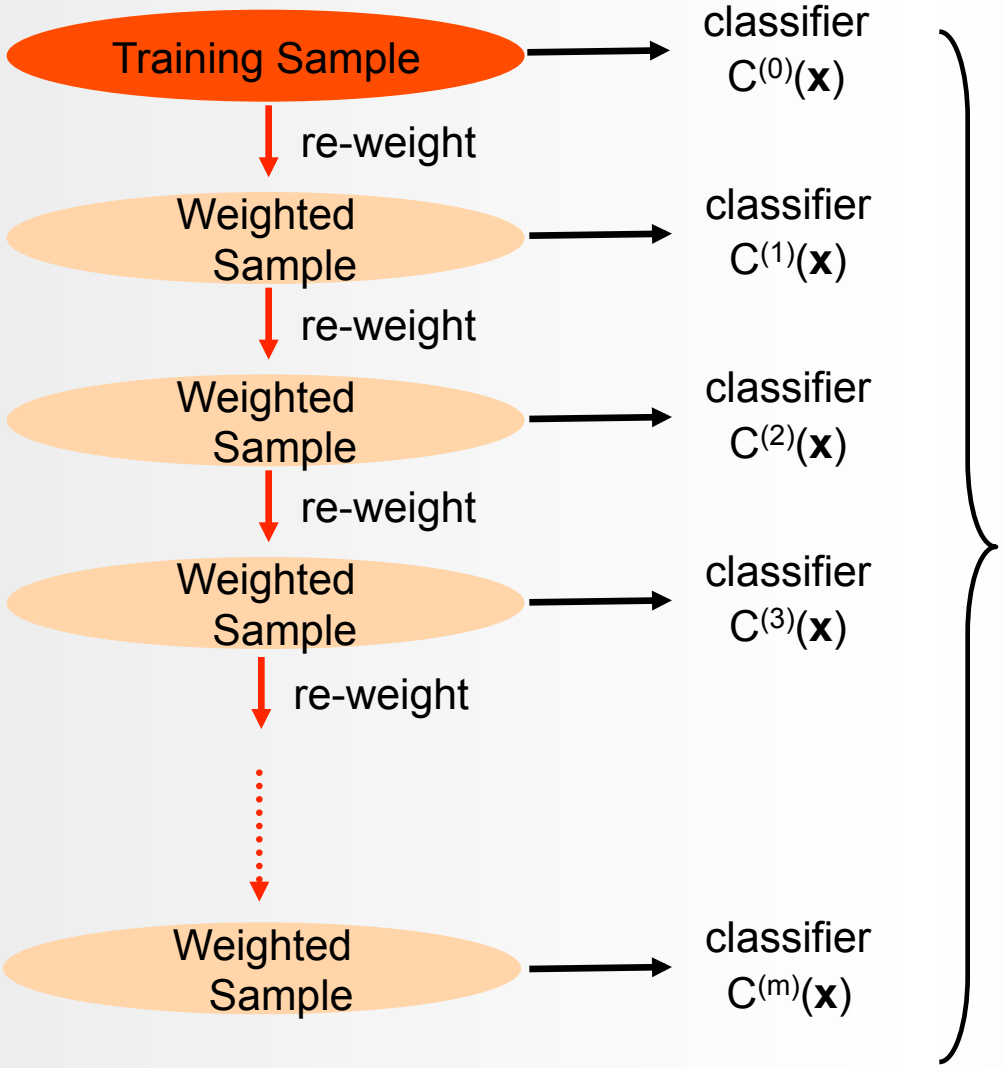


- Pruning algorithms are developed and applied on individual trees
  - optimally pruned single trees are not necessarily optimal in a forest !
  - actually they tend to be TOO big when boosted, no matter how hard you prune!

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:

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

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

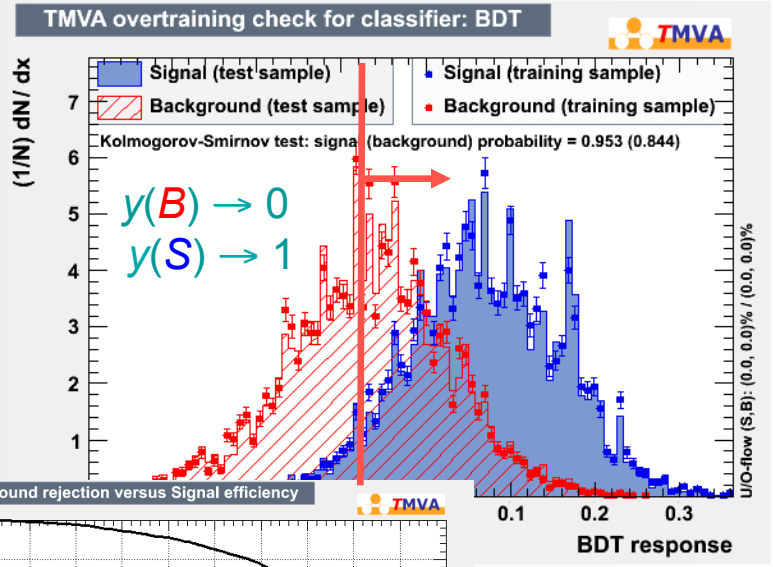
- AdaBoost weights the classifiers also using the error rate of the individual classifier according to:

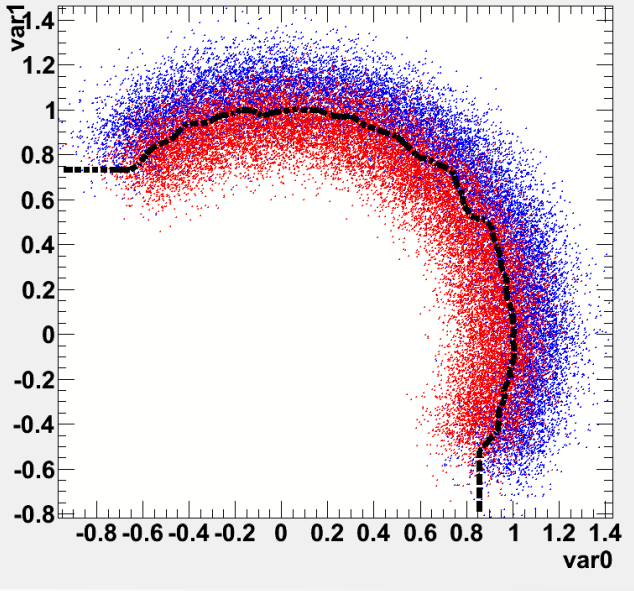
$$y(x) = \sum_{i}^{N_{Classifier}} \log\left(\frac{1-f_{err}^{(i)}}{f_{err}^{(i)}}\right) C^{(i)}(x)$$

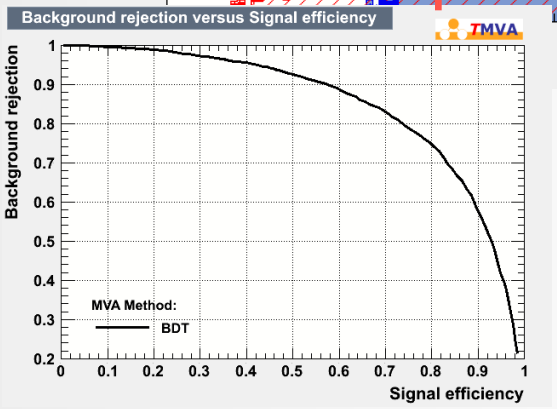
Boosted Decision Trees

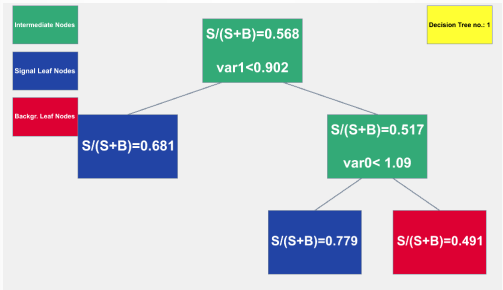
- Result of ONE Decision Tree for test event is either "Signal" or "Background"
  - the tree gives a fixed signal eff. and background rejection

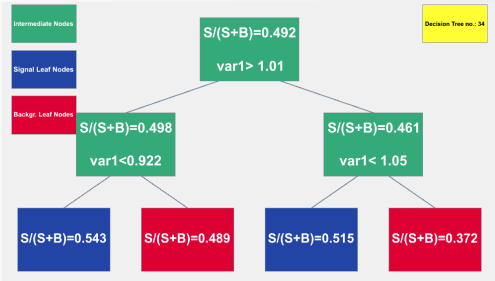
- For a whole Forest however:











AdaBoost in Pictures

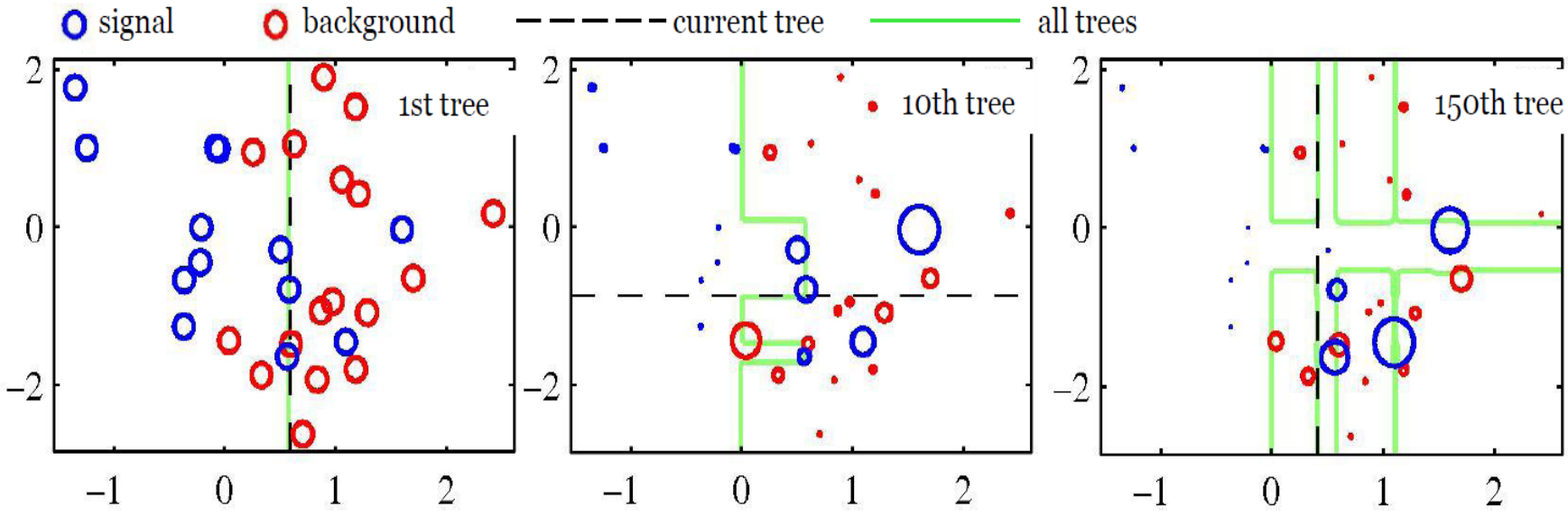
Start here:

equal event weights

misclassified events get

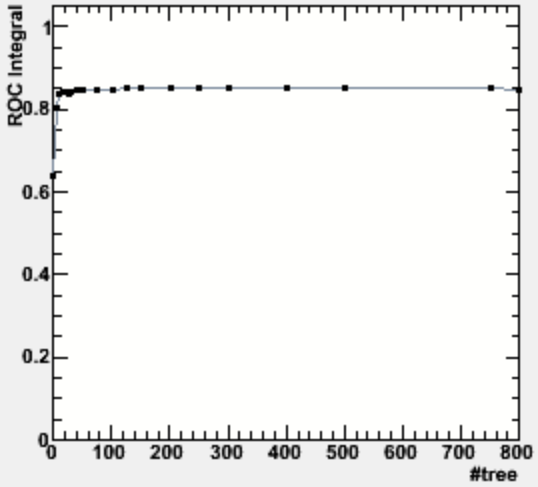
larger weights

… and so on

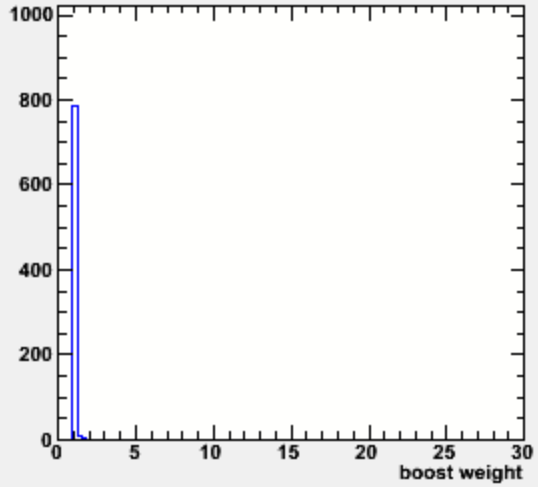


Boosted Decision Trees – Control Plots

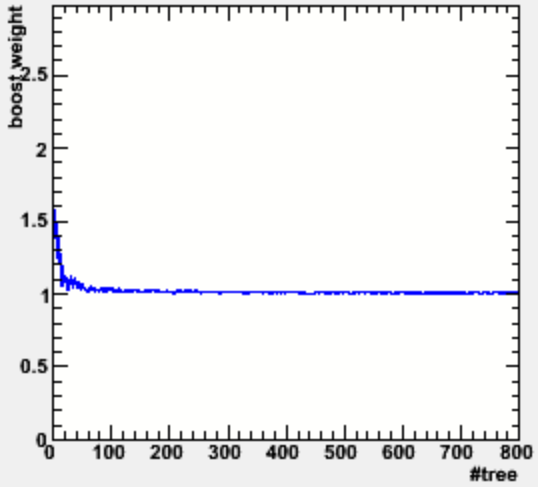
ROC Integral Vs iTree



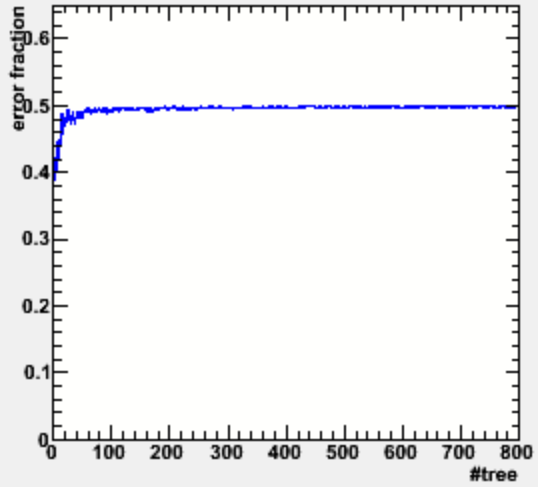
AdaBooost weight distribution



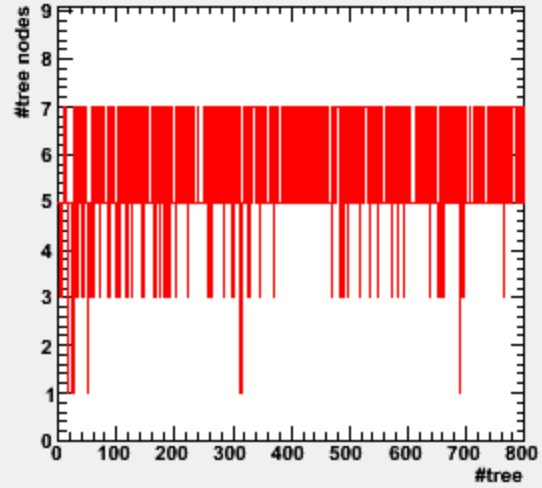
Boost weights vs tree



error fraction vs tree number

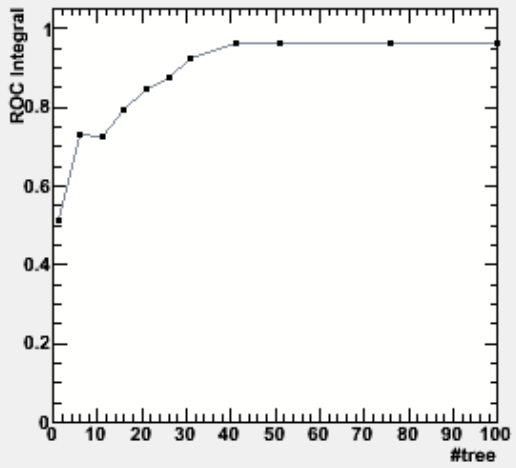


Nodes before/after pruning

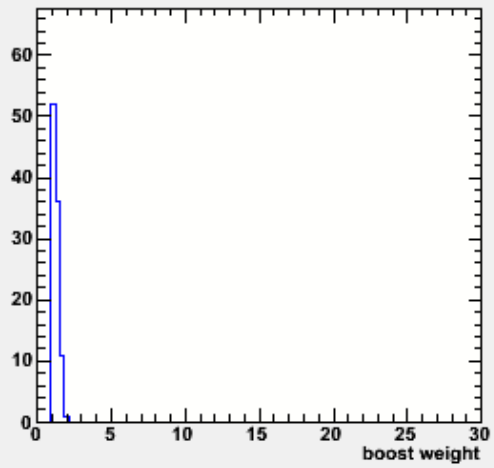


Boosted Decision Trees – Control Plots

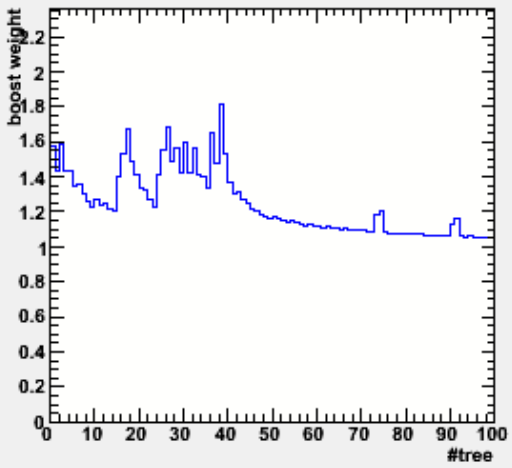
ROC Integral Vs iTree



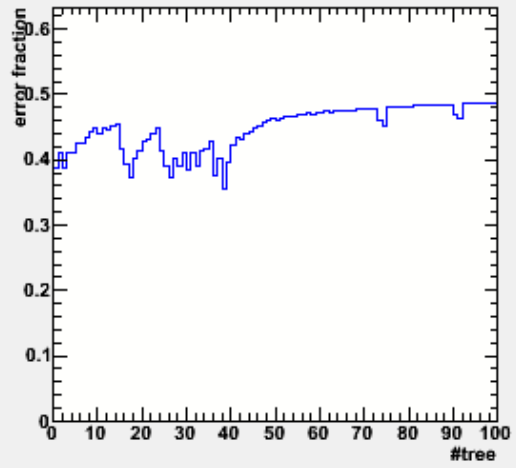
AdaBooost weight distribution



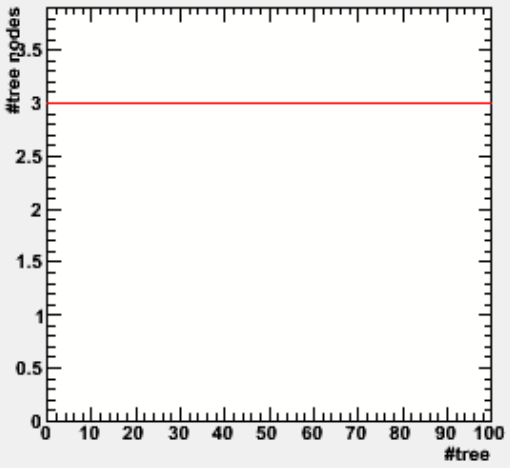
Boost weights vs tree



error fraction vs tree number



Nodes before/after pruning

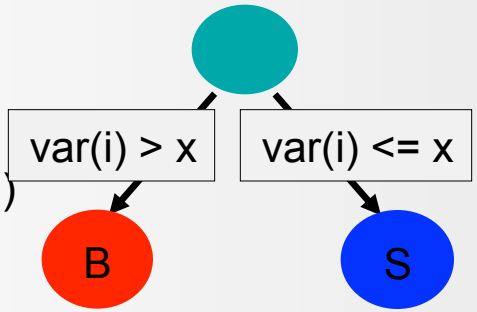


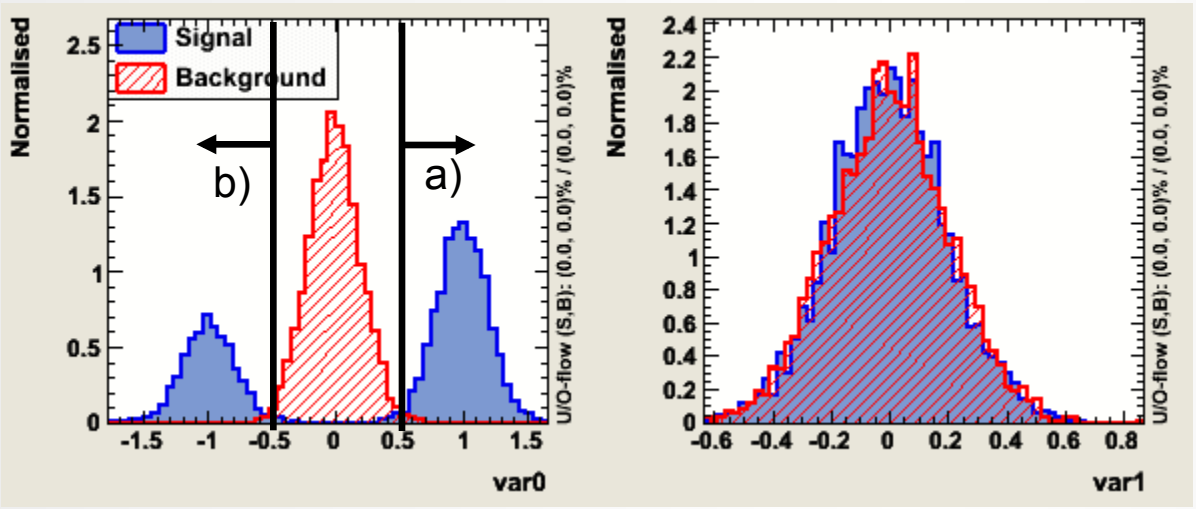
A more "difficult" example

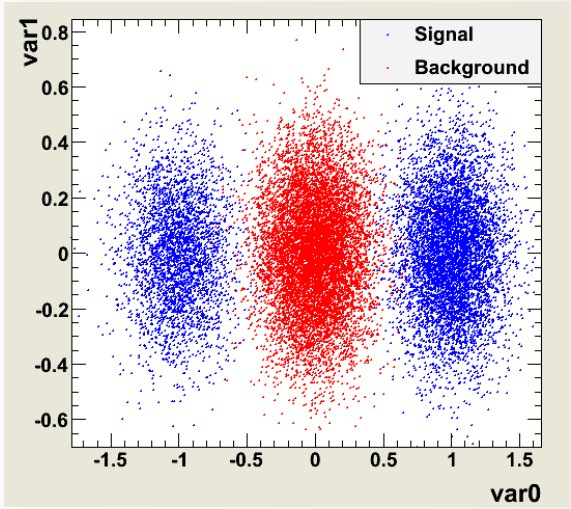
AdaBoost: A simple demonstration

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% ε_{bkg} ≈ 0% misclassified events in total 16.5%

or

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

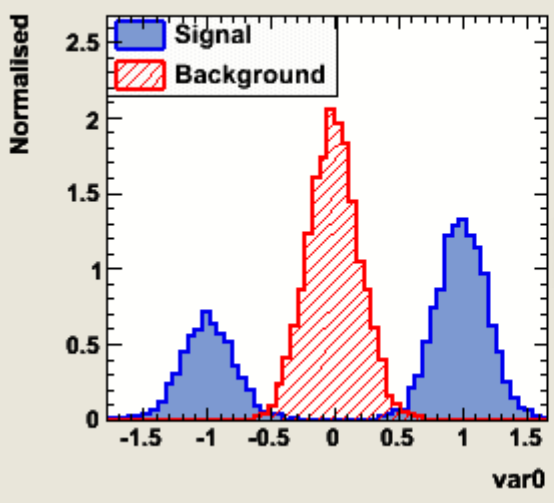
the training of a single decision tree stump will find "cut a)"

AdaBoost: A simple demonstration

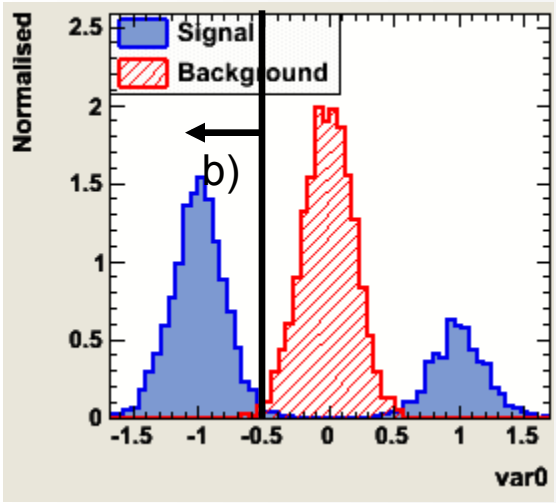
The first “tree”, choosing cut a) will give an error fraction: err = 0.165

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

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

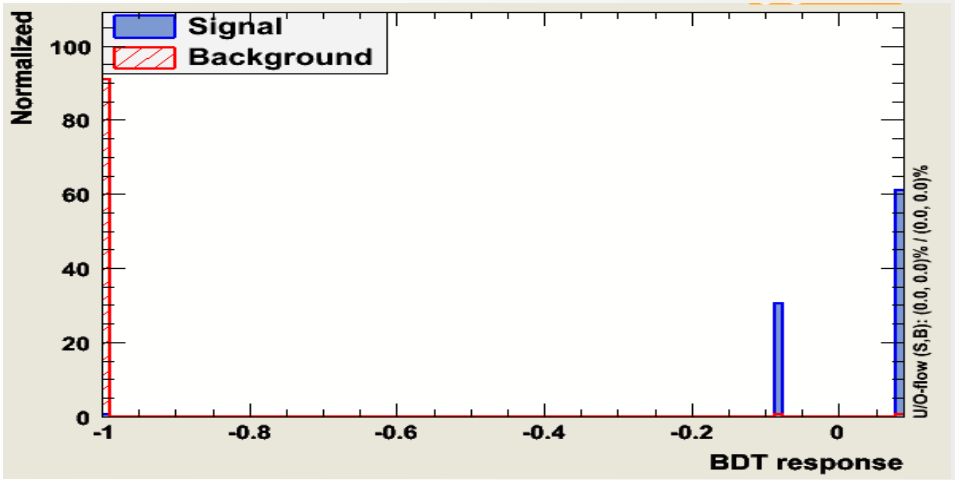


re-weight



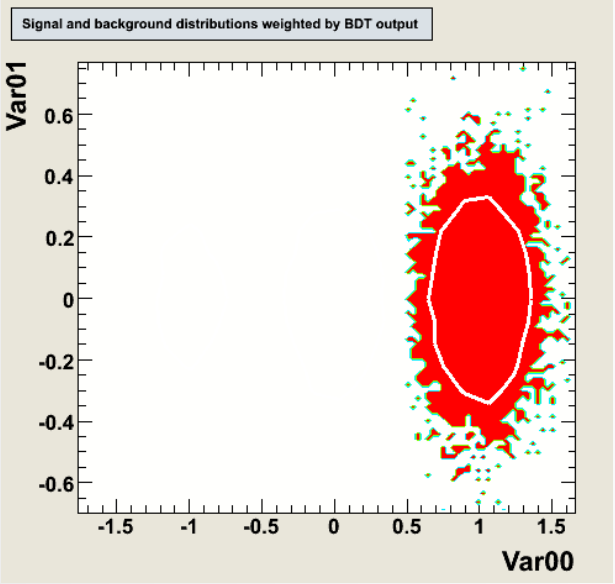
.. and hence will chose: “cut b)”: Var0 < -0.5

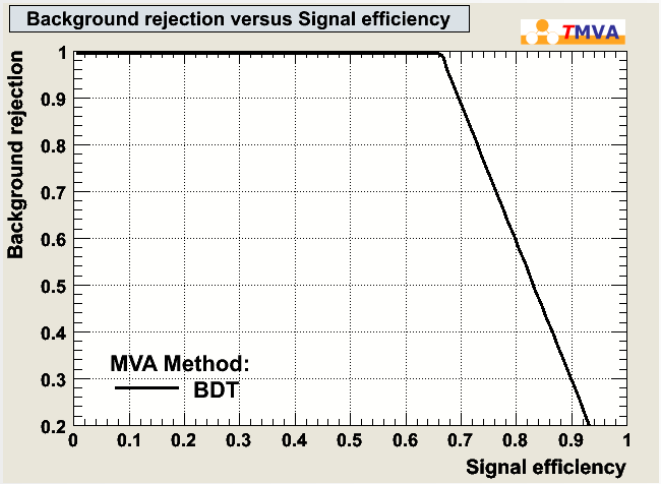
The combined classifier: Tree1 + Tree2
the (weighted) average of the response to a test event from both trees is able to separate signal from background as good as one would expect from the most powerful classifier



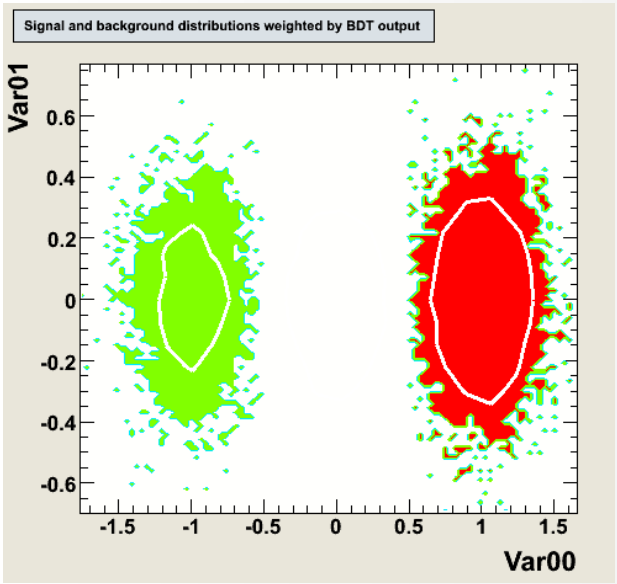
AdaBoost: A simple demonstration

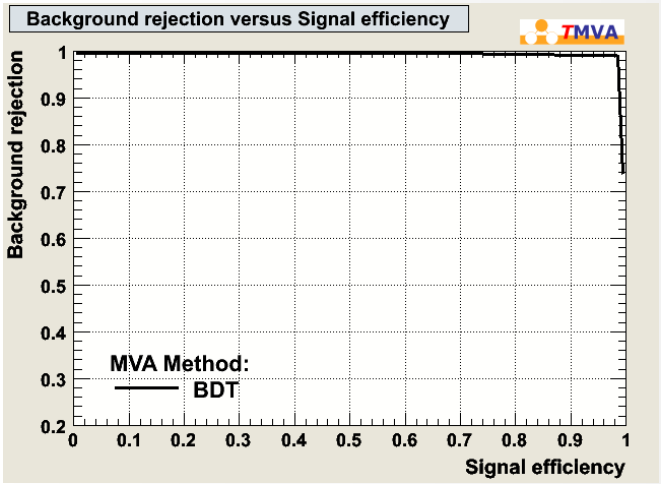
Only 1 tree “stump”





Only 2 tree “stumps” with AdaBoost





"A Statistical View of Boosting" (Friedman 1998 et.al)

- Boosted Decision Trees: two different interpretations
  - give events that are "difficult to categorize" more weight and average afterwards the results of all classifiers that were obtained with different weights
  - see each Tree as a "basis function" of a possible classifier →
    - boosting or bagging is just a mean to generate a set of "basis functions"
    - linear combination of basis functions gives final classifier or: final classifier is an expansion in the basis functions.

$$y(\vec{\alpha}, x) = \sum_{\text{tree}} \alpha_i T_i(x)$$

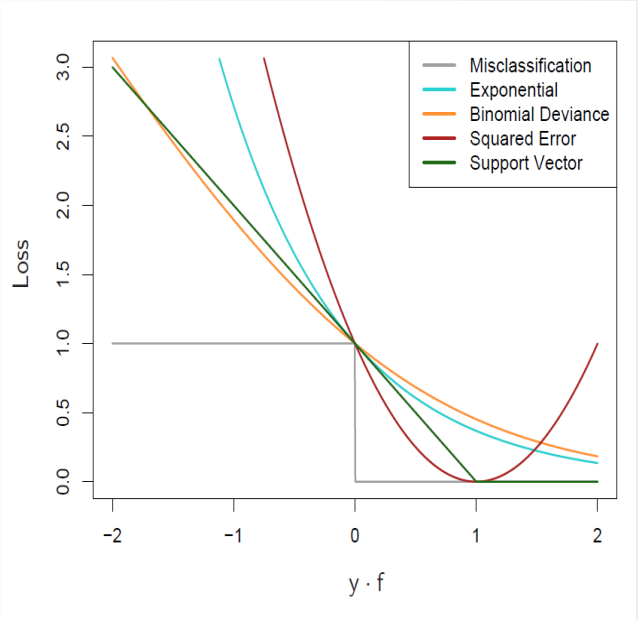
    - every "boosting" algorithm can be interpreted as optimising the loss function in a "greedy stagewise" manner
      - *i.e.* from the current point in the optimisation – *e.g.building of the decision tree forest-* :
        - chooses the parameters for the next boost step (weights) such that one moves a long the steepest gradient of the loss function
    - AdaBoost: "exponential loss function" = exp($-y_0 y(\alpha,x)$) where y_0=-1 (bkg), y_0=1 (signal)

Gradient Boost

- Gradient Boost is a way to implement "boosting" with arbitrary "loss functions" by approximating "somehow" the gradient of the loss function

- AdaBoost: Exponential loss $\exp(-y_0 y(\alpha,x))$ → theoretically sensitive to outliers

- Binomial log-likelihood loss $\ln(1 + \exp(-2y_0 y(\alpha,x))$ → more well behaved loss function,
  (the corresponding "GradientBoost" is implmented in TMVA)



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 bootstrap (or 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 vs Bagging and Randomised Forests

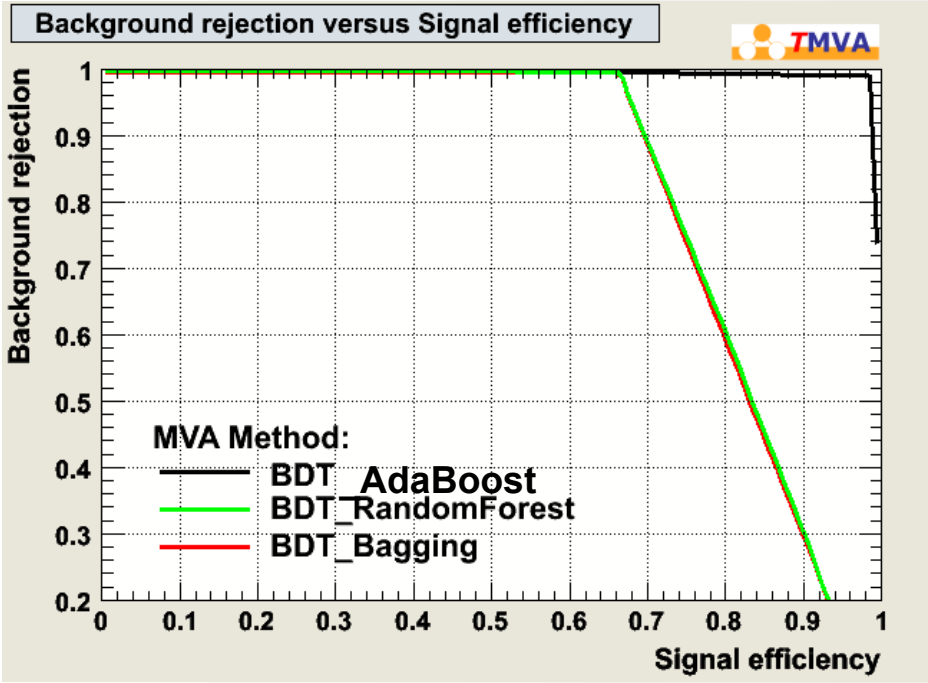
Sometimes people present “boosting” as nothing else then just “smearing” in order to make the Decision Trees more stable w.r.t statistical fluctuations in the training.

→clever “boosting” however can do more, than for example: for previous example of “three bumps”

- Random Forests
- Bagging

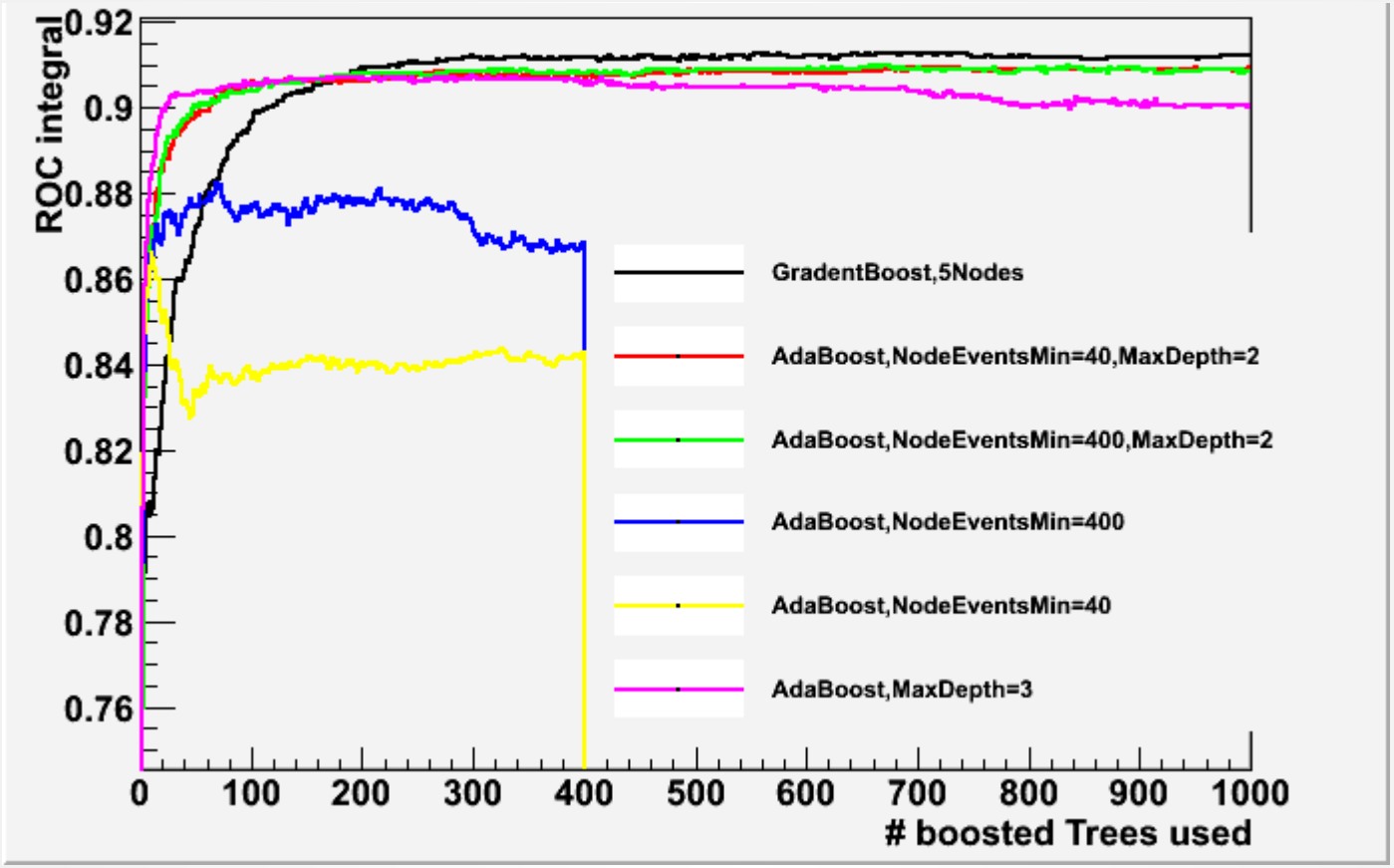
as in this case, pure statistical fluctuations are not enough to enhance the 2nd peak sufficiently

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 !

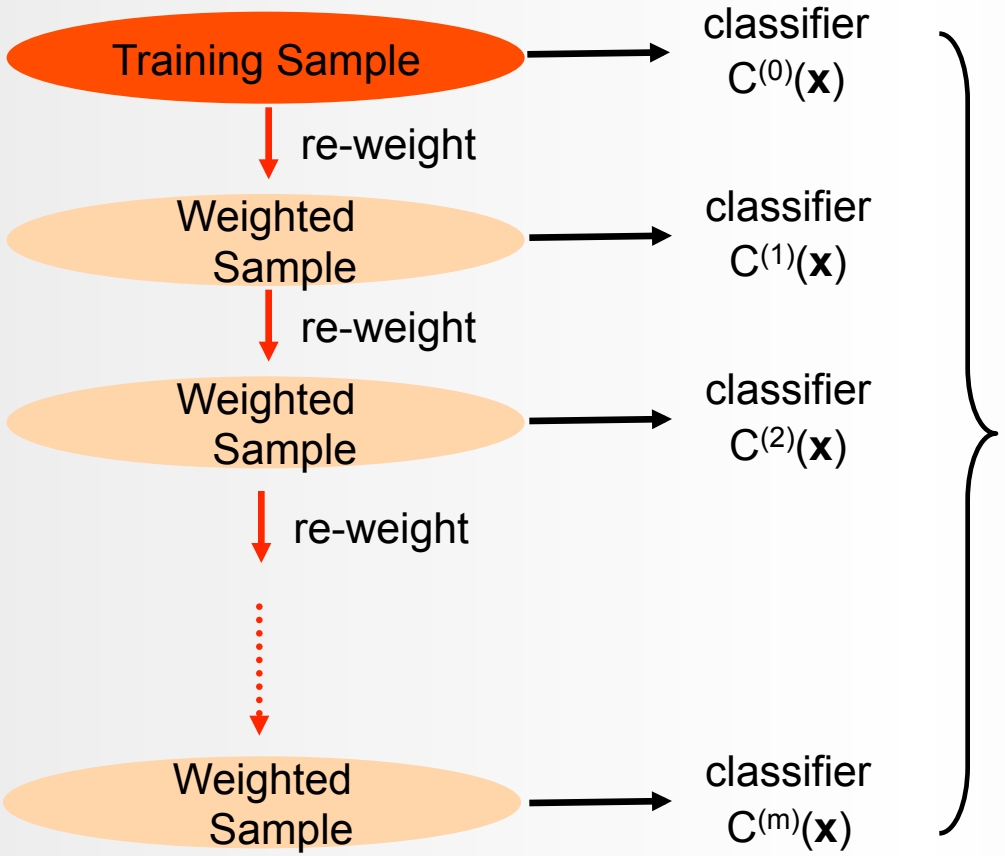
Boosting at Work



- Boosting seems to work best on “weak” classifiers (i.e. small, dum trees)
- Tuning (tree building) parameter settings are important
- For good out of the box performance: Large numbers of very small trees

Generalised Classifier Boosting

- Principle (just as in BDT): multiple training cycles, each time wrongly classified events get a higher event weight



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

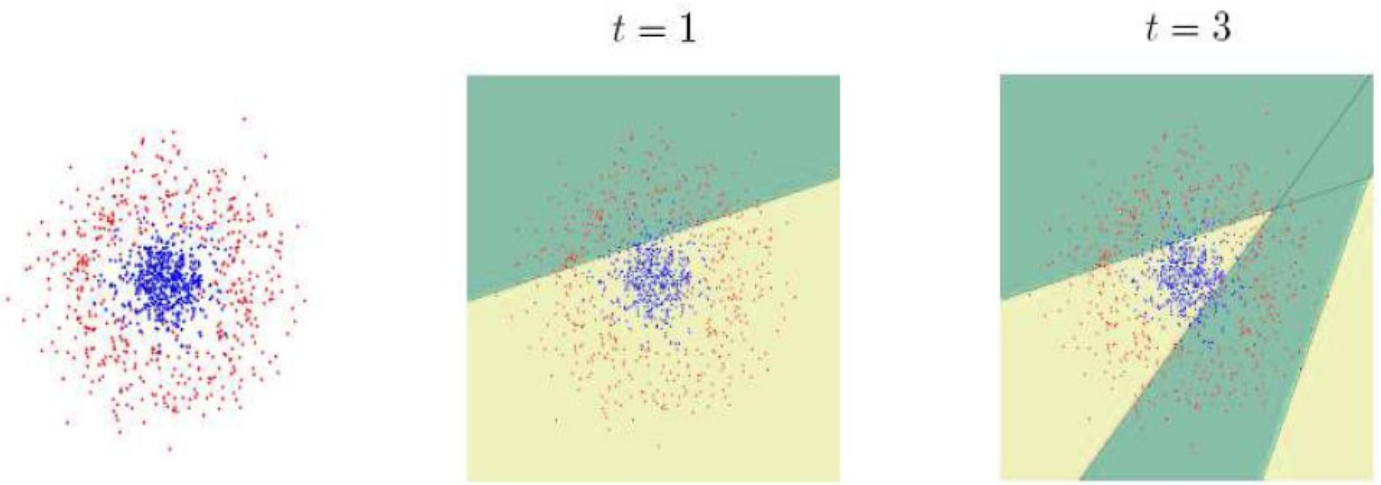
Response is weighted sum of each classifier response

Boosting might be interesting especially for simple (weak) Methods like Cuts, Linear Discriminants, simple (small, few nodes) MLPs

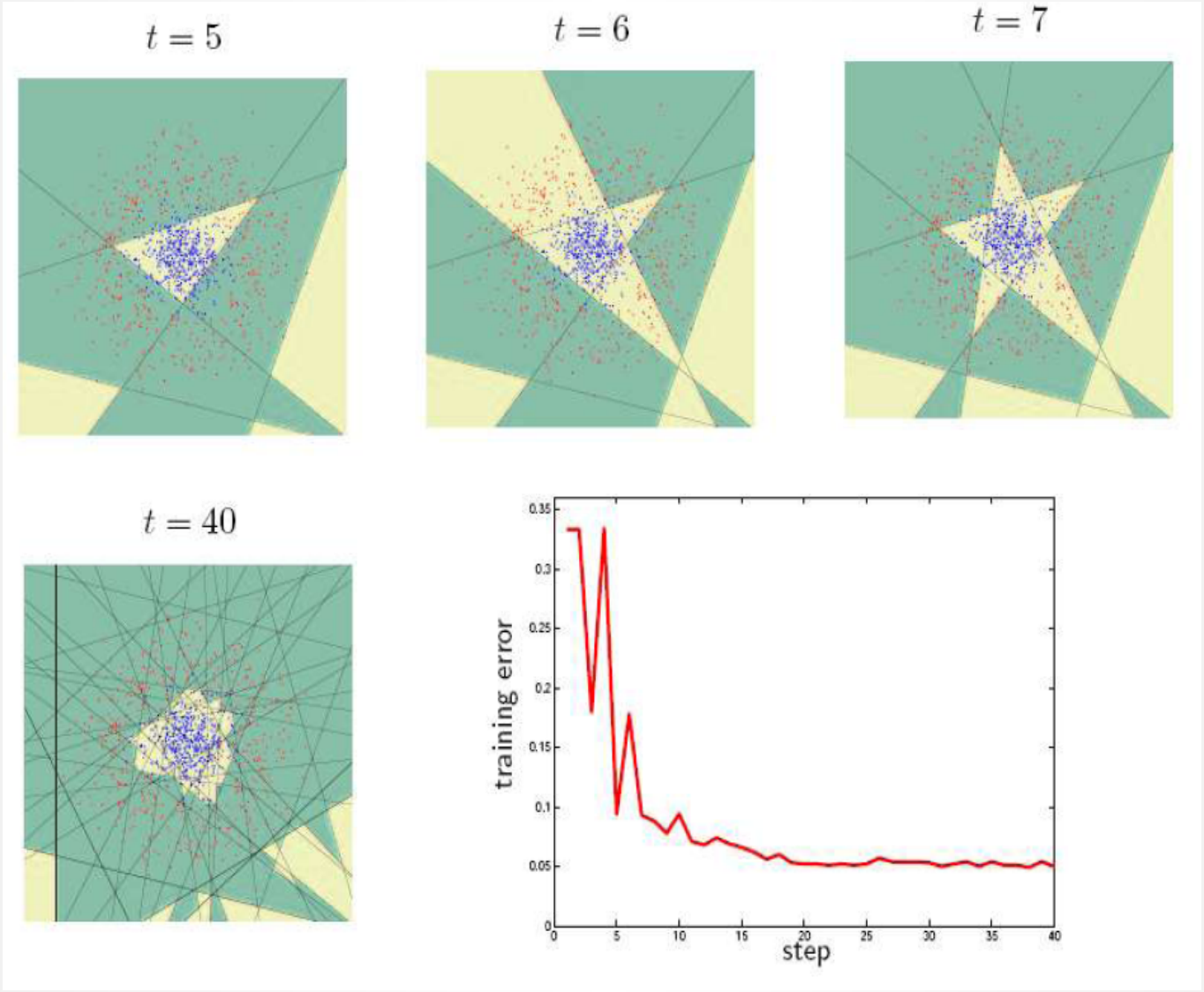
AdaBoost On a linear Classifier (e.g. Fisher)

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

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

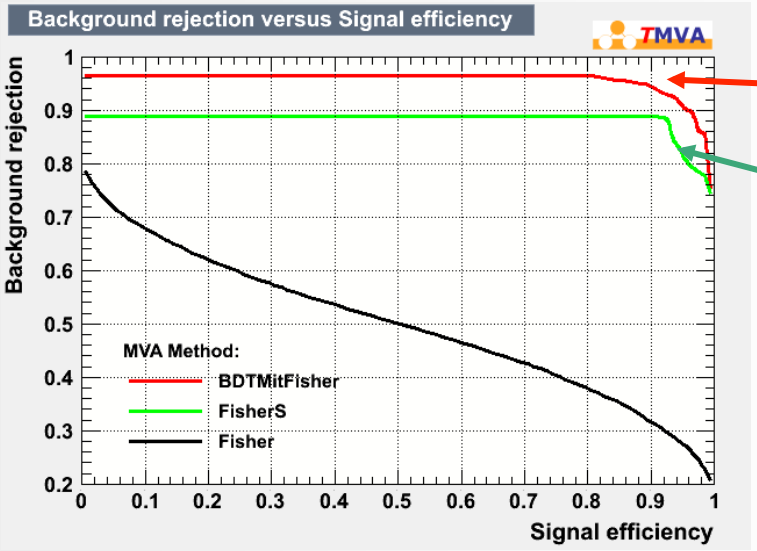


AdaBoost On a linear Classifier (e.g. Fisher)



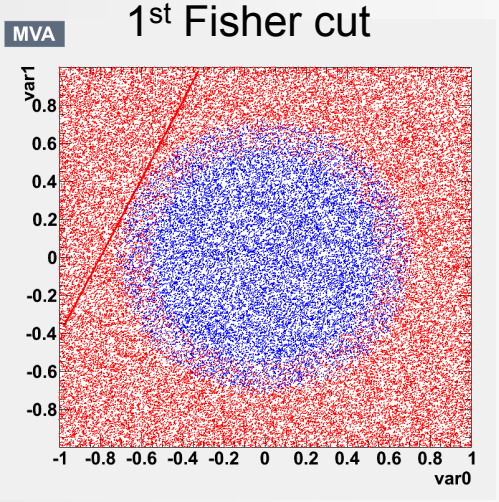
- Ups… there's still a problem in TMVA's generalized boosting. This example doesn't work yet !

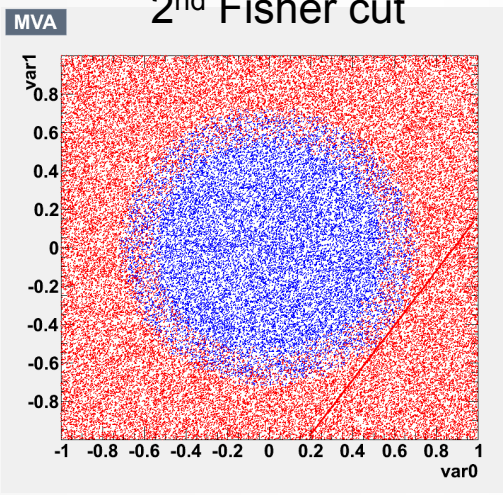
Boosting a Fisher Discriminant in TMVA…



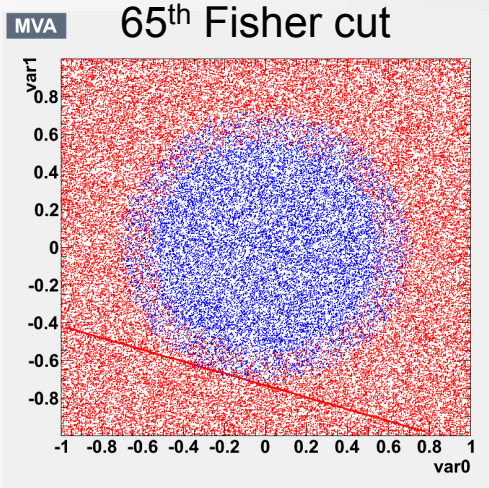
- 100 Boosts of a "Fisher Discriminant"
  - as Multivariate Tree split (yes.. it is in TMVA although I argued against it earlier. I hoped to cope better with linear correlations that way…)
  - generalised boosting of Fisher classifier

➔ Something isn't quite correct yet !





●●●●

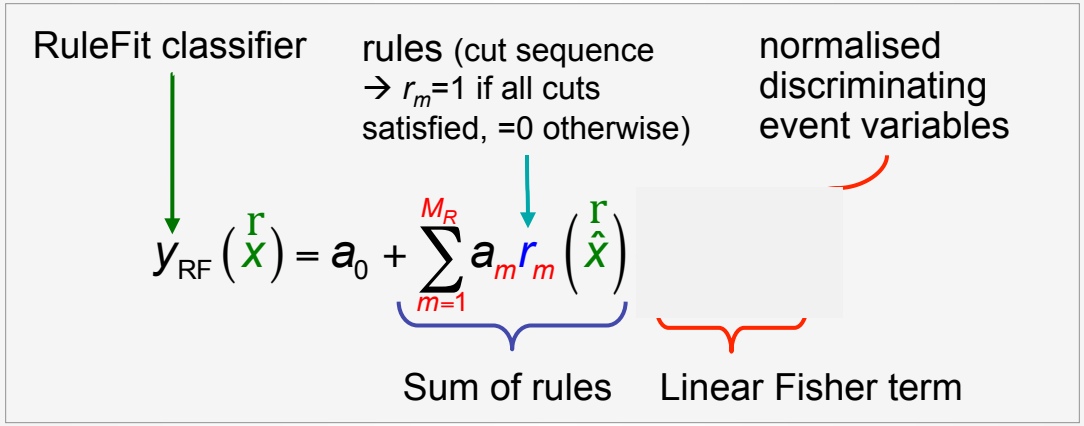


Learning with Rule Ensembles

- Following RuleFit approach by Friedman-Popescu

Friedman-Popescu, Tech Rep, Stat. Dpt, Stanford U., 2003

- Model is linear combination of *rules*, where a rule is a sequence of cuts (i.e. a branch of a decision tree)



- The problem to solve is
  - Create rule ensemble: use forest of decision trees
    - *pruning removes topologically equal rules (same variables in cut sequence)*
  - Add a "Fisher term" to capture linear correlations
  - Fit coefficients a_m, b_k: gradient direct regularization minimising *Risk* (Friedman et al.)

One of the elementary cellular automaton rules (Wolfram 1983, 2002). It specifies the next color in a cell, depending on its color and its immediate neighbors. Its rule outcomes are encoded in the binary representation $30=00011110_2$.

Regression Trees

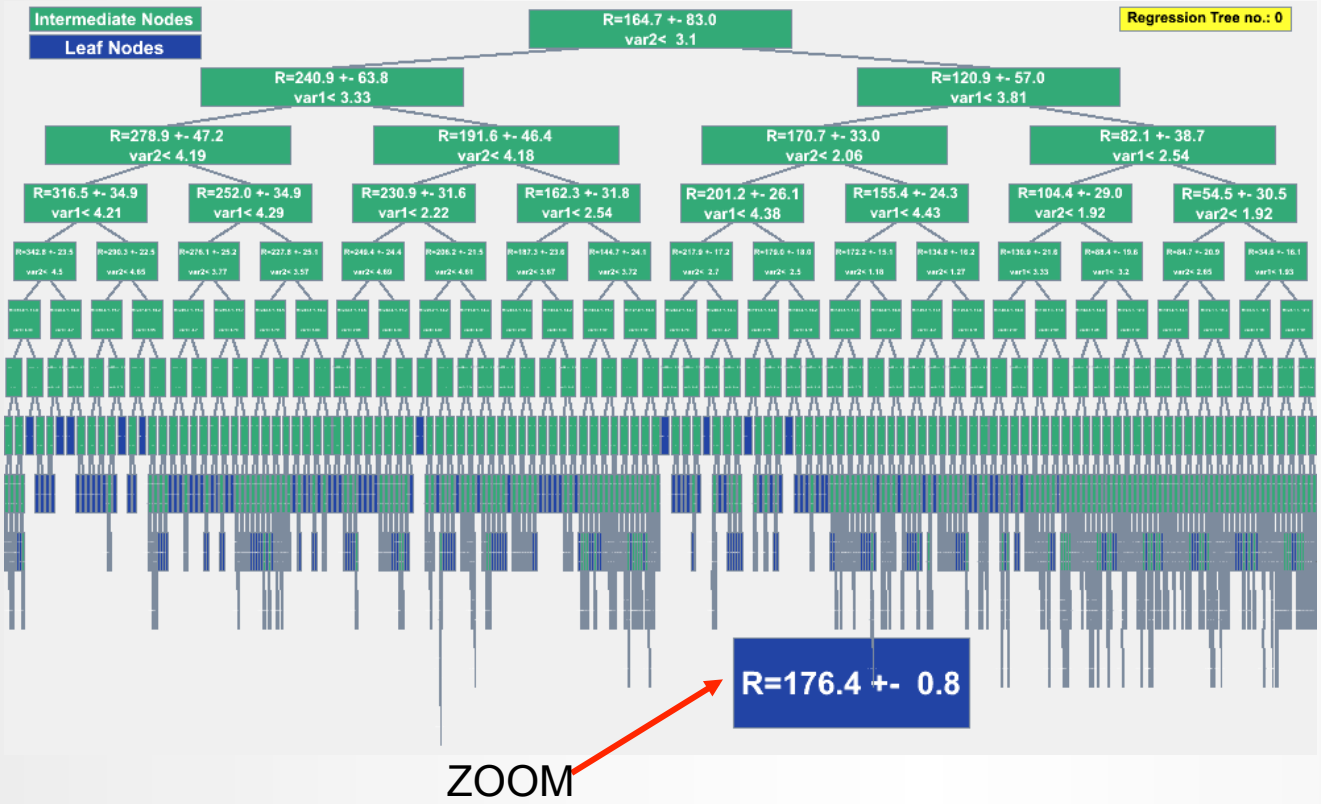
- Rather than calling leafs Signal or Background
  - → could also give them "values" (i.e. "mean value" of all values attributed to training events that end up in the node)
  - → Regression Tree

- Node Splitting: Separation Gain → Gain in Variance (RMS) of target function

- Boosting: error fraction → "distance" measure from the mean
  linear, square or exponential

- Use this to model ANY non analytic function of which you have "training data"
  i.e.
  - energy in your calorimeter as function of show parameters
  - training data from testbeam

Regression Trees



- Leaf Nodes:

One output value

Regression Trees seem to need DESPITE BOOSTING larger trees

Summary

- Boosted Decision Trees → a "brute force method" works "out of the box"
  - check tuning parameters anyway.
  - start with "small trees" (limit the maximum number of splits (tree depth)
  - automatic tuning parameter optimisation
    - first implementation is done, obviously needs LOTs of time!
  - be as careful as with "cuts" and check against data
- Boosting can (in principle) be applied to any (weak) classifier
- Boosted Regression Trees → at least as much "brute force"
  - little experience with yet.. but probably equally robust and powerful