

# Boosted decision trees

Yann Coadou

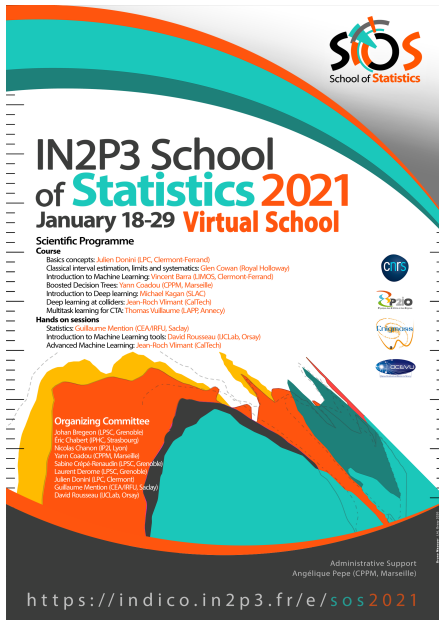
CPPM Marseille

SOS2021

Online edition, 22 January 2021



School of Statistics



**SOS**  
School of Statistics

## IN2P3 School of Statistics 2021

January 18-29 **Virtual School**

**Scientific Programme**

**Course**

- Basics concepts: Julien Donini (LPC, Clermont-Ferrand)
- Classical interval estimation, limits and systematics: Glen Cowan (Royal Holloway)
- Introduction to Machine Learning: Vincent Barra (IJMES, Clermont-Ferrand)
- Boosted Decision Trees: Yann Coadou (CPPM, Marseille)
- Introduction to Deep learning: Michael Kagan (SLAC)
- Deep learning at colliders: Jean-Roch Vlimant (CalTech)
- Multitask learning for CTA: Thomas Vautour (LAPP, Annecy)

**Hands on sessions**

- Statistics: Guillaume Mention (CEA/IRFU, Saclay)
- Introduction to Machine Learning tools: David Rousseau (JCLab, Orsay)
- Advanced Machine Learning: Jean-Roch Vlimant (CalTech)

**Organizing Committee**

- Johan Bregion (LPSG, Grenoble)
- Eric Chabert (IHC, Strasbourg)
- Nicolas Chauvin (IPSL, Lyon)
- Yann Coadou (CPPM, Marseille)
- Sabine Cappel-Fernaudin (LPSG, Grenoble)
- Laurent Doreme (LPSG, Grenoble)
- Julien Donini (LPC, Clermont)
- Guillaume Mention (CEA/IRFU, Saclay)
- David Rousseau (JCLab, Orsay)

**Administrative Support**

Angélique Pepe (CPPM, Marseille)

<https://indico.in2p3.fr/e/sos2021>

- 1 Introduction
- 2 Decision trees
- 3 Limitations
- 4 Boosted decision trees
- 5 Performance examples
- 6 BDTs in real physics cases
- 7 BDT systematics
- 8 Software
- 9 Conclusion
- 10 References

## Decision tree origin

- Machine-learning technique, widely used in social sciences. Originally data mining/pattern recognition, then medical diagnosis, insurance/loan screening, etc.



L. Breiman *et al.*, “Classification and Regression Trees” (1984)

## Basic principle

- Extend cut-based selection
  - many (most?) events do not have *all* characteristics of signal or background
  - try not to rule out events failing a particular criterion
- Keep events rejected by one criterion and see whether other criteria could help classify them properly

## Binary trees

- Trees can be built with branches splitting into many sub-branches
- In this lecture: mostly binary trees

## 1 Introduction

## 2 **Decision trees**

- Algorithm
- Tree hyperparameters
- Splitting a node
- Variable selection

## 3 Limitations

## 4 Boosted decision trees

## 5 Performance examples

## 6 BDTs in real physics cases

## Start with all events (signal and background) = first (root) node

- sort all events by each variable
- for each variable, find splitting value with best separation between two children
  - mostly signal in one child
  - mostly background in the other
- select variable and splitting value with best separation, produce two branches (nodes)
  - events failing criterion on one side
  - events passing it on the other

## Keep splitting

- Now have two new nodes. Repeat algorithm recursively on each node
- Can reuse the same variable
- Iterate until stopping criterion is reached
- Splitting stops: terminal node = leaf

- Consider signal ( $s_i$ ) and background ( $b_j$ ) events described by 3 variables:  $p_T$  of leading jet, top mass  $M_t$  and scalar sum of  $p_T$ 's of all objects in the event  $H_T$



- Consider signal ( $s_i$ ) and background ( $b_j$ ) events described by 3 variables:  $p_T$  of leading jet, top mass  $M_t$  and scalar sum of  $p_T$ 's of all objects in the event  $H_T$ 
  - sort all events by each variable:
    - $p_T^{s_1} \leq p_T^{b_{34}} \leq \dots \leq p_T^{b_2} \leq p_T^{s_{12}}$
    - $H_T^{b_5} \leq H_T^{b_3} \leq \dots \leq H_T^{s_{67}} \leq H_T^{s_{43}}$
    - $M_t^{b_6} \leq M_t^{s_8} \leq \dots \leq M_t^{s_{12}} \leq M_t^{b_9}$



- Consider signal ( $s_i$ ) and background ( $b_j$ ) events described by 3 variables:  $p_T$  of leading jet, top mass  $M_t$  and scalar sum of  $p_T$ 's of all objects in the event  $H_T$ 
  - sort all events by each variable:
    - $p_T^{s_1} \leq p_T^{b_{34}} \leq \dots \leq p_T^{b_2} \leq p_T^{s_{12}}$
    - $H_T^{b_5} \leq H_T^{b_3} \leq \dots \leq H_T^{s_{67}} \leq H_T^{s_{43}}$
    - $M_t^{b_6} \leq M_t^{s_8} \leq \dots \leq M_t^{s_{12}} \leq M_t^{b_9}$
  - best split (arbitrary unit):
    - $p_T < 56$  GeV, separation = 3
    - $H_T < 242$  GeV, separation = 5
    - $M_t < 105$  GeV, separation = 0.7

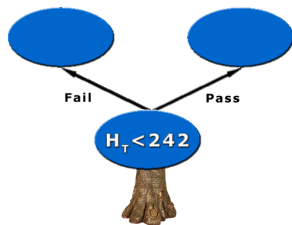




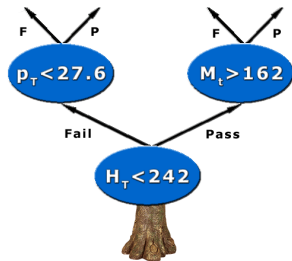
- Consider signal ( $s_i$ ) and background ( $b_j$ ) events described by 3 variables:  $p_T$  of leading jet, top mass  $M_t$  and scalar sum of  $p_T$ 's of all objects in the event  $H_T$ 
  - sort all events by each variable:
    - $p_T^{s_1} \leq p_T^{b_{34}} \leq \dots \leq p_T^{b_2} \leq p_T^{s_{12}}$
    - $H_T^{b_5} \leq H_T^{b_3} \leq \dots \leq H_T^{s_{67}} \leq H_T^{s_{43}}$
    - $M_t^{b_6} \leq M_t^{s_8} \leq \dots \leq M_t^{s_{12}} \leq M_t^{b_9}$
  - best split (arbitrary unit):
    - $p_T < 56 \text{ GeV}$ , separation = 3
    - $H_T < 242 \text{ GeV}$ , separation = 5
    - $M_t < 105 \text{ GeV}$ , separation = 0.7



- Consider signal ( $s_i$ ) and background ( $b_j$ ) events described by 3 variables:  $p_T$  of leading jet, top mass  $M_t$  and scalar sum of  $p_T$ 's of all objects in the event  $H_T$ 
  - sort all events by each variable:
    - $p_T^{s_1} \leq p_T^{b_{34}} \leq \dots \leq p_T^{b_2} \leq p_T^{s_{12}}$
    - $H_T^{b_5} \leq H_T^{b_3} \leq \dots \leq H_T^{s_{67}} \leq H_T^{s_{43}}$
    - $M_t^{b_6} \leq M_t^{s_8} \leq \dots \leq M_t^{s_{12}} \leq M_t^{b_9}$
  - best split (arbitrary unit):
    - $p_T < 56$  GeV, separation = 3
    - $H_T < 242$  GeV, separation = 5
    - $M_t < 105$  GeV, separation = 0.7
  - split events in two branches: pass or fail  $H_T < 242$  GeV



- Consider signal ( $s_i$ ) and background ( $b_j$ ) events described by 3 variables:  $p_T$  of leading jet, top mass  $M_t$  and scalar sum of  $p_T$ 's of all objects in the event  $H_T$ 
  - sort all events by each variable:
    - $p_T^{s_1} \leq p_T^{b_{34}} \leq \dots \leq p_T^{b_2} \leq p_T^{s_{12}}$
    - $H_T^{b_5} \leq H_T^{b_3} \leq \dots \leq H_T^{s_{67}} \leq H_T^{s_{43}}$
    - $M_t^{b_6} \leq M_t^{s_8} \leq \dots \leq M_t^{s_{12}} \leq M_t^{b_9}$
  - best split (arbitrary unit):
    - $p_T < 56$  GeV, separation = 3
    - $H_T < 242$  GeV, separation = 5
    - $M_t < 105$  GeV, separation = 0.7
  - split events in two branches: pass or fail  $H_T < 242$  GeV
  - Repeat recursively on each node



- Consider signal ( $s_i$ ) and background ( $b_j$ ) events described by 3 variables:  $p_T$  of leading jet, top mass  $M_t$  and scalar sum of  $p_T$ 's of all objects in the event  $H_T$

- sort all events by each variable:

- $p_T^{s_1} \leq p_T^{b_{34}} \leq \dots \leq p_T^{b_2} \leq p_T^{s_{12}}$
- $H_T^{b_5} \leq H_T^{b_3} \leq \dots \leq H_T^{s_{67}} \leq H_T^{s_{43}}$
- $M_t^{b_6} \leq M_t^{s_8} \leq \dots \leq M_t^{s_{12}} \leq M_t^{b_9}$

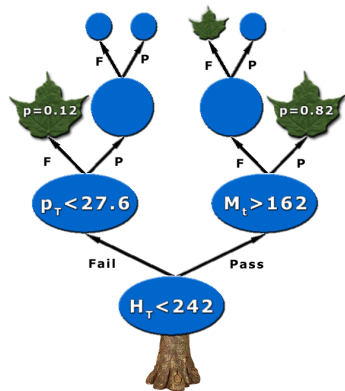
- best split (arbitrary unit):

- $p_T < 56$  GeV, separation = 3
- $H_T < 242$  GeV, separation = 5
- $M_t < 105$  GeV, separation = 0.7

- split events in two branches: pass or fail  $H_T < 242$  GeV

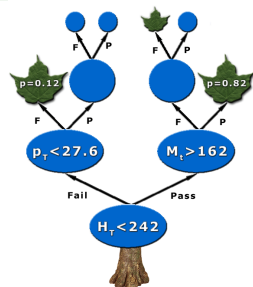
- Repeat recursively on each node

- Splitting stops: e.g. events with  $H_T < 242$  GeV and  $M_t > 162$  GeV are signal like ( $p = 0.82$ )



## Run event through tree

- Start from root node
- Apply first best cut
- Go to left or right child node
- Apply best cut for this node
- ...Keep going until...
- Event ends up in leaf



## DT Output

- Purity ( $\frac{s}{s+b}$ , with weighted events) of leaf, close to 1 for signal and 0 for background
- or binary answer (discriminant function +1 for signal, -1 or 0 for background) based on purity above/below specified value (e.g.  $\frac{1}{2}$ ) in leaf
- E.g. events with  $H_T < 242$  GeV and  $M_t > 162$  GeV have a DT output of 0.82 or +1

## Normalization of signal and background before training

- Balanced classes: same total weight for signal and background events ( $p = 0.5$ , maximal mixing)

## Selection of splits

- list of questions ( $variable_i < cut_i?$ , “Is the sky blue or overcast?”)
- goodness of split (separation measure)

## Decision to stop splitting (declare a node terminal)

- minimum leaf size (for statistical significance, e.g. 100 events)
- insufficient improvement from further splitting
- perfect classification (all events in leaf belong to same class)
- maximal tree depth (like-size trees choice or computing concerns)

## Assignment of terminal node to a class

- signal leaf if purity  $> 0.5$ , background otherwise

## Impurity measure $i(t)$

- maximal for equal mix of signal and background
- symmetric in  $p_{\text{signal}}$  and  $p_{\text{background}}$
- minimal for node with either signal only or background only
- strictly concave  $\Rightarrow$  reward purer nodes (favours end cuts with one smaller node and one larger node)

## Optimal split: figure of merit

- Decrease of impurity for split  $s$  of node  $t$  into children  $t_P$  and  $t_F$  (goodness of split):

$$\Delta i(s, t) = i(t) - p_P \cdot i(t_P) - p_F \cdot i(t_F)$$

- Aim: find split  $s^*$  such that:

$$\Delta i(s^*, t) = \max_{s \in \{\text{splits}\}} \Delta i(s, t)$$

- Maximising  $\Delta i(s, t) \equiv$  minimizing overall tree impurity

## Node purity

- Signal (background) event  $i$  with weight  $w_s^i$  ( $w_b^i$ )

$$p = \frac{\sum_{i \in \text{signal}} w_s^i}{\sum_{i \in \text{signal}} w_s^i + \sum_{j \in \text{bkg}} w_b^j}$$

- Signal purity (= purity)

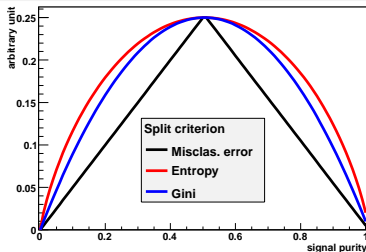
$$p_s = p = \frac{s}{s+b}$$

- Background purity

$$p_b = \frac{b}{s+b} = 1 - p_s = 1 - p$$

## Common impurity functions

- misclassification error  
 $= 1 - \max(p, 1 - p)$
- (cross) entropy  
 $= - \sum_{i=s,b} p_i \log p_i$
- Gini index (details in [▶ backup](#))



- Also cross section  $(-\frac{s^2}{s+b})$  and excess significance  $(-\frac{s^2}{b})$



## Reminder

- Need model giving good description of data

## Reminder

- Need model giving good description of data

## Playing with variables

- Number of variables:
  - not affected too much by “curse of dimensionality”
  - CPU consumption scales as  $nN \log N$  with  $n$  variables and  $N$  training events
- Insensitive to duplicate variables (give same ordering  $\Rightarrow$  same DT)
- Variable order does not matter: all variables treated equal
- Order of training events is irrelevant (batch training)
- Irrelevant variables:
  - no discriminative power  $\Rightarrow$  not used
  - only costs a little CPU time, no added noise
- Can use continuous and discrete variables, simultaneously

## Transforming input variables

- Completely insensitive to replacement of any subset of input variables by (possibly different) arbitrary strictly monotone functions of them:
  - let  $f : x_i \rightarrow f(x_i)$  be strictly monotone
  - if  $x > y$  then  $f(x) > f(y)$
  - ordering of events by  $x_i$  is the same as by  $f(x_i)$
  - $\Rightarrow$  produces the same DT
- Examples:
  - convert MeV  $\rightarrow$  GeV
  - no need to make all variables fit in the same range
  - no need to regularise variables (e.g. taking the log)
- $\Rightarrow$  Some immunity against outliers

## Transforming input variables

- Completely insensitive to replacement of any subset of input variables by (possibly different) arbitrary strictly monotone functions of them:
  - let  $f : x_i \rightarrow f(x_i)$  be strictly monotone
  - if  $x > y$  then  $f(x) > f(y)$
  - ordering of events by  $x_i$  is the same as by  $f(x_i)$
  - $\Rightarrow$  produces the same DT
- Examples:
  - convert MeV  $\rightarrow$  GeV
  - no need to make all variables fit in the same range
  - no need to regularise variables (e.g. taking the log)
- $\Rightarrow$  Some immunity against outliers

## Note about actual implementation

- The above is strictly true only if testing all possible cut values
- If there is some computational optimisation (e.g., check only 20 possible cuts on each variable), it may not work anymore

## Variable ranking (mean decrease impurity MDI)

- Ranking of  $x_i$ : add up decrease of impurity each time  $x_i$  is used
- Largest decrease of impurity = best variable

## Shortcoming: masking of variables

- $x_j$  may be just a little worse than  $x_i$  but will never be picked
  - $x_j$  is ranked as irrelevant
  - But remove  $x_i$  and  $x_j$  becomes very relevant
- ⇒ careful with interpreting ranking

## Solution: surrogate split

- Compare which events are sent left or right by optimal split and by any other split
- Give higher score to split that mimics better the optimal split
- Highest score = surrogate split
- Can be included in variable ranking
- Helps in case of missing data: replace optimal split by surrogate

## Permutation importance (mean decrease accuracy MDA)

- Applicable to any already trained classifier
- Randomly shuffle each variable in turn and measure decrease of performance
- Important variable  $\Rightarrow$  big loss of performance
- Can also be performed on validation sample
- Beware of correlations

## Choosing variables

- Usually try to have as few variables as possible
- But difficult: correlations, possibly large number to consider, large phase space with different properties in different regions
- *Brute force*: with  $n$  variables train all  $n$ ,  $n - 1$ , etc. combinations, pick best
- *Backward elimination*: train with  $n$  variables, then train all  $n - 1$  variables trees and pick best one; now train all  $n - 2$  variables trees starting from the  $n - 1$  variable list; etc. Pick optimal cost-complexity tree.
- *Forward greedy selection*: start with  $k = 1$  variable, then train all  $k + 1$  variables trees and pick the best; move to  $k + 2$  variables; etc.

## 1 Introduction

## 2 Decision trees

## 3 Limitations

- Training sample composition
- Pruning a tree
- Ensemble learning

## 4 Boosted decision trees

## 5 Performance examples

## 6 BDTs in real physics cases

- Small changes in sample can lead to very different tree structures (high variance)
- Performance on testing events may be as good, or not
- Not optimal to understand data from DT rules
- Does not give confidence in result:
  - DT output distribution discrete by nature
  - granularity related to tree complexity
  - tendency to have spikes at certain purity values (or just two delta functions at  $\pm 1$  if not using purity)



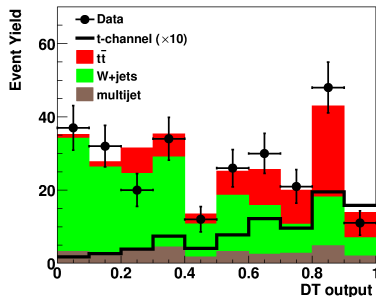
## Why prune a tree?

- Possible to get a perfect classifier on training events
- Mathematically misclassification error can be made as little as wanted
- E.g. tree with one class only per leaf (down to 1 event per leaf if necessary)
- Training error is zero
- But run new independent events through tree (testing or validation sample): misclassification is probably  $> 0$ , overtraining
- Pruning: eliminate subtrees (branches) that seem too specific to training sample:
  - a node and all its descendants turn into a leaf

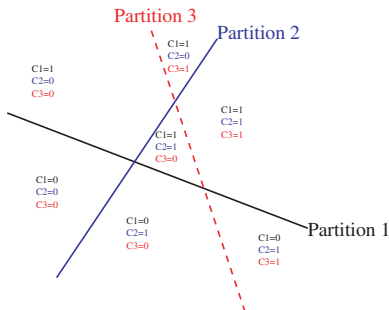
## Pruning algorithms (details in [▶ backup](#))

- Pre-pruning (early stopping condition like min leaf size, max depth)
- Expected error pruning (based on statistical error estimate)
- Cost-complexity pruning (penalise “complex” trees with many nodes/leaves)

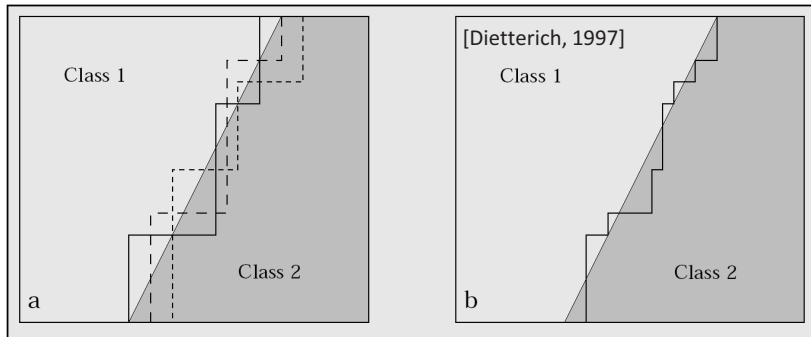
- ✓✓ Training is fast
- ✓✓ Human readable (not a black box, can interpret tree as selection rules or physics)
- ✓✓ Deals with continuous and discrete variables simultaneously
- ✓✓ No need to transform inputs
- ✓✓ Resistant to irrelevant variables
- ✓✓ Works well with many variables
- ✓✗ Good variables can be masked
- ✓ Very few parameters
- ✗ Unstable tree structure
- ✗ Piecewise nature of output
- ✗ Need at least as many training examples as variations in target function



- One tree:
  - one information about event (one leaf)
  - cannot really generalise to variations not covered in training set (at most as many leaves as input size)
- Many trees:
  - distributed representation**: number of intersections of leaves exponential in number of trees
  - many leaves contain the event  $\Rightarrow$  richer description of input pattern



- Build several trees and average the output



- K-fold cross-validation (good for small samples)
  - divide training sample  $\mathcal{L}$  in  $K$  subsets of equal size:  $\mathcal{L} = \bigcup_{k=1..K} \mathcal{L}_k$
  - Train tree  $T_k$  on  $\mathcal{L} - \mathcal{L}_k$ , test on  $\mathcal{L}_k$
  - DT output =  $\frac{1}{K} \sum_{k=1..K} T_k$
- Bagging, boosting, random forests, etc.

## 1 Introduction

## 2 Decision trees

## 3 Limitations

## 4 Boosted decision trees

- Introduction
- AdaBoost
- Overtraining?
- Clues to boosting performance
- Gradient boosting

## 5 Performance examples

## First provable algorithm by Schapire (1990)

- Train classifier  $T_1$  on  $N$  events
- Train  $T_2$  on new  $N$ -sample, half of which misclassified by  $T_1$
- Build  $T_3$  on events where  $T_1$  and  $T_2$  disagree
- Boosted classifier: MajorityVote( $T_1, T_2, T_3$ )

## First provable algorithm by Schapire (1990)

- Train classifier  $T_1$  on  $N$  events
- Train  $T_2$  on new  $N$ -sample, half of which misclassified by  $T_1$
- Build  $T_3$  on events where  $T_1$  and  $T_2$  disagree
- Boosted classifier: MajorityVote( $T_1, T_2, T_3$ )

## Then

- Variation by Freund (1995): boost by majority (combining many learners with fixed error rate)
- Freund&Schapire joined forces: 1<sup>st</sup> functional model **AdaBoost** (1996)

## First provable algorithm by Schapire (1990)

- Train classifier  $T_1$  on  $N$  events
- Train  $T_2$  on new  $N$ -sample, half of which misclassified by  $T_1$
- Build  $T_3$  on events where  $T_1$  and  $T_2$  disagree
- Boosted classifier: MajorityVote( $T_1, T_2, T_3$ )

## Then

- Variation by Freund (1995): boost by majority (combining many learners with fixed error rate)
- Freund&Schapire joined forces: 1<sup>st</sup> functional model **AdaBoost** (1996)

## When it really picked up in HEP

- MiniBooNe compared performance of different boosting algorithms and neural networks for particle ID (2005)
- D0 claimed first evidence for single top quark production (2006)
- CDF copied 😊 (2008). Both used BDT for single top observation



## What is boosting?

- General method, not limited to decision trees
- Hard to make a very good learner, but easy to make simple, error-prone ones (but still better than random guessing)
- Goal: combine such weak classifiers into a new more stable one, with smaller error

## Algorithm

- Training sample  $\mathbb{T}_k$  of  $N$  events. For  $i^{th}$  event:
  - weight  $w_i^k$
  - vector of discriminative variables  $x_i$
  - class label  $y_i = +1$  for signal,  $-1$  for background
- Pseudocode:
  - Initialise  $\mathbb{T}_1$
  - for  $k$  in  $1..N_{tree}$ 
    - train classifier  $T_k$  on  $\mathbb{T}_k$
    - assign weight  $\alpha_k$  to  $T_k$
    - modify  $\mathbb{T}_k$  into  $\mathbb{T}_{k+1}$
- Boosted output:  $F(T_1, \dots, T_{N_{tree}})$

- Introduced by Freund&Schapire in 1996
- Stands for *adaptive boosting*
- Learning procedure adjusts to training data to classify it better
- Many variations on the same theme for actual implementation
- Most common boosting algorithm around
- Usually leads to better results than without boosting

- Check which events of training sample  $\mathbb{T}_k$  are misclassified by  $T_k$ :
  - $\mathbb{I}(X) = 1$  if  $X$  is true, 0 otherwise
  - for DT output in  $\{\pm 1\}$ :  $\text{isMisclassified}_k(i) = \mathbb{I}(y_i \times T_k(x_i) \leq 0)$
  - or  $\text{isMisclassified}_k(i) = \mathbb{I}(y_i \times (T_k(x_i) - 0.5) \leq 0)$  in purity convention
  - misclassification rate:

$$R(T_k) = \varepsilon_k = \frac{\sum_{i=1}^N w_i^k \times \text{isMisclassified}_k(i)}{\sum_{i=1}^N w_i^k}$$

- Derive tree weight  $\alpha_k = \beta \times \ln((1 - \varepsilon_k)/\varepsilon_k)$
- Increase weight of misclassified events in  $\mathbb{T}_k$  to create  $\mathbb{T}_{k+1}$ :

$$w_i^k \rightarrow w_i^{k+1} = w_i^k \times e^{\alpha_k}$$

- Train  $T_{k+1}$  on  $\mathbb{T}_{k+1}$
- Boosted result of event  $i$ :

$$T(i) = \frac{1}{\sum_{k=1}^{N_{\text{tree}}} \alpha_k} \sum_{k=1}^{N_{\text{tree}}} \alpha_k T_k(i)$$

- Assume  $\beta = 1$

## Not-so-good classifier

- Assume error rate  $\varepsilon = 40\%$
- Then  $\alpha = \ln \frac{1-0.4}{0.4} = 0.4$
- Misclassified events get their weight multiplied by  $e^{0.4}=1.5$
- $\Rightarrow$  next tree will have to work a bit harder on these events

## Good classifier

- Error rate  $\varepsilon = 5\%$
- Then  $\alpha = \ln \frac{1-0.05}{0.05} = 2.9$
- Misclassified events get their weight multiplied by  $e^{2.9}=19 (!!)$
- $\Rightarrow$  being failed by a good classifier means a big penalty:
  - must be a difficult case
  - next tree will have to pay much more attention to this event and try to get it right

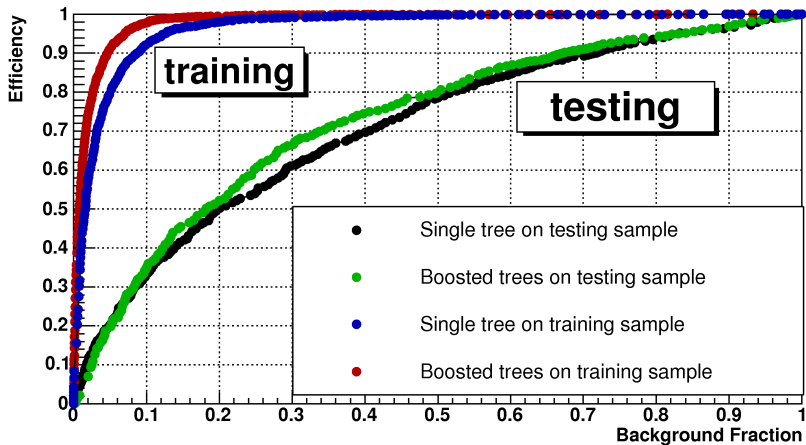
## Misclassification rate $\varepsilon$ on training sample

- Can be shown to be bound:
$$\varepsilon \leq \prod_{k=1}^{N_{tree}} 2\sqrt{\varepsilon_k(1 - \varepsilon_k)}$$
- If each tree has  $\varepsilon_k \neq 0.5$  (i.e. better than random guessing):  
*the error rate falls to zero for sufficiently large  $N_{tree}$*
- Corollary: training data is overfitted

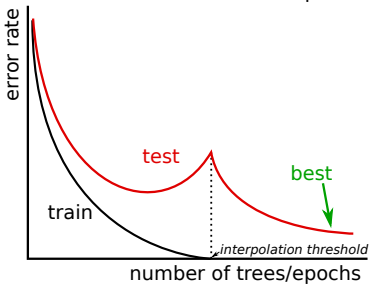
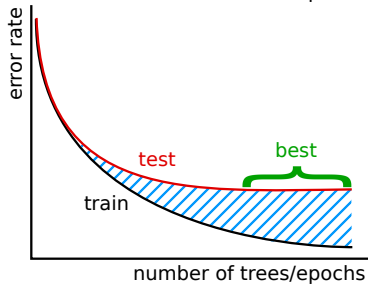
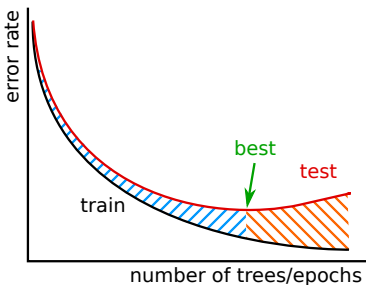
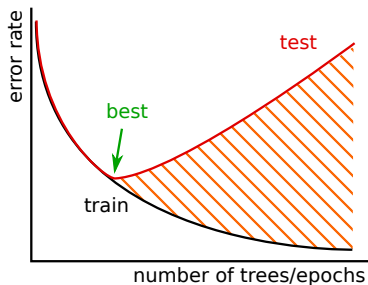
## Overtraining?

- Error rate on test sample may reach a minimum and then potentially rise. Stop boosting at the minimum.
- In principle AdaBoost *must* overfit training sample
- In many cases in literature, no loss of performance due to overtraining
  - may have to do with fact that successive trees get in general smaller and smaller weights
  - trees that lead to overtraining contribute very little to final DT output on validation sample

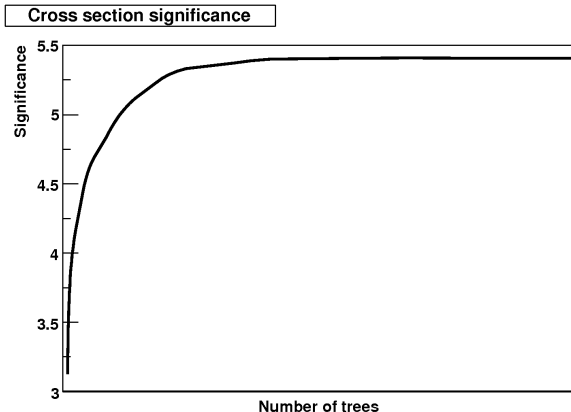
Efficiency vs. background fraction



- Clear overtraining, but still better performance after boosting



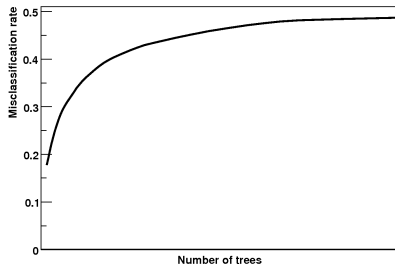
“bad” overtraining (overfitting) / “good” overtraining (still underfitting)



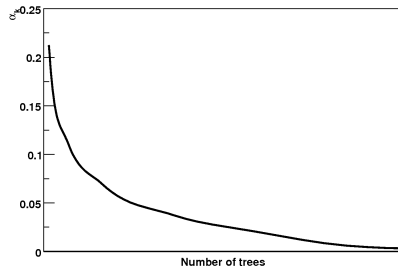
- More relevant than testing error
- Reaches plateau
- Afterwards, boosting does not hurt (just wasted CPU)
- Applicable to any other figure of merit of interest for your use case



Misclassification rate for each tree



Tree weight  $\alpha_k$



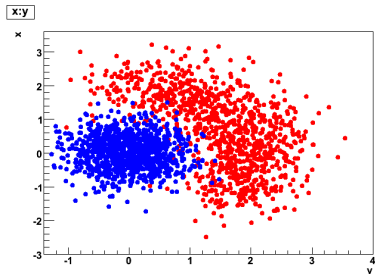
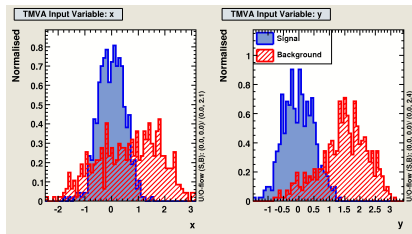
- First tree is best, others are minor corrections
- Specialised trees do not perform well on most events  $\Rightarrow$  decreasing tree weight and increasing misclassification rate
- Last tree is not better evolution of first tree, but rather a pretty bad DT that only does a good job on few cases that the other trees could not get right
- But adding trees may increase reliability of prediction: margins explanation
- See also double descent risk curve and interpolation regime

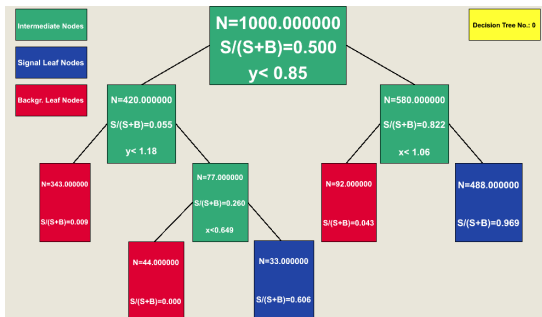
- AdaBoost recast in a statistical framework: corresponds to minimising an exponential loss
- Generalisation: formulate boosting as numerical optimisation problem, minimise loss function by adding trees using gradient descent procedure
- Build imperfect model  $F_k$  at step  $k$  (sometimes  $F_k(x) \neq y$ )
- Improve model:  $F_{k+1}(x) = F_k(x) + h_k(x) = y$ , or residual  $h_k(x) = y - F_k(x)$
- Train new classifier on residual
- Example: mean squared error loss function  

$$L_{\text{MSE}}(x, y) = \frac{1}{2} (y - F_k(x))^2$$
  - minimising loss  $J = \sum_i L_{\text{MSE}}(x_i, y_i)$  leads to  $\frac{\partial J}{\partial F_k(x_i)} = F_k(x_i) - y_i$   
 $\Rightarrow$  residual as negative gradient:  $h_k(x_i) = y_i - F_k(x_i) = -\frac{\partial J}{\partial F_k(x_i)}$
- Generalised to any differentiable loss function

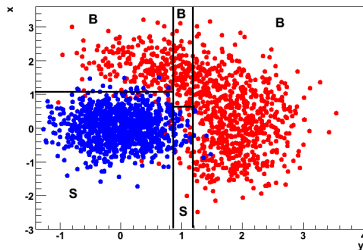
- 1 Introduction
- 2 Decision trees
- 3 Limitations
- 4 Boosted decision trees
- 5 Performance examples**
  - First is best
  - XOR problem
  - Circular correlation
  - Many small trees or fewer large trees?
- 6 BDTs in real physics cases

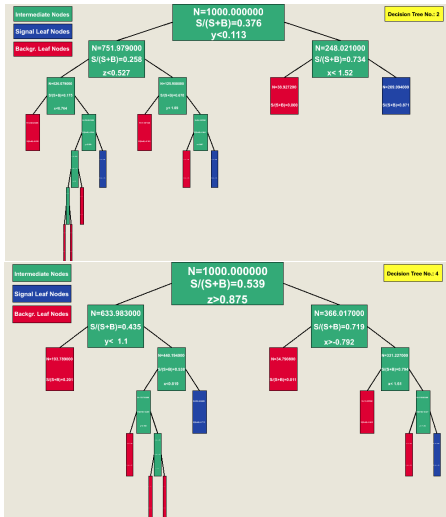
- Using ROOT and TMVA with basic code to make examples (more later)



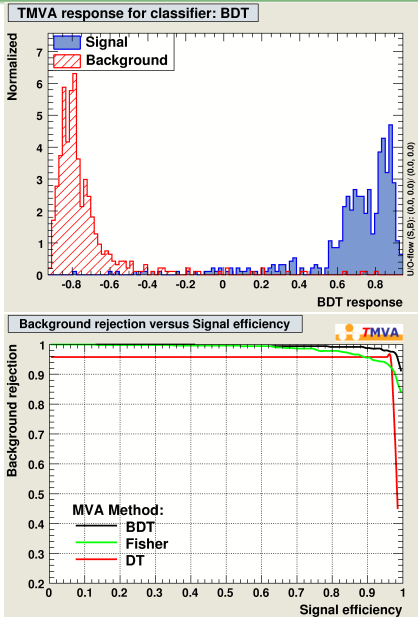


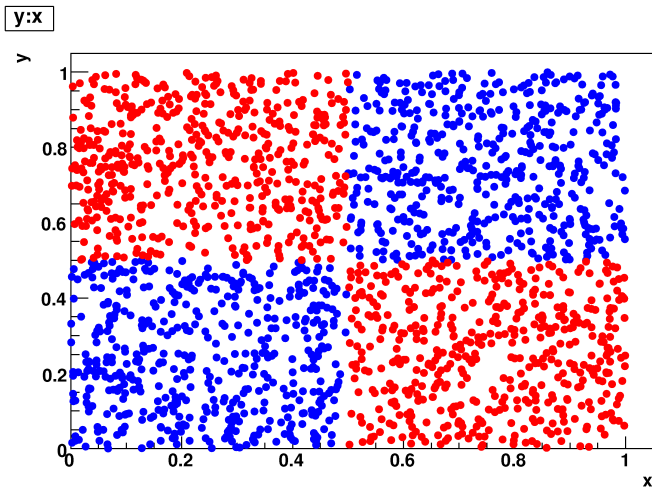
x:y



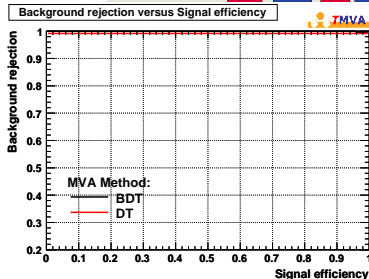
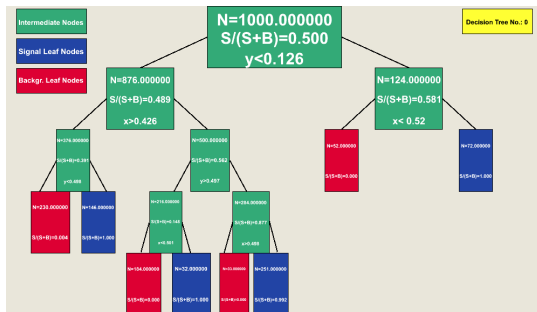
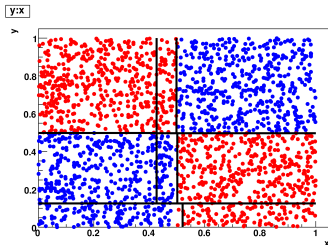


- ## Coadou (CPPM) — Boosted decision trees

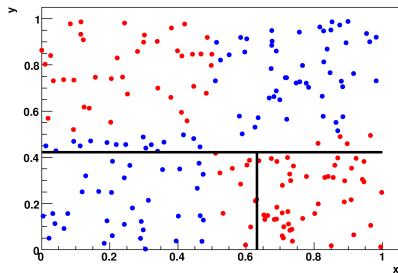








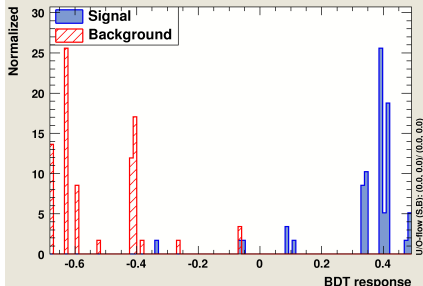
y:x



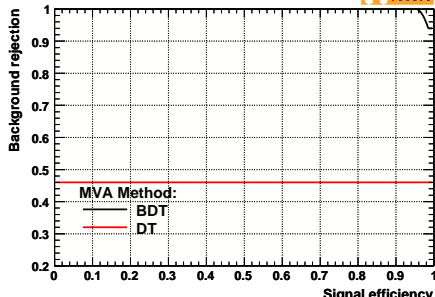
## Small statistics

- Single tree not so good
- BDT very good: high performance discriminant from combination of weak classifiers

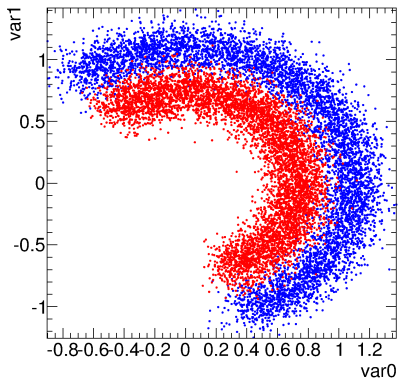
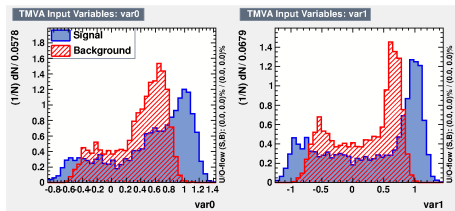
TMVA response for classifier: BDT



Background rejection versus Signal efficiency



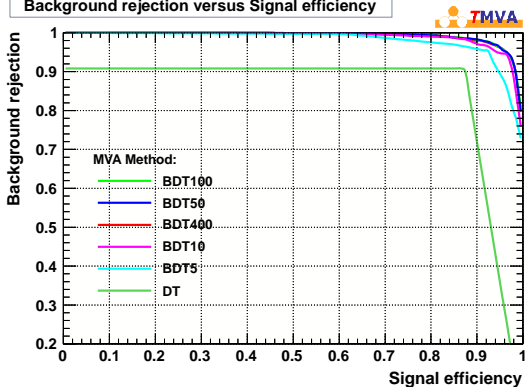
- Using TMVA and create\_circ macro from `$ROOTSYS/tutorials/tmva/createData.C` to generate dataset
- Plots: `TMVA::TMVAGui("filename");`



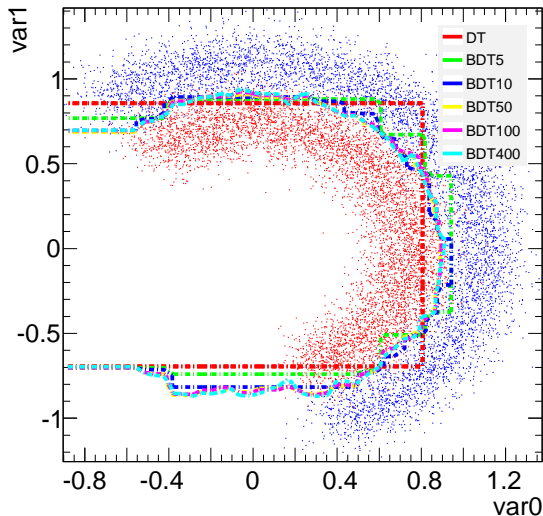
## Boosting longer (TMVA: NTrees)

- Compare performance of single DT and BDT with more and more trees (5 to 400)
- All other parameters at TMVA default (would be 400 trees)

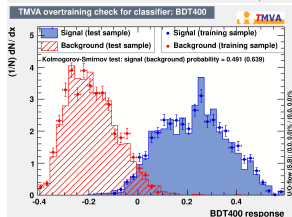
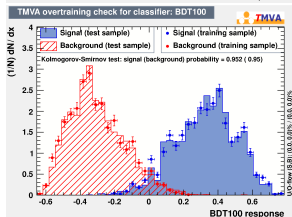
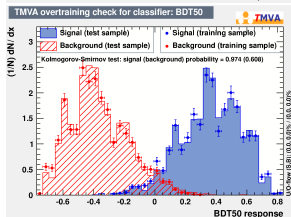
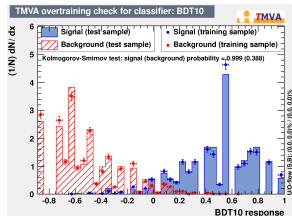
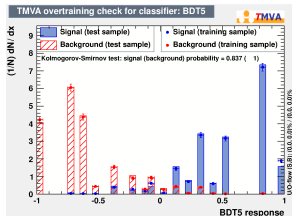
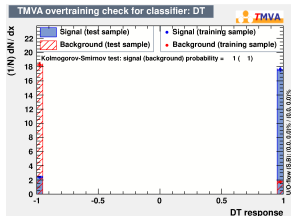
Background rejection versus Signal efficiency



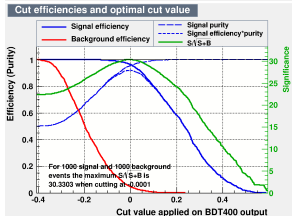
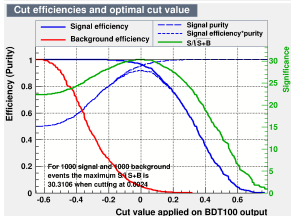
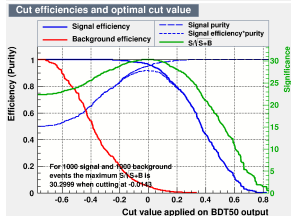
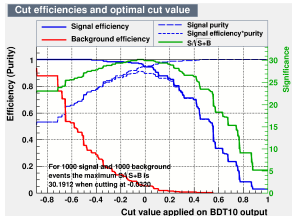
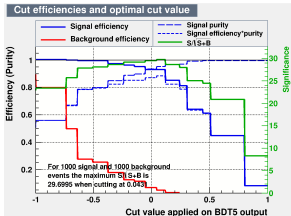
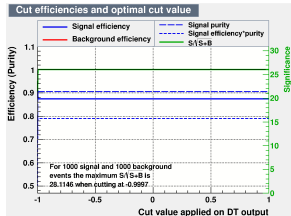
- Single (small) DT: not so good
- More trees  $\Rightarrow$  improve performance until saturation



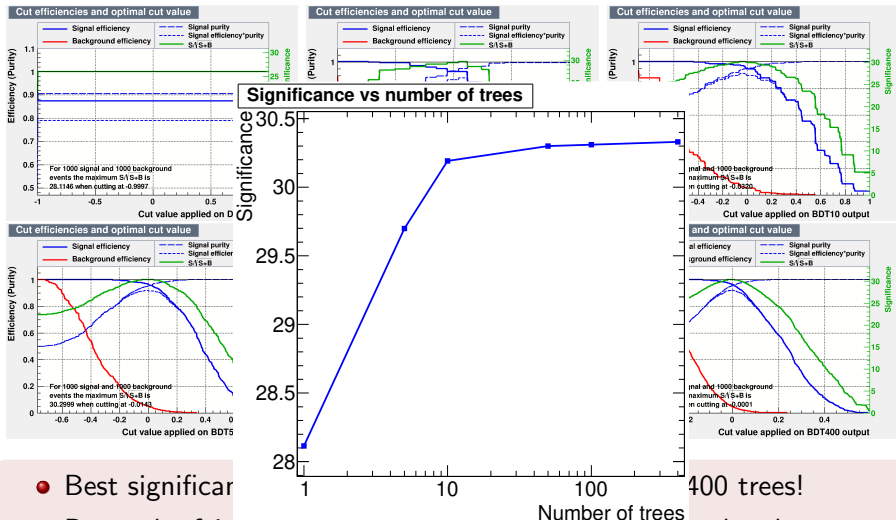
- Note: max tree depth = 3
- Single (small) DT: not so good. Note: a larger tree would solve this problem
- More trees  $\Rightarrow$  improve performance (less step-like, closer to optimal separation) until saturation
- Largest BDTs: wiggle a little around the contour  $\Rightarrow$  picked up features of training sample, that is, overtraining



- Better shape with more trees: quasi-continuous
  - Overtraining because of disagreement between training and testing?
- Let's see



- Best significance actually obtained with last BDT, 400 trees!
- But to be fair, equivalent performance with 10 trees already
- Less “stepped” output desirable?  $\Rightarrow$  maybe 50 is reasonable

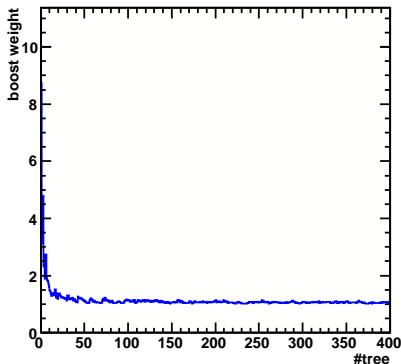


- Best significance
- But to be fair, equivalent performance with 10 trees already
- Less “stepped” output desirable?  $\Rightarrow$  maybe 50 is reasonable

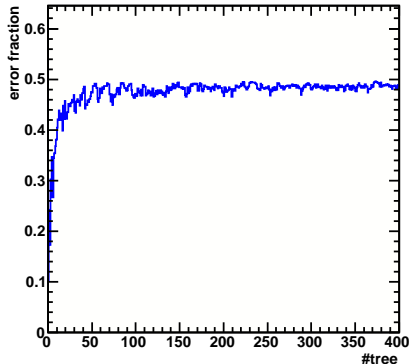


- Boosting weight decreases fast and stabilises
- First trees have small error fractions, then increases towards 0.5 (random guess)
- $\Rightarrow$  confirms that best trees are first ones, others are small corrections

Boost weights vs tree



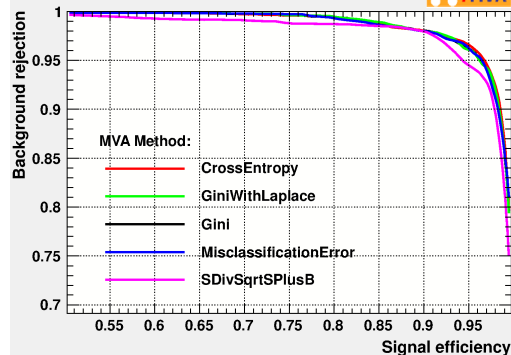
error fraction vs tree number



## Separation criterion for node splitting (TMVA: SeparationType)

- Compare performance of Gini, entropy, misclassification error,  $\frac{s}{\sqrt{s+b}}$
- All other parameters at TMVA default

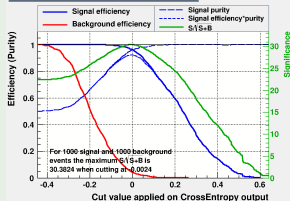
Background rejection versus Signal efficiency



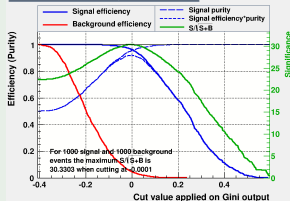
- Very similar performance (even zooming on corner)
- Small degradation (in this particular case) for  $\frac{s}{\sqrt{s+b}}$ : only criterion that does not respect good properties of impurity measure (see earlier: maximal for equal mix of signal and bkg, symmetric in  $p_{sig}$  and  $p_{bkg}$ , minimal for node with either signal only or bkg only, strictly concave)

## Performance in optimal significance

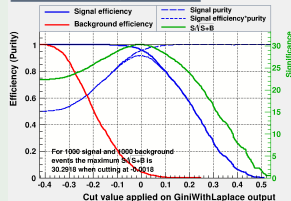
Cut efficiencies and optimal cut value



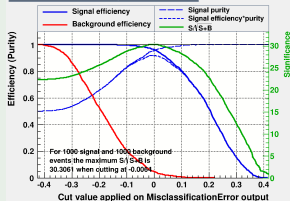
Cut efficiencies and optimal cut value



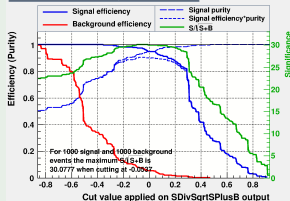
Cut efficiencies and optimal cut value



Cut efficiencies and optimal cut value

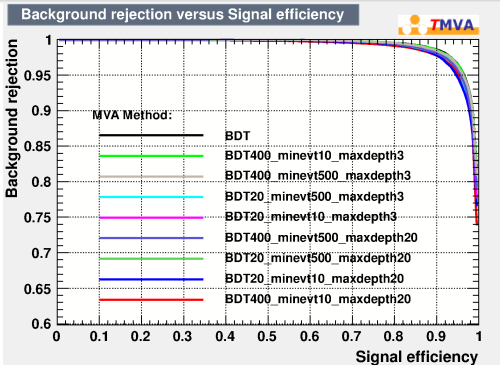
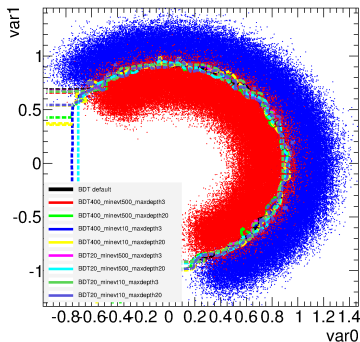


Cut efficiencies and optimal cut value



- Confirms previous page: very similar performance, worse for BDT optimised with significance!

- Using same `create_circ` macro but generating larger dataset to avoid stats limitations
- 20 or 400 trees; minimum leaf size: 10 or 500 events (`MinNodeSize`)
- Maximum depth (max # of cuts to reach leaf): 3 or 20 (`MaxDepth`)



- Overall: very comparable performance. Depends on use case.

## $\varepsilon$ -Boost (shrinkage)

- reweight misclassified events by a fixed  $e^{2\varepsilon}$  factor
- $T(i) = \sum_{k=1}^{N_{\text{tree}}} \varepsilon T_k(i)$

## $\varepsilon$ -LogitBoost

- reweight misclassified events by logistic function  $\frac{e^{-y_i T_k(x_i)}}{1 + e^{-y_i T_k(x_i)}}$
- $T(i) = \sum_{k=1}^{N_{\text{tree}}} \varepsilon T_k(i)$

## Real AdaBoost

- DT output is  $T_k(i) = 0.5 \times \ln \frac{p_k(i)}{1-p_k(i)}$  where  $p_k(i)$  is purity of leaf on which event  $i$  falls
- reweight events by  $e^{-y_i T_k(i)}$
- $T(i) = \sum_{k=1}^{N_{\text{tree}}} T_k(i)$
- $\varepsilon$ -HingeBoost, LogitBoost, Gentle AdaBoost, etc.

## Bagging (Bootstrap aggregating)

- Before building tree  $T_k$  take random sample of  $N$  events from training sample with replacement
- Train  $T_k$  on it
- Events not picked form “out of bag” validation sample
- Applicable to other techniques than DT
  - tends to produce more stable and better classifier

## Bagging (Bootstrap aggregating)

- Before building tree  $T_k$  take random sample of  $N$  events from training sample with replacement
- Train  $T_k$  on it
- Events not picked form “out of bag” validation sample
- Applicable to other techniques than DT
  - tends to produce more stable and better classifier

## Random forests

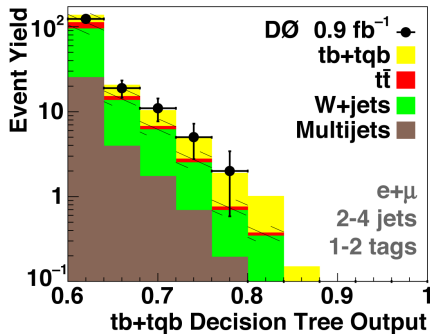
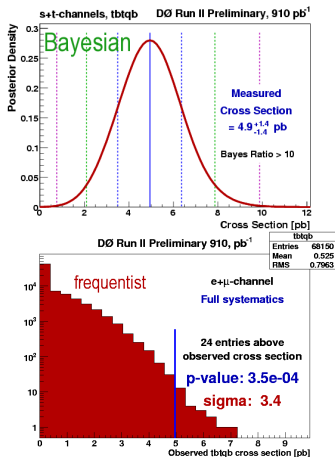
- Same as bagging
- In addition, pick random subset of variables to consider for each node split
- Two levels of randomisation, much more stable output
- Often as good as boosting

- 1 Introduction
- 2 Decision trees
- 3 Limitations
- 4 Boosted decision trees
- 5 Performance examples
- 6 BDTs in real physics cases**
  - Single top search at D0
- 7 BDT systematics



- Three multivariate techniques:  
BDT, Matrix Elements, BNN
- Most sensitive: BDT

$\sigma_{s+t} = 4.9 \pm 1.4 \text{ pb}$   
 $p\text{-value} = 0.035\% (3.4\sigma)$   
 SM compatibility: 11% ( $1.3\sigma$ )



$$\sigma_s = 1.0 \pm 0.9 \text{ pb}$$

$$\sigma_t = 4.2^{+1.8}_{-1.4} \text{ pb}$$

► Phys. Rev. D78, 012005 (2008)

## Object Kinematics

$p_T(\text{jet1})$   
 $p_T(\text{jet2})$   
 $p_T(\text{jet3})$   
 $p_T(\text{jet4})$   
 $p_T(\text{best1})$   
 $p_T(\text{notbest1})$   
 $p_T(\text{notbest2})$   
 $p_T(\text{tag1})$   
 $p_T(\text{untag1})$   
 $p_T(\text{untag2})$

## Angular Correlations

$\Delta R(\text{jet1}, \text{jet2})$   
 $\cos(\text{best1}, \text{lepton})_{\text{besttop}}$   
 $\cos(\text{best1}, \text{notbest1})_{\text{besttop}}$   
 $\cos(\text{tag1}, \text{alljets})_{\text{alljets}}$   
 $\cos(\text{tag1}, \text{lepton})_{\text{btaggedtop}}$   
 $\cos(\text{jet1}, \text{alljets})_{\text{alljets}}$   
 $\cos(\text{jet1}, \text{lepton})_{\text{btaggedtop}}$   
 $\cos(\text{jet2}, \text{alljets})_{\text{alljets}}$   
 $\cos(\text{jet2}, \text{lepton})_{\text{btaggedtop}}$   
 $\cos(\text{lepton}, Q(\text{lepton}) \times z)_{\text{besttop}}$   
 $\cos(\text{lepton}_{\text{besttop}}, \text{besttop}_{\text{CMframe}})$   
 $\cos(\text{lepton}_{\text{btaggedtop}}, \text{btaggedtop}_{\text{CMframe}})$   
 $\cos(\text{notbest}, \text{alljets})_{\text{alljets}}$   
 $\cos(\text{notbest}, \text{lepton})_{\text{besttop}}$   
 $\cos(\text{untag1}, \text{alljets})_{\text{alljets}}$   
 $\cos(\text{untag1}, \text{lepton})_{\text{btaggedtop}}$

## Event Kinematics

$A_{\text{planarity}}(\text{alljets}, W)$   
 $M(W, \text{best1})$  ("best" top mass)  
 $M(W, \text{tag1})$  ("b-tagged" top mass)  
 $H_T(\text{alljets})$   
 $H_T(\text{alljets} - \text{best1})$   
 $H_T(\text{alljets} - \text{tag1})$   
 $H_T(\text{alljets}, W)$   
 $H_T(\text{jet1}, \text{jet2})$   
 $H_T(\text{jet1}, \text{jet2}, W)$   
 $M(\text{alljets})$   
 $M(\text{alljets} - \text{best1})$   
 $M(\text{alljets} - \text{tag1})$   
 $M(\text{jet1}, \text{jet2})$   
 $M(\text{jet1}, \text{jet2}, W)$   
 $M_T(\text{jet1}, \text{jet2})$   
 $M_T(W)$   
 Missing  $E_T$   
 $p_T(\text{alljets} - \text{best1})$   
 $p_T(\text{alljets} - \text{tag1})$   
 $p_T(\text{jet1}, \text{jet2})$   
 $Q(\text{lepton}) \times \eta(\text{untag1})$   
 $\sqrt{s}$   
 $\text{Sphericity}(\text{alljets}, W)$

- Adding variables did not degrade performance
- Tested shorter lists, lost some sensitivity
- Same list used for all channels

## Object Kinematics

$p_T(\text{jet1})$   
 $p_T(\text{jet2})$   
 $p_T(\text{jet3})$   
 $p_T(\text{jet4})$   
 $p_T(\text{best1})$   
 $p_T(\text{notbest1})$   
 $p_T(\text{notbest2})$   
 $p_T(\text{tag1})$   
 $p_T(\text{untag1})$   
 $p_T(\text{untag2})$

## Angular Correlations

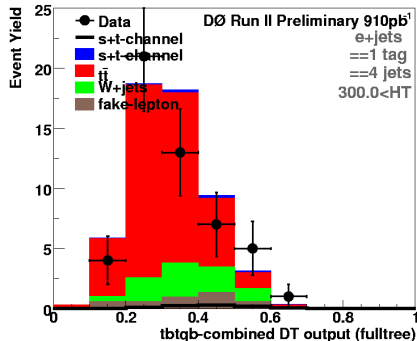
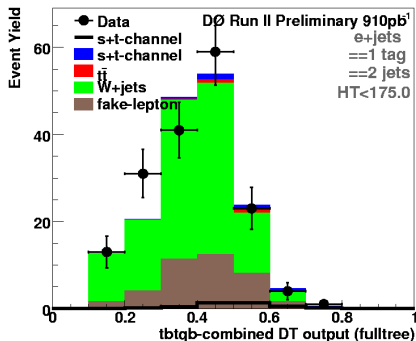
$\Delta R(\text{jet1}, \text{jet2})$   
 $\cos(\text{best1}, \text{lepton})_{\text{besttop}}$   
 $\cos(\text{best1}, \text{notbest1})_{\text{besttop}}$   
 $\cos(\text{tag1}, \text{alljets})_{\text{alljets}}$   
 $\cos(\text{tag1}, \text{lepton})_{\text{btaggedtop}}$   
 $\cos(\text{jet1}, \text{alljets})_{\text{alljets}}$   
 $\cos(\text{jet1}, \text{lepton})_{\text{btaggedtop}}$   
 $\cos(\text{jet2}, \text{alljets})_{\text{alljets}}$   
 $\cos(\text{jet2}, \text{lepton})_{\text{btaggedtop}}$   
 $\cos(\text{lepton}, Q(\text{lepton}) \times z)_{\text{besttop}}$   
 $\cos(\text{lepton}_{\text{besttop}}, \text{besttop}_{\text{CMframe}})$   
 $\cos(\text{lepton}_{\text{btaggedtop}}, \text{btaggedtop}_{\text{CMframe}})$   
 $\cos(\text{notbest}, \text{alljets})_{\text{alljets}}$   
 $\cos(\text{notbest}, \text{lepton})_{\text{besttop}}$   
 $\cos(\text{untag1}, \text{alljets})_{\text{alljets}}$   
 $\cos(\text{untag1}, \text{lepton})_{\text{btaggedtop}}$

## Event Kinematics

$A_{\text{planarity}}(\text{alljets}, W)$   
 $M(W, \text{best1})$  ("best" top mass)  
 $M(W, \text{tag1})$  ("b-tagged" top mass)  
 $H_T(\text{alljets})$   
 $H_T(\text{alljets} - \text{best1})$   
 $H_T(\text{alljets} - \text{tag1})$   
 $H_T(\text{alljets}, W)$   
 $H_T(\text{jet1}, \text{jet2})$   
 $H_T(\text{jet1}, \text{jet2}, W)$   
 $M(\text{alljets})$   
 $M(\text{alljets} - \text{best1})$   
 $M(\text{alljets} - \text{tag1})$   
 $M(\text{jet1}, \text{jet2})$   
 $M(\text{jet1}, \text{jet2}, W)$   
 $M_T(\text{jet1}, \text{jet2})$   
 $M_T(W)$   
 $\text{Missing } E_T$   
 $p_T(\text{alljets} - \text{best1})$   
 $p_T(\text{alljets} - \text{tag1})$   
 $p_T(\text{jet1}, \text{jet2})$   
 $Q(\text{lepton}) \times \eta(\text{untag1})$   
 $\sqrt{s}$   
 $\text{Sphericity}(\text{alljets}, W)$

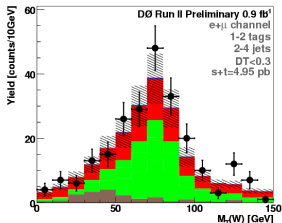
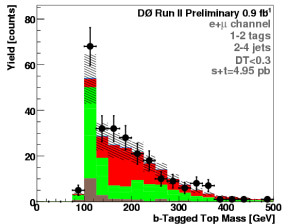
- Adding variables did not degrade performance
- Tested shorter lists, lost some sensitivity
- Same list used for all channels
- Best theoretical variable:  
 $H_T(\text{alljets}, W)$ .  
 But detector not perfect  $\Rightarrow$  capture the essence from several variations usually helps "dumb" MVA

- Validate method on data in no-signal region
- **“W+jets”**: = 2 jets,  
 $H_T(\text{lepton}, E_T^{\text{miss}}, \text{alljets}) < 175 \text{ GeV}$
- **“ttbar”**: = 4 jets,  
 $H_T(\text{lepton}, E_T^{\text{miss}}, \text{alljets}) > 300 \text{ GeV}$

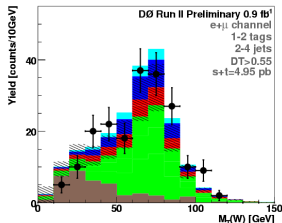
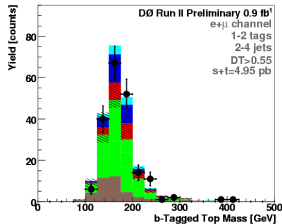


- Good agreement

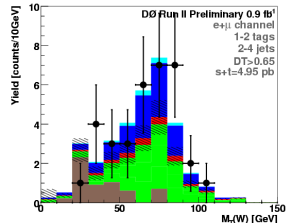
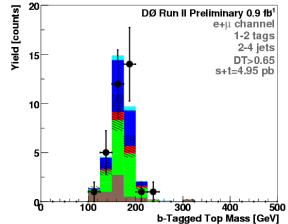
$DT < 0.3$



$DT > 0.55$

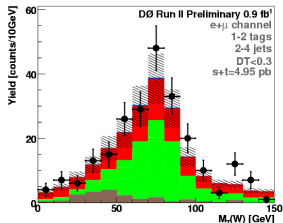
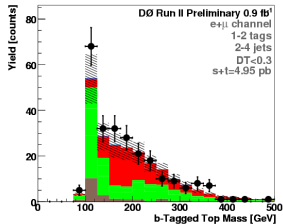


$DT > 0.65$

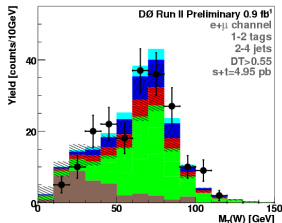
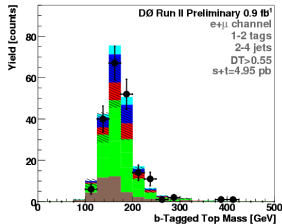


- High BDT region = shows masses of real  $t$  and  $W \Rightarrow$  expected
- Low BDT region = background-like  $\Rightarrow$  expected

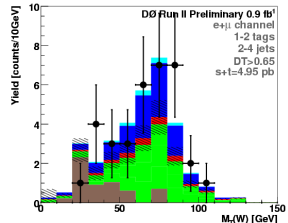
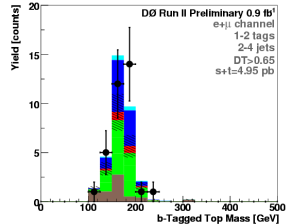
$DT < 0.3$



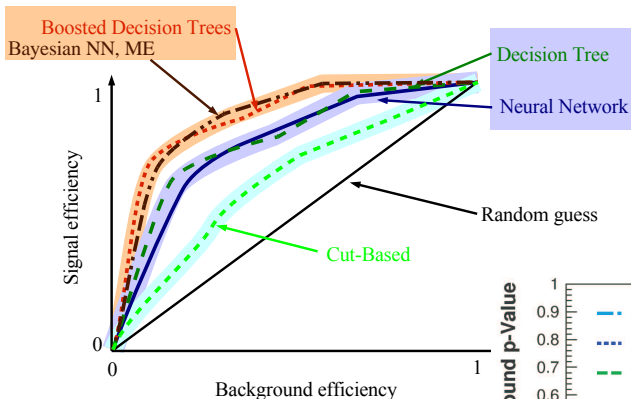
$DT > 0.55$



$DT > 0.65$



- High BDT region = shows masses of real  $t$  and  $W \Rightarrow$  expected
- Low BDT region = background-like  $\Rightarrow$  expected
- Above does NOT tell analysis is ok, but not seeing this could be a sign of a problem



- Cannot know *a priori* which method will work best
- $\Rightarrow$  Need to experiment with different techniques

Power curve

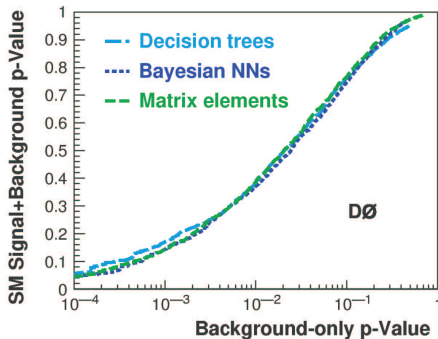


Figure 1 is a plot showing the Inverse Background Efficiency (Y-axis, logarithmic scale from 1 to  $10^3$ ) versus Signal Efficiency (X-axis, linear scale from 0 to 1). The plot compares the performance of three methods: Cuts (blue squares), BDT (red line), and Likelihood (black line). The Likelihood method shows the highest signal efficiency for a given background efficiency. The plot is labeled "tau performance" and "ATLAS Preliminary". The data points represent 2011 dijet data with  $\text{dt L} = 130 \text{ pb}^{-1}$  and  $40 \text{ GeV} < p_T \leq 100 \text{ GeV}$ .

**ATLAS**  
 $\sqrt{s} = 13 \text{ TeV}, 139 \text{ fb}^{-1}$

- ◆ Data
- ttW
- ttZ
- ttH
- Mid.-id.
- 1Mat. Conv.
- Low  $m_{\chi\chi}$
- HF jet
- Others
- ▨ Uncertainty
- tt
- ttZ
- ttH
- Mid-id.
- HF jet
- tt

Events / 0.1

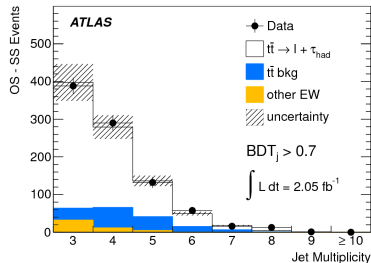
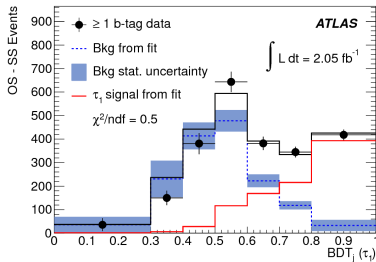
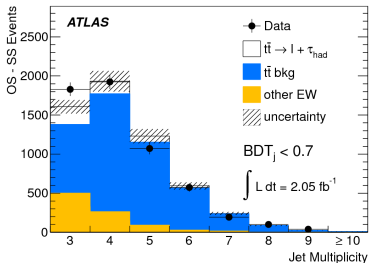
RDT score

Data / Pred.



► Phys.Lett. B717 (2012) 89-108

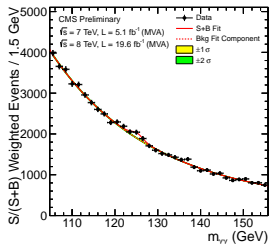
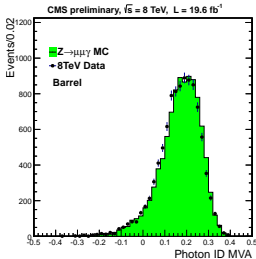
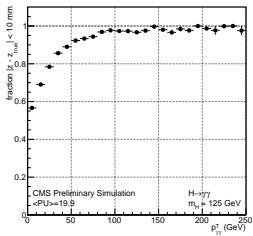
- BDT for tau ID: one to reject electrons, one against jets
- Fit BDT output to get tau contribution in data



► CMS-PAS-HIG-13-001

Hard to use more BDT in an analysis:

- vertex selected with BDT
- 2<sup>nd</sup> vertex BDT to estimate probability to be within 1cm of interaction point
- photon ID with BDT
- photon energy corrected with BDT regression
- event-by-event energy uncertainty from another BDT
- several BDT to extract signal in different categories

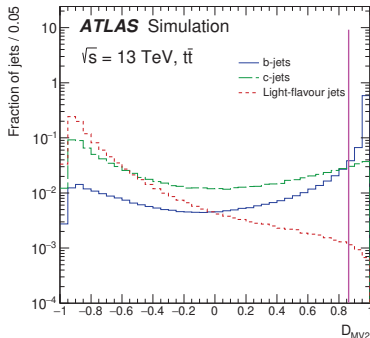
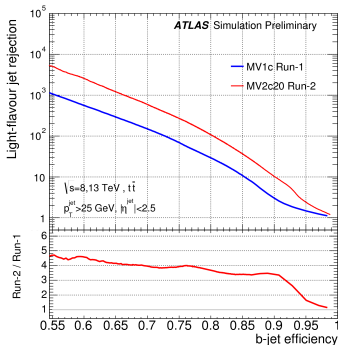


▶ ATL-PHYS-PUB-2015-022

▶ Eur. Phys. J. C **79** (2019) 970

▶ arXiv:1907.05120 [hep-ex]

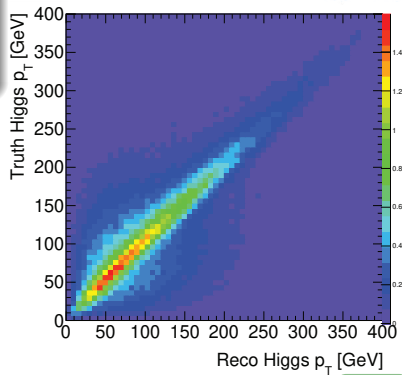
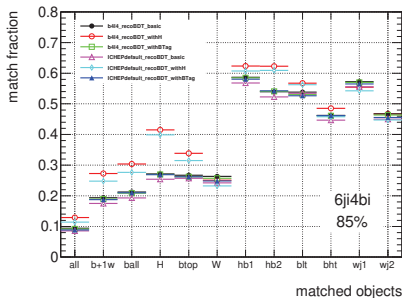
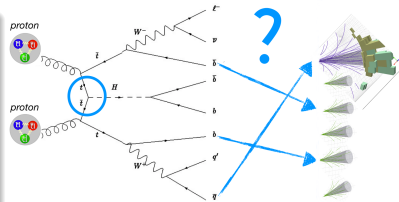
- Run 1 MV1c: NN trained from output of other taggers
- Run 2 MV2c20: BDT using feature variables of underlying algorithms (impact parameter, secondary vertices) and  $p_T$ ,  $\eta$  of jets
- Run 2: introduced IBL (new innermost pixel layer)  
 $\Rightarrow$  explains part of the performance gain, but not all



## $t\bar{t}H(b\bar{b})$ reconstruction

- Match jets and partons in high-multiplicity final state
- BDT trained on all combinations
- New inputs to classification BDT
- Access to Higgs  $p_T$ , origin of  $b$ -jets

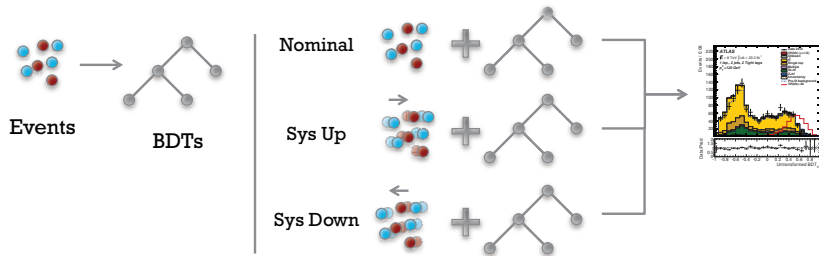
► Phys. Rev. D 97, 072016 (2018)



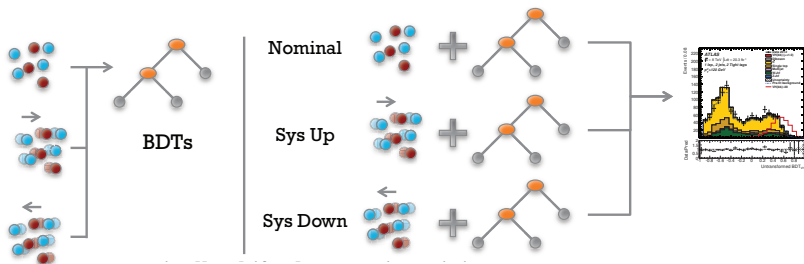
► thesis

► thesis

- No particular rule
- BDT output can be considered as any other cut variable (just more powerful). Evaluate systematics by:
  - varying cut value
  - retraining
  - calibrating, etc.
- Most common (and appropriate, I think): propagate other uncertainties (detector, theory, etc.) up to BDT output and check how much the analysis is affected
- More and more common: profiling.  
Watch out:
  - BDT output powerful
  - signal region (high BDT output) probably low statistics  
⇒ **potential recipe for disaster if modelling is not good**
- May require extra systematics, not so much on technique itself, but because it probes specific corners of phase space and/or wider parameter space (usually loosening pre-BDT selection cuts)

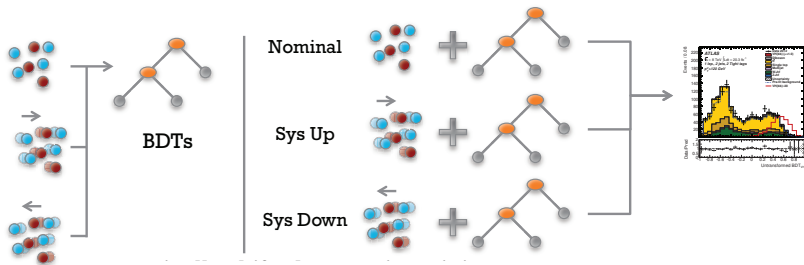


S. Hageböck

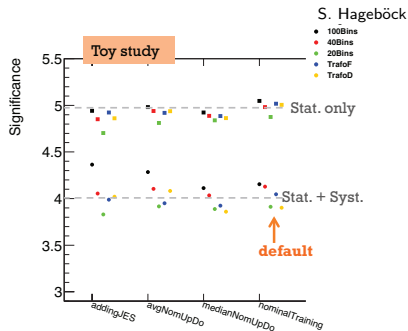


S. Hageböck

- Hope: seeing systematics-affected events during training may make the BDT less sensitive to systematic effects (data augmentation)



- Hope: seeing systematics-affected events during training may make the BDT less sensitive to systematic effects (data augmentation)





- Go for a fully integrated solution
  - use different multivariate techniques easily
  - **spend your time on understanding your data and model**
- Examples:
  - **TMVA** (Toolkit for MultiVariate Analysis)  
Integrated in ROOT, complete manual <https://root.cern/tmva>
  - scikit-learn (python) <http://scikit-learn.org>
- Dedicated to BDT:
  - XGBoost (popular in HEP) [arXiv:1603.02754](https://arxiv.org/abs/1603.02754)  
(note: cannot handle negative weights) <https://github.com/dmlc/xgboost>
  - LightGBM (Microsoft) <https://lightgbm.readthedocs.io>
  - CatBoost (Yandex) <https://catboost.ai/>

- **TMVA**: Toolkit for MultiVariate Analysis

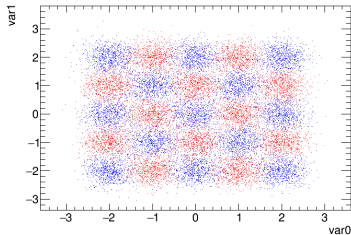
▶ <https://root.cern/tmva>

▶ <https://github.com/root-project/root/tree/master/tmva>

- Written by physicists
- In C++ (also python API), integrated in ROOT
- Quite complete manual
- Includes many different multivariate/machine learning techniques
- To compile, add appropriate header files in your code (e.g., `#include "TMVA/Factory.h"`) and this to your compiler command line:  
`'root-config --cflags --libs' -lTMVA`
- More complete examples of code: `$ROOTSYS/tutorials/tmva`
  - `createData.C` macro to make example datasets
  - classification and regression macros
  - also includes Keras examples (deep learning)
- Sometimes useful performance measures (more in these headers):  
`#include "TMVA/ROCCalc.h"`  
`TMVA::ROCCalc(TH1* S, TH1* B).GetROCIIntegral();`  
`#include "TMVA/Tools.h"`  
`TMVA::gTools().GetSeparation(TH1* S, TH1* B);`

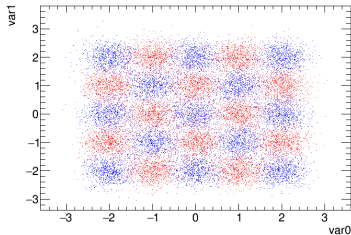
```
TFile* outputFile = TFile::Open("output.root","RECREATE");
TMVA::Factory *factory = new TMVA::Factory( "TMVAClassification", outputFile,
"!V:Color:DrawProgressBar:Transformations=I:AnalysisType=Classification");
```

```
TFile* outputFile = TFile::Open("output.root","RECREATE");
TMVA::Factory *factory = new TMVA::Factory( "TMVAClassification", outputFile,
"!V:Color:DrawProgressBar:Transformations=I:AnalysisType=Classification");
TFile* inputFile = new TFile("dataSchachbrett.root")
TTree* sig = (TTree*)inputFile->Get("TreeS");
TTree* bkg = (TTree*)inputFile->Get("TreeB");
double sigWeight = 1.0; double bkgWeight = 1.0;
TMVA::DataLoader *dataloader =
    new TMVA::DataLoader("dataset");
dataloader->AddSignalTree(sig, sigWeight);
dataloader->AddBackgroundTree(bkg, bkgWeight);
```



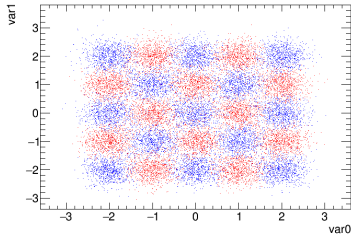
```

TFile* outputFile = TFile::Open("output.root","RECREATE");
TMVA::Factory *factory = new TMVA::Factory( "TMVAClassification", outputFile,
      "!V:Color:DrawProgressBar:Transformations=I:AnalysisType=Classification");
TFile* inputFile = new TFile("dataSchachbrett.root")
TTree* sig = (TTree*)inputFile->Get("TreeS");
TTree* bkg = (TTree*)inputFile->Get("TreeB");
double sigWeight = 1.0; double bkgWeight = 1.0;
TMVA::DataLoader *dataloader =
    new TMVA::DataLoader("dataset");
dataloader->AddSignalTree(sig, sigWeight);
dataloader->AddBackgroundTree(bkg, bkgWeight);
dataloader->AddVariable("var0", 'F');
dataloader->AddVariable("var1", 'F');
TCut mycut = "";
  
```



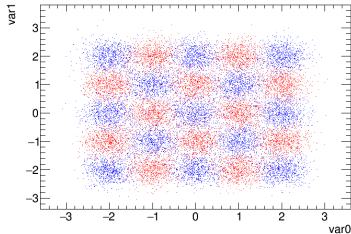
```

TFile* outputFile = TFile::Open("output.root","RECREATE");
TMVA::Factory *factory = new TMVA::Factory( "TMVAClassification", outputFile,
      "!V:Color:DrawProgressBar:Transformations=I:AnalysisType=Classification");
TFile* inputFile = new TFile("dataSchachbrett.root")
TTree* sig = (TTree*)inputFile->Get("TreeS");
TTree* bkg = (TTree*)inputFile->Get("TreeB");
double sigWeight = 1.0; double bkgWeight = 1.0;
TMVA::DataLoader *dataloader =
    new TMVA::DataLoader("dataset");
dataloader->AddSignalTree(sig, sigWeight);
dataloader->AddBackgroundTree(bkg, bkgWeight);
dataloader->AddVariable("var0", 'F');
dataloader->AddVariable("var1", 'F');
TCut mycut = "";
dataloader->PrepareTrainingAndTestTree(mycut, "SplitMode=Random");
  
```

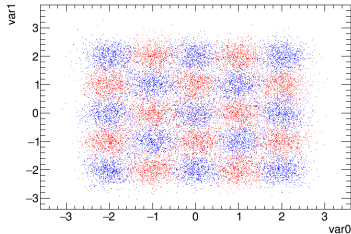


```

TFile* outputFile = TFile::Open("output.root","RECREATE");
TMVA::Factory *factory = new TMVA::Factory( "TMVAClassification", outputFile,
    "!V:Color:DrawProgressBar:Transformations=I:AnalysisType=Classification");
TFile* inputFile = new TFile("dataSchachbrett.root")
TTree* sig = (TTree*)inputFile->Get("TreeS");
TTree* bkg = (TTree*)inputFile->Get("TreeB");
double sigWeight = 1.0; double bkgWeight = 1.0;
TMVA::DataLoader *dataloader =
    new TMVA::DataLoader("dataset");
dataloader->AddSignalTree(sig, sigWeight);
dataloader->AddBackgroundTree(bkg, bkgWeight);
dataloader->AddVariable("var0", 'F');
dataloader->AddVariable("var1", 'F');
TCut mycut = "";
dataloader->PrepareTrainingAndTestTree(mycut,"SplitMode=Random");
factory->BookMethod(dataloader, TMVA::Types::kBDT, "BDT", "!H:!V:NTrees=400:
    MinNodeSize=4%:MaxDepth=5:BoostType=AdaBoost:AdaBoostBeta=0.15:nCuts=80");
factory->BookMethod(dataloader, TMVA::Types::kFisher, "Fisher", "!H:!V:Fisher");
    
```

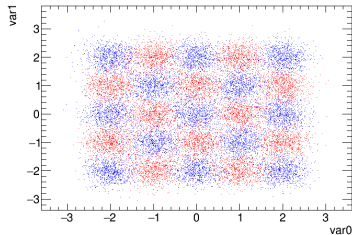


```
TFile* outputFile = TFile::Open("output.root","RECREATE");
TMVA::Factory *factory = new TMVA::Factory( "TMVAClassification", outputFile,
    "!V:Color:DrawProgressBar:Transformations=I:AnalysisType=Classification");
TFile* inputFile = new TFile("dataSchachbrett.root")
TTree* sig = (TTree*)inputFile->Get("TreeS");
TTree* bkg = (TTree*)inputFile->Get("TreeB");
double sigWeight = 1.0; double bkgWeight = 1.0;
TMVA::DataLoader *dataloader =
    new TMVA::DataLoader("dataset");
dataloader->AddSignalTree(sig, sigWeight);
dataloader->AddBackgroundTree(bkg, bkgWeight);
dataloader->AddVariable("var0", 'F');
dataloader->AddVariable("var1", 'F');
TCut mycut = "";
dataloader->PrepareTrainingAndTestTree(mycut,"SplitMode=Random");
factory->BookMethod(dataloader, TMVA::Types::kBDT, "BDT", "!H:!V:NTrees=400:
    MinNodeSize=4%:MaxDepth=5:BoostType=AdaBoost:AdaBoostBeta=0.15:nCuts=80");
factory->BookMethod(dataloader, TMVA::Types::kFisher, "Fisher", "!H:!V:Fisher");
factory->TrainAllMethods(); // Train MVAs using training events
```

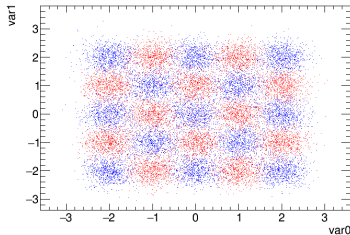




```
TFile* outputFile = TFile::Open("output.root","RECREATE");
TMVA::Factory *factory = new TMVA::Factory( "TMVAClassification", outputFile,
    "!V:Color:DrawProgressBar:Transformations=I:AnalysisType=Classification");
TFile* inputFile = new TFile("dataSchachbrett.root")
TTree* sig = (TTree*)inputFile->Get("TreeS");
TTree* bkg = (TTree*)inputFile->Get("TreeB");
double sigWeight = 1.0; double bkgWeight = 1.0;
TMVA::DataLoader *dataloader =
    new TMVA::DataLoader("dataset");
dataloader->AddSignalTree(sig, sigWeight);
dataloader->AddBackgroundTree(bkg, bkgWeight);
dataloader->AddVariable("var0", 'F');
dataloader->AddVariable("var1", 'F');
TCut mycut = "";
dataloader->PrepareTrainingAndTestTree(mycut,"SplitMode=Random");
factory->BookMethod(dataloader, TMVA::Types::kBDT, "BDT", "!H:!V:NTrees=400:
    MinNodeSize=4%:MaxDepth=5:BoostType=AdaBoost:AdaBoostBeta=0.15:nCuts=80");
factory->BookMethod(dataloader, TMVA::Types::kFisher, "Fisher", "!H:!V:Fisher");
factory->TrainAllMethods(); // Train MVAs using training events
factory->TestAllMethods(); // Evaluate all MVAs using test events
```

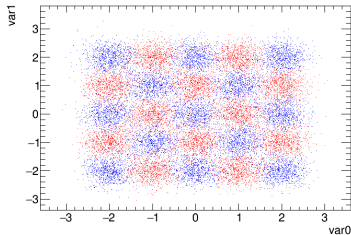


```
TFile* outputFile = TFile::Open("output.root","RECREATE");
TMVA::Factory *factory = new TMVA::Factory( "TMVAClassification", outputFile,
    "!V:Color:DrawProgressBar:Transformations=I:AnalysisType=Classification");
TFile* inputFile = new TFile("dataSchachbrett.root");
TTree* sig = (TTree*)inputFile->Get("TreeS");
TTree* bkg = (TTree*)inputFile->Get("TreeB");
double sigWeight = 1.0; double bkgWeight = 1.0;
TMVA::DataLoader *dataloader =
    new TMVA::DataLoader("dataset");
dataloader->AddSignalTree(sig, sigWeight);
dataloader->AddBackgroundTree(bkg, bkgWeight);
dataloader->AddVariable("var0", 'F');
dataloader->AddVariable("var1", 'F');
TCut mycut = "";
dataloader->PrepareTrainingAndTestTree(mycut,"SplitMode=Random");
factory->BookMethod(dataloader, TMVA::Types::kBDT, "BDT", "!H:!V:NTrees=400:
    MinNodeSize=4%:MaxDepth=5:BoostType=AdaBoost:AdaBoostBeta=0.15:nCuts=80");
factory->BookMethod(dataloader, TMVA::Types::kFisher, "Fisher", "!H:!V:Fisher");
factory->TrainAllMethods(); // Train MVAs using training events
factory->TestAllMethods(); // Evaluate all MVAs using test events
// ----- Evaluate and compare performance of all configured MVAs
factory->EvaluateAllMethods();
```



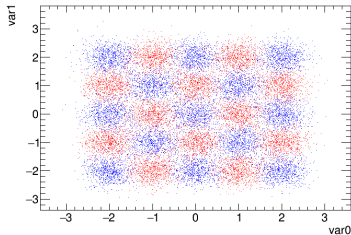
```

TFile* outputFile = TFile::Open("output.root","RECREATE");
TMVA::Factory *factory = new TMVA::Factory( "TMVAClassification", outputFile,
    "!V:Color:DrawProgressBar:Transformations=I:AnalysisType=Classification");
TFile* inputFile = new TFile("dataSchachbrett.root")
TTree* sig = (TTree*)inputFile->Get("TreeS");
TTree* bkg = (TTree*)inputFile->Get("TreeB");
double sigWeight = 1.0; double bkgWeight = 1.0;
TMVA::DataLoader *dataloader =
    new TMVA::DataLoader("dataset");
dataloader->AddSignalTree(sig, sigWeight);
dataloader->AddBackgroundTree(bkg, bkgWeight);
dataloader->AddVariable("var0", 'F');
dataloader->AddVariable("var1", 'F');
TCut mycut = "";
dataloader->PrepareTrainingAndTestTree(mycut,"SplitMode=Random");
factory->BookMethod(dataloader, TMVA::Types::kBDT, "BDT", "!H:!V:NTrees=400:
    MinNodeSize=4%:MaxDepth=5:BoostType=AdaBoost:AdaBoostBeta=0.15:nCuts=80");
factory->BookMethod(dataloader, TMVA::Types::kFisher, "Fisher", "!H:!V:Fisher");
factory->TrainAllMethods(); // Train MVAs using training events
factory->TestAllMethods(); // Evaluate all MVAs using test events
// ----- Evaluate and compare performance of all configured MVAs
factory->EvaluateAllMethods();
auto c1 = factory->GetROCCurve(dataloader); // Eager to compare performance
    
```



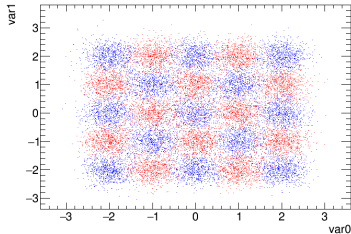
```

TFile* outputFile = TFile::Open("output.root","RECREATE");
TMVA::Factory *factory = new TMVA::Factory( "TMVAClassification", outputFile,
    "!V:Color:DrawProgressBar:Transformations=I:AnalysisType=Classification");
TFile* inputFile = new TFile("dataSchachbrett.root");
TTree* sig = (TTree*)inputFile->Get("TreeS");
TTree* bkg = (TTree*)inputFile->Get("TreeB");
double sigWeight = 1.0; double bkgWeight = 1.0;
TMVA::DataLoader *dataloader =
    new TMVA::DataLoader("dataset");
dataloader->AddSignalTree(sig, sigWeight);
dataloader->AddBackgroundTree(bkg, bkgWeight);
dataloader->AddVariable("var0", 'F');
dataloader->AddVariable("var1", 'F');
TCut mycut = "";
dataloader->PrepareTrainingAndTestTree(mycut,"SplitMode=Random");
factory->BookMethod(dataloader, TMVA::Types::kBDT, "BDT", "!H:!V:NTrees=400:
    MinNodeSize=4%:MaxDepth=5:BoostType=AdaBoost:AdaBoostBeta=0.15:nCuts=80");
factory->BookMethod(dataloader, TMVA::Types::kFisher, "Fisher", "!H:!V:Fisher");
factory->TrainAllMethods(); // Train MVAs using training events
factory->TestAllMethods(); // Evaluate all MVAs using test events
// ----- Evaluate and compare performance of all configured MVAs
factory->EvaluateAllMethods();
auto c1 = factory->GetROCCurve(dataloader); // Eager to compare performance
outputFile->Close();
delete factory; delete dataloader;
    
```



```

TFile* outputFile = TFile::Open("output.root","RECREATE");
TMVA::Factory *factory = new TMVA::Factory( "TMVAClassification", outputFile,
    "!V:Color:DrawProgressBar:Transformations=I:AnalysisType=Classification");
TFile* inputFile = new TFile("dataSchachbrett.root")
TTree* sig = (TTree*)inputFile->Get("TreeS");
TTree* bkg = (TTree*)inputFile->Get("TreeB");
double sigWeight = 1.0; double bkgWeight = 1.0;
TMVA::DataLoader *dataloader =
    new TMVA::DataLoader("dataset");
dataloader->AddSignalTree(sig, sigWeight);
dataloader->AddBackgroundTree(bkg, bkgWeight);
dataloader->AddVariable("var0", 'F');
dataloader->AddVariable("var1", 'F');
TCut mycut = "";
dataloader->PrepareTrainingAndTestTree(mycut,"SplitMode=Random");
factory->BookMethod(dataloader, TMVA::Types::kBDT, "BDT", "!H:!V:NTrees=400:
    MinNodeSize=4%:MaxDepth=5:BoostType=AdaBoost:AdaBoostBeta=0.15:nCuts=80");
factory->BookMethod(dataloader, TMVA::Types::kFisher, "Fisher", "!H:!V:Fisher");
factory->TrainAllMethods(); // Train MVAs using training events
factory->TestAllMethods(); // Evaluate all MVAs using test events
// ----- Evaluate and compare performance of all configured MVAs
factory->EvaluateAllMethods();
auto c1 = factory->GetROCCurve(dataloader); // Eager to compare performance
outputFile->Close();
delete factory; delete dataloader;
TMVA::TMVAGui("output.root");
    
```



```
TFile* inputFile = new TFile("dataSchachbrett.root");  
TTree* data = (TTree*)inputFile->Get("TreeS");  
Float_t var0=-99., var1=-99.;  
data->SetBranchAddress("var0", &var0);  
data->SetBranchAddress("var1", &var1);
```

```
TFile* inputFile = new TFile("dataSchachbrett.root");
TTree* data = (TTree*)inputFile->Get("TreeS");
Float_t var0=-99., var1=-99.;
data->SetBranchAddress("var0", &var0);
data->SetBranchAddress("var1", &var1);
TMVA::Reader *reader = new TMVA::Reader();
reader->AddVariable( "var0", &var0 );
reader->AddVariable( "var1", &var1 );
```

```
TFile* inputFile = new TFile("dataSchachbrett.root");
TTree* data = (TTree*)inputFile->Get("TreeS");
Float_t var0=-99., var1=-99.;
data->SetBranchAddress("var0", &var0);
data->SetBranchAddress("var1", &var1);
TMVA::Reader *reader = new TMVA::Reader();
reader->AddVariable( "var0", &var0 );
reader->AddVariable( "var1", &var1 );
reader->BookMVA( "My BDT", "dataset/weights/TMVAClassification_BDT.weights.xml");
reader->BookMVA( "Fisher discriminant",
    "dataset/weights/TMVAClassification_Fisher.weights.xml");
```



```

TFile* inputFile = new TFile("dataSchachbrett.root");
TTree* data = (TTree*)inputFile->Get("TreeS");
Float_t var0=-99., var1=-99.;
data->SetBranchAddress("var0", &var0);
data->SetBranchAddress("var1", &var1);
TMVA::Reader *reader = new TMVA::Reader();
reader->AddVariable( "var0", &var0 );
reader->AddVariable( "var1", &var1 );
reader->BookMVA( "My BDT", "dataset/weights/TMVAClassification_BDT.weights.xml");
reader->BookMVA( "Fisher discriminant",
    "dataset/weights/TMVAClassification_Fisher.weights.xml");
// ----- start your event loop
for (Long64_t ievt=0; ievt<10; ++ievt) {
    data->GetEntry(ievt);
    double bdt = reader->EvaluateMVA("My BDT");
    double fisher = reader->EvaluateMVA("Fisher discriminant");
    cout<<"var0="<<var0<<" var1="<<var1<<" BDT="<<bdt<<" Fisher="<<fisher<<endl;
}
delete reader;
inputFile->Close();

```

```
TFile* inputFile = new TFile("dataSchachbrett.root");
TTree* data = (TTree*)inputFile->Get("TreeS");
Float_t var0=-99., var1=-99.;
data->SetBranchAddress("var0", &var0);
data->SetBranchAddress("var1", &var1);
TMVA::Reader *reader = new TMVA::Reader();
reader->AddVariable( "var0", &var0 );
reader->AddVariable( "var1", &var1 );
reader->BookMVA( "My BDT", "dataset/weights/TMVAClassification_BDT.weights.xml");
reader->BookMVA( "Fisher discriminant",
    "dataset/weights/TMVAClassification_Fisher.weights.xml");
// ----- start your event loop
for (Long64_t ievt=0; ievt<10; ++ievt) {
    data->GetEntry(ievt);
    double bdt = reader->EvaluateMVA("My BDT");
    double fisher = reader->EvaluateMVA("Fisher discriminant");
    cout<<"var0="<<var0<<" var1="<<var1<<" BDT="<<bdt<<" Fisher="<<fisher<<endl;
}
delete reader;
inputFile->Close();
```

- How to compile in C++, training tricks: see [▶ backup](#)
- More complete tutorials:

[▶ https://github.com/lmoneta/tmva-tutorial](https://github.com/lmoneta/tmva-tutorial)

- Decision trees: natural extension to cut-based analysis
- Greatly improved performance with boosting (and also with bagging, random forests)
- Has become rather fashionable in HEP (soon has-been?)
- Possibly soon overpowered by deep learning algorithms, although trickier to optimise
- Whichever technique you use, expect a lot of scepticism: you will have to convince yourself and others that your advanced technique leads to meaningful and reliable results
  - ⇒ ensemble tests, use several techniques, compare to random grid search, etc. But DO NOT show them useless plots like BDT output on training and testing to measure overtraining, please!
- As with other advanced techniques,  
no point in using them if data are not understood and well modelled

- Decision trees: natural extension to cut-based analysis
- Greatly improved performance with boosting (and also with bagging, random forests)
- Has become rather fashionable in HEP (soon has-been?)
- Possibly soon overpowered by deep learning algorithms, although trickier to optimise
- Whichever technique you use, expect a lot of scepticism: you will have to convince yourself and others that your advanced technique leads to meaningful and reliable results  
 ⇒ ensemble tests, use several techniques, compare to random grid search, etc. But DO NOT show them useless plots like BDT output on training and testing to measure overtraining, please!
- As with other advanced techniques,  
 no point in using them if data are not understood and well modelled

**Upcoming reference book (in about six months)**

Artificial Intelligence for High Energy Physics

► <https://doi.org/10.1142/12200>

- PIDForest: Anomaly Detection via Partial Identification ▶ NeurIPS
- A Debiased MDI (Mean Decrease of Impurity) Feature Importance Measure for Random Forests ▶ NeurIPS
- MonoForest framework for tree ensemble analysis ▶ NeurIPS
- Faster Boosting with Smaller Memory (Yoav S Freund) ▶ NeurIPS
- Minimal Variance Sampling in Stochastic Gradient Boosting ▶ NeurIPS
- Regularized Gradient Boosting ▶ NeurIPS
- Partitioning Structure Learning for Segmented Linear Regression Trees ▶ NeurIPS
- Random Tessellation Forests ▶ NeurIPS
- Optimal Sparse Decision Trees ▶ NeurIPS
- Provably robust boosted decision stumps and trees against adversarial attacks ▶ NeurIPS
- Robustness Verification of Tree-based Models ▶ NeurIPS



L. Breiman, J.H. Friedman, R.A. Olshen and C.J. Stone, *Classification and Regression Trees*, Wadsworth, Stamford, 1984



R.E. Schapire, "The strength of weak learnability", *Machine Learning*, 5(2):197–227, 1990



Y. Freund, "Boosting a weak learning algorithm by majority", *Information and computation*. 121(2):256–285, 1995



Y. Freund and R.E. Schapire, "Experiments with a New Boosting Algorithm" in *Machine Learning: Proceedings of the Thirteenth International Conference*, edited by L. Saitta (Morgan Kaufmann, San Fransisco, 1996) p. 148



Y. Freund and R.E. Schapire, "A short introduction to boosting" *Journal of Japanese Society for Artificial Intelligence*, 14(5):771-780 (1999)











R. E. Schapire and Y. Freund, "Boosting: Foundations and Algorithms", *MIT Press*, 2012.



Y. Freund and R.E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting", *Journal of Computer and System Sciences*, 55(1):119–139, 1997



J.H. Friedman, T. Hastie and R. Tibshirani, "Additive logistic regression: a statistical view of boosting", *The Annals of Statistics*, 28(2), 377–386, 2000

-  J. H. Friedman, "Greedy function approximation: A gradient boosting machine", *Ann. Statist.* 29 (2001) 1189
-  T. Hastie, R. Tibshirani, and J. Friedman, "The Elements of Statistical Learning: Data Mining, Inference, and Prediction (2nd edition)", *Springer Series in Statistics*, 2009
-  S. Shalev-Shwartz and S. Ben-David, "Understanding Machine Learning: From Theory to Algorithms", *Cambridge University Press*, 2014
-  M. Belkin, D. Hsu, S. Ma, and S. Mandal, "Reconciling modern machine-learning practice and the classical bias–variance trade-off", *PNAS* 116 (2019) 15849, arXiv:1812.11118 [stat.ML]
-  L. Breiman, "Bagging Predictors", *Machine Learning*, 24 (2), 123–140, 1996
-  L. Breiman, "Random forests", *Machine Learning*, 45 (1), 5–32, 2001
-  B.P. Roe, H.-J. Yang, J. Zhu, Y. Liu, I. Stancu, and G. McGregor, Nucl. Instrum. Methods Phys. Res., Sect.A 543, 577 (2005); H.-J. Yang, B.P. Roe, and J. Zhu, Nucl. Instrum.Methods Phys. Res., Sect. A 555, 370 (2005)
-  V. M. Abazov *et al.* [D0 Collaboration], "Evidence for production of single top quarks", *Phys. Rev. D* **78**, 012005 (2008)

Backup



## Defined for many classes

- $$\text{Gini} = \sum_{i,j \in \{\text{classes}\}}^{i \neq j} p_i p_j$$

## Statistical interpretation

- Assign random object to class  $i$  with probability  $p_i$ .
- Probability that it is actually in class  $j$  is  $p_j$
- $\Rightarrow$  Gini = probability of misclassification

## For two classes (signal and background)

- $i = s, b$  and  $p_s = p = 1 - p_b$
- $\Rightarrow \text{Gini} = 1 - \sum_{i=s,b} p_i^2 = 2p(1 - p) = \frac{2sb}{(s+b)^2}$
- Most popular in DT implementations
- Usually similar performance to e.g. entropy

## Pre-pruning

- Stop tree growth during building phase
- Already seen: minimum leaf size, minimum separation improvement, maximum depth, etc.
- Careful: early stopping condition may prevent from discovering further useful splitting

## Expected error pruning

- Grow full tree
- When result from children not significantly different from result of parent, prune children
- Can measure statistical error estimate with binomial error  $\sqrt{p(1-p)/N}$  for node with purity  $p$  and  $N$  training events
- No need for testing sample
- Known to be “too aggressive”

- Idea: penalise “complex” trees (many nodes/leaves) and find compromise between good fit to training data (larger tree) and good generalisation properties (smaller tree)
- With misclassification rate  $R(T)$  of subtree  $T$  (with  $N_T$  nodes) of fully grown tree  $T_{max}$ :

$$\text{cost complexity } R_\alpha(T) = R(T) + \alpha N_T$$

$\alpha$  = complexity parameter

- Minimise  $R_\alpha(T)$ :
  - small  $\alpha$ : pick  $T_{max}$
  - large  $\alpha$ : keep root node only,  $T_{max}$  fully pruned
- First-pass pruning, for terminal nodes  $t_L, t_R$  from split of  $t$ :
  - by construction  $R(t) \geq R(t_L) + R(t_R)$
  - if  $R(t) = R(t_L) + R(t_R)$  prune off  $t_L$  and  $t_R$

- For node  $t$  and subtree  $T_t$ :
  - if  $t$  non-terminal,  $R(t) > R(T_t)$  by construction
  - $R_\alpha(\{t\}) = R_\alpha(t) = R(t) + \alpha (N_T = 1)$
  - if  $R_\alpha(T_t) < R_\alpha(t)$  then branch has smaller cost-complexity than single node and should be kept
  - at critical  $\alpha = \rho_t$ , node is preferable
  - to find  $\rho_t$ , solve  $R_{\rho_t}(T_t) = R_{\rho_t}(t)$ , or:  $\rho_t = \frac{R(t) - R(T_t)}{N_T - 1}$
  - node with smallest  $\rho_t$  is *weakest link* and gets pruned
  - apply recursively till you get to the root node
- This generates sequence of decreasing cost-complexity subtrees
- Compute their true misclassification rate on validation sample:
  - will first decrease with cost-complexity
  - then goes through a minimum and increases again
  - pick this tree at the minimum as the best pruned tree
- Note: best pruned tree may not be optimal in a forest

- To make code compilable (and MUCH faster)
  - Need ROOT and TMVA corresponding header files
  - e.g., for Train.C:

```
#include "TFile.h"
#include "TTree.h"
#include "TMVA/Factory.h"
#include "TMVA/DataLoader.h"
#include "TMVA/TMVAGui.h"
```

- Need a “main” function

```
int main() {
    Train();
    return 0;
}
```

- Compilation:

```
g++ Train.C 'root-config --cflags --libs' -lTMVA -lTMVAGui -o TMVATrainer
```

- Train.C: file to compile
- TMVATrainer: name of executable
- -lTMVAGui: just because of TMVA::TMVAGui("output.root");

- Common technique: train on even event numbers, test on odd event numbers (and vice versa)
- Can also think of more than two-fold
- Achieve in TMVA by replacing:

```
dataloader->AddSignalTree(sig, sigWeight);
dataloader->AddBackgroundTree(bkg, bkgWeight);
```

- with:

```
TString trainString = "(eventNumber % 2 == 0)";
TString testString = "!" + trainString;
dataloader->AddTree(sig, "Signal", sigWeight, trainString.Data(), "Training");
dataloader->AddTree(sig, "Signal", sigWeight, testString.Data(), "Test");
dataloader->AddTree(bkg, "Background", bkgWeight, trainString.Data(), "Training");
dataloader->AddTree(bkg, "Background", bkgWeight, testString.Data(), "Test");
```

- Use individual event weights:

```
string eventWeight = "TMath::Abs(eventWeight)"; //Compute event weight
dataloader->SetSignalWeightExpression(eventWeight);
dataloader->SetBackgroundWeightExpression(eventWeight); //Can differ
```