MultiVariate Analysis Tutorial

Francesco Giuseppe Gravili

F.G. Gravili - INFN Section of Lecce
Department of Mathematics and Physics *Ennio de Giorgi*
Università del Salento

XIII Workshop ATLAS Italia - 2017, 27th October

Introduction

Most HEP analyses require descrimination of signal from background:

- Event level, e.g. Higgs searches
- Track level, e.g. particle ID
- Cone level, e.g. τ-vs-jet reconstrucion
- Lifetime and flavor tagging, e.g. b-tagging
- SUSY/Exotic searches

The MultiVariate input information used for that has various sources:

- Kinematic variables, e.g. momenta, decay angles
- Event properties, e.g. jet/lepton multiplicity
- Detector response, e.g. muon hits

References

Almost everything can be found on the official TMVA[1] website, even if it's not fully up-to-date...

- Home page: `http://tmva.sourceforge.net`
- User Guide:
  `http://tmva.sourceforge.net/docu/TMVAUsersGuide.pdf`
- Public Talks: `http://tmva.sourceforge.net/talks.shtml`

Additional material available on `https://indico.cern.ch/event/472305/`

Please, download the compressed archive for this tutorial:
`https://cernbox.cern.ch/index.php/s/NhWEUHxQqFD6eIt`

[1] ***T****oolkit for* ***M****ulti****V****ariate* ***A****nalysis*

MultiVariate Analysis

- Analysis techniques, e.g. the ones for new particle searches, are performed by reducing the background, exploiting some appropriate cuts on specific discriminating variables, over the signal looked for.
- Cut values are based upon comparison between Signal distribution and Monte Carlo ones for background processes.
- This procedure is correct if such background processes are well known and Monte Carlo has been tuned accordingly in order to reproduce real data.
- MultiVariate techniques are exploited because they can combine the discriminating power of several input variables, being able to take into account internal correlations (linear and/or not), thus reducing the complexity of a problem.

Event Classification and Regression

Let's suppose we have two categories of events, H_0 and H_1:

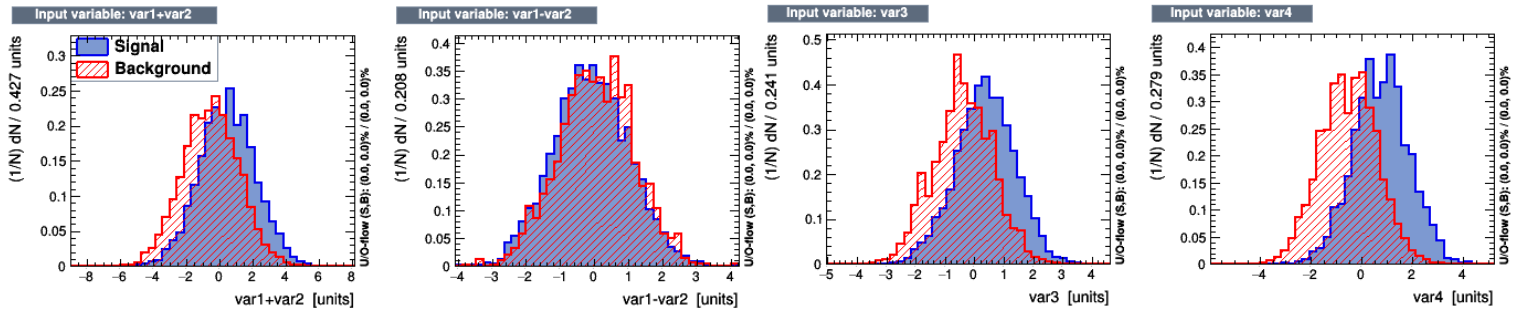
- A set of discriminating input variable $x_1, x_2, \cdots, x_N$
- Which cut(s) in the *input parameter space* to set a decision boundary to discriminate H_0 from H_1 or viceversa?
  - Linear
  - Non-Linear $\rightarrow$ Wide range of options available, maybe too much...

More generally, a classification problem corresponds to a *discretized regression* problem, the process that estimates the parameter values of a function, which predicts the value of a response variable (or vector) in terms of the values of other variables (the *input* variables).

The problem is not trivial, expecially if the parameter space is multidimensional...

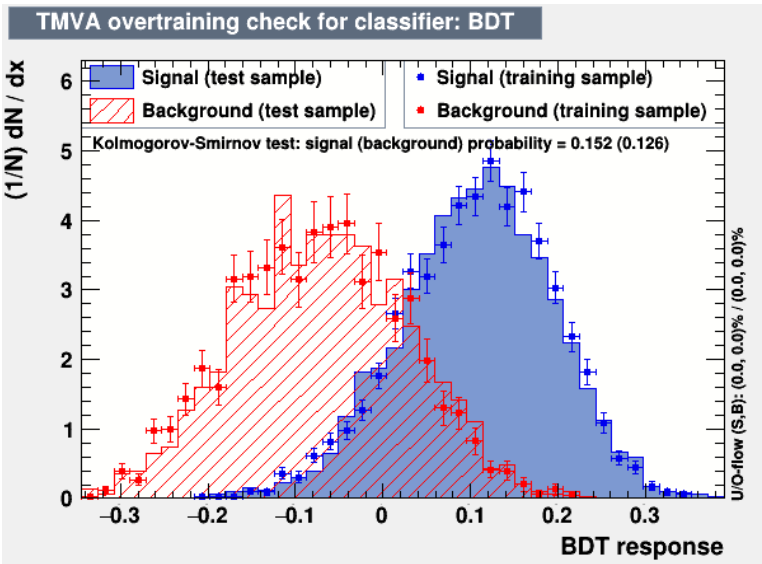
MultiVariate Event Classification in N-dim

Each event, Signal or Background, has N measured variables:



Aim is to find a mapping from N-dimensional input-observable space to 1-dimensional output

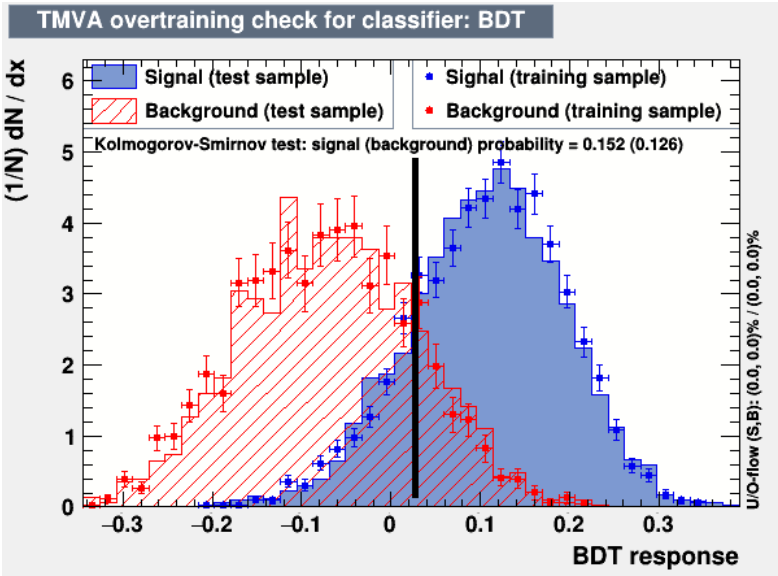
$$y(\vec{x}) : \mathbb{R}^N \to \mathbb{R}$$



$y(\vec{x})$ *response* is the "test statistic" in N-dimensional space of input variables

In our case $y(\vec{x})$ corresponds to the response of BDT (*Boosted Decision Trees*) method

MultiVariate Event Classification in N-dim



Applying a cut at $y(\vec{x} = \vec{x_0}) \equiv y_0$

Signal density

$$P(y, S) \equiv P(y|S)P(S)$$

Background density

$$P(y, B) \equiv P(y|B)P(B)$$

- Probability Distribution Functions of $\vec{x}$, $PDF_S(y)$ and $PDF_B(y)$, are used to set the selection cut
- Most important parameters are efficiency and purity:
  - Efficiency: Fraction of signal events accepted over the total ones, i.e. $N_s(y > y_0)/N_s$
  - Purity: Fraction of signal events in the $y > y_0$ area, i.e. $N_s(y > y_0)/(N_s + N_b)(y > y_0)$
- Let $H(y)$ be Heaviside step function:
  - Signal loss: $M_S \int H(y_0 - y)P(y, S)\, dy$
  - Background contamination: $M_B \int H(y - y_0)P(y, B)\, dy$

M_S and M_B misclassifing factors

Event Classification: Bayesian statistics

Given an initial set of measured variables $\vec{x} = \{x_1, x_2, \cdots, x_N\}$:

$$P(class = C|y) = \frac{P(y|C)P(C)}{P(y)}$$

- $P(class = C|y) \rightarrow$ Posterior probability
- $P(y|C) \rightarrow$ *Likelihood*, which depends on $\vec{x}$ and the mapping function
- $P(C) \rightarrow$ Prior probability to observe an event of class-C, i.e. fraction of signal events w.r.t. the total ones, $N_S/(N_S + N_B)$
- $P(y) \rightarrow$ Global probability density of making a measure of $y(\vec{x})$, i.e. $\sum_c P(y|c)P(c)$ with c running on the total classes available

Event Classification: Bayesian statistics

Back to our case, it needs to minimize the quantity

$$M(y_0) = M_S \int H(y_0 - y)P(y, S)\, dy + M_B \int H(y - y_0)P(y, B)\, dy$$

with respect to boundary y_0:

$$\begin{aligned} 0 &= M_S \int \delta(y_0 - y)P(y, S)\, dy - M_B \int \delta(y - y_0)P(y, B)\, dy \\ &= M_S P(y_0, S) - M_B P(y_0, B) \end{aligned}$$

The Bayes Discriminant is defined as

$$BD \equiv \frac{M_B}{M_S} = \frac{P(y_0, S)}{P(y_0, B)} = \frac{P(y_0|S)P(S)}{P(y_0|B)P(B)}$$

Event Classification: Bayesian statistics

The boundary y_0 may be hard to establish, especially in case of overlap of $PDF_S(y)$ and $PDF_B(y)$. However, the quantity

$$P(S|y) = \frac{BD}{1+BD} = \frac{M_B}{M_S + M_B}$$

is what should be achieved to minimize misclassification cost, i.e. achieving classification with the fewest mistakes. By fixing $P(S|y)$, decision boundary is defined.

The criterion usually adopted to discriminate Signal from Background is:

$$\frac{P(S|y)}{P(B|y)} = \frac{P(y|S)}{P(y|B)} \cdot \frac{P(S)}{P(B)} > \epsilon$$

Posterior odds ratio

Likelihood ratio as discriminating function

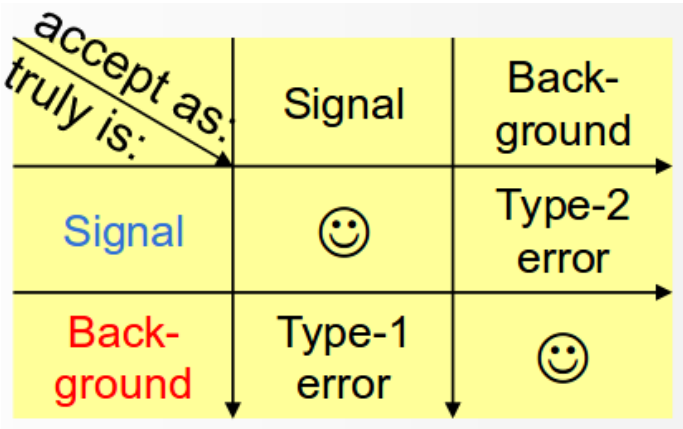
Prior odds ratio

$\propto$ efficiency and purity

Event Classification: Errors

When selecting Signal events from Background ones, two errors may occur:

- Type-1 errors
  - Classify event as Class-C even though it is not
  - Loss of purity
- Type-2 errors
  - Fail to identify an event from Class-C as such
  - Loss of efficiency



If Ω is the region (from test outcome) where an event is accepted as Signal

- Background selection efficiency $\equiv \alpha = \int_{\Omega} P(y|B)\, dy \equiv$ *p-value*
- Signal selection efficiency $\equiv 1 - \beta = 1 - \int_{\bar{\Omega}} P(y|S)\, dy$

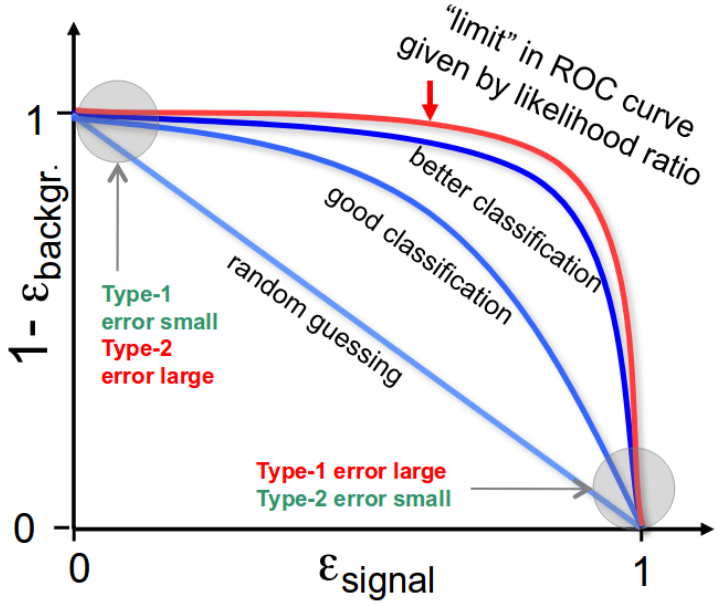
Neyman-Pearson Lemma

Given a *Likelihood ratio*:

Neyman-Pearson Lemma

For each selection efficiency, the Likelihood ratio used as selection criterion gives the best possible background rejection.

In other words, it maximises the area under the *Receiver Operation Characteristics* (ROC) curve.



By varying the cut chosen for the ratio, the working point (efficiency and purity) moves along the ROC curve.

How to choose the cut? No golden rule for it...Need to know prior odds...

- Measurement of signal cross section $\rightarrow \max(S/\sqrt{S+B})$
- Discovery of a signal $\rightarrow \max(S/\sqrt{B})$
- Precision measurement $\rightarrow$ High purity p
- Event selection $\rightarrow$ High efficiency ϵ

"Real" Event Classification

- Unfortunately, the true probability densities functions are typically unknown, making N-P Lemma not very useful...
- A few sets of known (already classified) events can be found
- Commonly, Monte Carlo simulations are used in order to have an estimation of Signal and Background
- Simulated events are split in two main statistically independent categories:
  - **Training** events to estimate $P(y|C)$ and find out Likelihood ratio, as well as a region where emphasizing Signal and Background separation
  - **Testing** events to have a cross-check for the classifier performance

It doesn't exist an exact recipe to follow; everything must be tuned according to one's analysis purposes

TMVA

The *Toolkit for MultiVariate Analysis* provides a ROOT-integrated environment for the processing and parallel evaluation of sophisticated multivariate classification techniques.

The idea behind it is to provide a large variety of powerful multivariate classifiers in one common environment, allowing for easy use and comparison of the different classification techniques for any given problem.

The TMVA software package consists of object-oriented implementations in C++/ROOT and it currently includes:

- Rectangular cut optimisation;
- Projective and Multi-dimensional likelihood estimation (PDE range-search and k-NN);
- Linear and nonlinear discriminant analysis (H-Matrix, Fisher, FDA);
- Artificial Neural Networks
- Boosted Decision Trees
- Support Vector Machine
- Predictive learning via rule ensambles (RuleFit)

TMVA

A typical TMVA analysis consists of two main steps:

- *Training phase*: training, testing and evaluation of classifiers using data samples with known signal and background composition. The chosen classifier computes optimal values of some tunable parameters w (weights) of the mapping function $y(\vec{x})$ in order to maximize Signal-Background separation.
- *Application phase*: using selected trained classifiers to classify unknown data samples.

These two steps will be illustrated later with toy data samples...

Moreover, before starting the training phase, a ***Preanalysis*** and a ***Preprocessing*** take place: their aim is to provide variables according to the method(s) invoked and apply cut(s) to the input variables.

In the preanalysis step ***correlation coefficients*** of the input variable are calculated, as well as a first ***variable ranking***, to be replaced by a new one, specific for each method invoked. ***At the end of this phase, an *.xml file is generated which contains aforesaid weights***.

TMVA: Global Methods Overview

CRITERIA		MVA METHOD: Cuts	Likeli-hood	PDE-RS / k-NN	PDE-Foam	H-Matrix	Fisher / LD	MLP	BDT	Rule-Fit	SVM
Perfor-mance	No or linear correlations	⋆	⋆⋆	⋆	⋆	⋆	⋆⋆	⋆⋆	⋆	⋆⋆	⋆
	Nonlinear correlations	○	○	⋆⋆	⋆⋆	○	○	⋆⋆	⋆⋆	⋆⋆	⋆⋆
Speed	Training	○	⋆⋆	⋆⋆	⋆⋆	⋆⋆	⋆⋆	⋆	⋆	⋆	○
	Response	⋆⋆	⋆⋆	○	⋆	⋆⋆	⋆⋆	⋆⋆	⋆	⋆⋆	⋆
Robust-ness	Overtraining	⋆⋆	⋆	⋆	⋆	⋆⋆	⋆⋆	⋆	⋆	⋆	⋆⋆
	Weak variables	⋆⋆	⋆	○	○	⋆⋆	⋆⋆	⋆	⋆⋆	⋆	⋆
Curse of dimensionality		○	⋆⋆	○	○	⋆⋆	⋆⋆	⋆	⋆	⋆	
Transparency		⋆⋆	⋆⋆	⋆	⋆	⋆⋆	⋆⋆	○	○	○	○

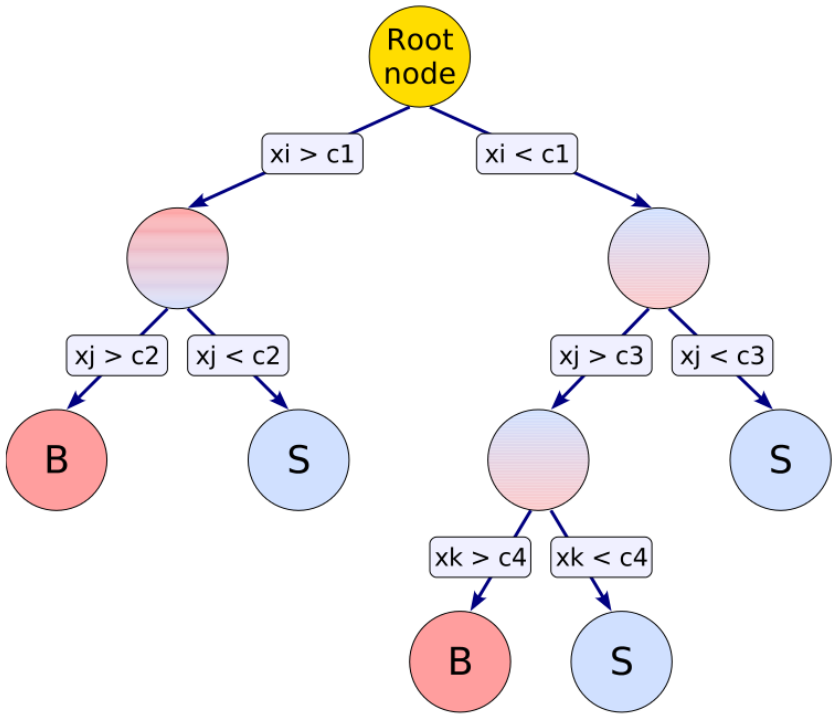
Table 6: Assessment of MVA method properties. The symbols stand for the attributes "good" (⋆⋆), "fair" (⋆) and "bad" (○). "Curse of dimensionality" refers to the "burden" of required increase in training statistics and processing time when adding more input variables. See also comments in the text. The FDA method is not listed here since its properties depend on the chosen function.

Decision Trees are sometimes referred to as the best "out of the box" classifiers because little tuning is required in order to obtain reasonably good results

A disadvantage of BDT is the need of large training samples in order to avoid that results strongly depend on the training sample, thus leading to a performance deterioration when it is measured on an independent sample ("overtraining" effect)

B Decision Trees

A **Decision Tree** is a binary tree structured classifier.



Repeated left/right (yes/no) decisions are taken on one single variable at a time until a stop criterion is fulfilled

The phase space is split this way into many regions (nodes): the algorithm is repeated recursively on each node, classifying events as Signal or Background, depending on the majority of training events that end up in the final *leaf* node.

Boosted DT

The **boosting** of a decision tree extends this concept from one tree to several trees which form a **forest**; an event is classified on the basis of a majority vote done by each tree of the forest.

The trees are derived from the same training ensemble by *reweighting* events and are finally combined into a single classifier, given by a (weighted) average of the individual decision trees.

Boosting stabilizes the response of the decision trees with respect to fluctuations in the training sample and is able to considerably enhance the performance w.r.t. a single tree.

4 different boosting algorithms are available for decision trees in TMVA:

- Ada(ptive)Boost, the most popular boosting algorithm
- Gradient Boost
- Bagging
- Randomised Trees

BDT: A few words on Systematics...

- No particular rule to evaluate them
- BDT output can be considered as any other variable used for selection
- Most common procedure: propagate other uncertainties up to BDT output and check how much the analysis is affected
- Some extra systematics may be required, not so much on technique itself, but because it probes specific corners of phase space and/or wider parameter space, e.g. usually loosening pre-BDT selection cuts

Classification

From lxplus, it is just needed to setup root as usual

```
setupATLAS && lsetup root
```

Launching commands[2] (If no method is given, default ones will be trained):

```
root -l 'TMVAClassification.C("Method1,Method2,...,MethodN")'
root -l TMVAClassification.C\(\"Method1,Method2,...,MethodN\"\)
```

Input file(s) opening:

```
TFile *input = TFile::Open( "tmva_example.root" );
```

Acquiring Signal and Background Trees:

```
TTree *Signal     = (TTree*)input->Get("TreeS");
TTree *Background = (TTree*)input->Get("TreeB");
```

Output file opening:

```
TFile* outputFile = TFile::Open( "TMVA.root", "RECREATE" );
```

[2] If batch mode is needed, just add the *-b* flag

Classification

Creating pointer to *factory*[3] object:

TMVA::Factory *f = new TMVA::Factory("<JobName>", outputFile, "<options>");

Standard configuration:

TMVA::Factory *factory = new TMVA::Factory("TMVAClassification", outputFile, "!V:!Silent:Color:DrawProgressBar:Transformations=I;D;P;G,D: AnalysisType=Classification");

Options separated by a column (:) - Most important ones:

- !V → Verbose mode disabled
- !Silent → Batch mode disabled
- Transformations on input variables → Identity (I), Decorrelation (D), PCA (P), Uniform (U), Gaussianisation followed by Decorrelation (G,D)
- AnalysisType → Set the analysis type: Classification, Regression, Multiclass, Auto

[3] It handles communication of the user with data and the MVA methods

Classification

Creating pointer to *dataloader* container:

```
TMVA::DataLoader *dataloader=new TMVA::DataLoader("dataset");
```

Adding input/spectator variables to this container:

```
dataloader->AddVariable( "myvar1 := var1+var2", "var1+var2", "units", 'F' );
dataloader->AddVariable( "var3", "var3", "units", 'F' );

dataloader->AddSpectator( "spec1 := var1*2",  "Spectator 1", "units", 'F' );
```

- A simple or a derived variable can be added
- The types 'F' stands for both float and double, while 'I' for int, short, char, and unsigned integers
- Spectator variables won't be used in the MVA training, but will appear in the final "TestTree".

Classification

Assigning global weights per tree:

```
Double_t signalWeight = 1.0, backgroundWeight = 1.0;
dataloader->AddSignalTree     ( Signal,     signalWeight );
dataloader->AddBackgroundTree ( Background, backgroundWeight );
```

Setting individual event weights (variables must exist in the original TTree):

```
dataloader->SetSignalWeightExpression( "weight" );
dataloader->SetBackgroundWeightExpression( "weight" );
```

Apply additional cuts on samples (may be different):

```
TCut mycuts = "abs(var1)<0.5 && abs(var2-0.5)<1";
TCut mycutb = "abs(var1)<0.5";
```

Classification

Creating Training and Test trees:

dataloader->PrepareTrainingAndTestTree(preselectionCut, "<options>");

Standard configuration:

dataloader->PrepareTrainingAndTestTree(mycuts, mycutb,
"nTrain_Signal=0:nTrain_Background=0:SplitMode=Random:
NormMode=NumEvents:!V");

- Number of events for Training and Testing can be chosen by setting the corresponding variables: *nTrain_Signal, nTrain_Background, nTest_Signal, nTest_Background*
- If nTrain_Signal and nTrain_Background are both 0, the total sample is split in half for training and testing.
- SplitMode → Method of picking training and testing events (Random, Alternate, Block)

Classification

Creating Training and Test trees:

dataloader->PrepareTrainingAndTestTree(preselectionCut, "<options>");

Standard configuration:

dataloader->PrepareTrainingAndTestTree(mycuts, mycutb,
"nTrain_Signal=0:nTrain_Background=0:SplitMode=Random:
NormMode=NumEvents:!V");

- NormMode → Overall renormalisation of event-by-event weights used in the training:
  - None: No renormalisation
  - NumEvents: average weight of 1 per event, independently for Signal and Background
  - EqualNumEvents: average weight of 1 per event for signal and sum of weights for background equal to sum of weights for signal

Classification

Booking chosen method:

factory->BookMethod(dataloader, TMVA::Types::kBDT, "BDT",
"!H:!V:NTrees=850:MinNodeSize=2.5%:MaxDepth=3:BoostType=AdaBoost:
AdaBoostBeta=0.5:UseBaggedBoost:BaggedSampleFraction=0.5:
SeparationType=GiniIndex:nCuts=20");

Most important settings:

- NTrees $\rightarrow$ Number of trees in the forest
- MaxDepth $\rightarrow$ Max depth of the decision tree allowed
- nCuts $\rightarrow$ Number of grid points in variable range used in finding optimal cut in node splitting. If $nCuts = -1$, an algorithm will be invoked to test all possible cuts on the training sample and find the best one.
- BoostType $\rightarrow$ Boosting type for the trees in the forest
- AdaBoostBeta $\rightarrow$ Learning rate for AdaBoost algorithm

Full list on `http://tmva.sourceforge.net/optionRef.html`

Classification

Training MVAs using the set of training events:

factory->TrainAllMethods();

Evaluating all MVAs using the set of test events:

factory->TestAllMethods();

Evaluating and compare performance of all configured MVAs:

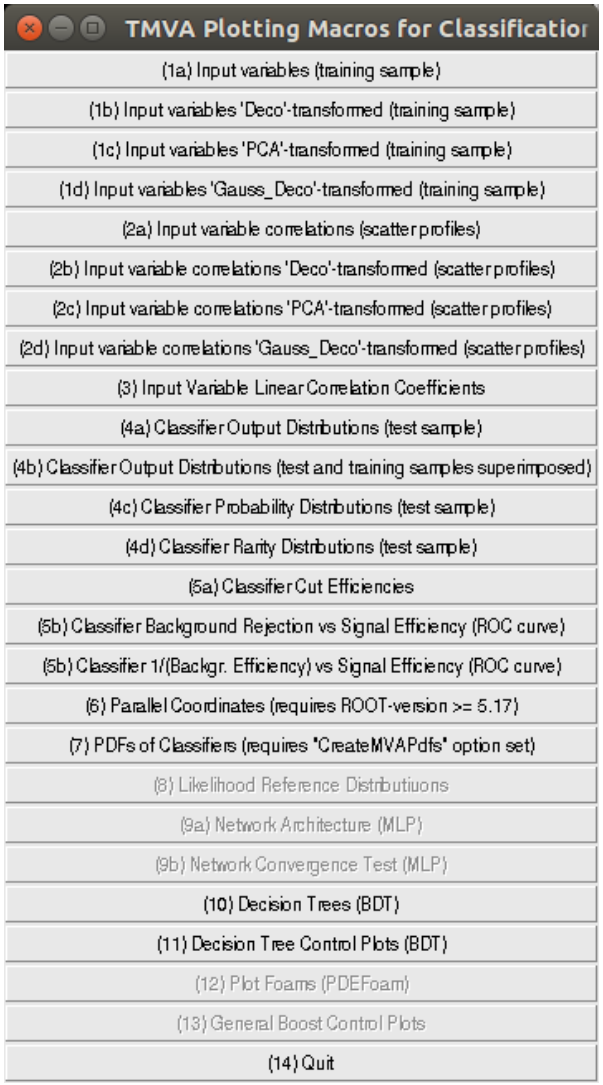
factory->EvaluateAllMethods();

Launching the GUI:

if(!gROOT->IsBatch()) TMVA::TMVAGui(outfileName);

Weight file is under:
$PWD/dataset/weights/TMVAClassification_BDT.weights.xml
One *.xml file for each method trained

Classification - TMVA Gui



Several plots available for Signal (S) and Background (B):

- Input variables with and without preprocessing
- Correlation scatter profiles and linear coefficients
- Classifier outputs for test and training samples (Watch out for overtraining!!)
- Classifier *Rarity* distributions
- Classifier *significance* with optimal cuts
- B rejection versus S efficiency
- Classifier specific plots
- Decision trees visualization

Application

Launching commands[4] (If no method is given, default ones will be trained):

```
root -l 'TMVAClassificationApplication.C("Method1,...,MethodN")'
root -l TMVAClassificationApplication.C\(\"Method1,...,MethodN\"\)
```

Creating pointer to *Reader*[5] object

```
TMVA::Reader *reader = new TMVA::Reader( "!Color:!Silent" );
```

Creating variables whose name must corresponds in name and type to those given in the weight file(s) used:

```
Float_t var1, var3, spec1;
reader->AddVariable( "myvar1 := var1+var2", &var1 );
reader->AddVariable( "var3",                &var3 );

reader->AddSpectator( "spec1 := var1*2",   &spec1 );
```

[4] If batch mode is needed, just add the *-b* flag

[5] Analogously to the Factory, it interfaces communication between the user application and the MVA methods

Application

Booking Classifier(s):

```
reader->BookMVA( "methodName", "path/to/weightfile" );
```

Starting event loop on tree, e.g. Signal:

```
TFile *input = TFile::Open( "tmva_class_example.root" );
TTree* theTree = (TTree*)input->Get("TreeS");
for (Long64_t ievt=0; ievt<theTree->GetEntries(); ievt++) { ... }
```

Evaluating Classifier output:

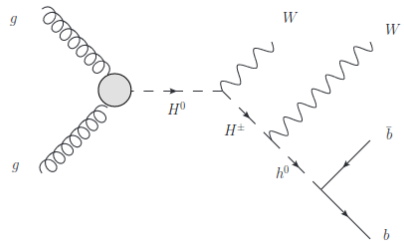
```
reader->EvaluateMVA( "BDT method" )
```

The last command will return a number, i.e. it can be used to fill an histogram, a leaf or whatever is needed.

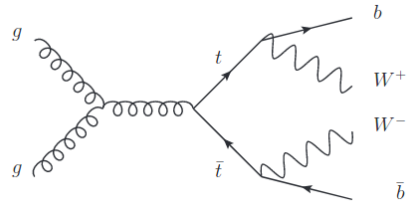
```
if (Use["BDT"])  histBdt->Fill( reader->EvaluateMVA( "BDT method" ) );
```

3 toy-datasets for machine-learning purposes with 100k, 10k and 1k data (signal and background), publicly[6] available:

Signal process



Background process

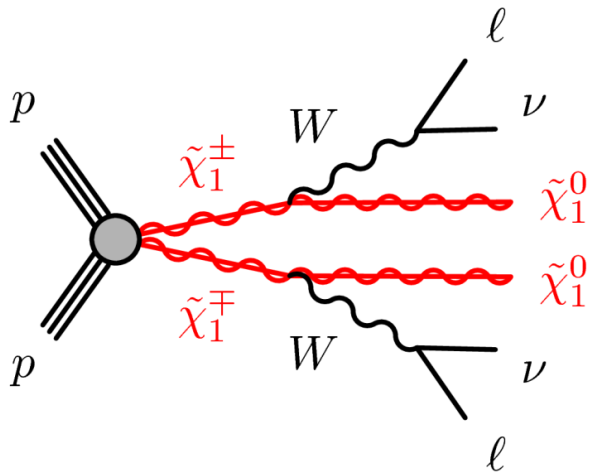


- Data converted from *csv* format to ROOT-*TTrees* (one for Signal and one for Background) in the same TFile
- 28 variables to play around
- *Input features were standardized over the entire set with mean 0 and standard deviation 1, except for those features with values strictly greater than 0 which were scaled so that the mean value was 1.*

[6] Download link and reference article on `https://arxiv.org/abs/1402.4735`

Analysis comparison: Cut&Count

Process:



Preselection cuts for benchmark signal $m(\tilde{\chi}_1^{\pm}, \tilde{\chi}_1^0) = (100, 25)$, luminosity $L = 36.1$ fb^{-1}

- OS leptons, with $m_{ll} > 20$ GeV
- $p_T^{l1} > 25$ GeV && $p_T^{l2} > 20$ GeV
- Njets < 2, with $|\eta| < 2.4$ and $E > 20$ GeV
- DF and SF (with $|m_{ll} - m_Z| > 20$ GeV) all together

Analysis comparison: Cut&Count (vs BDT)

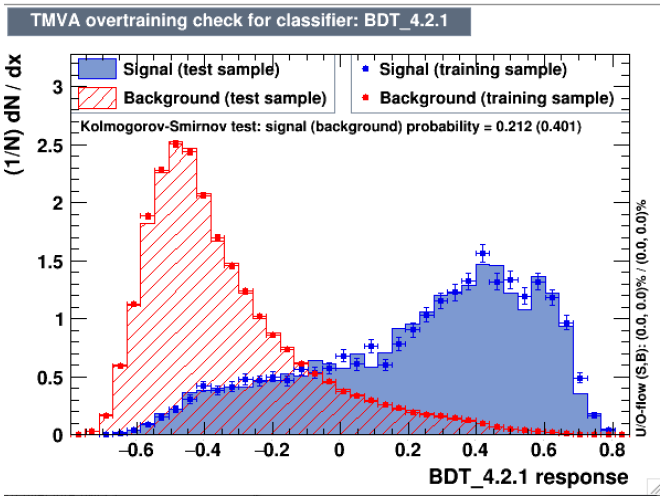
Cuts	Signal (100,25)	$t\bar{t}$	VV $2l2\nu$	$Z \to l^+l^-$ $l = e, \mu, \tau$	Signif. $S/\sqrt{S+B}$
After Preselection	3370.2	51276.1	37075.5	2047746.8	
$E_T^{miss} > 50$ GeV	1478.7	32965.2	15374.8	51345.9	
$E_T^{miss}/M_{eff} > 0.35$	907.0	12597.2	9337.0	32854.4	
$\lvert p_b^{ll}\rvert < 50$ GeV	767.5	4016.5	7830.0	12400.6	
$\lvert\Delta\eta\rvert < 1.3$	616.7	2644.3	5089.8	10550.0	
$m_T(l_1 - E_T^{miss})$	407.1	2161.6	3150.3	1890.2	
$35 < m_{T2} < 70$ GeV	280.5	1067.9	1959.3	672.1	
$nJet = 0$	217.5	314.1	1336.4	323.0	**4.64**
$nJet = 1$	63.0	753.8	622.9	349.1	1.49
$nJet = 1$ && $isB1$	62.4	354.4	617.3	325.3	**1.69**

Combining the two highlighted conditions, a total significance of **4.70** is obtained

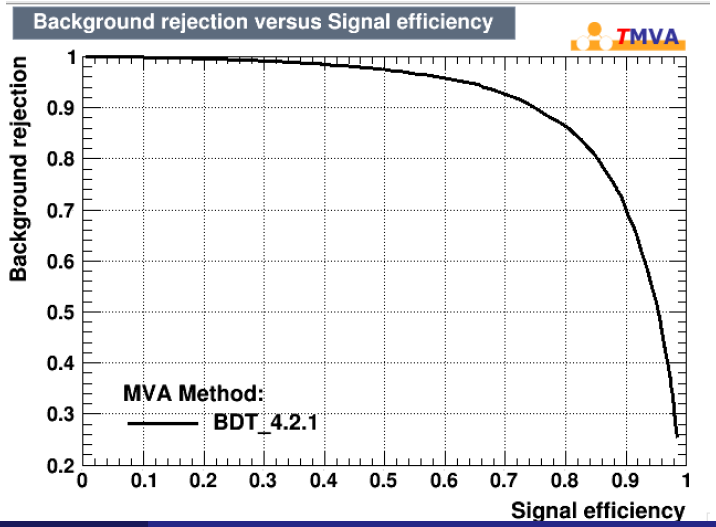
Analysis comparison: BDT

Keep in mind to look at

Classifier output



ROC Curve

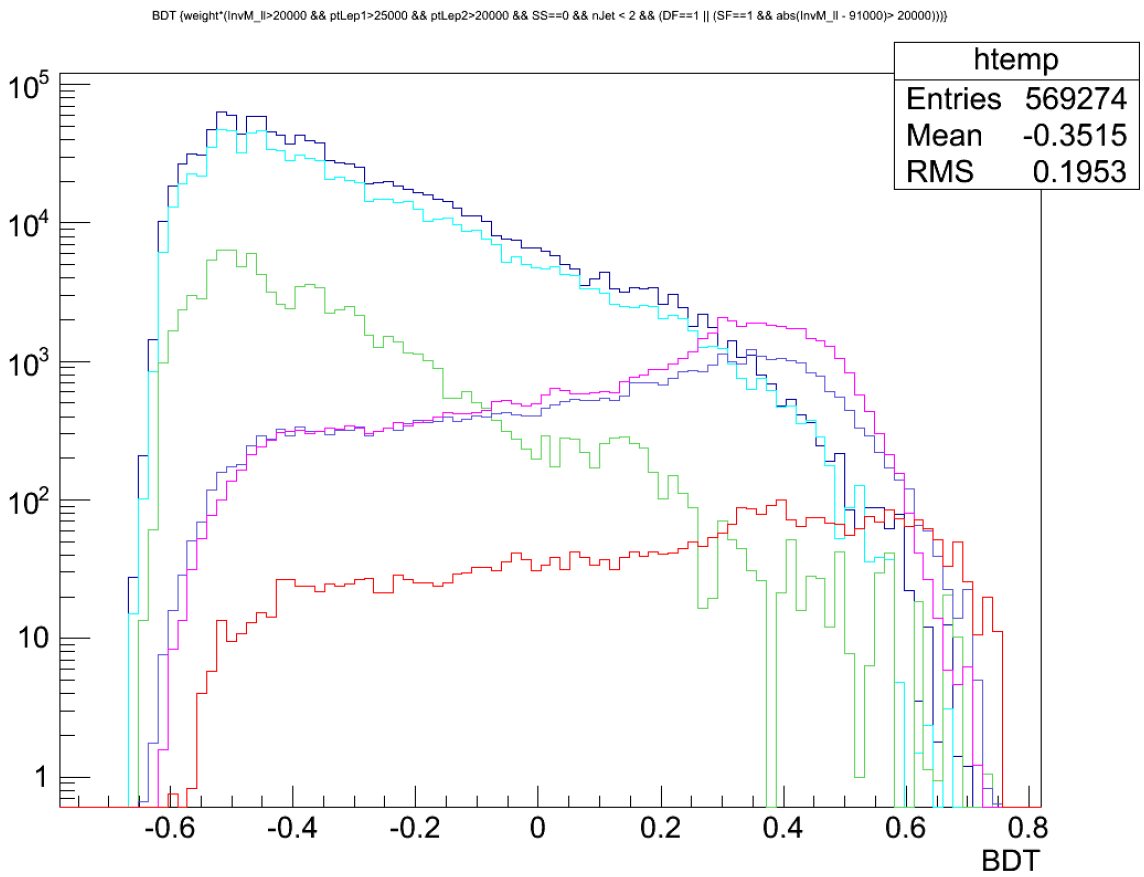


Variable ranking (method specific)

```
: Ranking result (top variable is best ranked)
: ----------------------------------------------------------------
: Rank : Variable                    : Variable Importance
: ----------------------------------------------------------------
:    1 : met_over_meff               : 1.454e-01
:    2 : EtMiss_tst                  : 1.120e-01
:    3 : MT2                         : 9.695e-02
:    4 : Mtransv1                    : 9.234e-02
:    5 : abs(phiLep1-phiLep2)        : 9.053e-02
:    6 : InvM_ll                     : 8.980e-02
:    7 : abs(etaLep1-etaLep2)        : 7.088e-02
:    8 : abs(phiLep1-EtMiss_tstPhi)  : 6.112e-02
:    9 : nJet                        : 5.701e-02
:   10 : pbllmod                     : 5.657e-02
:   11 : abs(phiLep2-EtMiss_tstPhi)  : 5.519e-02
:   12 : ptLep2                      : 3.735e-02
:   13 : ptLep1                      : 3.491e-02
: ----------------------------------------------------------------
```

Analysis comparison: BDT

- **Signal(100,25)**
- $t\bar{t}$
- $2l2\nu$
- $Z \to ee$
- $Z \to \mu\mu$
- $Z \to \tau\tau$



Analysis comparison: BDT (vs Cut&Count)

Cuts	Signal (100,25)	$t\bar{t}$	VV $2l2\nu$	$Z \to l^+l^-$ $l = e, \mu, \tau$	Signif. $S/\sqrt{S+B}$
After Preselection	3370.2	51276.1	37075.5	2047746.8	
$BDT > 0.5$	820.4	2676.3	1942.7	970.4	**10.2**
$BDT > 0.62$	295.3	74.3	192.9	77.8	**11.7**
$BDT > 0.63$	255.0	56.6	159.4	65.4	**11.0**
$BDT > 0.65$	176.5	26.5	87.8	49.1	**9.6**
$BDT > 0.65$ && $nJet = 0$	176.5	26.5	87.8	49.1	**9.6**
$BDT > 0.65$ && $nJet = 1$	0	0	0	0	

Back-up Slides

Ada(ptive) Boost: The most popular boosting algorithm

Misclassified events during the training of a decision tree are given a **higher event weight** in the training of the following one.

Starting with the original event weights when training the first decision tree, the subsequent one is trained using a **modified** event sample where the weights of previously misclassified events are multiplied by a common ***boost weight*** α; the boost weight is derived from the **misclassification rate** E of the previous tree, $\alpha = (1 - E)/E$

The result of an individual classifier is $h(\vec{x})$, with $\vec{x}$ being the n-tuple of input variables, encoded for Signal and Background as $h(\vec{x}) = +1$ and -1, respectively.

AdaBoost performs **best on weak classifiers**, but it lacks robustness in presence of outliers or mislabelled data points, because of exponential loss.

Gradient Boost

Let $F(\vec{x})$ be the function under estimation, assumed to be a weighted sum of parametrised base functions $f(x; a_m)$, so-called **weak-learners**. Thus each base function in this expansion corresponds to a decision tree

$$F(\vec{x}|P) = \sum_{m=0}^{M} \beta_m f(x; a_m) \qquad P \in \{\beta_m; a_m\}$$

The boosting procedure is employed to adjust the parameters P such that the deviation between the model response $F(\vec{x})$ and the true value y obtained from the training sample is **minimised**. The **deviation** is measured by the ***loss-function*** $L(F, y)$

The AdaBoost exponential loss leads to some reweighting algorithm in regression trees. The GradientBoost algorithm attempts to cure this weakness by implementing the *binomial log-likelihood* loss:

$$L(F, y) = \ln(1 + e^{-2F(\vec{x})y})$$

Bagging and Randomised Trees

Bagging: A resampling technique where a classifier is repeatedly trained using resampled training events such that the combined classifier represents an **average** of the individual classifiers.

Bagging does not aim at enhancing a weak classifier, but it effectively smears over statistical representations of the training data and is hence suited to **stabilise** the response of a classifier, w.r.t. **statistical fluctuations** in the training sample.

Randomised Trees: Each tree is grown in such a way that at each split only a **random subset** of all variables is considered. Moreover, each tree in the forest is grown using only a (resampled) subset of the original training events.