Boosted decision trees

Yann Coadou

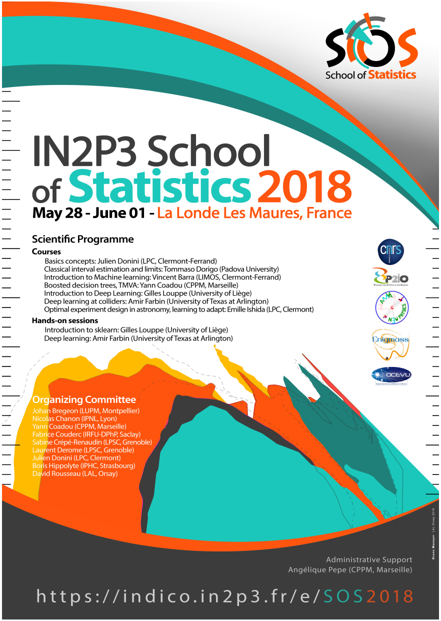
CPPM Marseille

SOS2018

La Londe-les-Maures, 28 May 2018

School of Statistics

Outline



1 Introduction
2 Growing a tree
3 Tree (in)stability
4 Boosting
5 BDT performance
6 Concrete examples
7 Other averaging techniques
8 BDTs in real physics cases
9 BDT systematics
10 Software
11 Conclusion
12 References

Before we go on...

!!! VERY IMPORTANT !!!

Understand your inputs well
before you start playing with
multivariate techniques and machine learning

Introduction

Decision tree origin

- Machine-learning technique, widely used in social sciences.
  Originally data mining/pattern recognition, then medical diagnostic, 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 Growing a tree
3 Tree (in)stability
4 Boosting
5 BDT performance
6 Concrete examples
7 Other averaging techniques
8 BDTs in real physics cases
9 BDT systematics
10 Software
11 Conclusion
12 References

Tree building algorithm

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

Algorithm example

- 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

Algorithm example

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

Algorithm example

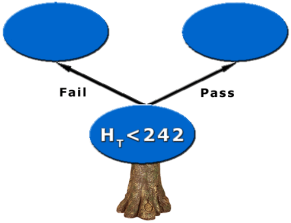
- Consider signal (s_i) and background (b_j) events described by 3 variables: p_T of leading jet, top mass M_t and scalar sum of p_T's of all objects in the event H_T
  - sort all events by each variable:
    - $p_T^{s_1} \leq p_T^{b_{34}} \leq \cdots \leq p_T^{b_2} \leq p_T^{s_{12}}$
    - $H_T^{b_5}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

Algorithm example

- 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:
    - $p_T^{s_1} \leq p_T^{b_{34}} \leq \cdots \leq p_T^{b_2} \leq p_T^{s_{12}}$
    - $H_T^{b_5} \leq H_T^{b_3} \leq \cdots \leq H_T^{s_{67}} \leq H_T^{s_{43}}$
    - $M_t^{b_6} \leq M_t^{s_8} \leq \cdots \leq M_t^{s_{12}} \leq M_t^{b_9}$
  - best split (arbitrary unit):
    - $p_T < 56$ GeV, separation $= 3$
    - $H_T < 242$ GeV, separation $= 5$
    - $M_t < 105$ GeV, separation $= 0.7$

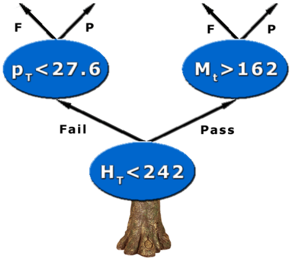
Algorithm example

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



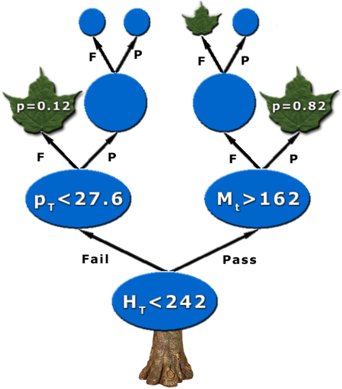
Algorithm example

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



Algorithm example

- 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:
    - $p_T^{s_1} \leq p_T^{b_{34}} \leq \cdots \leq p_T^{b_2} \leq p_T^{s_{12}}$
    - $H_T^{b_5} \leq H_T^{b_3} \leq \cdots \leq H_T^{s_{67}} \leq H_T^{s_{43}}$
    - $M_t^{b_6} \leq M_t^{s_8} \leq \cdots \leq M_t^{s_{12}} \leq M_t^{b_9}$
  - 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 ($p = 0.82$)



Decision tree output

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 $\left(\frac{s}{s+b}\right.$, with weighted events$\left.\right)$ 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 $+1$

Tree construction parameters

Normalization of signal and background before training

- 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

Splitting a node

Impurity measure i(t)

- maximal for equal mix of signal and background
- symmetric in p_{signal} and $p_{background}$
- 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:

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

Stopping condition

- See previous slide
- When not enough improvement ($\Delta i(s^*, t) < \beta$)
- Careful with early-stopping conditions

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

Splitting a node: examples

Node purity

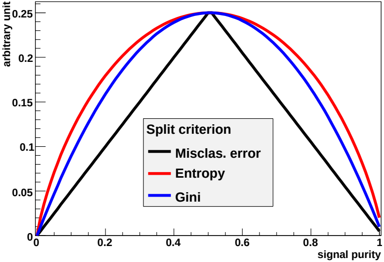
- Signal (background) event i with weight w_s^i (w_b^i)

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

- 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 ($-\frac{s^2}{b}$)

Variable selection I

Reminder

- Need model giving good description of data

Variable selection I

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

Variable selection II

Transforming input variables

- Completely insensitive to the replacement of any subset of input variables by (possibly different) arbitrary strictly monotone functions of them:
  - let $f : x_i \to f(x_i)$ 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 $\to$ 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

Variable selection II

Transforming input variables

- Completely insensitive to the 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 selection III

Variable ranking

- 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

  $\Rightarrow$ 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

1 Introduction
2 Growing a tree
3 **Tree (in)stability**
4 Boosting
5 BDT performance
6 Concrete examples
7 Other averaging techniques
8 BDTs in real physics cases
9 BDT systematics
10 Software
11 Conclusion
12 References

Tree instability: training sample composition

- Small changes in sample can lead to very different tree structures
- 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 ± 1 if not using purity)

Pruning a tree

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)

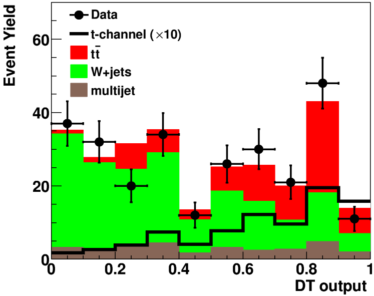
- Pre-pruning (early stopping condition like min leaf size, max depth)
- Expected error pruning (based on statistical error estimate)
- Cost-complexity pruning (penalise “complex” trees with many nodes/leaves)

Decision tree score card

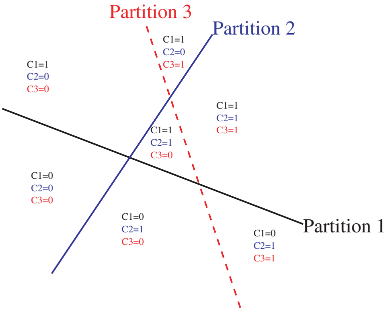
- ✔ Training is fast
- ✔ Human readable (not a black box, can interpret tree as selection rules or physics)

- ✔ 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
- ✔✘ Not that "original" in HEP anymore
- ✘ Unstable tree structure
- ✘ Piecewise nature of output
- ✘ Need at least as many training examples as variations in target function

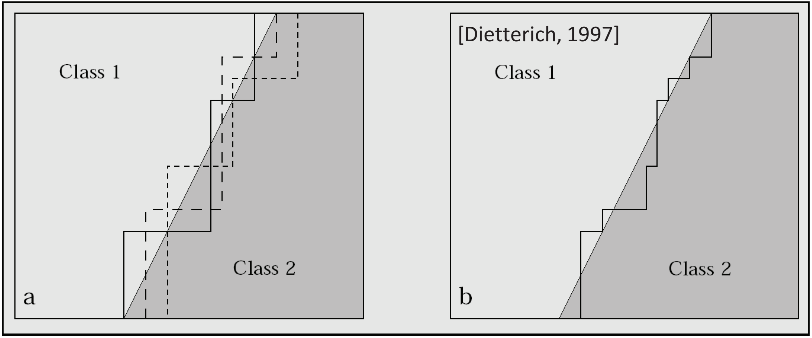


- One tree:
  - one information about event (one leaf)
  - cannot really generalise to variations not covered in training set (at most as many leaves as input size)
- Many trees:
  - distributed representation: number of intersections of leaves exponential in number of trees
  - many leaves contain the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



Tree (in)stability solution: averaging

- 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. Growing a tree
3. Tree (in)stability
4. **Boosting**
5. BDT performance
6. Concrete examples
7. Other averaging techniques
8. BDTs in real physics cases
9. BDT systematics
10. Software
11. Conclusion
12. References

Boosting: a brief history

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)

Boosting: a brief history

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: 1st functional model AdaBoost (1996)

Boosting: a brief history

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: 1st 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 α_k to T_k
  modify $\mathbb{T}_k$ into $\mathbb{T}_{k+1}$
- Boosted output: $F(T_1, .., T_{N_{tree}})$

AdaBoost

- 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

AdaBoost algorithm

- 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 $\{\pm 1\}$: $\text{isMisclassified}_k(i) = \mathbb{I}\big(y_i \times T_k(x_i) \leq 0\big)$
  - or $\text{isMisclassified}_k(i) = \mathbb{I}\big(y_i \times (T_k(x_i) - 0.5) \leq 0\big)$ 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 $\mathbb{T}_{k+1}$:

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

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

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

AdaBoost by example

- 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

AdaBoost error rate

Misclassification rate ε 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):

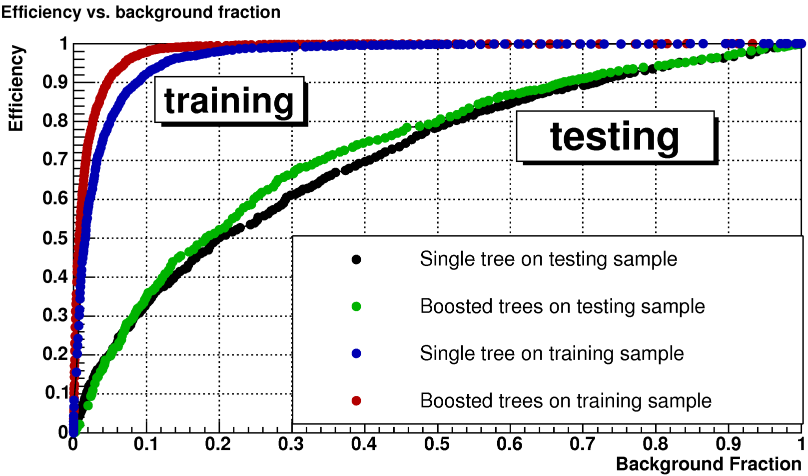
the error rate falls to zero for sufficiently large N_{tree}

- Corollary: training data is over fitted

Overtraining?

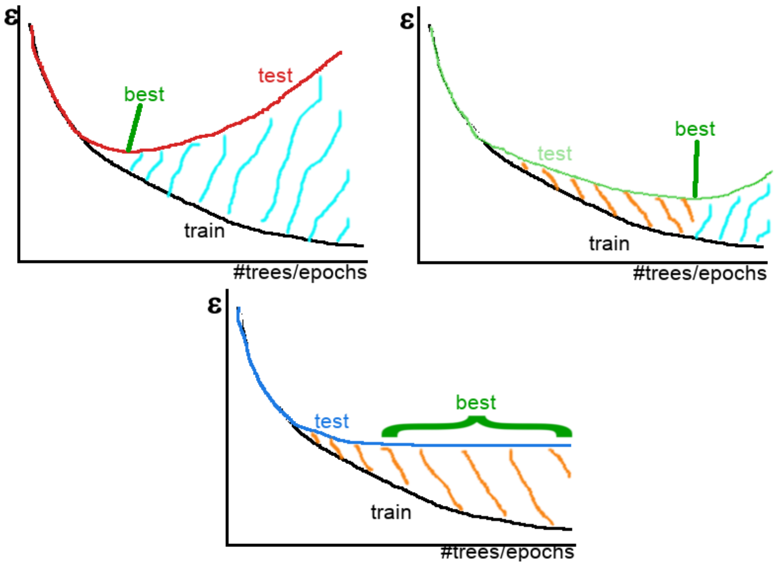
- Error rate on test sample may reach a minimum and then potentially rise. Stop boosting at the minimum.
- In principle AdaBoost *must* overfit training sample
- In many cases in literature, no loss of performance due to overtraining
  - may have to do with fact that successive trees get in general smaller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

Training and generalisation error



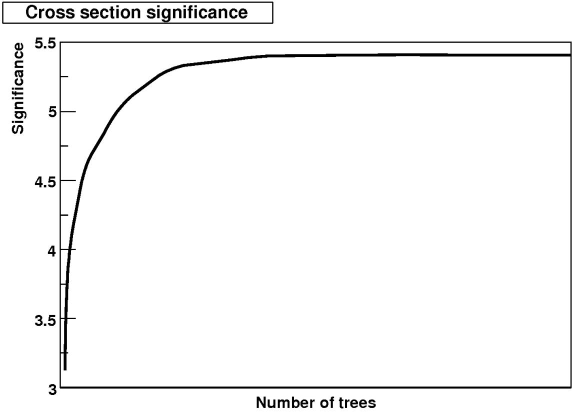
- Clear overtraining, but still better performance after boosting

Overtraining estimation: good or bad?



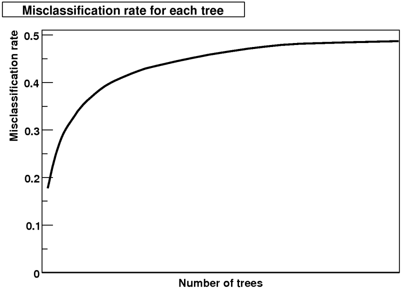
"good" overtraining / "bad" overtraining

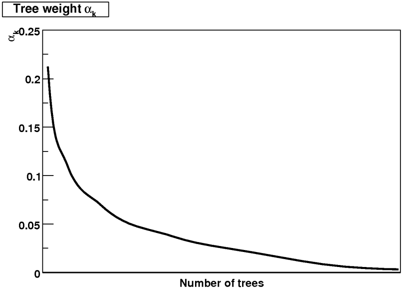
Cross section significance $(s/\sqrt{s+b})$



- 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

Clues to boosting performance

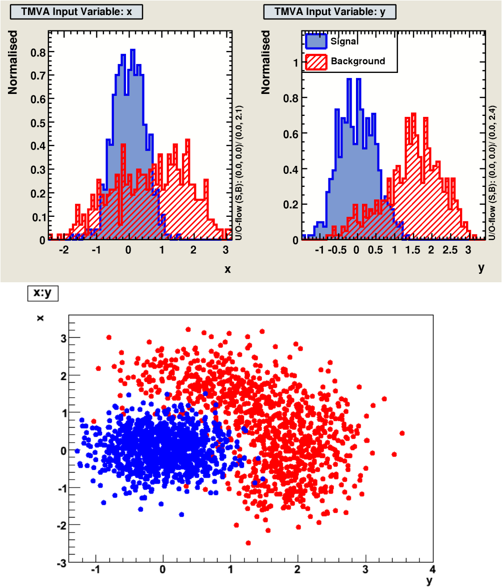




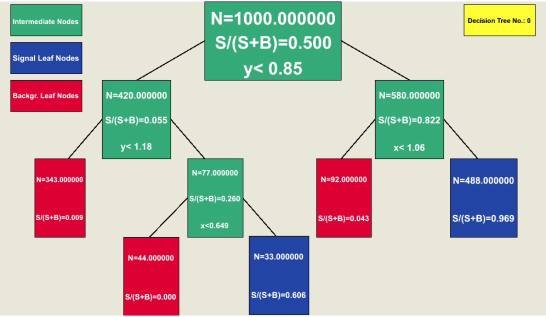
- First tree is best, others are minor corrections
- Specialised trees do not perform well on most events $\Rightarrow$ decreasing tree weight and increasing misclassification rate
- Last tree is not better evolution of first tree, but rather a pretty bad DT that only does a good job on few cases that the other trees could not get right

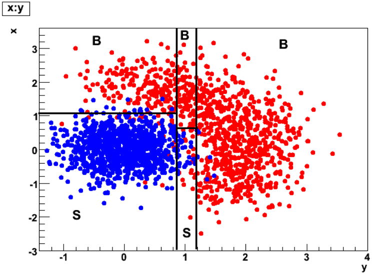
1. Introduction
2. Growing a tree
3. Tree (in)stability
4. Boosting
5. BDT performance
6. **Concrete examples**
7. Other averaging techniques
8. BDTs in real physics cases
9. BDT systematics
10. Software
11. Conclusion
12. References

Concrete example

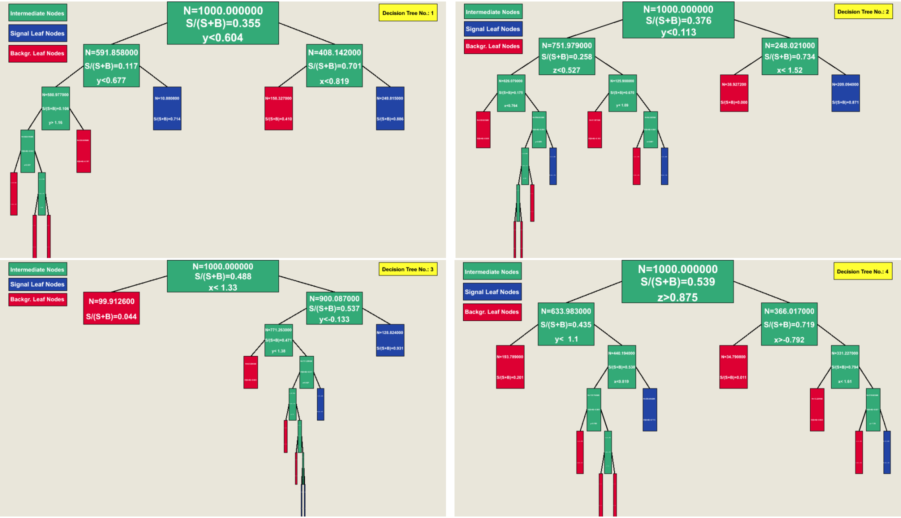


Concrete example



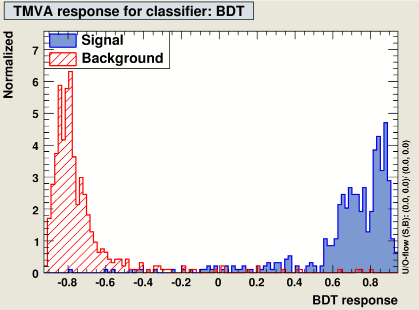


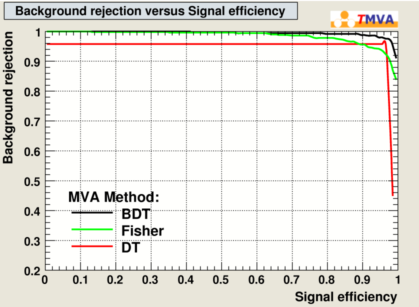
Concrete example



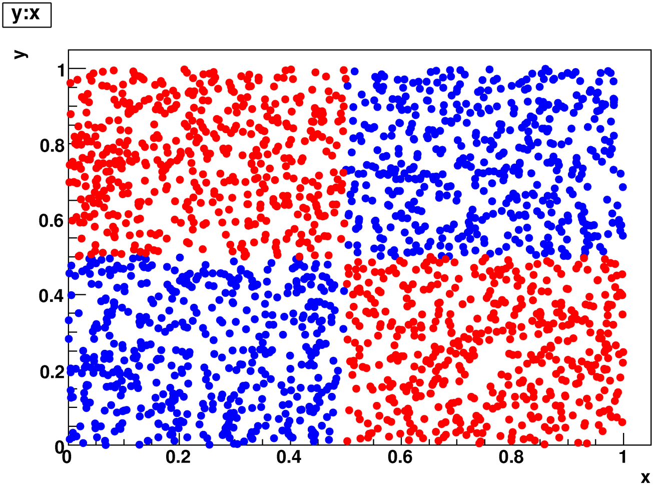
- Specialised trees

Concrete example

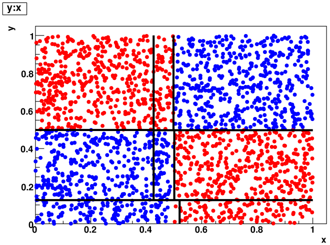


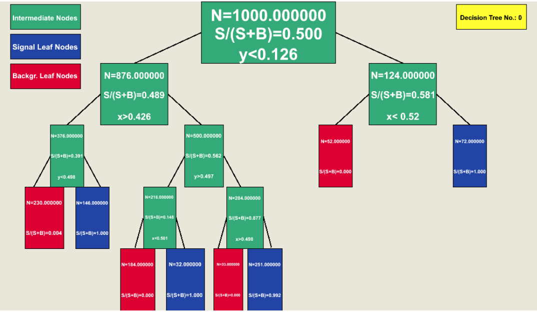


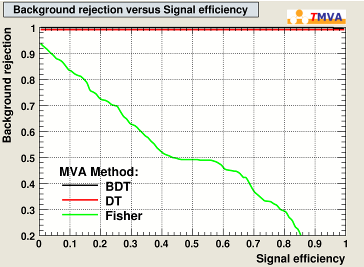
Concrete example: XOR



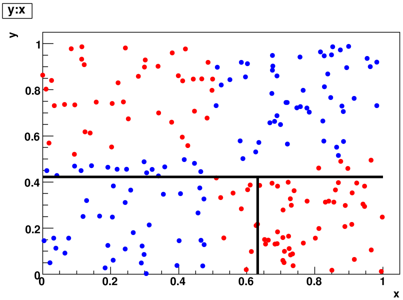
Concrete example: XOR





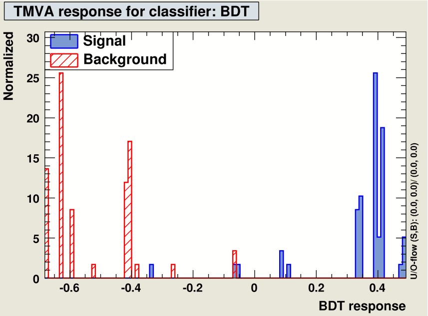


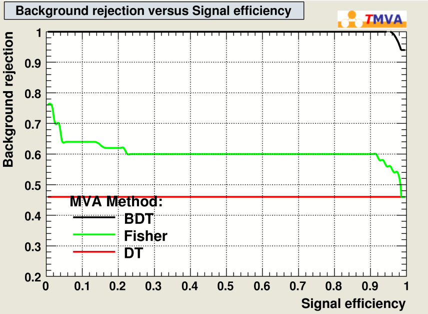
Concrete example: XOR with 100 events



Small statistics

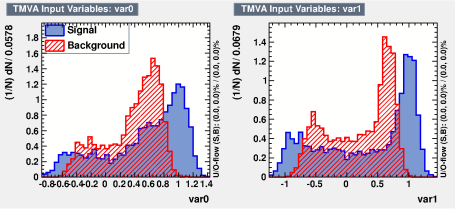
- Single tree or Fischer discriminant not so good
- BDT very good: high performance discriminant from combination of weak classifiers

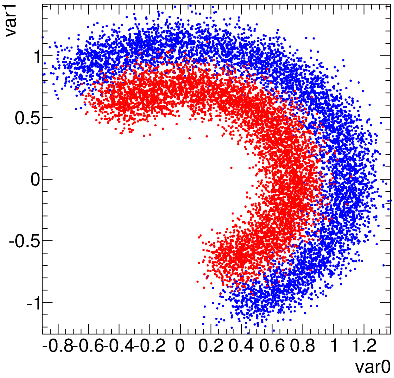




Circular correlation

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

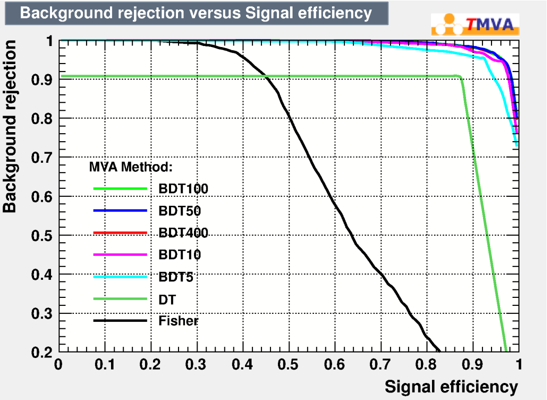




Circular correlation

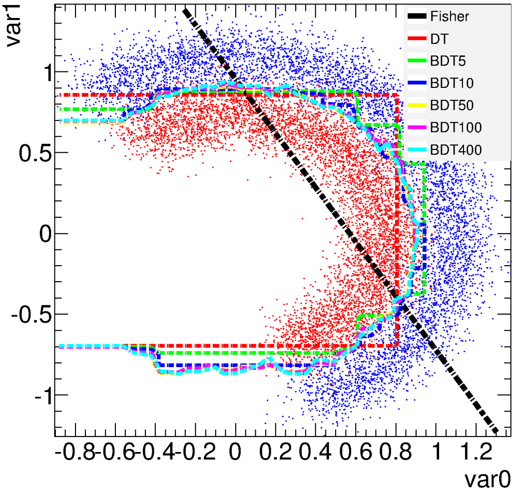
Boosting longer (TMVA: `NTrees`)

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



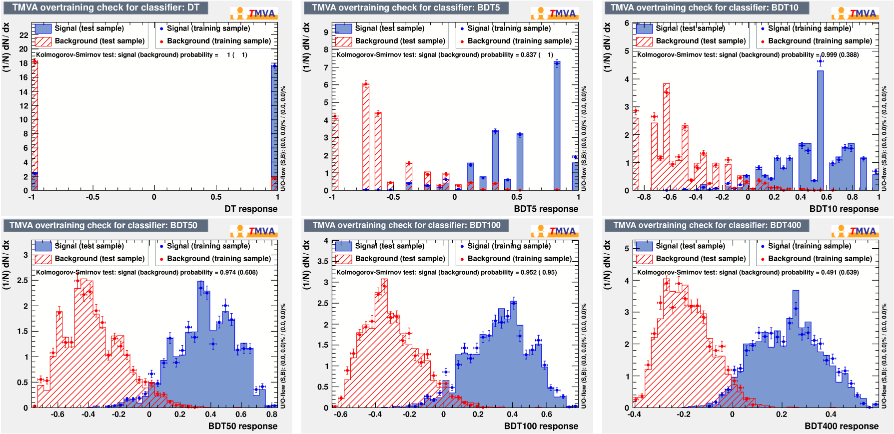
- Fisher bad (expected)
- Single (small) DT: not so good
- More trees ⇒ improve performance until saturation

Decision contours



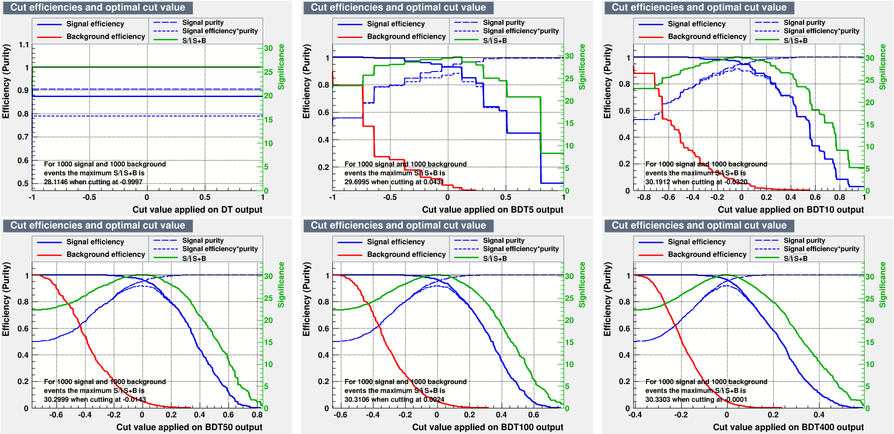
- Fisher bad (expected)
- Note: max tree depth = 3
- Single (small) DT: not so good. Note: a larger tree would solve this problem
- More trees ⇒ improve performance (less step-like, closer to optimal separation) until saturation
- Largest BDTs: wiggle a little around the contour ⇒ picked up features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

Training/testing output



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

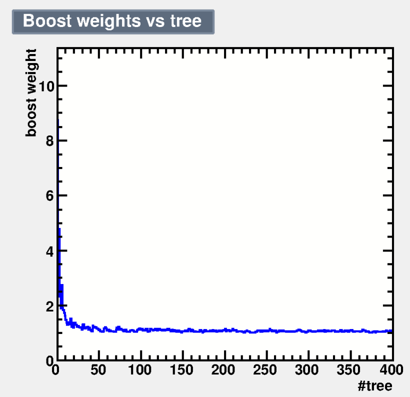
Performance in optimal significance

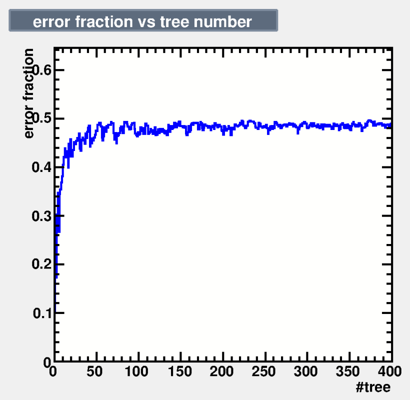


- Best significance actually obtained with last BDT, 400 trees!
- But to be fair, equivalent performance with 10 trees already
- Less “stepped” output desirable? $\Rightarrow$ maybe 50 is reasonable

Control plots

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

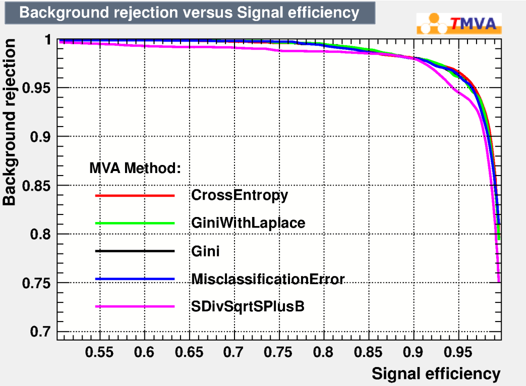




Circular correlation

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

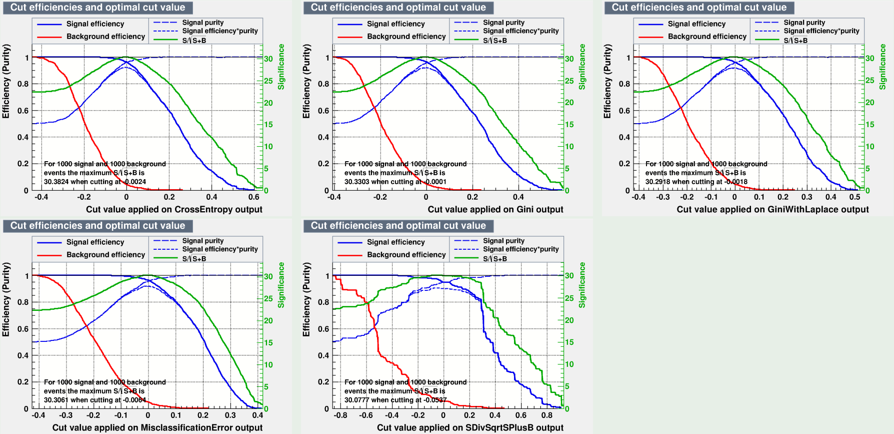
- Compare performance of Gini, entropy, misclassification error, $\frac{s}{\sqrt{s+b}}$
- All other parameters at TMVA default



- 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)

Circular correlation

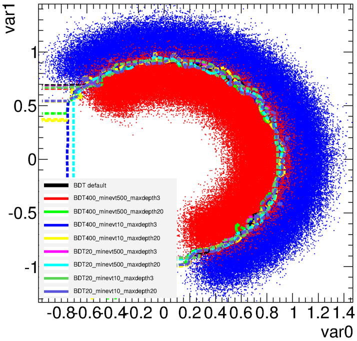
Performance in optimal significance

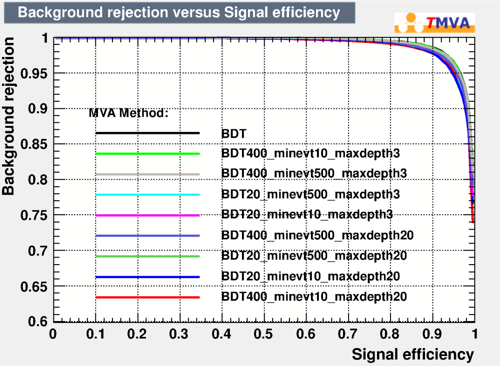


- Confirms previous page: very similar performance, worse for BDT optimised with significance!

Many small trees or fewer large trees?

- 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.

- **TMVA**: Toolkit for MultiVariate Analysis
- http://tmva.sourceforge.net 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 --glibs` -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):

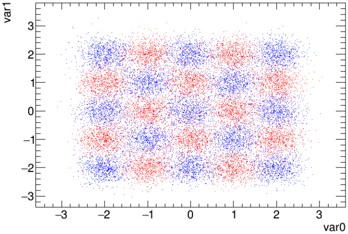
```
#include "TMVA/ROCCalc.h"
TMVA::ROCCalc(TH1* S,TH1* B).GetROCIntegral();
#include "TMVA/Tools.h"
TMVA::gTools().GetSeparation(TH1* S,TH1* B);
```

Training with TMVA (Train.C)

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

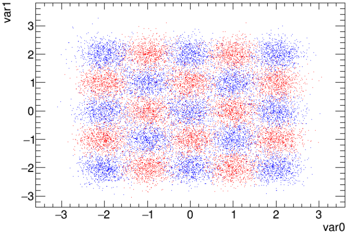
Training with TMVA (Train.C)

```
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);
```



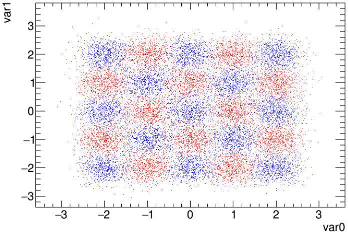
Training with TMVA (Train.C)

```
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 = "";
```



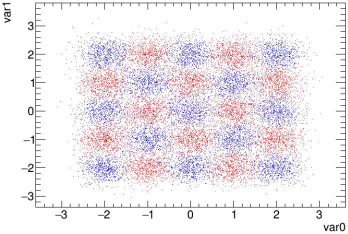
Training with TMVA (Train.C)

```
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");
```



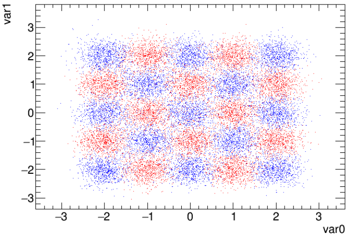
Training with TMVA (Train.C)

```
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");
```



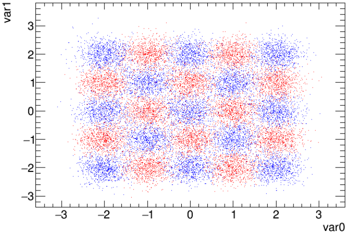
Training with TMVA (Train.C)

```
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
```



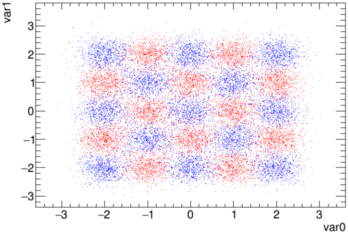
Training with TMVA (Train.C)

```
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
```



Training with TMVA (Train.C)

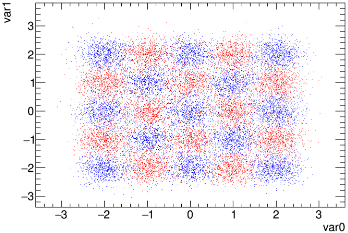
```
TFile* outputFile = TFile::Open("output.root","RECREATE");
TMVA::Factory *factory = new TMVA::Factory( "TMVAClassification", outputFile,
    "!V:Color:DrawProgressBar:Transformations=I:AnalysisType=Classification");
TFile* inputFile = new TFile("dataSchachbrett.root");
TTree* sig = (TTree*)inputFile->Get("TreeS");
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();
```

Training with TMVA (Train.C)

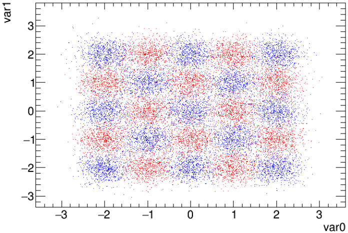
```
TFile* outputFile = TFile::Open("output.root","RECREATE");
TMVA::Factory *factory = new TMVA::Factory( "TMVAClassification", outputFile,
    "!V:Color:DrawProgressBar:Transformations=I:AnalysisType=Classification");
TFile* inputFile = new TFile("dataSchachbrett.root");
TTree* sig = (TTree*)inputFile->Get("TreeS");
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();
outputFile->Close();
delete factory; delete dataloader;
```

Training with TMVA (Train.C)

```
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();
outputFile->Close();
delete factory; delete dataloader;
```




```
TMVA::TMVAGui("output.root");
```

Apply classifier with TMVA (Apply.C)

```
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);
```

Apply classifier with TMVA (Apply.C)

```
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 );
```

Apply classifier with TMVA (Apply.C)

```
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");
```

Apply classifier with TMVA (Apply.C)

```
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();
```

Apply classifier with TMVA (Apply.C)

```
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();
```

- More complete tutorial:
  - https://github.com/lmoneta/tmva-tutorial/tree/IML-tutorial-2018/tutorial_IML2018

Other boosting algorithms

ε-Boost (shrinkage)

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

ε-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 $e^{-y_i T_k(i)}$
- $T(i) = \sum_{k=1}^{N_{\text{tree}}} T_k(i)$

- ε-HingeBoost, LogitBoost, Gentle AdaBoost, GradientBoost, etc.

Other averaging techniques

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

Other averaging techniques

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

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

Other averaging techniques

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

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

Trimming

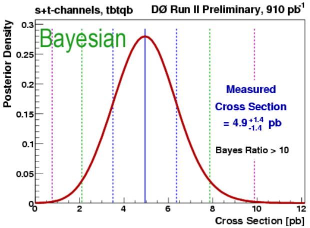
- Not exactly the same. Used to speed up training
- After some boosting, very few high weight events may contribute
- $\Rightarrow$ ignore events with too small a weight

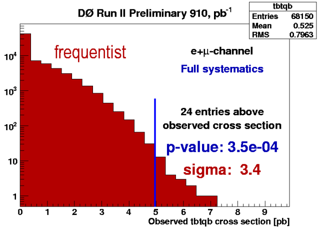
1 Introduction
2 Growing a tree
3 Tree (in)stability
4 Boosting
5 BDT performance
6 Concrete examples
7 Other averaging techniques
8 **BDTs in real physics cases**
9 BDT systematics
10 Software
11 Conclusion
12 References

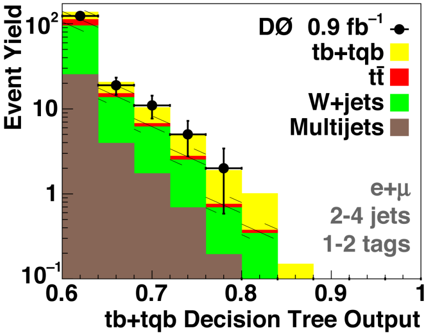
Single top production evidence at D0 (2006)

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

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







$\sigma_s = 1.0 \pm 0.9$ pb
$\sigma_t = 4.2^{+1.8}_{-1.4}$ pb

Phys. Rev. D**78**, 012005 (2008)

Decision trees — 49 input variables

Object Kinematics

p_T(jet1)
p_T(jet2)
p_T(jet3)
p_T(jet4)
p_T(best1)
p_T(notbest1)
p_T(notbest2)
p_T(tag1)
p_T(untag1)
p_T(untag2)

Angular Correlations

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

Event Kinematics

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

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

Decision trees — 49 input variables

Object Kinematics

p_T(jet1)
p_T(jet2)
p_T(jet3)
p_T(jet4)
p_T(best1)
p_T(notbest1)
p_T(notbest2)
p_T(tag1)
p_T(untag1)
p_T(untag2)

Angular Correlations

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

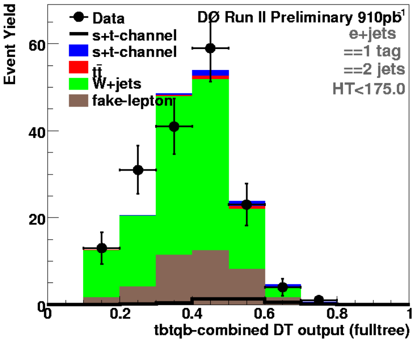
Event Kinematics

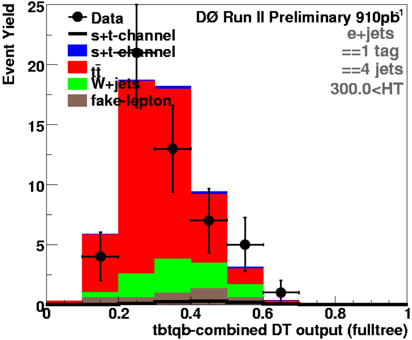
Aplanarity(alljets,W)
M(W,best1) ("best" top mass)
M(W,tag1) ("b-tagged" top mass)
H_T(alljets)
H_T(alljets−best1)
H_T(alljets−tag1)
H_T(alljets,W)
H_T(jet1,jet2)
H_T(jet1,jet2,W)
M(alljets)
M(alljets−best1)
M(alljets−tag1)
M(jet1,jet2)
M(jet1,jet2,W)
M_T(jet1,jet2)
M_T(W)
Missing E_T
p_T(alljets−best1)
p_T(alljets−tag1)
p_T(jet1,jet2)
Q(lepton)$\times\eta$(untag1)
$\sqrt{\hat{s}}$
Sphericity(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(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, $H_T(\text{lepton},\not{E}_T,\text{alljets}) < 175$ GeV
- **"ttbar":** $= 4$ jets, $H_T(\text{lepton},\not{E}_T,\text{alljets}) > 300$ GeV

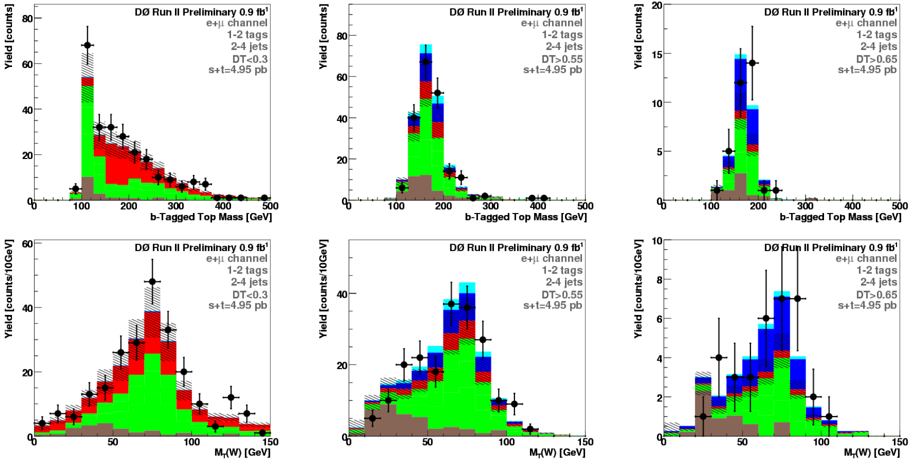




- Good agreement

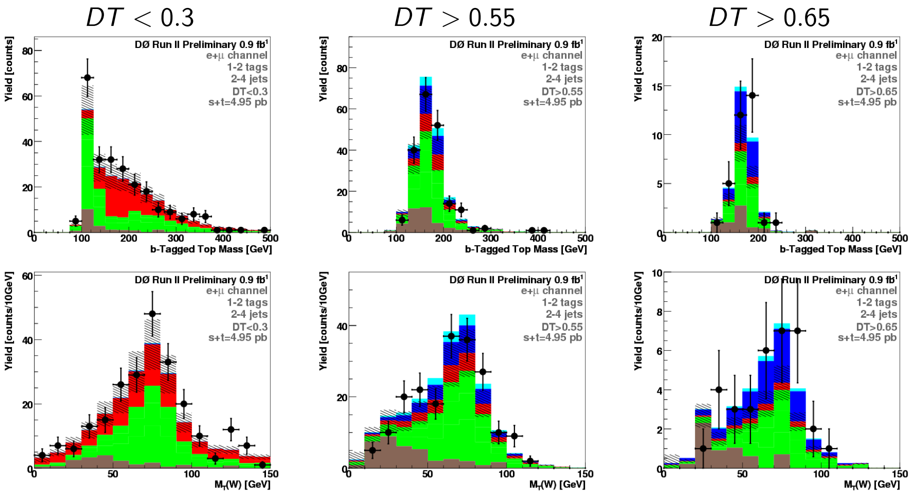
Boosted decision tree event characteristics

$DT < 0.3$ $\quad$ $DT > 0.55$ $\quad$ $DT > 0.65$



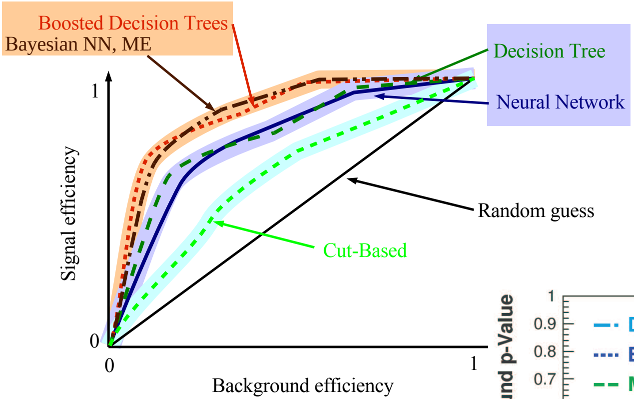
- High BDT region = shows masses of real t and W $\Rightarrow$ expected
- Low BDT region = background-like $\Rightarrow$ expected

Boosted decision tree event characteristics



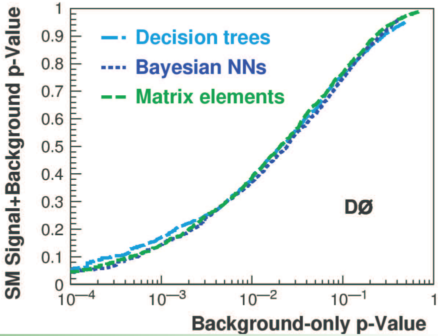
- High BDT region = shows masses of real t and W ⇒ expected
- Low BDT region = background-like ⇒ expected
- Above does NOT tell analysis is ok, but not seeing this could be a sign of a problem

Comparison for D0 single top evidence



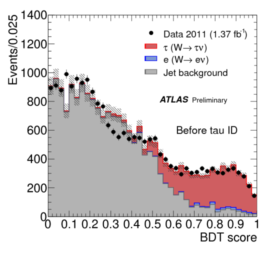
- Cannot know *a priori* which method will work best
- ⇒ Need to experiment with different techniques

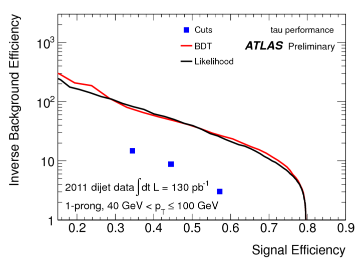
Power curve



ATLAS tau identification

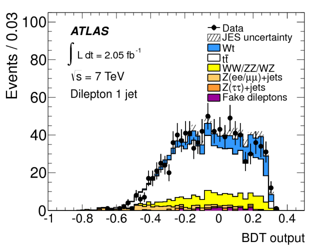
- Now used both offline and online
- Systematics: propagate various detector/theory effects to BDT output and measure variation





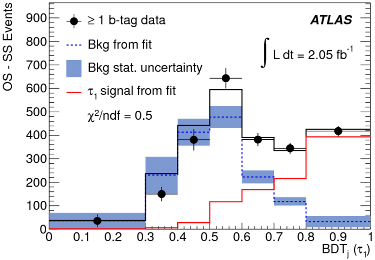
ATLAS Wt production evidence

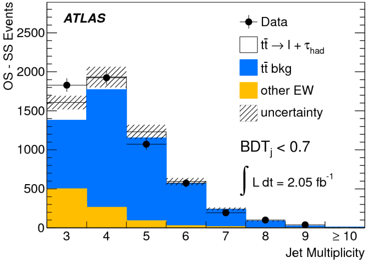
- Phys.Lett. B716 (2012) 142-159
- BDT output used in final fit to measure cross section
- Constraints on systematics from profiling

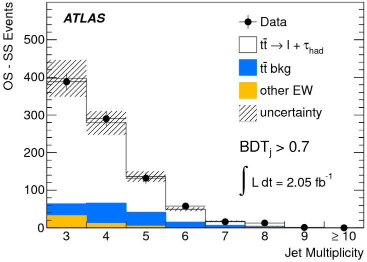


Phys.Lett. B717 (2012) 89-108

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





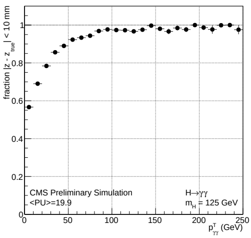


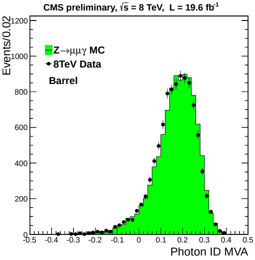
BDT in HEP: CMS $H \to \gamma\gamma$ result

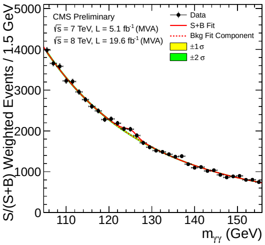
CMS-PAS-HIG-13-001

Hard to use more BDT in an analysis:

- vertex selected with BDT
- 2^{nd} 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



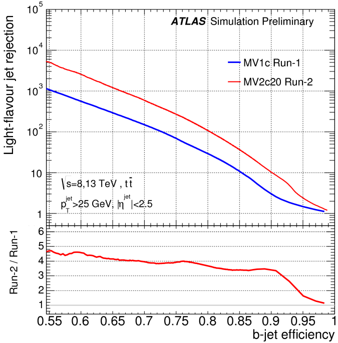


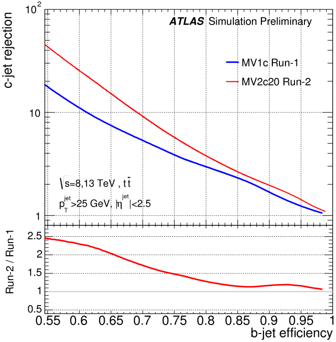


BDT in HEP: ATLAS b-tagging in Run 2

ATL-PHYS-PUB-2015-022

- 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, η of jets
- Run 2: introduced IBL (new innermost pixel layer)
  $\Rightarrow$ explains part of the performance gain, but not all

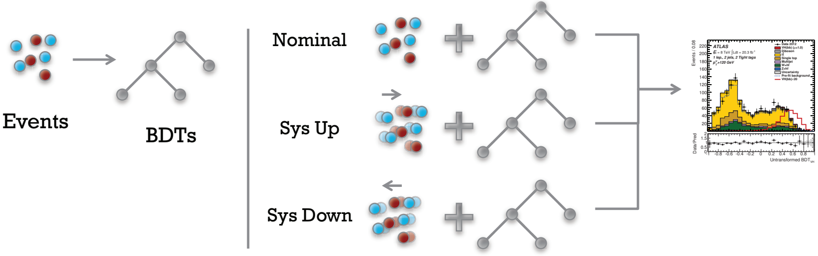




BDT and systematics

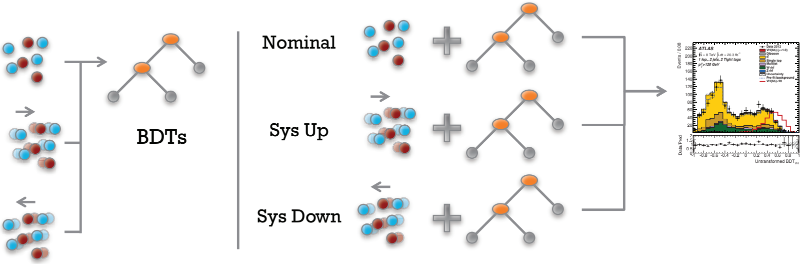
- No particular rule
- BDT output can be considered as any other cut variable (just more powerful). Evaluate systematics by:
  - varying cut value
  - retraining
  - calibrating, etc.
- Most common (and appropriate, I think): propagate other uncertainties (detector, theory, etc.) up to BDT ouput and 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
    $\Rightarrow$ 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)

BDT and systematics



S. Hageböck

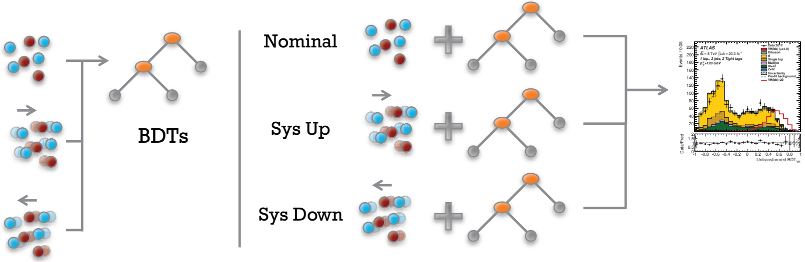
BDT and systematics



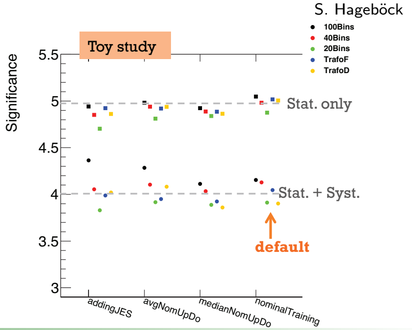
S. Hageböck

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

BDT and systematics



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



Boosted decision tree software

- Go for a fully integrated solution
  - use different multivariate techniques easily
  - spend your time on understanding your data and model
- Examples:
  - Weka. Written in Java, open source, very good published manual. Not written for HEP but very complete ▸ http://www.cs.waikato.ac.nz/ml/weka/
  - StatPatternRecognition ▸ http://statpatrec.sourceforge.net/
  - *T*MVA (Toolkit for MultiVariate Analysis) Integrated in ROOT, complete manual ▸ http://tmva.sourceforge.net
  - scikit-learn (python) [see G. Louppe's tutorial] ▸ http://scikit-learn.org
  - pylearn2 (python) ▸ https://github.com/lisa-lab/pylearn2
- Dedicated to BDT: XGBoost ▸ arXiv:1603.02754 ▸ https://github.com/dmlc/xgboost

Conclusion

- Decision trees have been around for some time in social sciences
- Natural extension to cut-based analysis
- Greatly improved performance with boosting (and also with bagging, random forests)
- Has become rather fashionable in HEP
- 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 people that your advanced technique leads to meaningful and reliable results
  $\Rightarrow$ 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, please!
- As with other advanced techniques,
  no point in using them if data are not understood and well modelled

References I

- L. Breiman, J.H. Friedman, R.A. Olshen and C.J. Stone, *Classification and Regression Trees*, Wadsworth, Stamford, 1984
- J.R. Quinlan, "Induction of decision trees", *Machine Learning*, 1(1):81–106, 1986
- J.R. Quinlan, "Simplifying decision trees", *International Journal of Man-Machine Studies*, 27(3):221–234, 1987
- 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)*

References II

- 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
- 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

Splitting a node: Gini index of diversity

Defined for many classes

- 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 $p_s = p = 1 - p_b$
- $\Rightarrow$ 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

Pruning a tree I

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"

Pruning a tree II: cost-complexity pruning

- 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 α: pick T_{max}
  - large α: 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

Pruning a tree III: cost-complexity pruning

- 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 ρ_t, solve $R_{\rho_t}(T_t) = R_{\rho_t}(t)$, or: $\rho_t = \dfrac{R(t) - R(T_t)}{N_T - 1}$
  - node with smallest ρ_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