EPJ Web of Conferences **4**, 02003 (2010)
DOI:10.1051/epjconf/20100402003

Decision trees

Yann COADOU
CERN, Genève

1 Introduction

Decision trees are a machine learning technique not (yet) commonly used in high energy physics, although it has been widely used in the social sciences. It was first developed in the context of data mining and pattern recognition, and gained momentum in various fields, including medical diagnostic, insurance and loan screening, and optical character recognition (OCR) of handwritten text.

It was developed and formalised by Breiman *et al.* [1] who proposed the CART algorithm (Classification And Regression Trees) with a complete and functional implementation of decision trees.

The basic principle is rather simple : it consists in extending a simple cut-based analysis into a multivariate technique by continuing to analyse events that fail a particular criterion. Many, if not most, events do not have all characteristics of either signal or background. If that were the case then an analysis with a few criteria would allow to easily extract the signal. The concept of a decision tree is therefore to not reject right away events that fail a criterion, and instead check whether other criteria may help to classify these events properly.

In principle a decision tree can deal with multiple output classes, each branch splitting in many subbranches. In these proceedings only binary trees will be considered, with only two possible classes : signal and background.

Section 2 describes how a decision tree is constructed and what parameters can influence its development. Section 3 provides some insights into some of the intrinsic limitations of a decision tree and how to address some of them. One possible extension of decision trees, boosting, is introduced in Section 4, and other techniques trying to reach the same goal as boosting are presented in Section 5. Conclusions are summarised in Section 6 and references to available decision tree software are given in Section 7.

While starting with this powerful multivariate technique, it is important to remember that before applying it to real data, it is crucial to have a good understanding of the data and of the model used to describe them. Any discrepancy between the data and model will provide an artificial separation that the decision trees will use, misleading the analyser. The hard part (and interest) of the analysis is in building the proper model, not in extracting the signal. But once this is properly done, decision trees provide a very powerful tool to increase the significance of any analysis.

2 Growing a tree

Mathematically, decision trees are rooted binary trees (as only trees with two classes, signal and background, are considered). An example is shown in Fig. 1. A decision tree starts from an initial node, the root node. Each node can be recursively split into two daughters or branches, until some stopping condition is reached. The different aspects of the process leading to a full tree, indifferently referred to as growing, training, building or learning, are described in the following sections.

2.1 Algorithm

Consider a sample of signal (s_i) and background (b_j) events, each with weight w_i^s and w_j^b respectively, described by a set $\vec{x}_i$ of variables. This sample constitutes the root node of a new decision tree.

Starting from this root node, the algorithm proceeds as follows :

1. If the node satisfies any stopping criterion, declare it as terminal (that is, a leaf) and exit the algorithm.
2. Sort all events according to each variable in $\vec{x}$
3. For each variable, find the splitting value that gives the best separation between two children, one with mostly signal events, the other with mostly background events (see Section 2.4 for details). If the separation cannot be improved by any splitting, turn the node into a leaf and exit the algorithm.
4. Select the variable and splitting value leading to the best separation and split the node in two new nodes (branches), one containing events that fail the criterion and one with events that satisfy it.
5. Apply recursively from step 1 on each node

At each node, all variables can be considered, even if they have been used in a previous iteration : this allows to find intervals of interest in a particular variable, instead of limiting oneself to using each variable only once.

It should be noted that a decision tree is human readable : one can trace exactly which criteria an event satisfied in order to reach a particular leaf. It is therefore possible to interpret a tree in terms, e.g., of physics, defining selection rules, rather than only as a mathematical object.

In order to make the whole procedure clearer, let us take the tree in Fig. 1 as an example. Consider that all events are described by three variables : the transverse momentum, p_T, of the leading jet, the reconstructed top quark mass M_t and the scalar sum of p_T's of all reconstructed objects in the event, H_T. All signal and background events make up the root node.

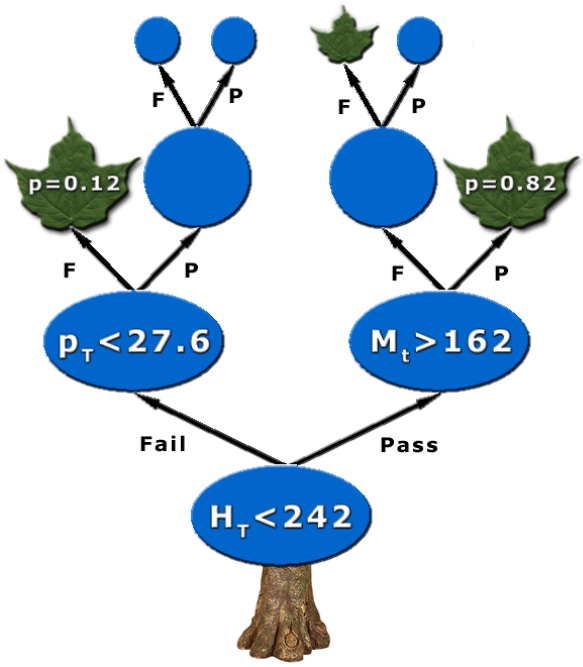

Fig. 1: Graphical representation of a decision tree. Blue ellipses and disks are internal nodes with their associated splitting criterion ; green leaves are terminal nodes with purity p.

Following the above algorithm one should first sort all events according to each variable :
– $p_T^{s_1} \leq p_T^{b_{34}} \leq \cdots \leq p_T^{b_2} \leq p_T^{s_{12}}$,
– $H_T^{b_5} \leq H_T^{b_3} \leq \cdots \leq H_T^{s_{67}} \leq H_T^{s_{43}}$,
– $M_t^{b_6} \leq M_t^{s_8} \leq \cdots \leq M_t^{s_{12}} \leq M_t^{b_9}$,
where superscript s_i (b_j) represents signal (background) event i (j). Using some measure of separation one may find that the best splitting for each variable is (arbitrary unit) :
– $p_T < 56$ GeV, separation $= 3$,
– $H_T < 242$ GeV, separation $= 5$,
– $M_t < 105$ GeV, separation $= 0.7$.

One would conclude that the best split is to use $H_T < 242$ GeV, and would create two new nodes, the left one with events failing this criterion and the right one with events satisfying it. One can now apply

the same algorithm recursively on each of these new nodes. As an example consider the right-hand-side node with events that satisfied $H_T < 242$ GeV. After sorting again all events in this node according to each of the three variables, it was found that the best criterion was $M_t > 162$ GeV, and events were split accordingly into two new nodes. This time the right-hand-side node satisfied one of the stopping conditions and was turned into a leaf. From signal and background training events in this leaf, the purity was computed as $p = 0.82$ (see the next Section).

2.2 Decision tree output

The decision tree output for a particular event i is defined by how its $\vec{x}_i$ variables behave in the tree :

1. Starting from the root node, apply the first criterion on $\vec{x}_i$.
2. Move to the passing or failing branch depending on the result of the test.
3. Apply the test associated to this node and move left or right in the tree depending on the result of the test.
4. Repeat step 3 until the event ends up in a leaf.
5. The decision tree output for event i is the value associated with this leaf.

There are several conventions used for the value attached to a leaf. It can be the purity $p = \frac{s}{s+b}$ where s (b) is the sum of weights of signal (background) events that ended up in this leaf during training. It is then bound to $[0, 1]$, close to 1 for signal and close to 0 for background.

It can also be a binary answer, signal or background (mathematically typically $+1$ for signal and 0 or -1 for background) depending on whether the purity is above or below a specified critical value (e.g. $+1$ if $p > \frac{1}{2}$ and -1 otherwise).

Looking again at the tree in Fig. 1, the leaf with purity $p = 0.82$ would give an output of 0.82, or $+1$ as signal if choosing a binary answer with a critical purity of 0.5.

2.3 Tree parameters

The number of parameters of a decision tree is relatively limited. The first one is not specific to decision trees and applies to most techniques requiring training : how to normalise signal and background before starting the training ? Conventionally the sums of weights of signal and background events are chosen to be equal, giving the root node a purity of 0.5, that is, an equal mix of signal and background.

Other parameters concern the selection of splits. One first needs a list of questions to ask, like "is variable $x_i < \text{cut}_i$?", requiring a list of discriminating variables and a way to evaluate the best separation between signal and background events (the goodness of the split). Both aspects are described in more detail in Sections 2.4 and 2.5.

The splitting has to stop at some point, declaring such nodes as terminal leaves. Conditions to satisfy can include :

- a minimum leaf size. A simple way is to require at least N_{min} training events in each node after splitting, in order to ensure statistical significance of the purity measurement, with a statistical uncertainty $\sqrt{N_{min}}$. It becomes a little bit more complicated with weighted events, as is normally the case in high energy physics applications. One may then want to consider using the effective number of events instead :

$$N_{\mathit{eff}} = \frac{\left(\sum_{i=1}^{N} w_i\right)^2}{\sum_{i=1}^{N} w_i^2},$$

  for a node with N events associated to weights w_i ($N_{\mathit{eff}} = N$ for unweighted events). This would ensure a proper statistical uncertainty.
- having reached perfect separation (all events in the node belong to the same class).
- an insufficient improvement with further splitting.
- a maximal tree depth. One can decide that a tree cannot have more than a certain number of layers (for purely computational reasons or to have like-size trees).

Finally a terminal leaf has to be assigned to a class. This is classically done by labelling the leaf as signal if $p > 0.5$ and background otherwise.

2.4 Splitting a node

The core of a decision tree algorithm resides in how a node is split into two. Consider an impurity measure $i(t)$ for node t, which describes to what extent the node is a mix of signal and background. Desirable features of such a function are that it should be :

- maximal for an equal mix of signal and background (no separation).
- minimal for nodes with either only signal or only background events (perfect separation).
- symmetric in signal and background purities, as isolating background is as valuable as isolating signal.
- strictly concave in order to reward purer nodes. This tends to favour end cuts with one smaller node and one larger node.

A figure of merit can be constructed with this impurity function, as the decrease of impurity for a split S of node t into two children t_P (pass) and t_F (fail) :

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

where p_P (p_F) is the fraction of events that passed (failed) split S.

The goal is to find the split S^* that maximises the decrease of impurity :

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

It will result in the smallest residual impurity, which minimises the overall tree impurity.

Some decision tree applications use an alternative definition of the decrease of impurity [2] :

$$\Delta i(S,t) = i(t) - \min\big(p_P \cdot i(t_P), p_F \cdot i(t_F)\big),$$

which may perform better for, e.g., particle identification.

A stopping condition can be defined using the decrease of impurity : one may decide to not split a node if $\Delta i(S^*,t)$ is less than some predefined value. One should nonetheless always be careful when using such early-stopping criteria, as sometimes a seemingly very weak split may allow child nodes to be powerfully split further.

For signal (background) events with weights w_s^i (w_b^i) the purity is defined as :

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

Simplifying this expression one can write down the signal purity (or simply purity) as $p_s = p = \frac{s}{s+b}$ and the background purity as $p_b = \frac{b}{s+b} = 1 - p_s = 1 - p$. Common impurity functions (exhibiting most of the desired features mentioned previously) are illustrated in Fig. 2 :

- the misclassification error : $1 - \max(p, 1-p)$,
- the (cross) entropy [1] : $-\sum_{i=s,b} p_i \log p_i$,
- the Gini index of diversity.

For a problem with any number of classes, the Gini index [3] is defined as :

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

The statistical interpretation is that if one assigns a random object to class i with probability p_i, the probability that it is actually in class j is p_j and the Gini index is the probability of misclassification.

In the case of two classes, signal and background, then $i = s$ and $j = b$, $p_s = p = 1 - p_b$ and the Gini index is :

$$\text{Gini} = 1 - \sum_{i=s,b} p_i^2 = 2p(1-p) = \frac{2sb}{(s+b)^2}.$$

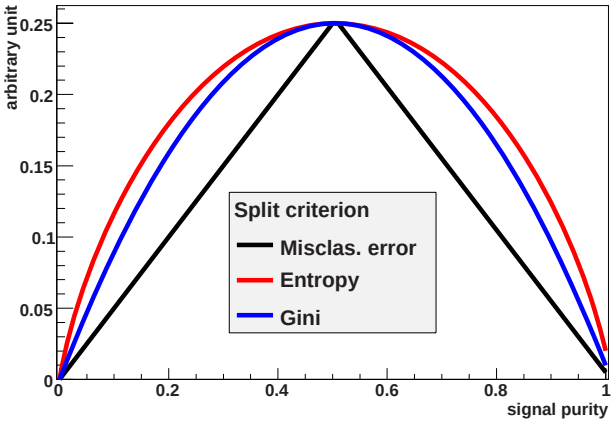

Fig. 2: Various popular impurity measures as a function of signal purity.

The Gini index is the most popular in decision tree implementations. It typically leads to similar performance to entropy.

Other measures are also used sometimes, which do not satisfy all criteria listed previously but attempt at optimising signal significance, usually relevant in high energy physics applications :

- cross section significance : $-\frac{s^2}{s+b}$,
- excess significance : $-\frac{s^2}{b}$.

2.5 Variable selection

As stated before, the data and model have to be in good agreement before starting to use a decision tree (or any other analysis technique). This means that all variables used should be well described. Forgetting this prerequisite will jeopardise the analysis.

Overall decision trees are very resilient to most features associated to variables. They are not too much affected by the "curse of dimensionality", which forbids the use of too many variables in most multivariate techniques. For decision trees the CPU consumption scales as $nN \log N$ with n variables and N training events. It is not uncommon to encounter decision trees using tens or hundreds of variables.

A decision tree is immune to duplicate variables : the sorting of events according to each of them would be identical, leading to the exact same tree. The order in which variables are presented is completely irrelevant : all variables are treated equal. The order of events in the training samples is also irrelevant.

If variables are not discriminating, they will simply be ignored and will not add any noise to the decision tree. The final performance will not be affected, it will only come with some CPU overhead during both training and evaluation.

Decision trees can deal easily with both continuous and discrete variables, simultaneously.

Another typical task before applying a multivariate technique is to transform input variables by for instance making them fit in the same range or taking the logarithm to regularise the variable. This is totally unnecessary with decision trees, which are completely insensitive to the replacement of any subset of input variables by (possibly different) arbitrary strictly monotone functions of them. The explanation is trivial. Let $f : x_i \rightarrow f(x_i)$ be a strictly monotone function : if $x > y$ then $f(x) > f(y)$. Then any ordering of events by x_i is the same as by $f(x_i)$, which means that any split on x_i will create the same separation as a split on $f(x_i)$, producing the same decision tree. This means that decision trees have some immunity against outliers.

Until now the splits considered have always answered questions of the form "Is $x_i < c_i$?", while it is also possible to make linear combinations of input variables and ask instead "Is $\sum_i a_i x_i < c_i$?", where $a = (a_1, .., a_n)$ is a set of coefficients such that $||a||^2 = \sum_i a_i^2 = 1$. One would then choose the optimal split $S^*(a*)$ and the set of linear coefficients a^* that maximises $\Delta i(S(a), t)$. This is in practise rather tricky to implement and very CPU intensive. This approach is also powerful only if strong linear correlations exist between variables. If this is the case, a simpler approach would consist in first decorrelating the input variables and then feeding them to the decision tree. Even if not doing this decorrelation, a decision tree will find the correlations but in a very suboptimal way, by successive approximations, adding complexity to the tree structure.

It is possible to rank variables in a decision tree. To rank variable x_i one can add up the decrease of impurity for each node where variable x_i was used to split. The variable with the largest decrease of impurity is the best variable.

There is nevertheless a shortcoming with variable ranking in a decision tree : variable masking. Variable x_j may be just a little worse than variable x_i and would end up never being picked in the decision tree growing process. Variable x_j would then be ranked as irrelevant, and one would conclude this variable has no discriminating power. But if one would remove x_i, then x_j would become very relevant.

There is a solution to this feature, called surrogate splits [1]. For each split, one compares which training events pass or fail the optimal split to which events pass or fail a split on another variable. The split that mimics best the optimal split is called the surrogate split. One can then take this into consideration when ranking variables. This has applications in case of missing data : one can then replace the optimal split by the surrogate split.

3 Tree (in)stability

Despite all the nice features presented above, decision trees are known to be relatively unstable. If trees are too optimised for the training sample, they may not generalise very well to unknown events. This can be mitigated with pruning, described in Section 3.1.

A small change in the training sample can lead to drastically different tree structures, rendering the physics interpretation a bit less straightforward. For sufficiently large training samples, the performance of these different trees will be equivalent, but on small training samples variations can be very large. This doesn't give too much confidence in the result.

Moreover a decision tree output is by nature discrete, limited by the purities of all leaves in the tree. This means that to decrease the discontinuities one has to increase the tree size and complexity, which may not be desirable or even possible. Then the tendency is to have spikes in the output distribution at specific purity values, as illustrated in Fig. 3, or even two delta functions at ± 1 if using a binary answer rather than the purity output.

Section 3.2 describes how these shortcomings can for the most part be addressed by averaging.

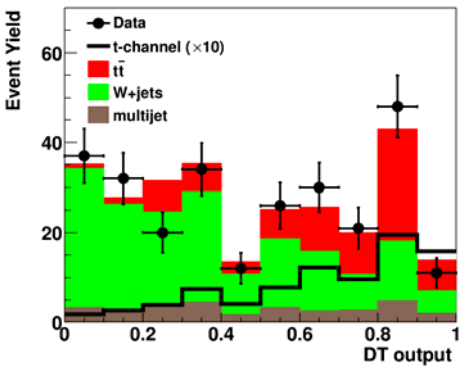

Fig. 3: Comparison of signal, backgrounds and data for one of the decision tree outputs in an old D0 single top quark search. The discrete output of a decision tree is clearly visible, but data are well reproduced by the model.

3.1 Pruning a tree

When growing a tree, each node contains fewer and fewer events, leading to an increase of the statistical uncertainty on each new split. The tree will therefore tend to become more and more specialised, focusing on properties of the training sample that may not reflect the expected result, had there been infinite statistics to train on.

A first approach to mitigate this effect, sometimes referred to as pre-pruning, has already been described in Section 2, using stopping conditions. They included requiring a minimum number of events in each node or a minimum amount of separation improvement. The limitation is that requiring too big a minimum leaf size or too much improvement may prevent further splitting that could be very beneficial.

Another approach consists in building a very large tree and then cutting irrelevant branches by turning

an internal node and all its descendants into a leaf, removing the corresponding subtree. This is post-pruning, or simply pruning.

Why would one wish to prune a decision tree ? It is possible to get a perfect classifier on the training sample : mathematically the misclassification rate can be made as little as one wants on training events. For instance one can build a tree such that each leaf contains only one class of events (down to one event per leaf if necessary). The training error is then zero. But when passing events through the tree that were not seen during training, the misclassification rate will most likely be non-zero, showing signs of overtraining. Pruning helps in avoiding such effects, by eliminating subtrees (branches) that are deemed too specific to the training sample.

There are many different pruning algorithms available. Only three of them, among the most commonly used, are presented briefly.

Expected error pruning was introduced by Quinlan [4] in the ID3 algorithm. A full tree is first grown. One then computes the approximate expected error for a node (using the Laplace error estimate), and compares it to the weighted sum of expected errors from its children. If the expected error of the node is less than that of the children, then the node is pruned. This algorithm is fast and does not require a separate sample of events to do the pruning, but it is also known to be too aggressive : it tends to prune large subtrees containing branches with good separation power.

Reduced error pruning was also introduced by Quinlan [4], and requires a separate pruning sample. Starting from terminal leaves, the misclassification rate on the pruning sample for the full tree is compared to the misclassification rate when a node is turned into a leaf. If the simplified tree has better performance, the subtree is pruned. This operation is repeated until further pruning increases the misclassification rate.

Cost-complexity pruning is part of the CART algorithm [1]. The idea is to penalise complex trees (with many nodes and/or leaves) and to find a compromise between a good fit to training data (requiring a larger tree) and good generalisation properties (easier achieved with a smaller tree).

Consider a fully grown decision tree, T_{max}. For any subtree T (with N_T nodes) of T_{max} with a misclassification rate $R(T)$ one can define the cost complexity $R_\alpha(T)$:

$$R_\alpha(T) = R(T) + \alpha N_T,$$

where α is a complexity parameter. When trying to minimise $R_\alpha(T)$, a small α would help picking T_{max} (no cost for complexity) while a large α would keep only the root node, T_{max} being fully pruned. The optimally pruned tree is somewhere in the middle.

In a first pass, for terminal nodes t_P and t_F emerging from the split of node t, by construction $R(t) \geq R(t_P) + R(t_F)$. If these quantities are equal, then one can prune off t_P and t_F.

Now, for node t and subtree T_t, by construction $R(t) > R(T_t)$ if t is non-terminal. The cost complexity for node t is :

$$R_\alpha(\{t\}) = R_\alpha(t) = R(t) + \alpha$$

since $N_t = 1$. If $R_\alpha(T_t) < R_\alpha(t)$, then the branch T_t has smaller cost-complexity than the single node $\{t\}$ and should be kept. But for a critical value $\alpha = \rho_t$, obtained by solving $R_{\rho_t}(T_t) = R_{\rho_t}(t)$, that is :

$$\rho_t = \frac{R(t) - R(T_t)}{N_T - 1},$$

pruning the tree and making t a leaf becomes preferable. The node with the smallest ρ_t is the weakest link and gets pruned. The algorithm is applied recursively on the pruned tree until it is completely pruned, leaving only the root node.

This generates a sequence of decreasing cost-complexity subtrees. For each of them (from T_{max} to the root node), one then computes their misclassification rate on the validation sample. It will first decrease, and then go through a minimum before increasing again. The optimally pruned tree is the one corresponding to the minimum.

3.2 Averaging several trees

Pruning was shown to be helpful in maximising the generalisation potential of a single decision tree. It nevertheless doesn't address other shortcomings of trees like the discrete output or the sensitivity of the tree structure to the training sample composition. A way out is to proceed with averaging several trees, with the added potential bonus that the discriminating power may increase, as briefly described elsewhere in these proceedings [5].

Such a principle was introduced from the beginning with the so-called V-fold cross-validation [1], a useful technique for small samples. After dividing a training sample $\mathcal{L}$ into V subsets of equal size, $\mathcal{L} = \bigcup_{v=1..V} \mathcal{L}_v$, one can train a tree T_v on the $\mathcal{L} - \mathcal{L}_v$ sample and test it on $\mathcal{L}_v$. This produces V decision trees, whose outputs are combined into a final discriminant :

$$\frac{1}{V} \sum_{v=1..V} T_v.$$

Following this simple-minded approach, many other averaging techniques have been developed, after realising the enormous advantage it provided. Bagging, boosting and random forests are such techniques and will be described in the following Section.

4 Boosting

As will be shown in this section, the boosting algorithm has turned into a very successful way of improving the performance of any type of classifier, not only decision trees. After a short history of boosting in Section 4.1, the generic algorithm is presented in Section 4.2 and a specific implementation (AdaBoost) is described in Section 4.3. Boosting is illustrated with a few examples in Section 4.4. Finally other examples of boosting implementations are given in Section 4.5.

4.1 A brief history of boosting

Boosting has appeared quite recently. The first provable algorithm of boosting was proposed by Schapire in 1990 [6]. It worked in the following way :

- train a classifier T_1 on a sample of N events.
- train T_2 on a new sample with N events, half of which were misclassified by T_1.
- build T_3 on events where T_1 and T_2 disagree.

The boosted classifier was defined as a majority vote on the outputs of T_1, T_2 and T_3.

In 1995 Freund followed up on this idea [7], introducing boosting by majority. It consisted in combining many learners with a fixed error rate. This was an impractical prerequisite for a viable automated algorithm, but was a stepping stone to the proposal by Freund&Schapire of the first functional boosting algorithm, called AdaBoost [8].

Boosting, and in particular boosted decision trees, have become increasingly popular in high energy physics. The MiniBooNe experiment first compared the performance of different boosting algorithms and artificial neural networks for particle identification [9]. The D0 experiment was the first to use boosted decision trees for a search, which lead to the first evidence (and now observation) of single top quark production [10].

4.2 Boosting algorithm

Boosting is a general technique which is not limited to decision trees, although it is often used with them. It can apply to any classifier, e.g., neural networks. It is hard to make a very good discriminant, but is it relatively easy to make simple ones which are certainly more error-prone but are still performing at least marginally better than random guessing. Such discriminants are called weak classifiers [5]. The goal of boosting is to combine such weak classifiers into a new, more stable one, with a smaller error rate and better performance.

Consider a training sample T_k containing N events. The i^{th} event is associated with a weight w_i^k, a vector of discriminating variables $\vec{x}_i$ and a class label $y_i = +1$ for signal, -1 for background. The pseudocode for a generic boosting algorithm is :

```
Initialise T_1
for k in 1..N_tree
  train classifier T_k on T_k
  assign weight α_k to T_k
  modify T_k into T_{k+1}
```

The boosted output is some function $F(T_1, .., T_{N_{tree}})$, typically a weighted average :

$$F(i) = \sum_{k=1}^{N_{tree}} \alpha_k T_k(\vec{x}_i).$$

Thanks to this averaging, the output becomes quasi-continuous, mitigating one of the limitations of single decision trees.

4.3 AdaBoost

One particularly successful implementation of the boosting algorithm is AdaBoost, introduced by Freund&Schapire [8]. AdaBoost stands for adaptive boosting, referring to the fact that the learning procedure adjusts itself to the training data in order to classify it better. There are many variations on the same theme for the actual implementation, and it is the most common boosting algorithm. It typically leads to better results than without boosting, up to the Bayes limit as will be seen later.

An actual implementation of the AdaBoost algorithm works as follows. After having built tree T_k one should check which events in the training sample T_k are misclassified by T_k, hence defining the misclassification rate $R(T_k)$. In order to ease the math, let us introduce some notations. Define $\mathsf{I} : X \to \mathsf{I}(X)$ such that $\mathsf{I}(X) = 1$ if X is true, and 0 otherwise. One can now define a function that tells whether an event is misclassified by T_k. In the decision tree output convention of returning only $\{\pm 1\}$ it gives :

$$\text{isMisclassified}_k(i) = \mathsf{I}\big(y_i \times T_k(i) \leq 0\big),$$

while in the purity output convention (with a critical purity of 0.5) it leads to :

$$\text{isMisclassified}_k(i) = \mathsf{I}\big(y_i \times (T_k(i) - 0.5) \leq 0\big).$$

The misclassification rate is now :

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

This misclassification rate can be used to derive a weight associated to tree T_k :

$$\alpha_k = \beta \times \ln \frac{1 - \epsilon_k}{\epsilon_k},$$

where β is a free boosting parameter to adjust the strength of boosting (it was set to 1 in the original algorithm).

The core of the AdaBoost algorithm resides in the following step : each event in T_k has its weight changed in order to create a new sample T_{k+1} such that :

$$w_i^k \to w_i^{k+1} = w_i^k \times e^{\alpha_k \cdot \text{isMisclassified}_k(i)}.$$

This means that properly classified events are unchanged from T_k to T_{k+1}, while misclassified events see their weight increased by a factor e^{α_k}. The next tree T_{k+1} is then trained on the T_{k+1} sample. This next tree will therefore see a different sample composition with more weight on misclassified events, and will therefore try harder to classify properly difficult events that tree T_k failed to identify correctly. The

final AdaBoost result for event i is :

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

As an example, assume for simplicity the case $\beta = 1$. A not-so-good classifier, with a misclassification rate $\epsilon = 40\%$ would have a corresponding $\alpha = \ln \frac{1-0.4}{0.4} = 0.4$. All misclassified events would therefore get their weight multiplied by $e^{0.4} = 1.5$, and the next tree will have to work a bit harder on these events. Now consider a good classifier with an error rate $\epsilon = 5\%$ and $\alpha = \ln \frac{1-0.05}{0.05} = 2.9$. Now misclassified events get a boost of $e^{2.9} = 19$ and will contribute decisively to the structure of the next tree ! This shows that being failed by a good classifier brings a big penalty : it must be a difficult case, so the next tree will have to pay much more attention to this event and try to get it right.

It can be shown [11] that the misclassification rate ϵ of the boosted result on the training sample is bounded from above :

$$\epsilon \leq \prod_{k=1}^{N_{tree}} 2\sqrt{\epsilon_k(1-\epsilon_k)}.$$

If each tree has $\epsilon_k \neq 0.5$, that is to say, if it does better than random guessing, then the conclusion is quite remarkable : the error rate falls to zero for a sufficiently large N_{tree} ! A corollary is that the training data is overfitted.

Overtraining is usually regarded has a negative feature. Does this mean that boosted decision trees are doomed because they are too powerful on the training sample ? Not really. What matters most is not the error rate on the training sample, but rather the error rate on a testing sample. This may well decrease at first, reach a minimum and increase again as the number of trees increases. In such a case one should stop boosting when this minimum is reached. It has been observed that quite often boosted decision trees do not go through such a minimum, but rather tend towards a plateau in testing error. One could then decide to stop boosting after having reached this plateau.

In a typical high energy physics problem, the error rate may not even be what one wants to optimise. A good figure of merit on the testing sample would rather be the significance. Figure 4 (top left) illustrates this behaviour, showing how the significance saturates with an increasing number of boosting cycles. One could argue one should stop before the end and save resources, but at least the performance does not deteriorate with increasing boosting.

Another typical curve one tries to optimise is the signal efficiency versus the background efficiency. Figure 4 (top right) clearly exemplifies the interesting property of boosted decision trees. The performance is clearly better on the training sample than on the testing sample (the training curves are getting very close to the upper left corner of perfect separation), with a single tree or with boosting, a clear sign of overtraining. But the boosted tree is still performing better than the single tree on the testing sample, proof that it does not suffer from this overtraining.

People have been wondering why boosting leads to such features, with typically no loss of generalisation performance due to overtraining. No clear explanation has emerged yet, but some ideas have come up. It may have to do with the fact that during the boosting sequence, the first tree is the best while the others are successive minor corrections, which are given smaller weights. This is shown at the bottom of Fig. 4, where the misclassification rate of each new tree separately is actually increasing, while the corresponding tree weight is decreasing. This is no surprise : during boosting the successive trees are specialising on specific event categories, and can therefore not perform as well on other events. So the trees that lead to a perfect fit of the training data are contributing very little to the final decision tree output on the testing sample. When boosting decision trees, the last tree is not an evolution of the first one that performs better, quite the contrary. The first tree is typically the best, while others bring dedicated help for misclassified events. The power of boosting does not rely in the last tree in the sequence, but rather in combining a suite of trees that focus on different events.

Finally a probabilistic interpretation of AdaBoost was proposed [12] which gives some insight into the

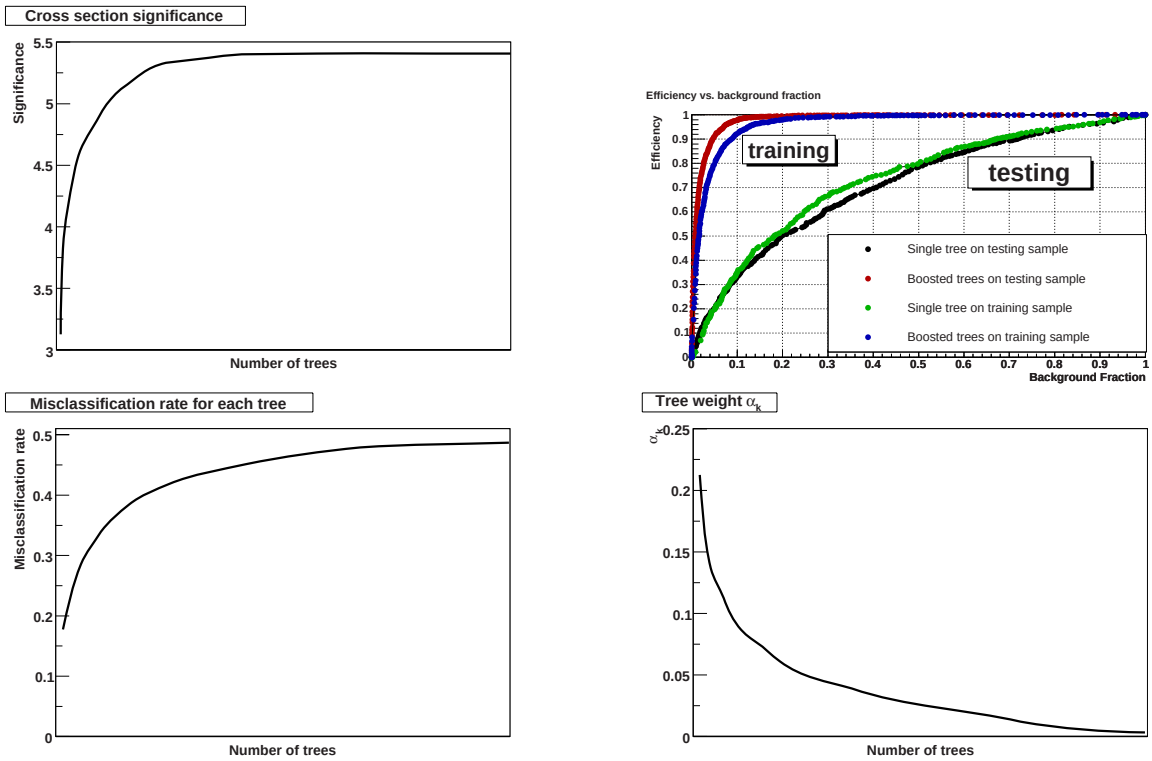

Fig. 4: Behaviour of boosting. Top left : Significance as a function of the number of boosted trees. Top right : Signal efficiency vs. background efficiency for single and boosted decision trees, on the training and testing samples. Bottom left : Misclassification rate of each tree as a function of the number of boosted trees. Bottom right : Weight of each tree as a function of the number of boosted trees.

performance of boosted decision trees. It can be shown that for a boosted output T flexible enough :

$$e^{T(i)} = \frac{p(S|i)}{p(B|i)} = BD(i),$$

where one recognises the Bayes discriminant [13]. This means that the AdaBoost algorithm will tend towards the Bayes limit, the maximal separation one can hope to reach.

4.4 Boosting practical examples

The examples of this section were produced using the TMVA package [14] and starting from code provided by G. Cowan [15]. Consider a system described by two variables, x and y, as shown in Fig. 5.

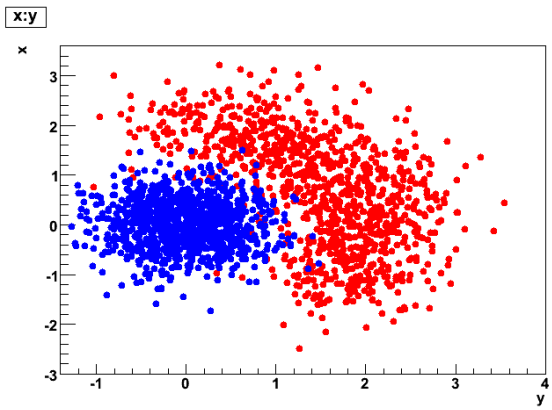

Fig. 5: x : y correlation of input variables used to illustrate how decision trees work. Signal is in blue, background in red.

Building a first tree leads to the result shown in Fig. 6 (top), as a decision tree (left) and with criteria applied in the x : y plane (right). This single decision tree is already performing quite well. Its output is either $+1$ (signal) or -1 (background), so the only way to use the output is to keep either -1 or $+1$ testing candidates. After applying boosting, the separation between signal and background can be improved further. The boosted decision tree output is shown in Fig. 6 (bottom left). The bottom right plot in this figure shows the background versus signal efficiency curves for the first decision tree, for the boosted decision trees and for a Fisher discriminant analysis (for comparison), all run on the same testing events. The boosted decision trees perform best, with more freedom than on a single tree to choose a working point (either choose a signal efficiency or background rejection).

Another way of showing how a decision tree can handle complicated inputs is the XOR problem, a small version of the checkerboard, illustrated in Fig. 7. With enough statistics (left column), even a single tree is already able to find more or less the optimal separation, so boosting cannot actually do much better. On the other hand this type of correlations is a killer for a Fisher discriminant, which fairs as bad as random guessing.

One can repeat the exercise, this time with limited statistics (right column in Fig. 7). Now a single tree is not doing such a good job anymore and the Fisher discriminant is completely unreliable. Boosted decision trees, on the other hand, are doing almost as well as with full statistics, separating almost perfectly signal and background. This illustrates very clearly how the combination of weak classifiers (see for instance the lousy performance of the first tree) can generate a high performance discriminant with a boosting algorithm.

4.5 Other boosting algorithms

AdaBoost is but one of many boosting algorithms. It is also referred to as discrete AdaBoost to distinguish it from other AdaBoost flavours. The Real AdaBoost algorithm [12] defines each decision tree output as :

$$T_k(i) = 0.5 \times \ln \frac{p_k(i)}{1 - p_k(i)},$$

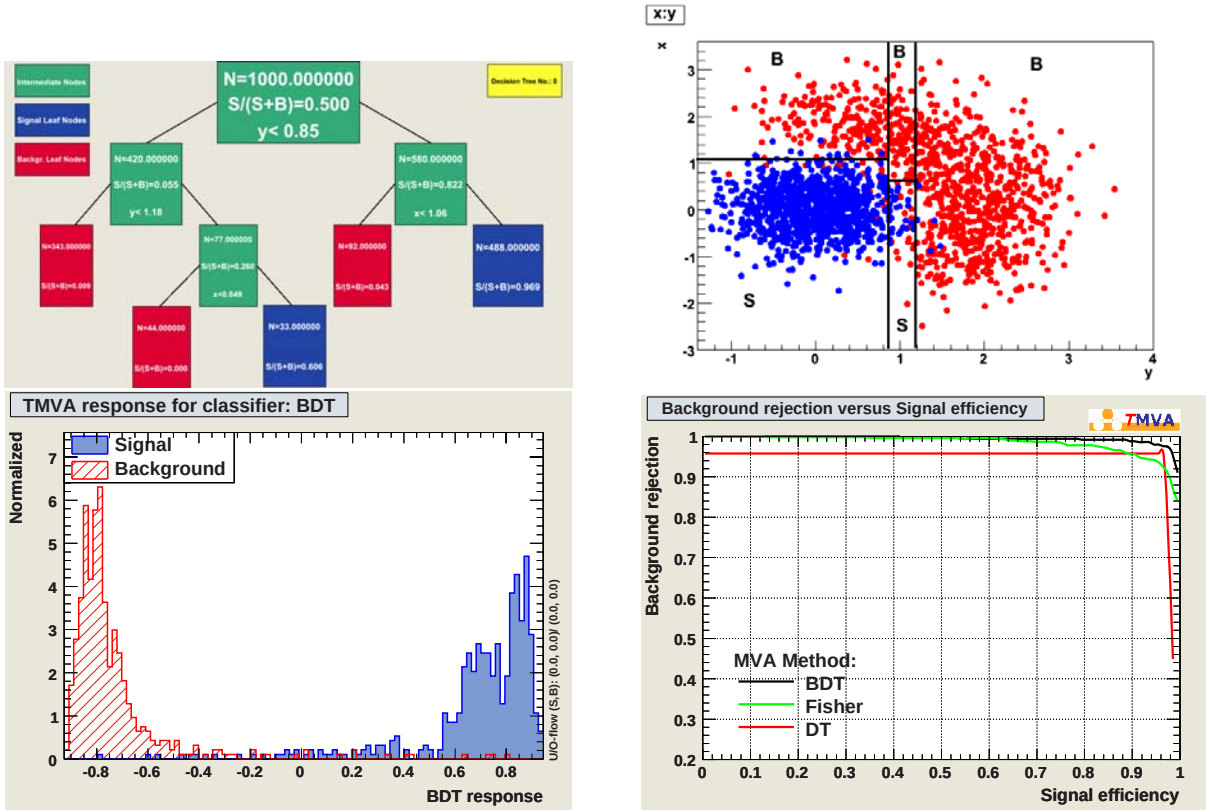

Fig. 6: Top left : First decision tree built on x and y. Top right : Criteria applied in this first tree, shown in the $x : y$ plane ; S (B) areas are labelled as signal (background). Bottom left : Output of the boosted decision trees. Bottom right : background vs. signal efficiency curves for the first decision tree (red), for the boosted decision trees (black) and for a Fisher discriminant analysis (green), all run on the same testing events.

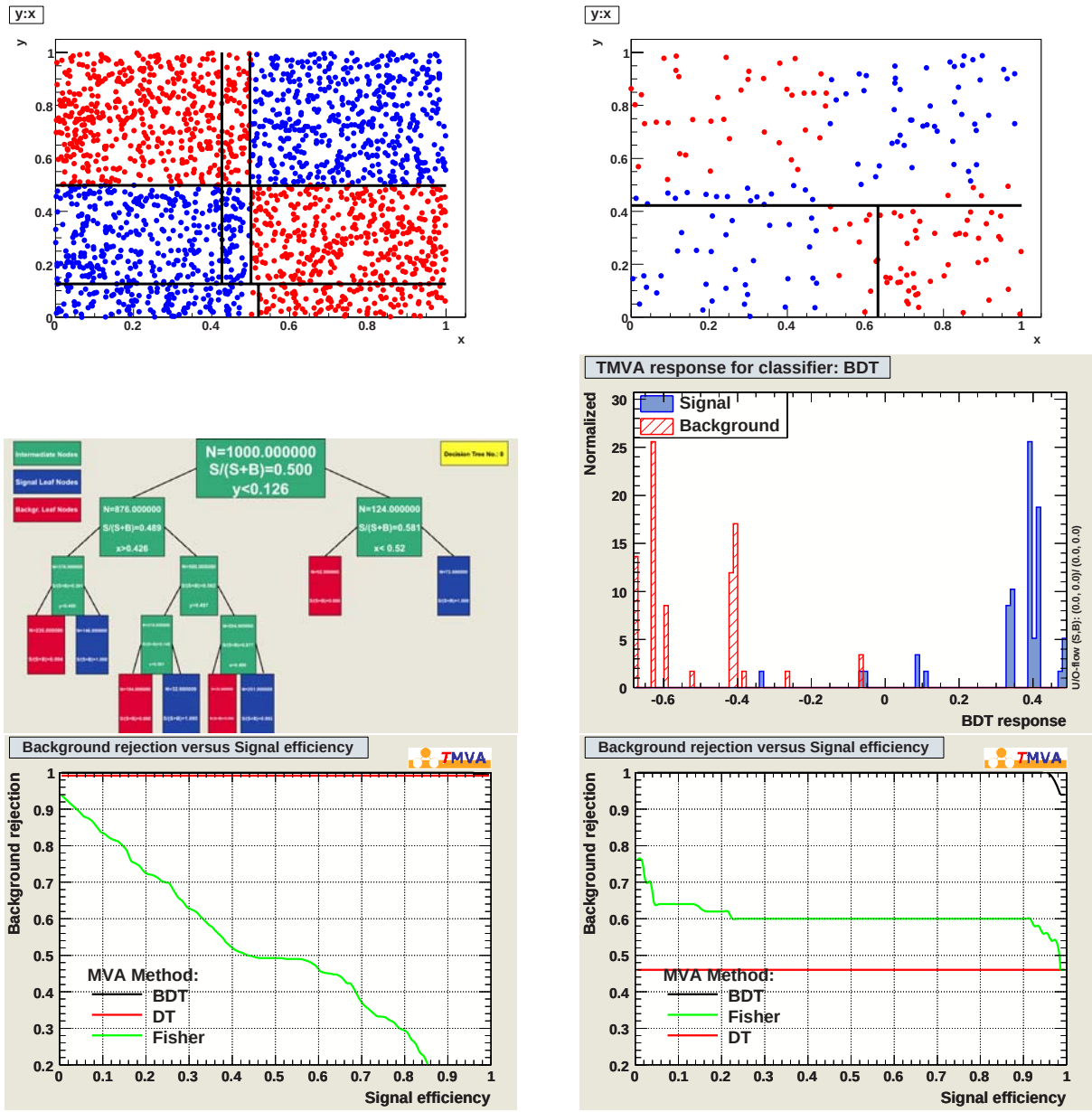

Fig. 7: The XOR problem. Signal is in blue, background in red. The left columns uses sufficient statistics, while the right column has a limited number of training events. The top plots show the signal and background distributions as well as the criteria used by the first decision tree. The middle left plot shows the first decision tree for the large statistics case. The middle right plot shows the boosted decision tree output for the limited statistics case. Bottom plots illustrate the background vs. signal efficiency curves for the first decision tree (red), for the boosted decision trees (black) and for a Fisher discriminant analysis (green), all run on the same testing events.

where $p_k(i)$ is the purity of the leaf on which event i falls. Events are reweighted as :

$$w_i^k \to w_i^{k+1} = w_i^k \times e^{-y_i T_k(i)}$$

and the boosted result is $T(i) = \sum_{k=1}^{N_{\text{tree}}} T_k(i)$. Gentle AdaBoost and LogitBoost (with a logistic function) [12] are other variations.

ϵ-Boost, also called shrinkage [16], consists in reweighting misclassified events by a fixed factor $e^{2\epsilon}$ rather than the tree-dependent α_k factor of AdaBoost. ϵ-LogitBoost [9] is reweighting them with a logistic function $\frac{e^{-y_i T_k(i)}}{1+e^{-y_i T_k(i)}}$. ϵ-HingeBoost [9] is only dealing with misclassified events :

$$w_i^k \to w_i^{k+1} = \mathsf{I}(y_i \times T_k(i) \leq 0).$$

Finally one can cite the adaptive version of the "boost by majority" [7] algorithm, called BrownBoost [17]. It works in the limit where each boosting iteration makes an infinitesimally small contribution to the total result, modelling this limit with the differential equations that govern Brownian motion.

5 Other averaging techniques

As mentioned in Section 3.2 the key to improving a single decision tree performance and stability is averaging. Other techniques than boosting exist, some are briefly described below. As with boosting, the name of the game is to introduce statistical perturbations to randomise the training sample, hence increasing the predictive power of the ensemble of trees [5].

Bagging (Bootstrap AGGregatING) was proposed in 1994 [18]. It consists in training trees on different bootstrap samples drawn randomly with replacement from the training sample. Events that are not picked for the bootstrap sample form an "out of bag" validation sample. The bagged output is the simple average of all such trees.

Random forests is bagging with an extra level of randomisation [19]. Before splitting a node, only a random subset of input variables is considered. The fraction can vary for each split for yet another level of randomisation.

Trimming is not exactly an averaging technique per se but can be used in conjunction with another technique, in particular boosting, to speed up the training process. After some boosting cycles, it is possible that very few events with very high weight are making up most of the total training sample weight. One may then decide to ignore events with very small weights, hence introducing again some minor statistical perturbations and speeding up the training.

These techniques, as was the case for boosting, are actually not limited to decision trees. Bagging applies to any training, random forests apply to any classifier where an extra level of randomisation is possible during the training, and trimming applies in particular to any boosting algorithm on any classifier. In particular one could build a boosted random forest with trimming.

6 Conclusion

In this lecture we have seen what decision trees are and how to construct them, as a powerful multivariate extension of a cut-based analysis. They are a very convincing technique starting to make its way in high energy physics. Advantages are numerous : their training is fast, they lead to human-readable results (not black boxes) with possible interpretation by a physicist, can deal easily with all sorts of variables and with many of them, with in the end relatively few parameters.

Decision trees are, however, not perfect and suffer from the piecewise nature of their output and a high sensitivity to the content of the training sample. These shortcoming are for a large part addressed by averaging the results of several trees, each built after introducing some statistical perturbation in the training sample. Among the most popular such techniques, boosting (and its AdaBoost incarnation) was described in detail, providing ideas as to why it seems to be performing so well and being very resilient against overtraining. Other averaging techniques were briefly described.

Boosted decision trees have now become quite fashionable in high energy physics. Following the steps of MiniBooNe for particle identification and D0 for the first evidence and observation of single top quark production, other experiments and analyses are starting to use them, in particular at the LHC. This warrants a word a caution. Despite recent successes in several high profile results, boosted decision trees cannot be thought of as the best multivariate technique around. Most multivariate techniques will in principle tend towards the Bayes limit, the maximum achievable separation, given enough statistics, time and information. But in real life resources and knowledge are limited and it is impossible to know *a priori* which method will work best on a particular problem. The only way is to test them. Figure 8 illustrates this situation for the single top quark production evidence at D0 [10]. In the end boosted decision trees performed only marginally better than Bayesian neural networks and the matrix elements technique, but all three analyses were very comparable, as shown by their power curves. The boosted decision tree analysis profited, however, of the fast turnaround of decision tree training in order to perform many valuable cross checks.

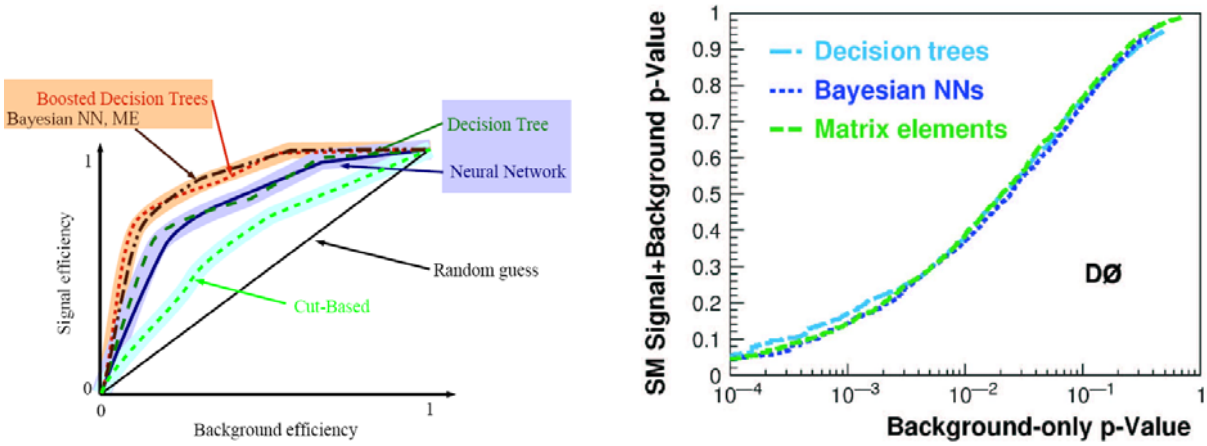

Fig. 8: Comparison of several analysis techniques used in the D0 search for single top quark production [10]. Left : Signal vs. background efficiency curves for random guessing, a cut-based analysis, artificial neural networks, decision trees, Bayesian neural networks, the matrix elements technique and boosted decision trees. Right : Power curves (p-value of the signal+background hypothesis vs. p-value of the signal-only hypothesis) for the boosted decision trees, Bayesian neural networks and matrix elements.

Finally, it cannot be stated often enough that using multivariate techniques is only the very last step of an analysis and is meaningful if and only if a proper model has been built that describes the data very well in all the variables one wishes to feed into the analysis. This also means there is no point in carefully optimising such analyses right now at the LHC, when no real data are available.

7 Software

Many implementations of decision trees exist on the market. Historical algorithms like CART [1], ID3 [20] and its evolution C4.5 [21] are available in many different computing languages. The original MiniBoone [9] code is available at http ://www-mhp.physics.lsa.umich.edu/∼roe/, so is the StatPattern-Recognition [2] code at http ://sourceforge.net/projects/statpatrec, and LHC experiments have various implementations in their software.

I would recommend a different approach, which is to use an integrated solution able to handle decision trees but also other techniques and flavours, allowing to run several of them to find the best suited to a given problem. Weka is a data mining software written in Java, open source, with a very good published manual. It was not written for HEP but is very complete. Details can be found at http ://www.cs.waikato.ac.nz/ml/weka/.

Another recent development which becomes popular in HEP is TMVA [14]. It is now integrated in

ROOT, which makes it convenient to use, comes with a complete manual and was extensively presented during this school [22]. It is available online at http ://tmva.sourceforge.net or directly in ROOT.

8 Acknowledgements

I would like to thank the organisers for inviting me to teach this lecture on decision trees. The school program, lectures, attendance and location were very enjoyable.

Références

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

[2] I. Narsky, "StatPatternRecognition : A C++ Package for Statistical Analysis of High Energy Physics Data", arXiv :physics/0507143v1, 2005.

[3] C. Gini, "Variabilità e Mutabilità" (1912), reprinted in *Memorie di Metodologica Statistica*, edited by E. Pizetti and T. Salvemini, Rome : Libreria Eredi Virgilio Veschi, 1955.

[4] J.R. Quinlan, "Simplifying decision trees", *International Journal of Man-Machine Studies*, 27(3) :221–234, 1987.

[5] H. Prosper, *Multivariate discriminants*, "Ensemble learning", these proceedings.

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

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

[8] 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) p. 148, 1996.

[9] 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).

[10] V. M. Abazov *et al.* [D0 Collaboration], "Evidence for production of single top quarks and first direct measurement of $|V_{tb}|$", Phys. Rev. Lett. **98**, 181802 (2007) ; V. M. Abazov *et al.*, "Evidence for production of single top quarks", Phys. Rev. D**78**, 012005 (2008) ; V. M. Abazov *et al.*, "Observation of single top quark production", submitted to Phys. Rev. Lett.

[11] 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.

[12] 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.

[13] H. Prosper, *Multivariate discriminants*, "Optimal classification", these proceedings.

[14] A. Höcker *et al.*, "TMVA : Toolkit for multivariate data analysis", PoS A **CAT**, 040 (2007) [arXiv :physics/0703039].

[15] G. Cowan, "Multivariate statistical methods and data mining in particle physics", *CERN Academic Training Lectures*, June 2008.
http ://indico.cern.ch/conferenceDisplay.py ?confId=24827

[16] J.H. Friedman, "Greedy function approximation : a gradient boosting machine", *The Annals of Statistics*, 29 (5), 1189–1232, 2001.

[17] Y. Freund, "An adaptive version of the boost by majority algorithm", *Machine Learning*, 43 (3), 293–318, 2001.

[18] L. Breiman, "Bagging Predictors", *Machine Learning*, 24 (2), 123–140, 1996.

[19] L. Breiman, "Random forests", *Machine Learning*, 45 (1), 5–32, 2001.

[20] J.R. Quinlan, "Induction of decision trees", *Machine Learning*, 1(1) :81–106, 1986.

[21] J.R. Quinlan, *C4.5 : programs for machine learning*, Morgan Kaufmann Publishers Inc., San Francisco, CA, 1993.

[22] A. Höcker, *L'utilisation de TMVA*, these proceedings.