Statistical Tools in Collider Experiments

Multivariate analysis
in high energy physics

Lecture 2

Pauli Lectures - 07/02/2012

Nicolas Chanon - ETH Zürich

ETH Institute for
Particle Physics

Outline

1. Introduction
2. Multivariate methods
3. Optimization of MVA methods
4. Application of MVA methods in HEP
5. Understanding Tevatron and LHC results

Lecture 2. Multivariate methods

Multivariate analysis : Definitions

MultiVariate Analysis :

- Set of statistical analysis methods that simultaneously analyze multiple measurements (variables) on the object studied
- Variables can be dependent or correlated in various ways

Classification / regression :

- **Classification** : discriminant analysis to separate classes of events, given already known results on a training sample
- **Regression** : analysis which provides an output variable taken into account the correlations of the input variables

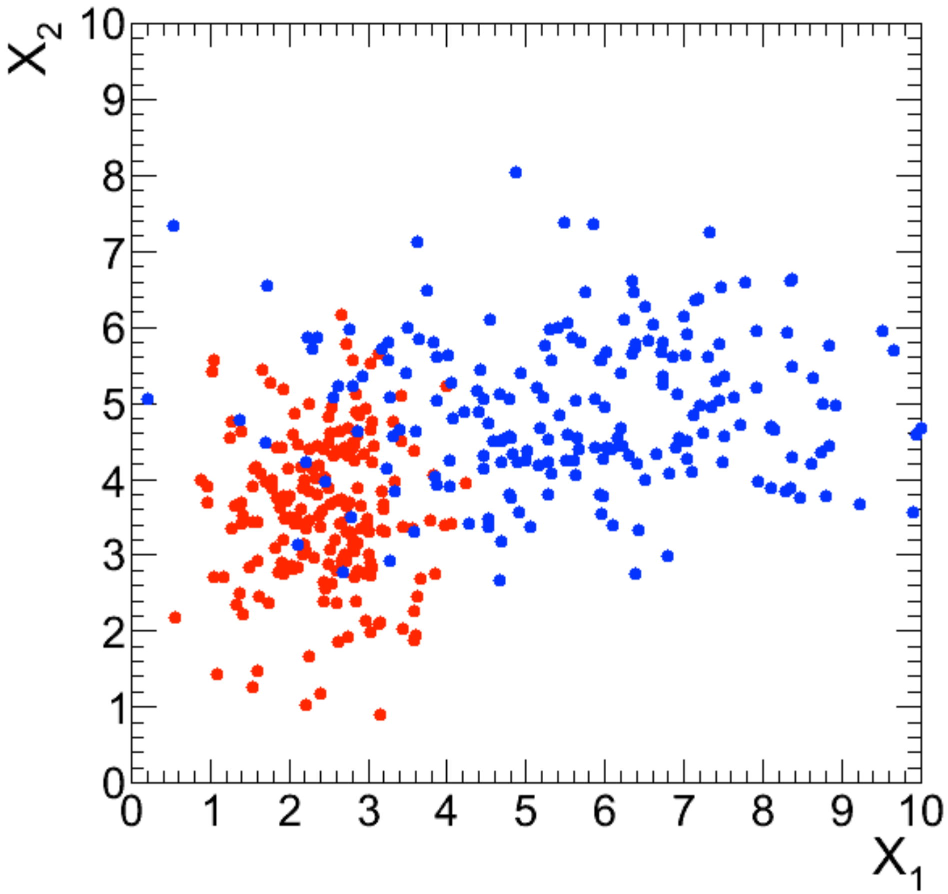
Statistical learning :

- **Supervised learning :** the multivariate method is trained over a sample were the result is known (e.g. Monte-Carlo simulation of signal and background)
- **Unsupervised learning :** no prior knowledge is required. The algorithm will cluster events in an optimal way

Event classification

- Focus here on **supervised learning for classification**.
- Use case in particle physics : **signal/background discrimination**

- Assume we have two populations (signal and background) and two variables

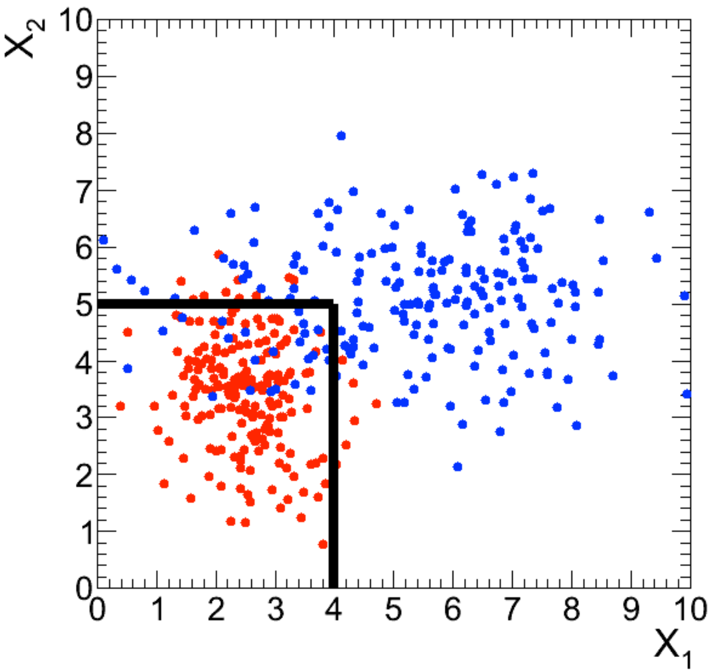


- How to decorrelate, what decision boundary (on X1 and X2) to choose, to decide if an event is signal or background ?

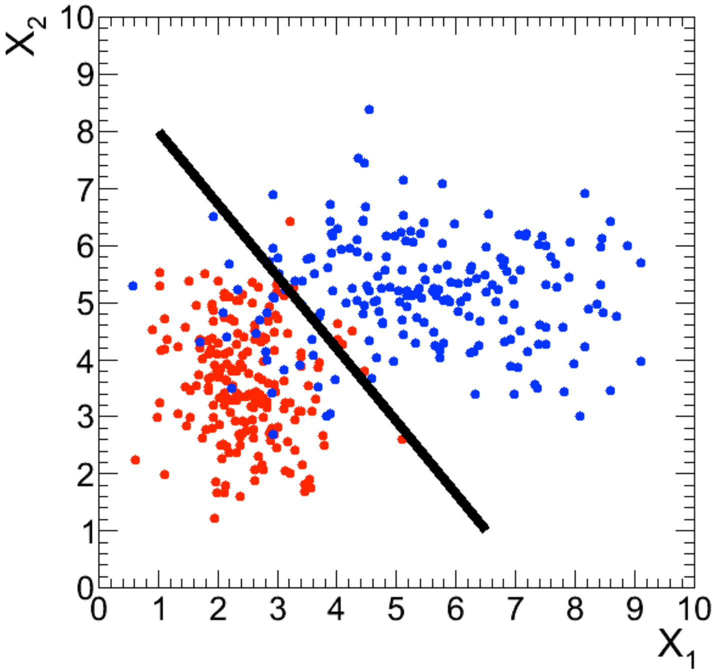
Event classification

- **Possible solutions :** rectangular cuts, Fisher, non-linear contour

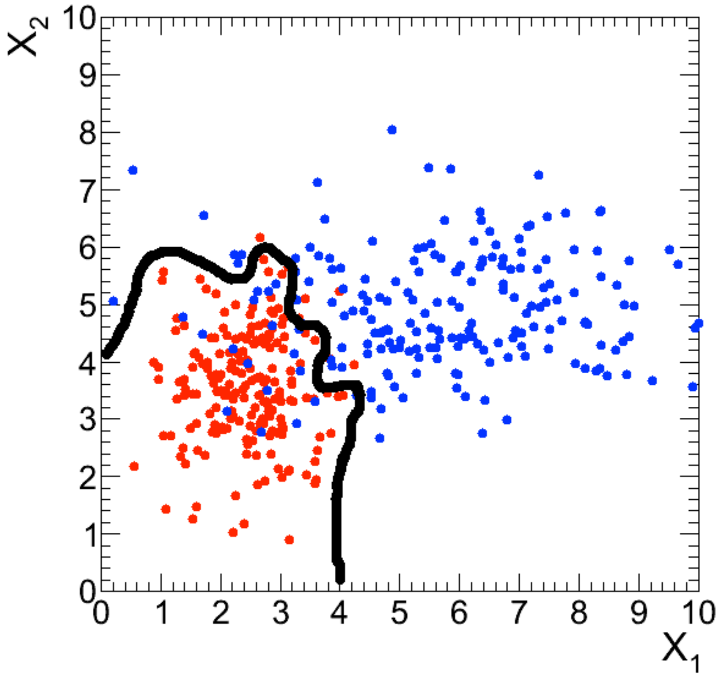
Rectangular cuts



Linear (Fisher)

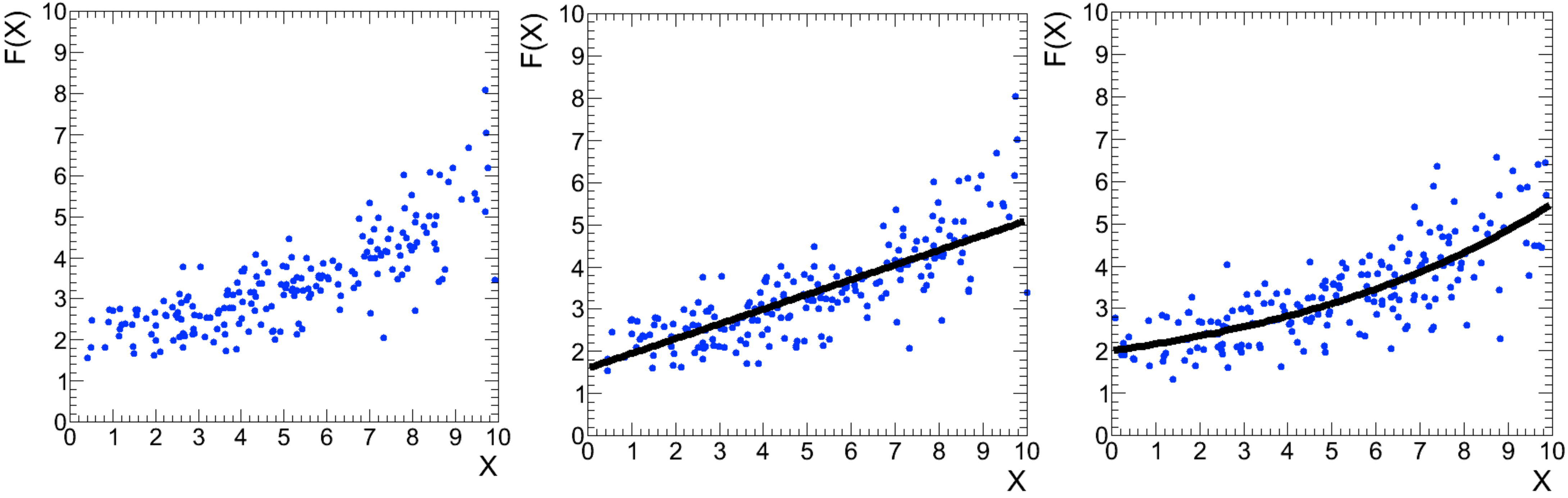


Non-linear (BDT, NN...)



Regression

- Assume we have one set of measurements.
- How to approximate the law underlying such measurement ?
- If the value of the function in each point is known, this is an example of **supervised regression**.
- If F(X) is not known this is an example of **unsupervised regression**



Plenty of multivariate methods...

Example of MVA methods :

- Rectangular cut optimization
- Fisher
- Likelihood

- Neural network

- Decision tree
- Support Vector Machine
- ...

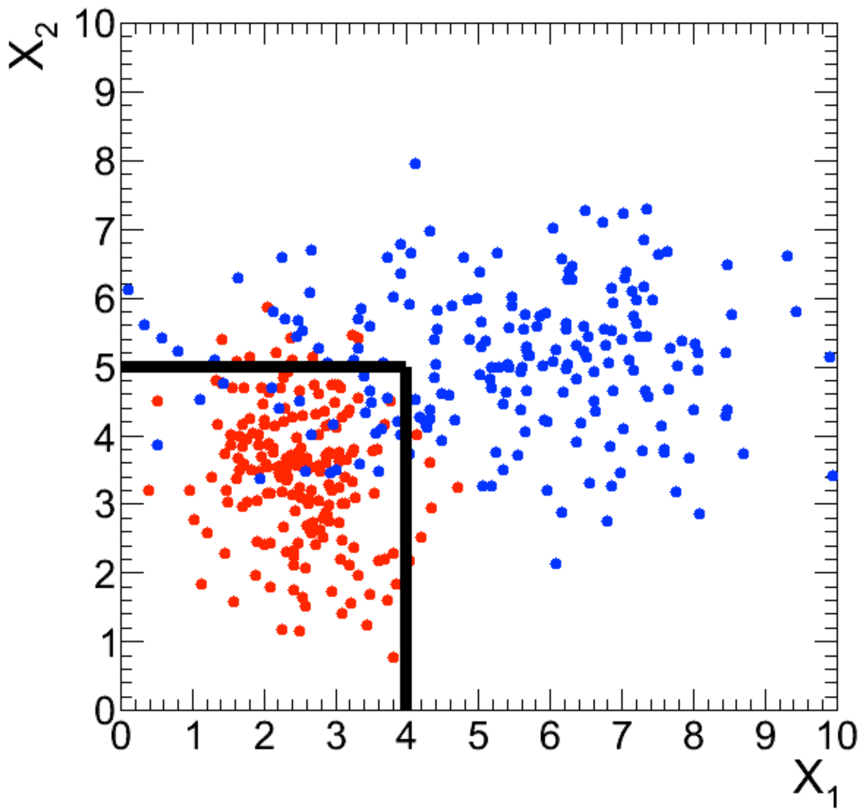
Characteristics :
- Level of complexity and transparency
- Performance in term of background rejection
- Way of dealing with non-linear correlations
- Speed of training
- Robustness while increasing the number of input variables
- Robustness against overtraining

Rectangular cuts

- Simplest multivariate method, very intuitive
- All HEP analyses are using rectangular cuts, not always completely optimized

Rectangular cuts optimization :
- Grid search, Monte-Carlo sampling
- Genetic algorithm
- Simulated annealing

Characteristics :
- Difficult to discriminate signal from background if too much correlations
- Optimization difficult to handle with high number of variables



Define the signal region :

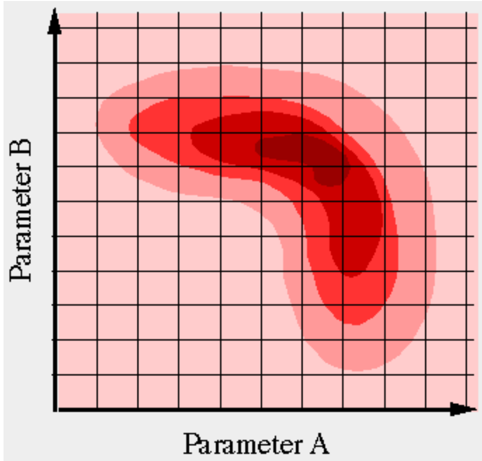
a1 < x1 < a2,

b1 < x2 < b2

...

Cut optimization

How to find the best set of cuts for a given criterion ?



Grid search

- Try N points (usually very large) of the phase-space equally spaced in each dimensions

=> Impossible with high number of variables (too much CPU time)

Monte-Carlo sampling

- Try N points randomly chosen in the phase space

=> Usually performs better, but still non optimal

Both are good global minimum finder but have poor accuracy

Examples of criterion :

- Maximize the signal efficiency for a given background rejection
- Maximize the significance

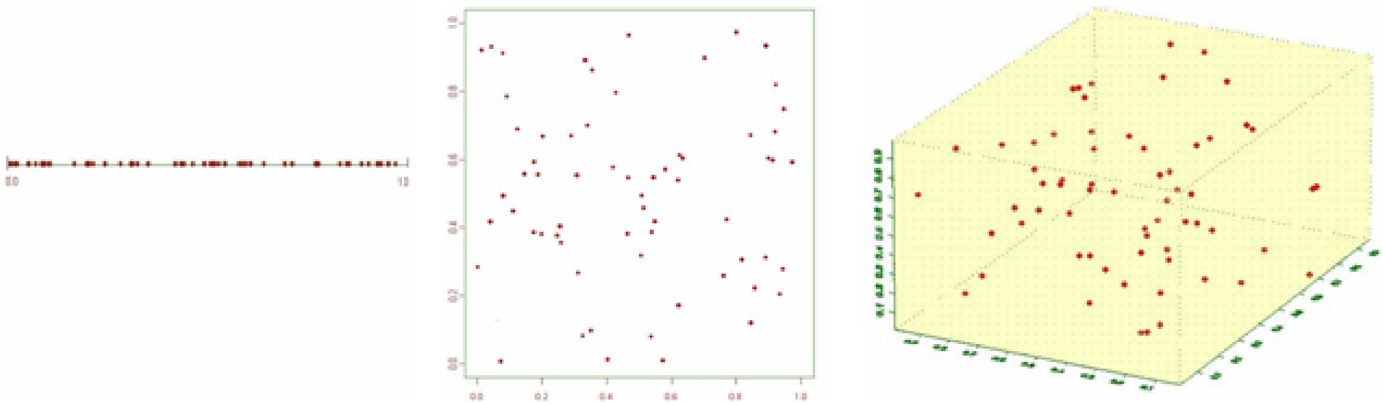
Curse of dimensionality

Grid search and Monte-Carlo sampling suffer from the curse of dimensionality :

- For one variables, trying 100 working points is easy

- For two variables, 100 working points will lead to not well covered phase-space because each points have more distance between them
- 100x100 points should be used

- Increasing number of variables will lead this algorithm to be impossible in practice

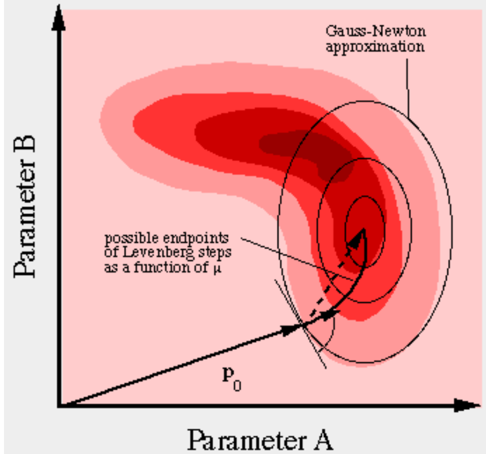


Optimization methods

Quadratic interpolation

- Compute the function (say the significance) in 3 points. Interpolate with a quadratic function and go to the minimum. Repeat the operation.

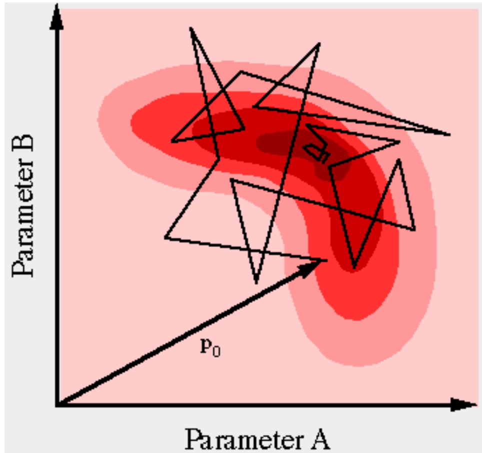
=> Problem if no minimum but a maximum is found (work around exist)



Gradient descent

- At each point, go in the gradient direction. This should lead to a minimum.

=> This method is not the fastest since the gradient direction *at each step* is not always the direction of the minimum.

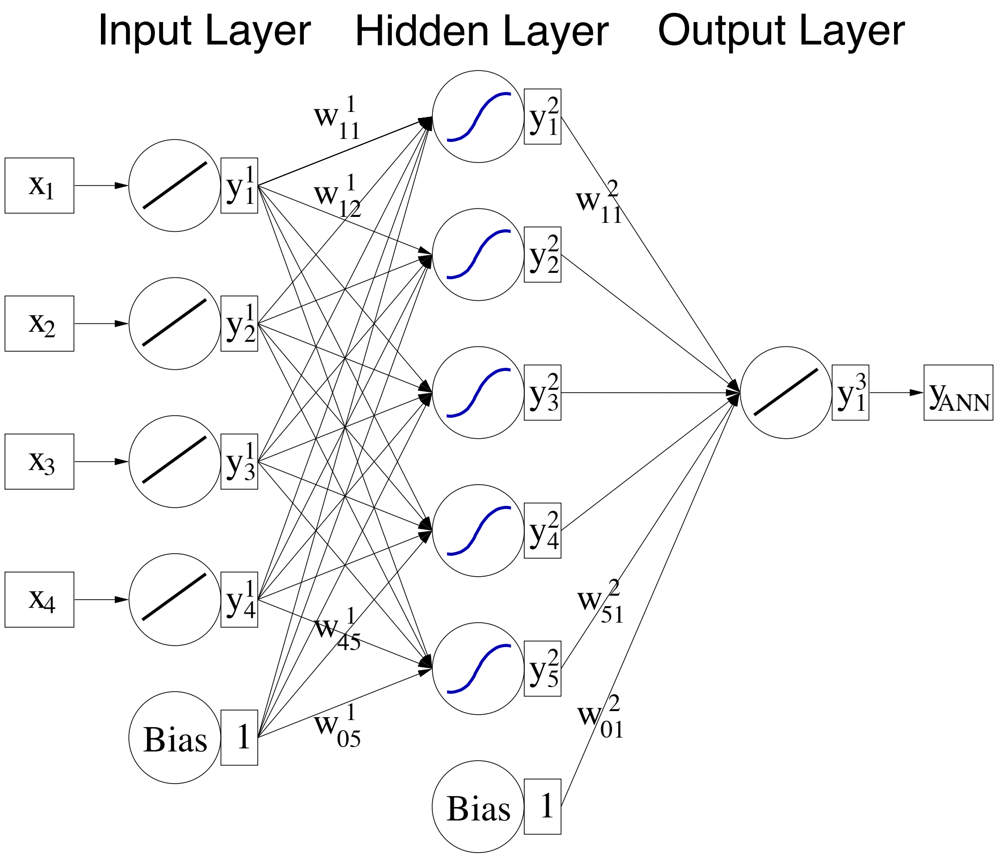


Both methods are good to find local minima

- **MINUIT package** uses a combination : gradient-driven search, using variable metric, can use quadratic Newton-type solution
- Other methods exist : **genetic algorithms, simulated annealing**

Neural network

- Most commonly used : the **multi-layer perceptron**
- Composed of **neurons** taking as input a linear combination of the previous neuron outputs
- **Activation function** (usually tanh) transforms the linear combination
- **Weights** for each neurons are found during the training phase by minimizing the error on the neural network output



- Neural networks are universal approximators : takes advantage of correlations

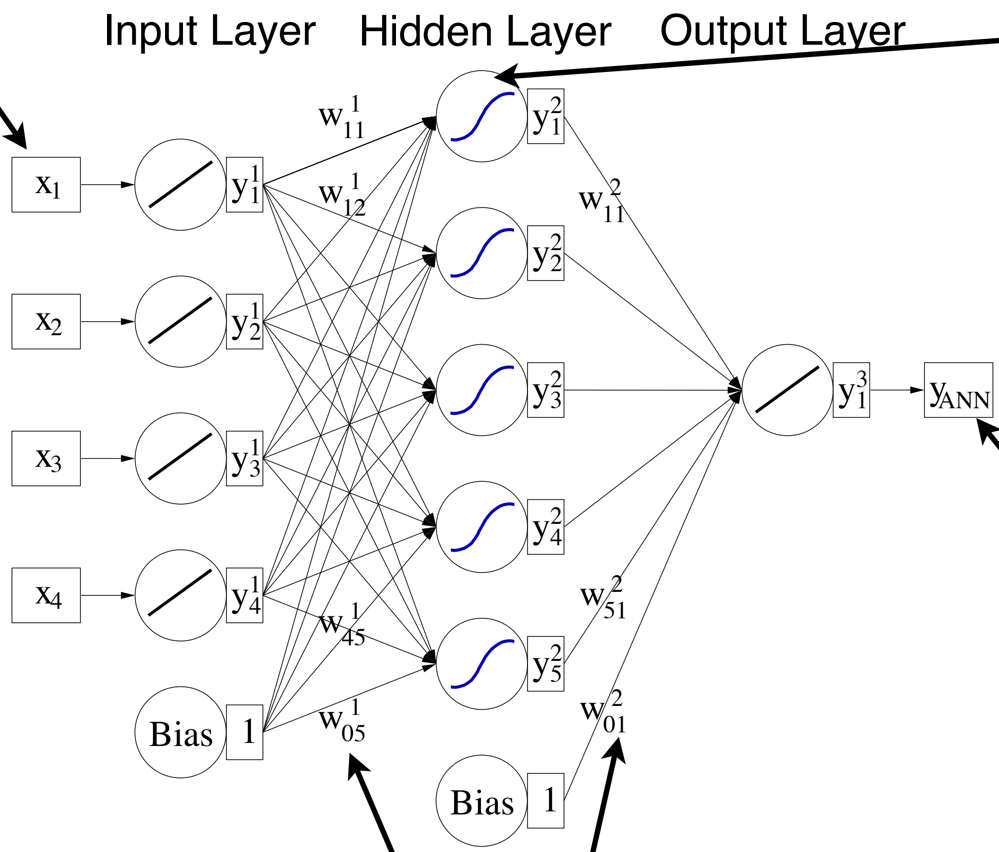
- Quite stable against overtraining and against increasing number of variables

Neural network : structure

Hidden layer

Input variables

Activation functions



Multi-layer perceptron : most popular neural network

- Here : only one hidden layer

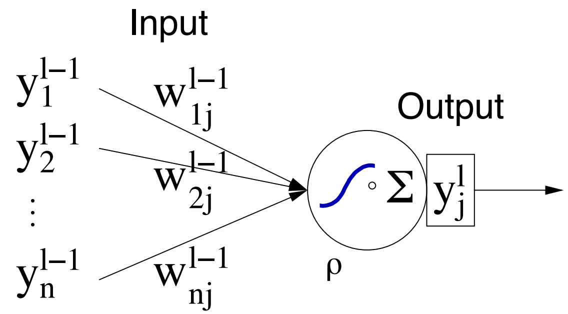
Output variable

Weights used for the linear combination

Neural network : structure

Given input values for the variables, how to compute the output ?

- Start from a set of **input variables** fed to the **input layer**



- For each neuron in the **hidden layer** :
- Compute a **weighted sum** of the input variables (linear combination) fed as input to the hidden neuron

- Transform the input with an **activation function** : usually tanh or sigmoid

$$x \to \begin{cases} x & \mathit{Linear}, \\ \frac{1}{1+e^{-kx}} & \mathit{Sigmoid}, \\ \frac{e^x - e^{-x}}{e^x + e^{-x}} & \mathit{Tanh}, \\ e^{-x^2/2} & \mathit{Radial}. \end{cases}$$

- If there is more hidden layers, repeat the operation for each neuron of the new hidden layer, taken as input the output of the previous layer

- The output layer performs a weighted sum of the previous hidden layer output

$$y_{\mathrm{ANN}} = \sum_{j=1}^{n_{\mathrm{h}}} y_j^{(2)} w_{j1}^{(2)} = \sum_{j=1}^{n_{\mathrm{h}}} \tanh\left(\sum_{i=1}^{n_{\mathrm{var}}} x_i w_{ij}^{(1)} \right) \cdot w_{j1}^{(2)}$$

Neural network : training

How to compute the weights ?

- By minimization of the error, defined as : $\sum_{a=1}^{N} \frac{1}{2} \left(y_{\mathrm{ANN},a} - \hat{y}_a\right)^2$

Where yANN is the output and ŷ is the desired response : -1 for background, +1 for signal.

Remember that we have : $y_{\mathrm{ANN}} = \sum_{j=1}^{n_\mathrm{h}} y_j^{(2)} w_{j1}^{(2)} = \sum_{j=1}^{n_\mathrm{h}} \tanh\left(\sum_{i=1}^{n_\mathrm{var}} x_i w_{ij}^{(1)}\right) \cdot w_{j1}^{(2)}$

We will minimize the error using the gradient descent method : this is called the back-propagation of errors :

$$\mathbf{w}^{(\rho+1)} = \mathbf{w}^{(\rho)} - \eta \boldsymbol{\nabla}_{\mathbf{w}} E$$

Weights connected to the output layer are updated by :

$$\Delta w_{j1}^{(2)} = -\eta \sum_{a=1}^{N} \frac{\partial E_a}{\partial w_{j1}^{(2)}} = -\eta \sum_{a=1}^{N} \left(y_{\mathrm{ANN},a} - \hat{y}_a\right) y_{j,a}^{(2)}$$

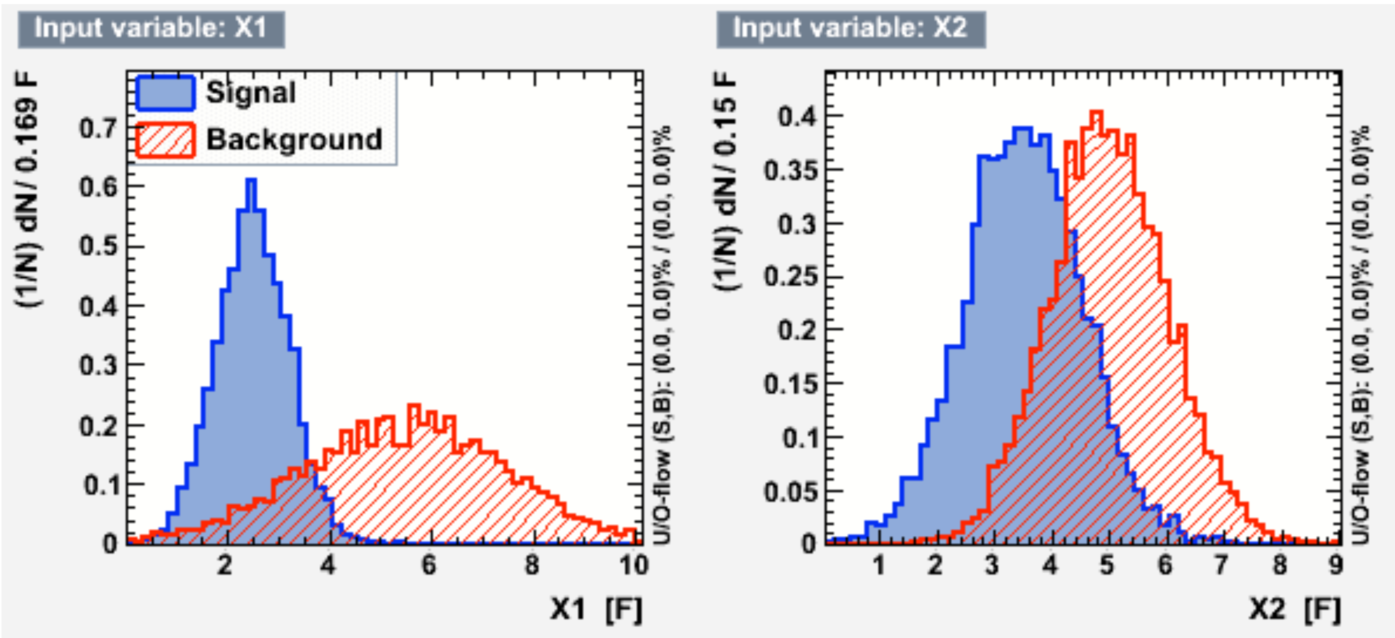
And weights connected to the hidden layer are therefore updated with :

$$\Delta w_{ij}^{(1)} = -\eta \sum_{a=1}^{N} \frac{\partial E_a}{\partial w_{ij}^{(1)}} = -\eta \sum_{a=1}^{N} \left(y_{\mathrm{ANN},a} - \hat{y}_a\right) y_{j,a}^{(2)}(1 - y_{j,a}^{(2)}) w_{j1}^{(2)} x_{i,a}$$

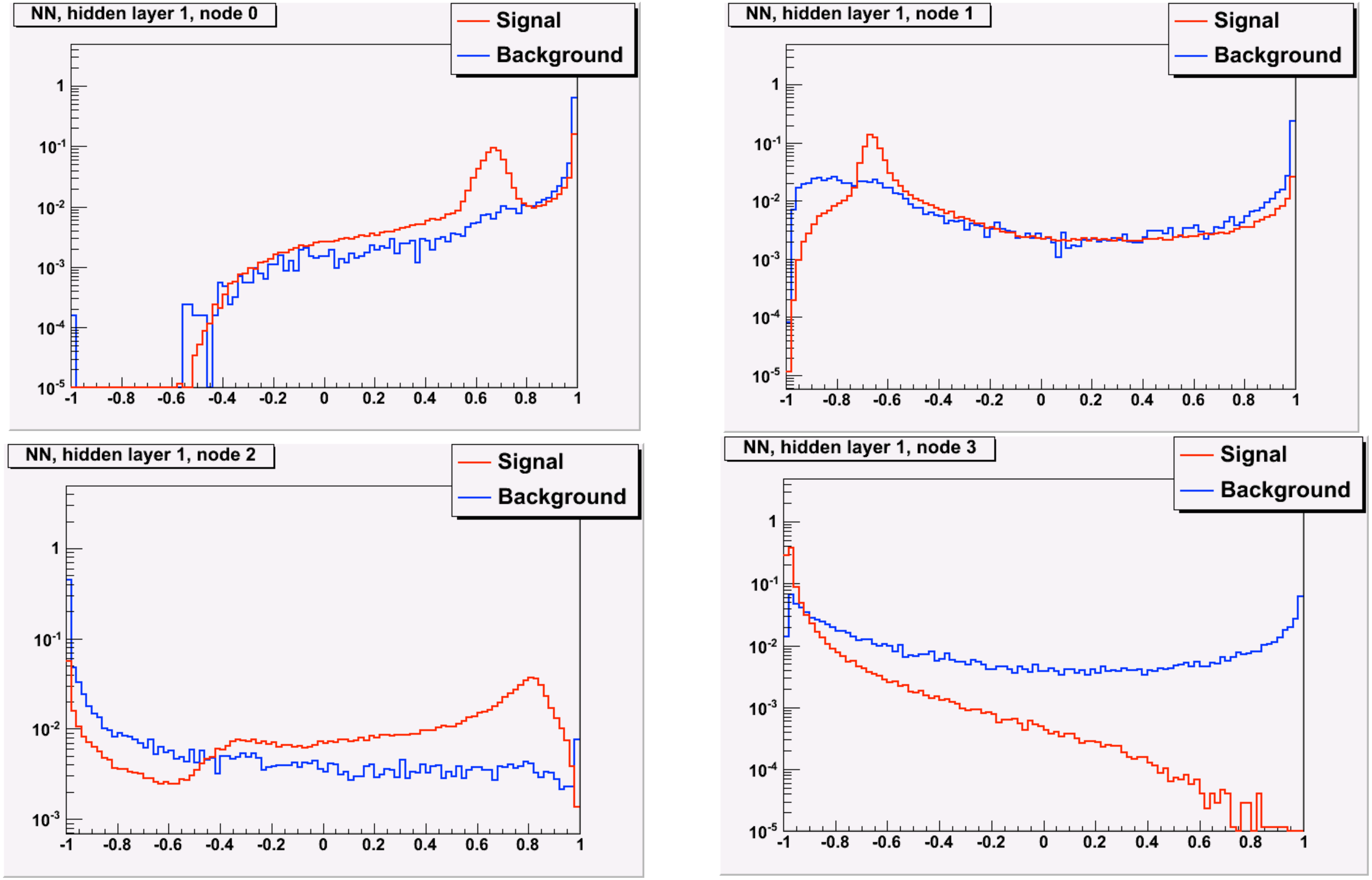
Neural network : input

Input variables :

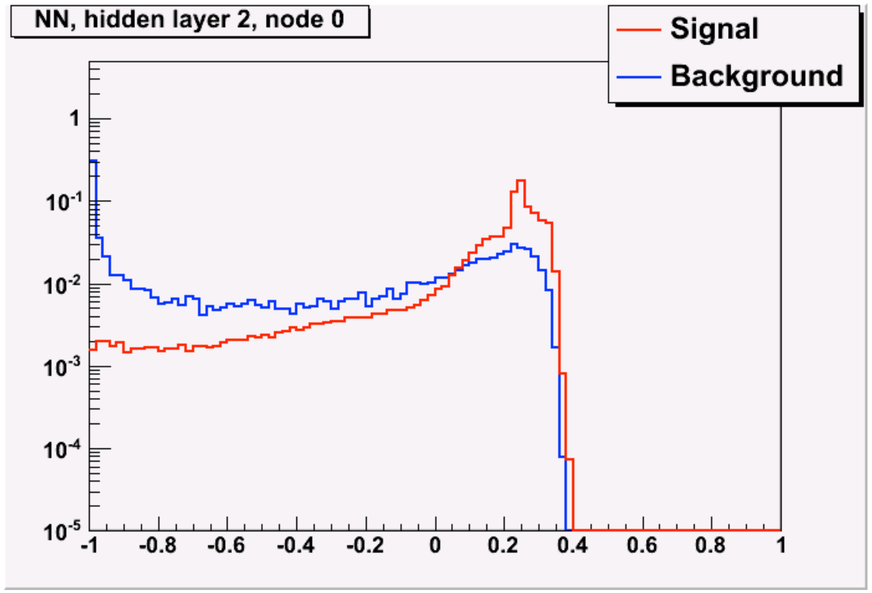
- Can be correlated (NN uses correlations)
- To improve the NN performance, should avoid unuseful variables (too much correlated, too low discrimination power)
- They can be transformed to improve their discrimination power before the training

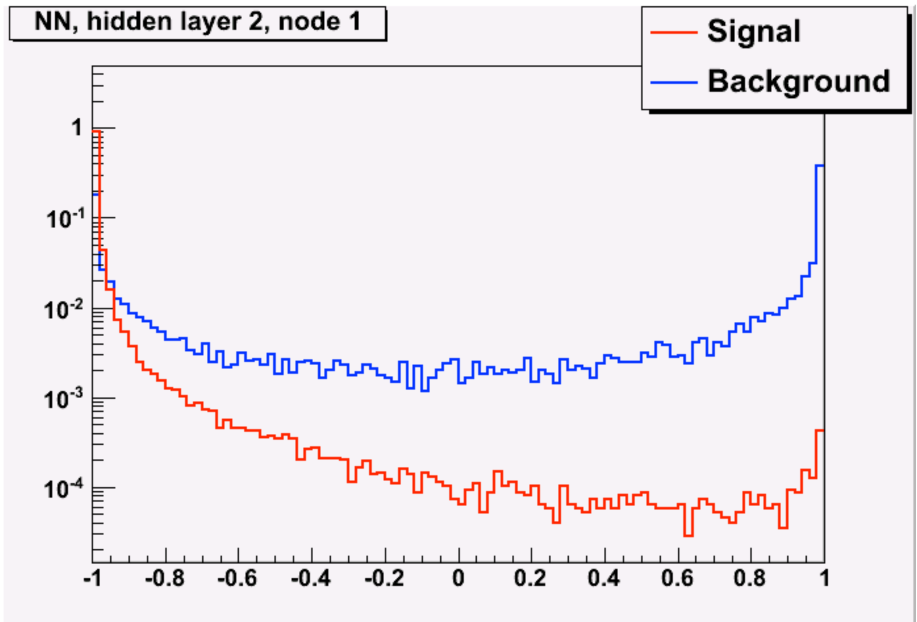


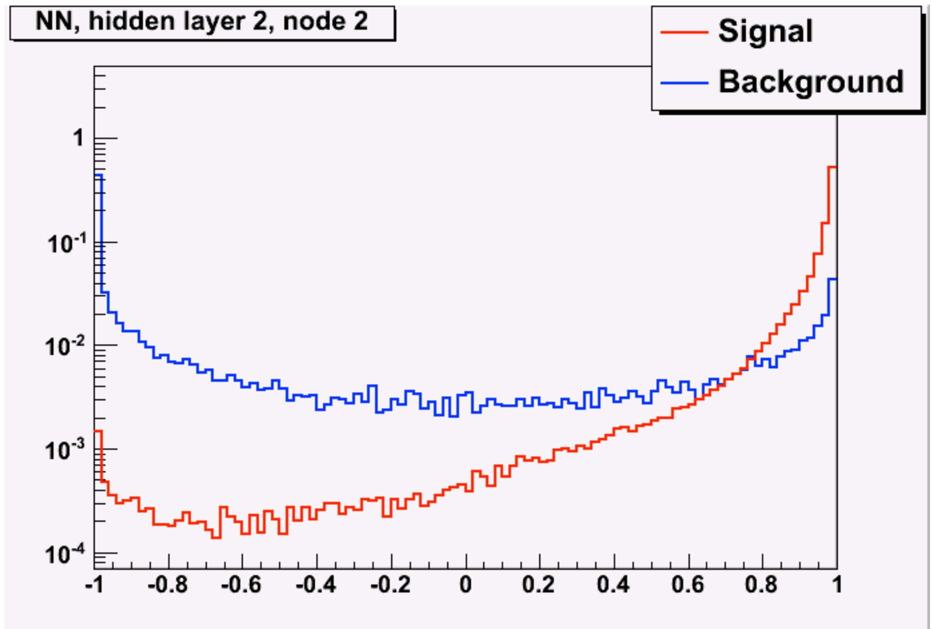
Neural network : neurons



Neural network : neurons

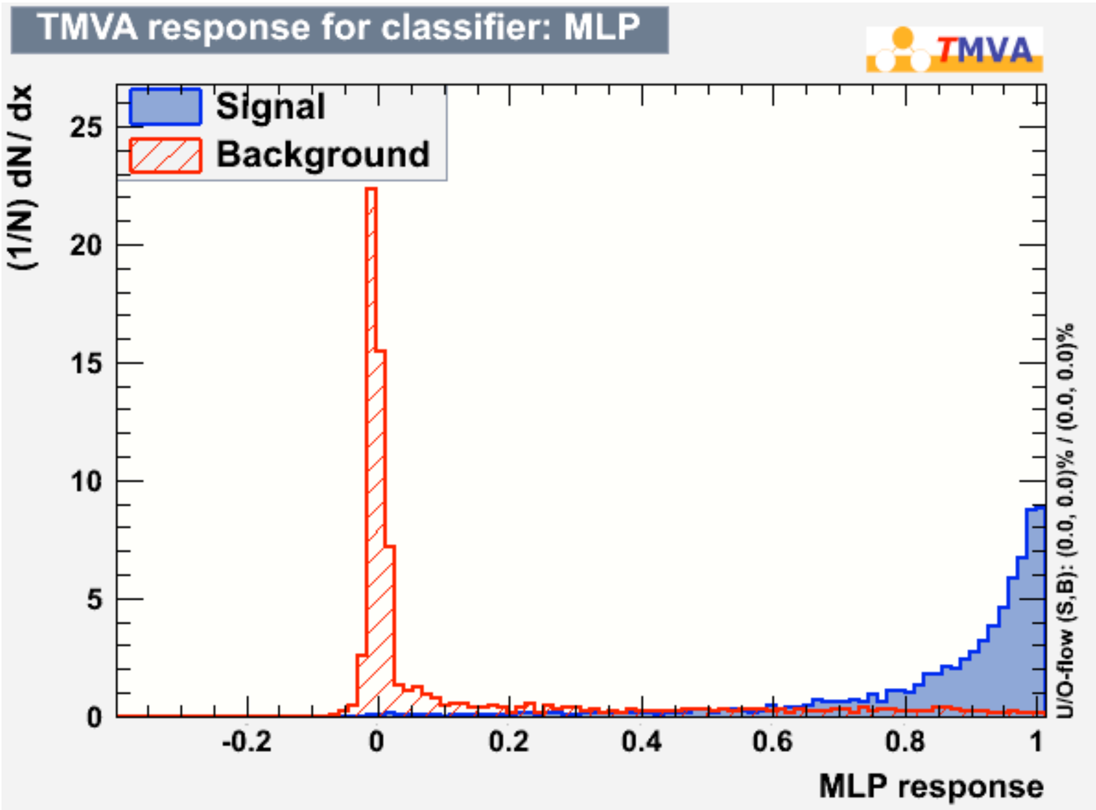






Neural network : output

- The neural network output can be real or integer
- For most of the HEP applications it is more interesting to have a a real-valued variable
- If the training is successful, background should peak at -1 (or 0) and signal at +1
- Shape depends a lot on the NN parameters (layers, epochs...)
- Discrimination power achieved depend a lot on the problems.

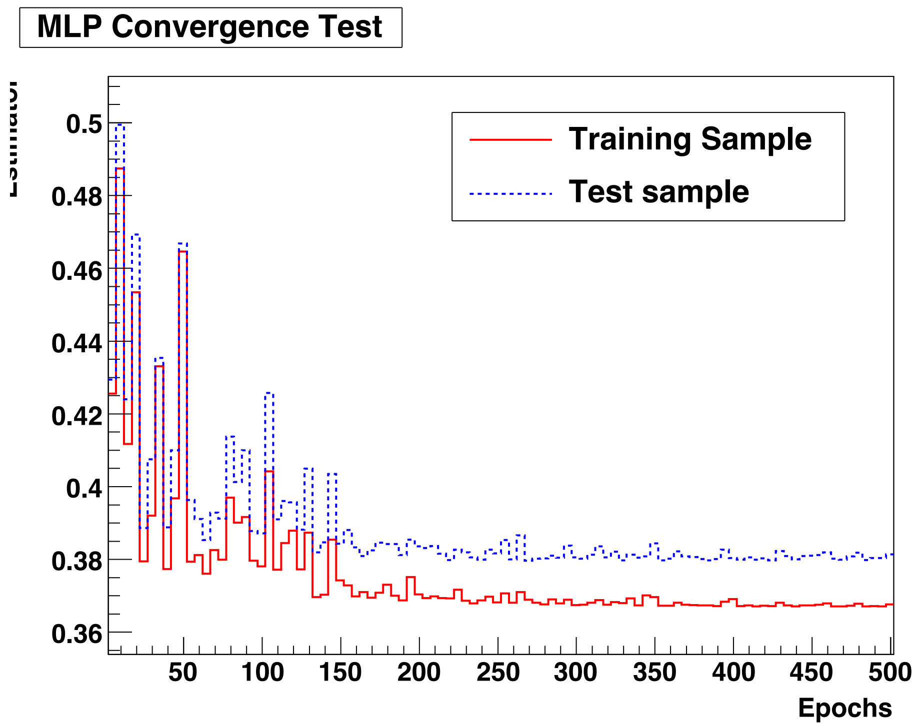


Neural network : error

- Training error : $$\sum_{a=1}^{N} \frac{1}{2} \left(y_{\mathrm{ANN},a} - \hat{y}_a\right)^2$$

- One can compare, at each iteration (epoch), what is the NN error for the training and the test sample.

- Errors decrease with epochs in both training and test samples.
- Usually it stabilizes
- But with more epochs, it can happen that the test sample will have an error which will increase again

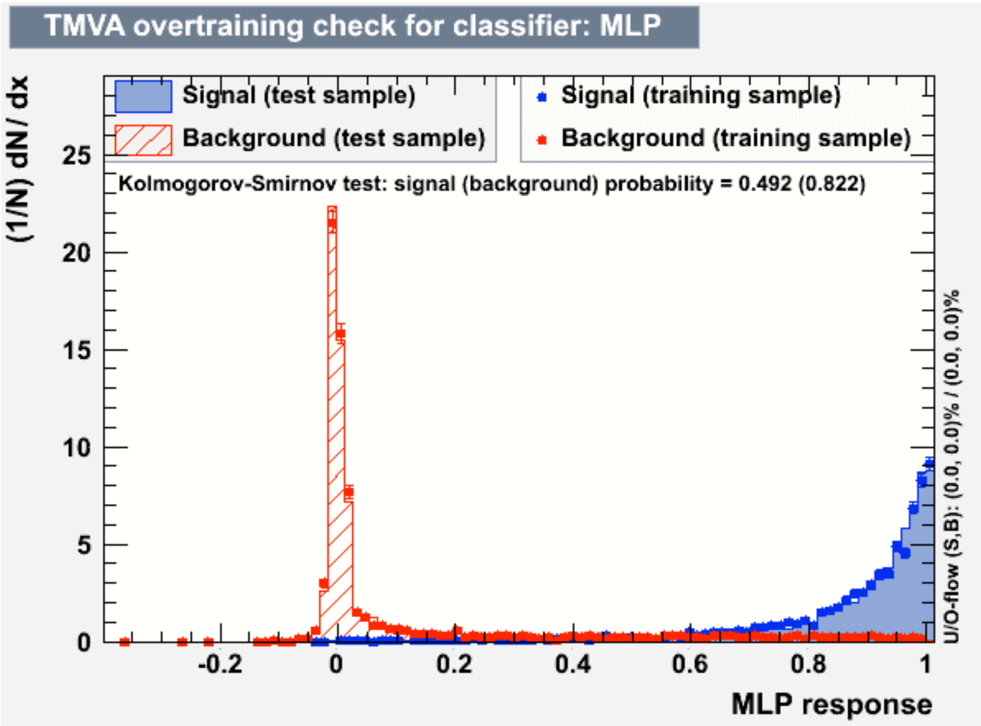


=> **Overtraining** :
- The neural network was trained to recognize even the statistical fluctuations of the training sample and is therefore not suitable for any test sample

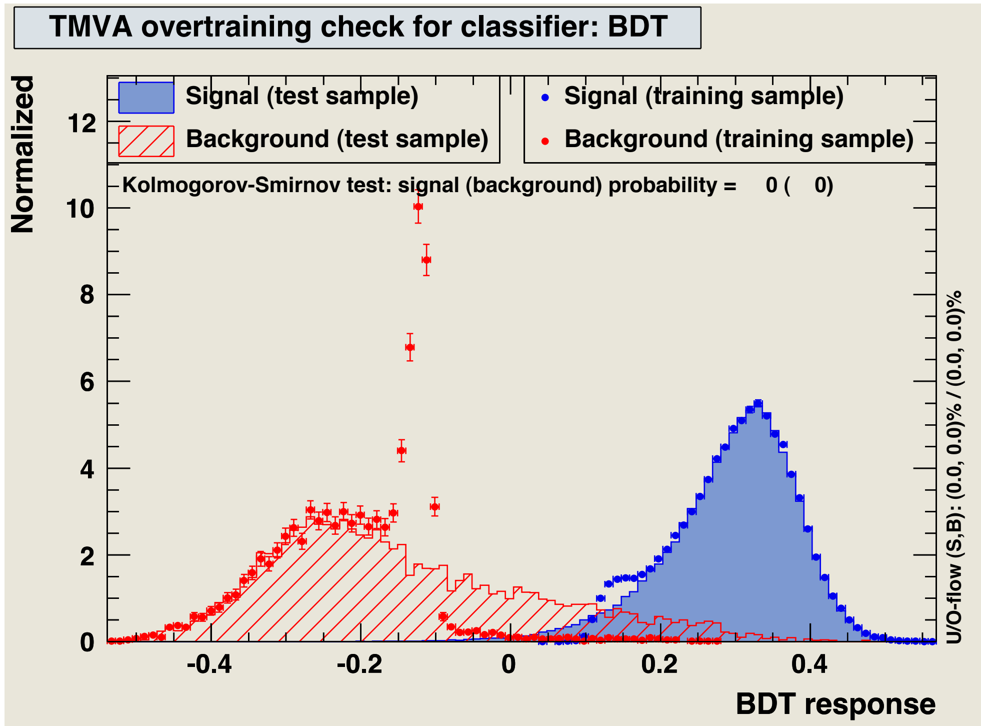
Neural network : overtraining

- Simple check : NN output for the training and test sample.
- Both samples should have the same shape, with the statistical uncertainties

Not overtrained



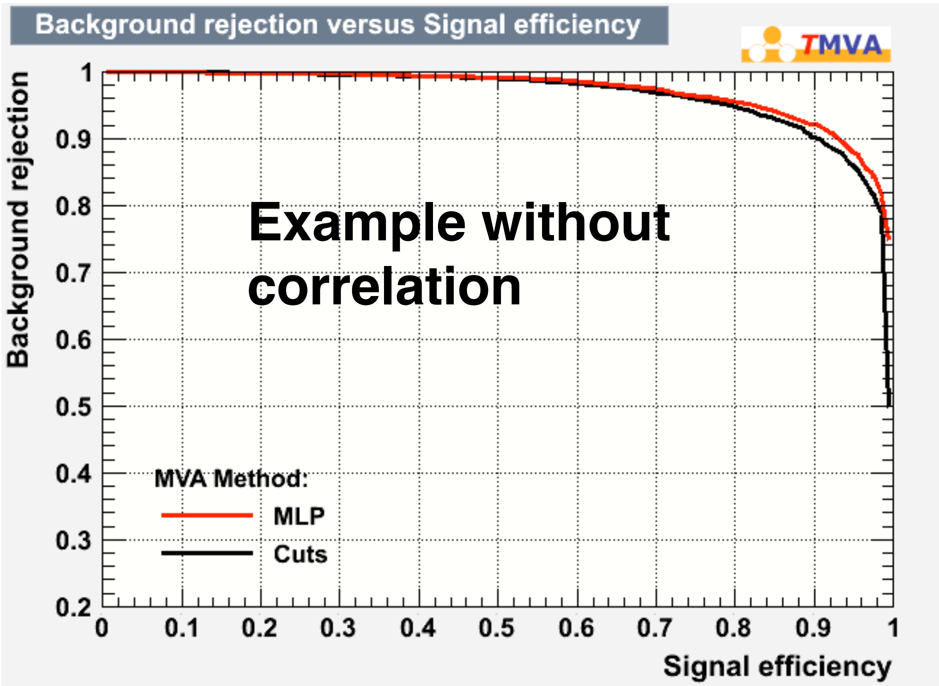
Overtrained

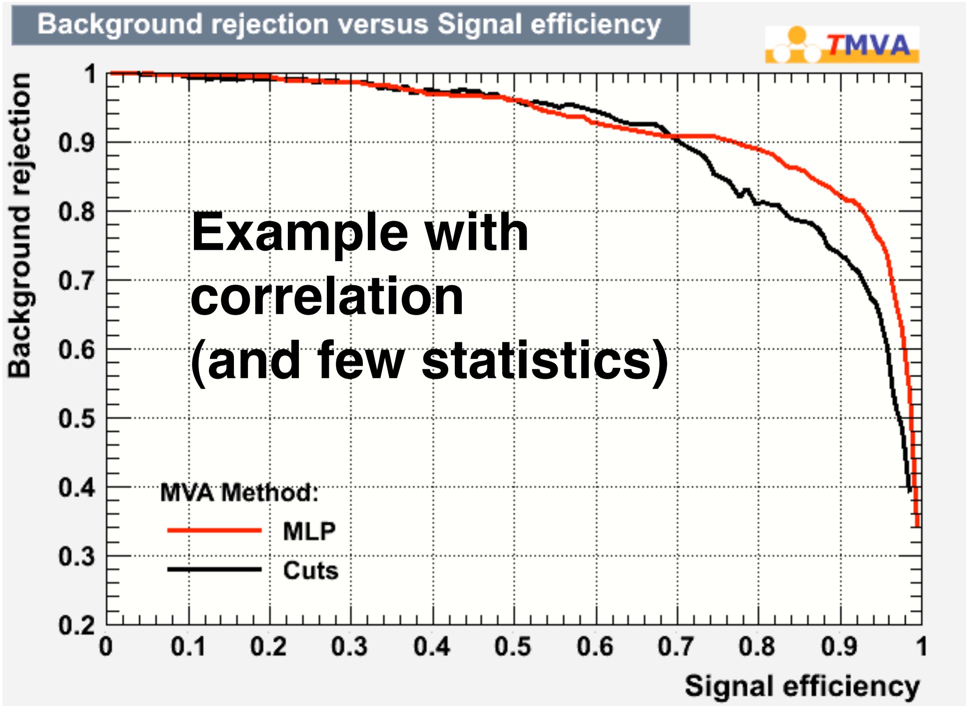


Neural network : performance

Usual figure of merit to check the performance :

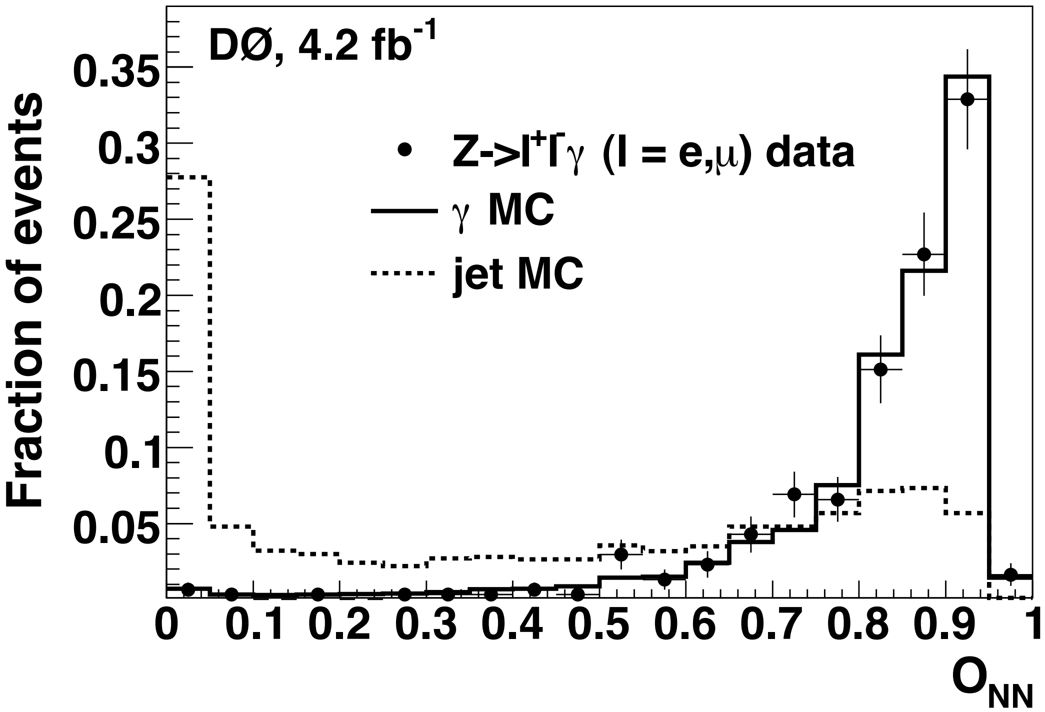
- Scan the performance varying the cut on the network output
- Plot the signal efficiency versus background efficiency (or background rejection). Each cut on the NN output is one point on the figure.
- The NN performs (almost all the time) better than the rectangular cut

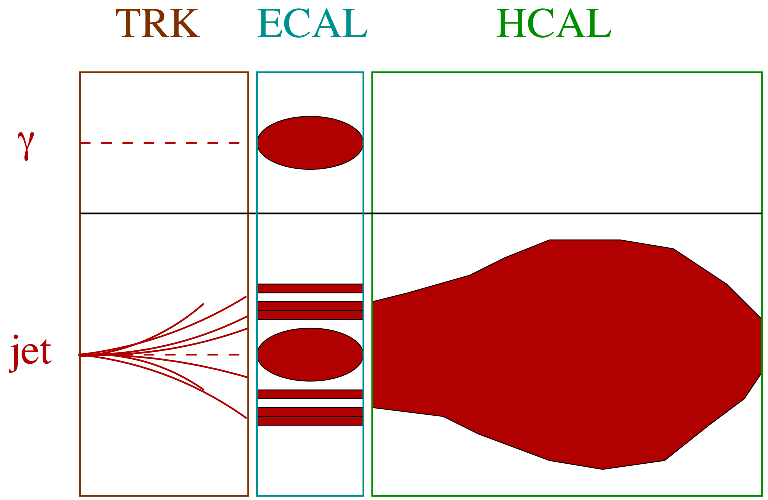


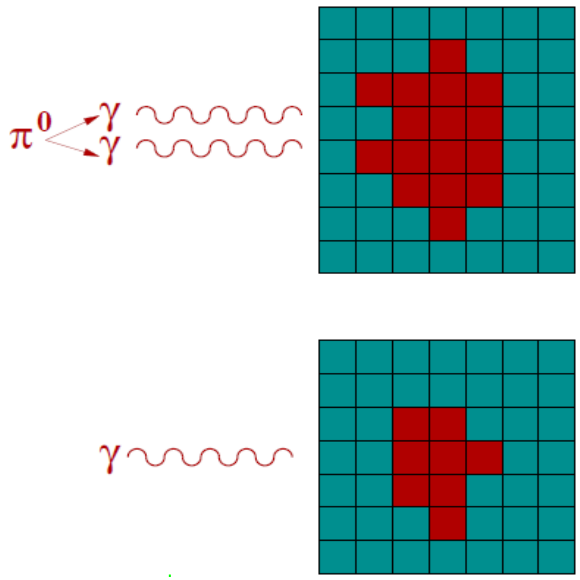


Neural network : examples in HEP

Photon identification at D0 and applications







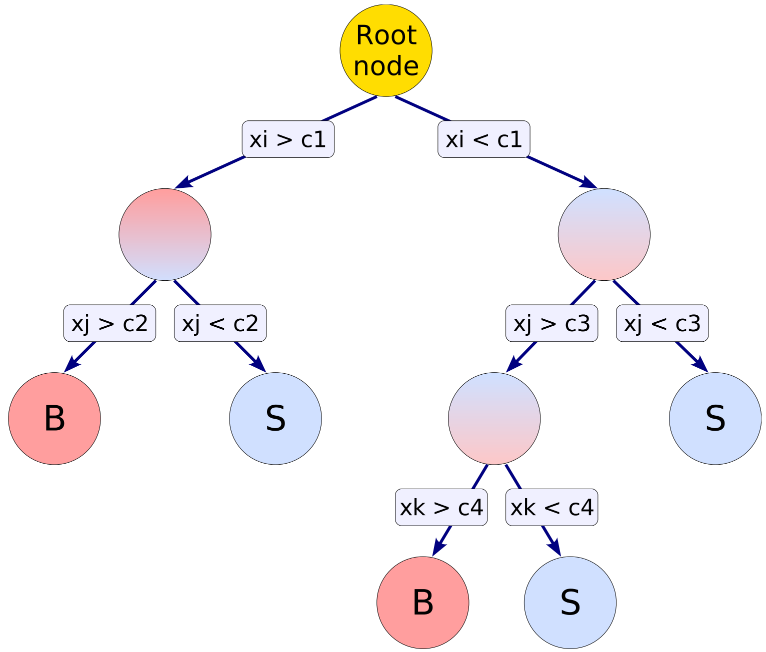
Goal : discriminate photons against neutral mesons in jets

Neural network input variables :

- Shape of the calorimeter energy deposit
- Track variables in an isolation cone around the photon

Decision tree

- A **decision tree** is a binary tree : a sequence of cuts paving the phase-space of the input variables
- Repeated yes/no decisions on each variables are taken for an event until a stop criterion is fulfilled
- Trained to maximize the purity of signal nodes (or the impurity of background nodes)

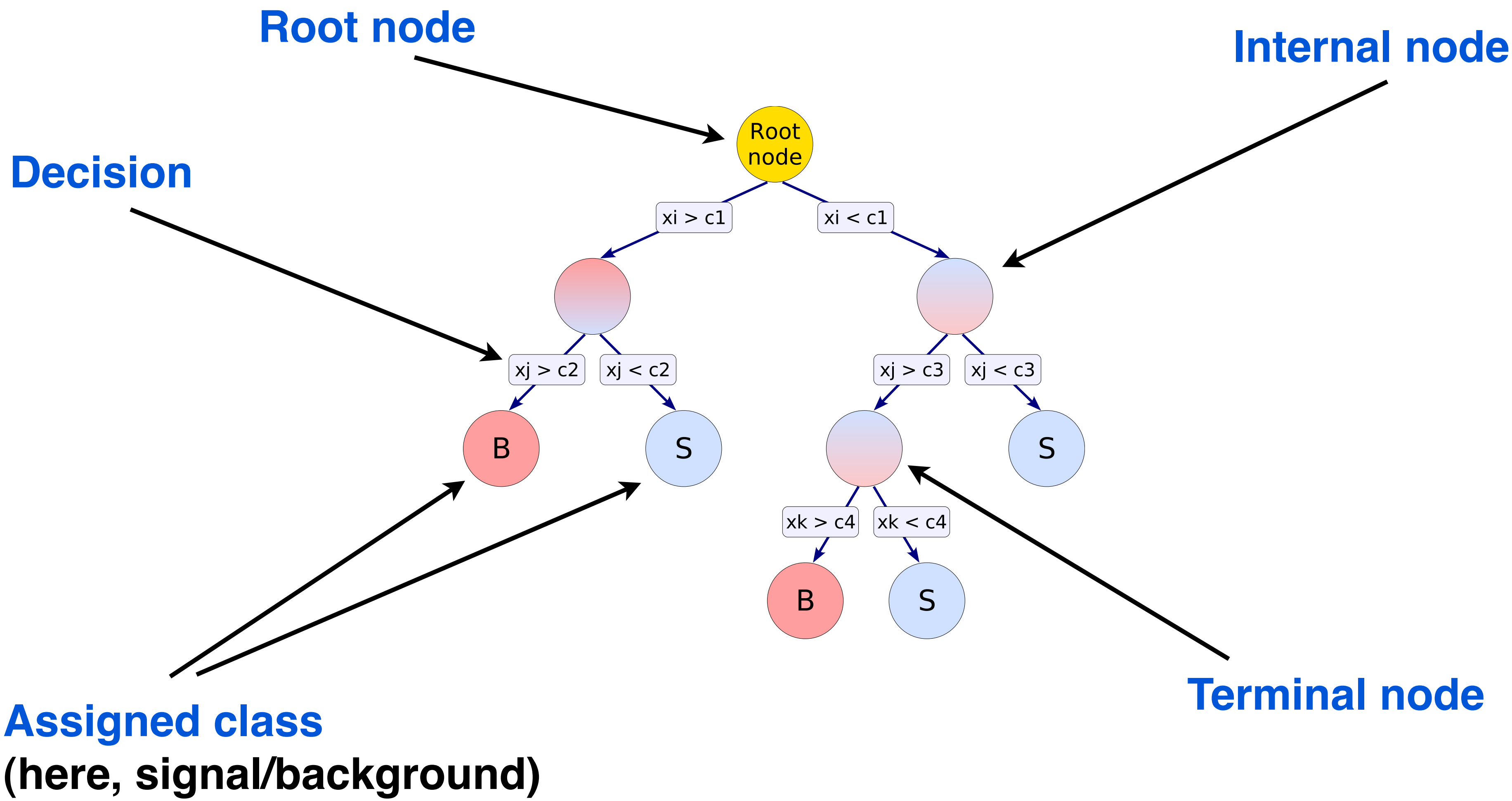


- Decision trees are **extremely sensitive to the training samples**, therefore to overtraining

- To stabilize their performance, one uses different techniques :
  - **Boosting**
  - Bagging
  - Random forests

Decision tree : structure

- Similar to rectangular cuts, but each cut depends on the previous one
- Classifies from a set of attributes. Each node splits the data according to one attribute

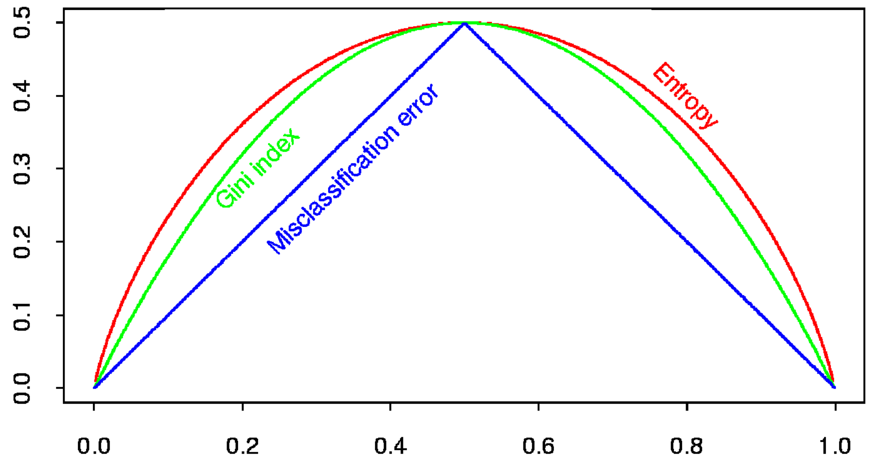


Decision tree : training

- **Training a decision tree** : process that defines the splitting criteria for each node.
- Start with the root node, the split in two subsets of training events. Go through the same algorithm for the next splitting operation
- Repeat until the whole tree is built

- Splitting criterion found **maximizing the signal/background separation**.
- Different criteria available. Usually one uses the

  Gini Index : p.(1-p) where p is the signal purity
- Note that it is symmetric between signal and background
- Selects the variable and cut value that optimises the increase in the separation index between the parent node and the sum of the indices of the two daughter nodes, weighted by their relative fraction of events.



$$Criterion = Gini_{father} - Gini_{left\ son} - Gini_{right\ son}$$

Decision tree : overtraining

Advantages :

- Decision trees are independent of monotonous variable transformations
- Weak variables are ignored and do not deteriorate performance

- But **Decision trees are extremely sensitive to the training samples**, therefore to overtraining
- Slightly different training samples can lead to radically different DT

- To stabilize Decision Tree performance, one can use different techniques.
    - **Boosting**
    - Bagging
    - Random forests
    - Pruning

Decision tree : boosting

Boosting :
- **Sequentially apply the DT algorithm** to **reweighted** (**boosted**) versions of the training data
- Take a weighted **majority** vote of the sequence of DT algorithms produced.
- Boosting allows also to **increase the performance**.
- Works very well on non-optimal decision tree (small number of nodes...)

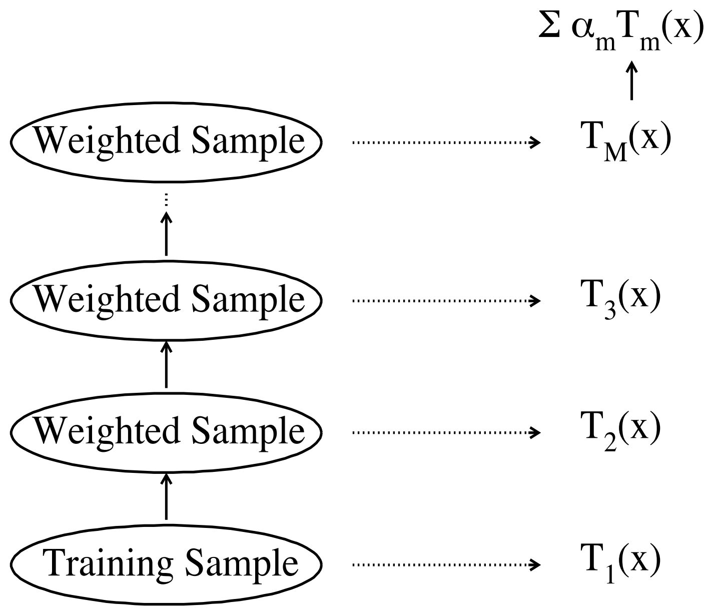


FIG. 2: Schematic of a boosting procedure.

Most famous implementation in **AdaBoost (adaptive boost) :**
- **Events misclassified** during the training of a decision tree are given a **higher event weight**
- Events are **reweighted** depending on the **error** of the previous tree

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

$$\alpha = \frac{1 - \text{err}}{\text{err}}$$

- The **output** of the BDT is :
  where hi=+1 or -1.

$$y_{\text{Boost}}(\mathbf{x}) = \frac{1}{N_{\text{collection}}} \cdot \sum_{i}^{N_{\text{collection}}} \ln(\alpha_i) \cdot h_i(\mathbf{x})$$

AdaBoost : event weight

error on the mth tree

$$err_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq T_m(x_i))}{\sum_{i=1}^{N} w_i}$$

I=1 if the event is misclassified (0 otherwise)

$$\alpha_m = \beta \times \ln((1 - err_m)/err_m)$$

weight of the ith event :

$$w_i \longrightarrow w_i \times e^{\alpha_m I(y_i \neq T_m(x_i))}$$

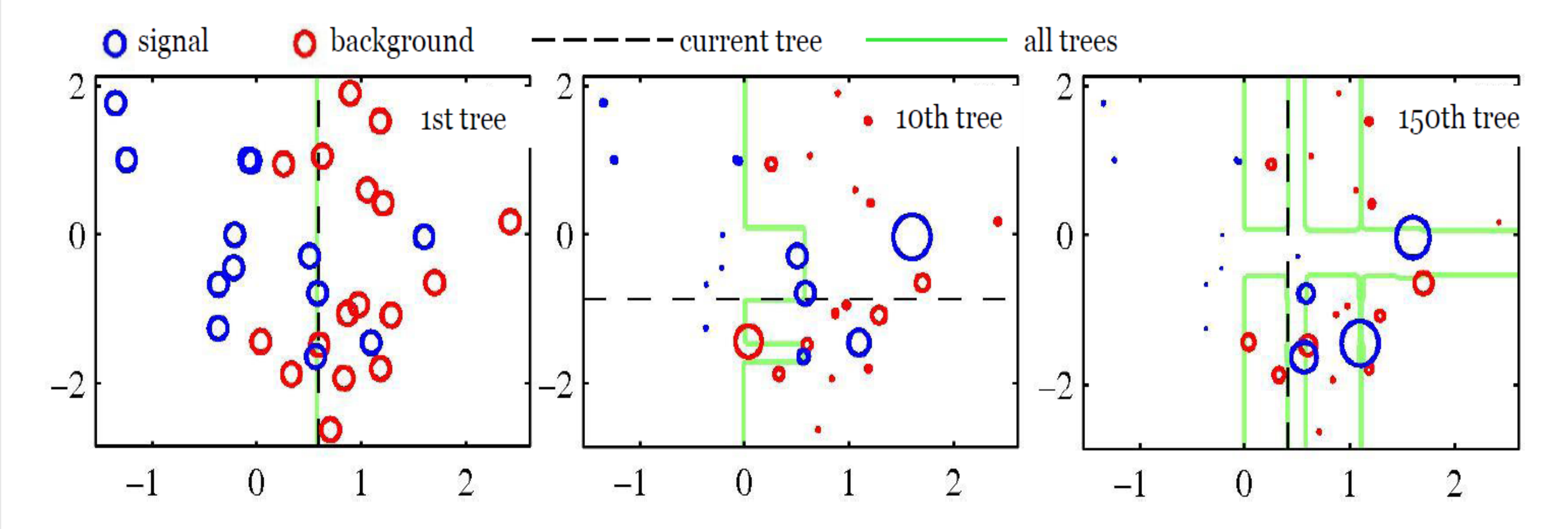
Start here:

equal event weights

misclassified events get

larger weights

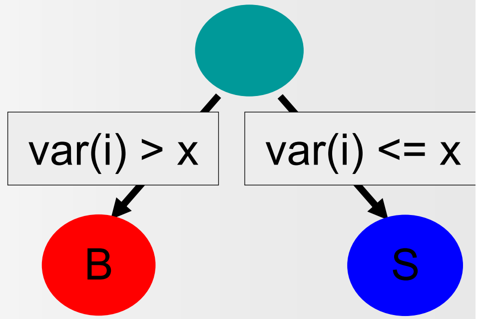
… and so on

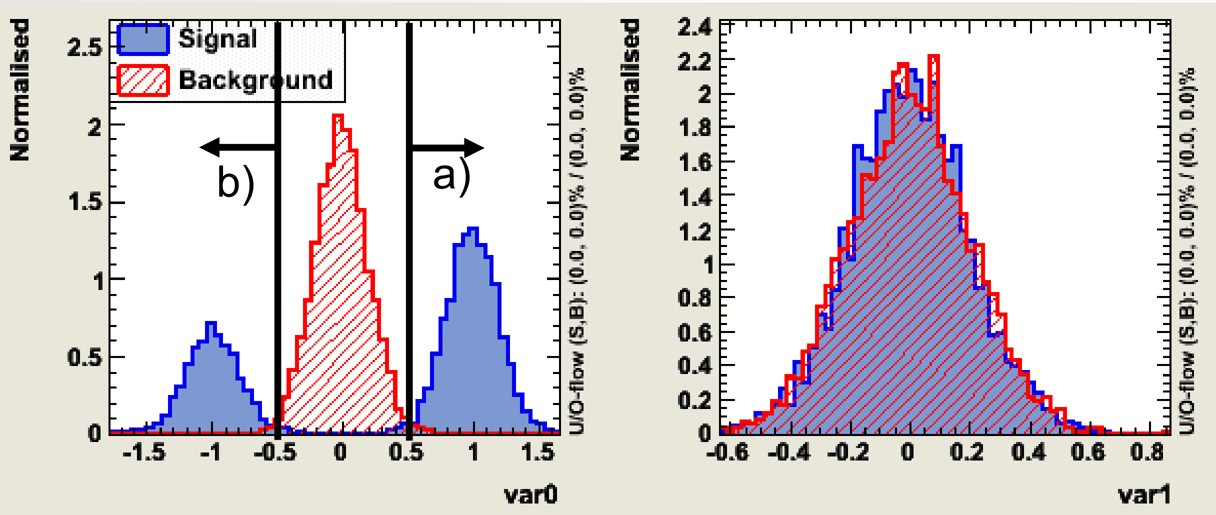


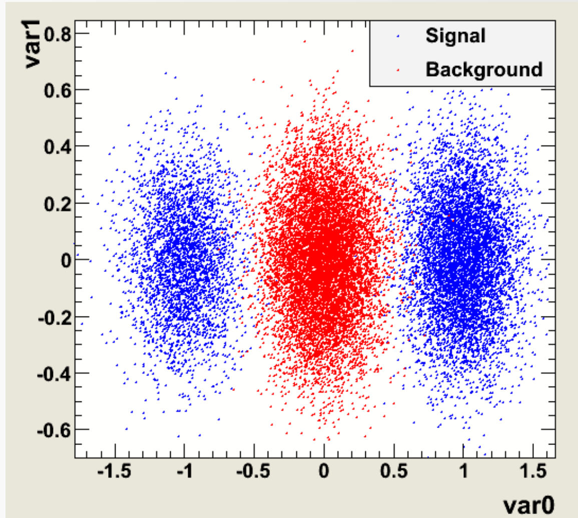
BDT : example

The example: (somewhat artificial…but nice for demonstration) :

- Data file with three “bumps”
- Weak classifier (i.e. one single simple “cut” ↔ decision tree stumps)







Two reasonable cuts: a) Var0 > 0.5 → ε_{signal}=66% ε_{bkg} ≈ 0% misclassified events in total 16.5%

or

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

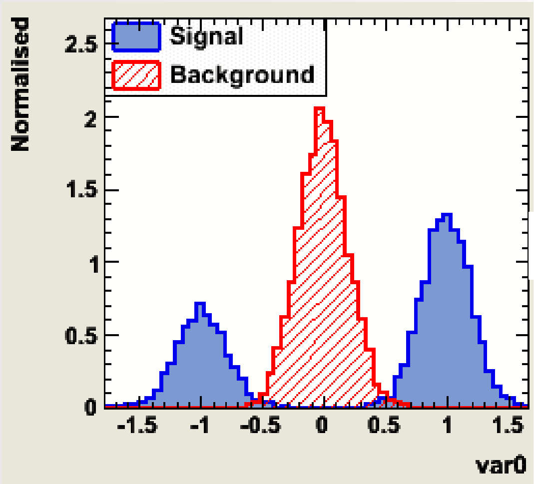
the training of a single decision tree stump will find “cut a)”

BDT : example

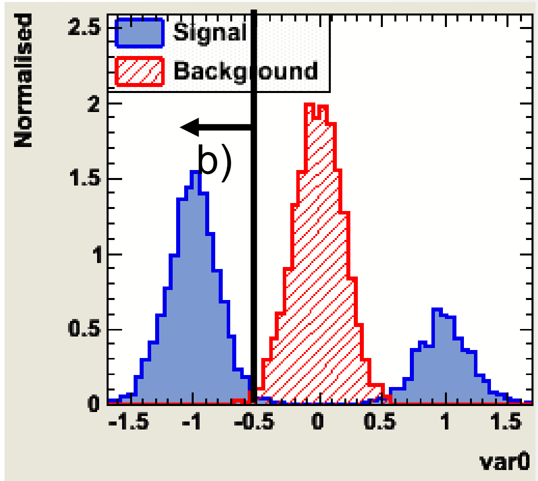
The first "tree", choosing cut a) will give an error fraction: err = 0.165

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

➔ the next "tree" sees essentially the following data sample:

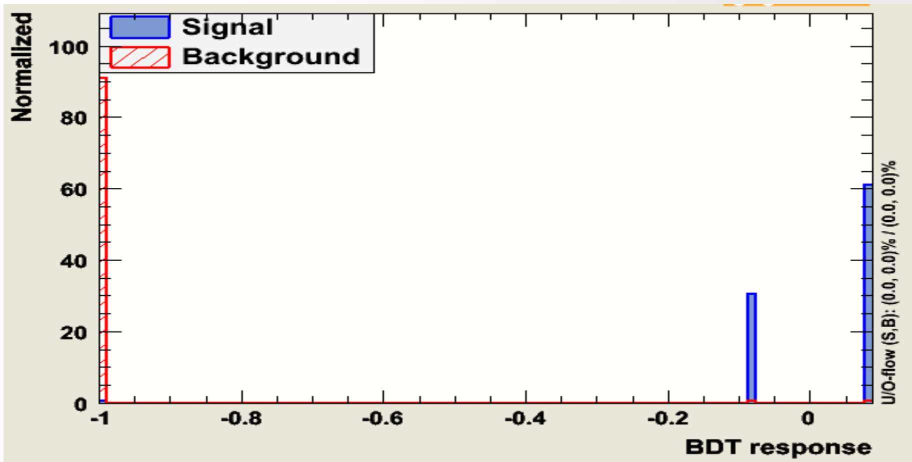


re-weight



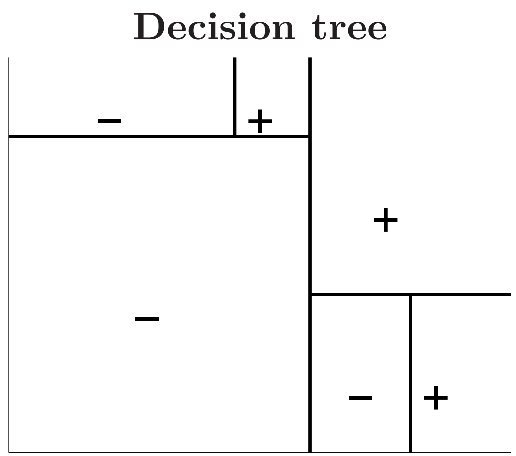
.. and hence will chose: "cut b)": Var0 < -0.5

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



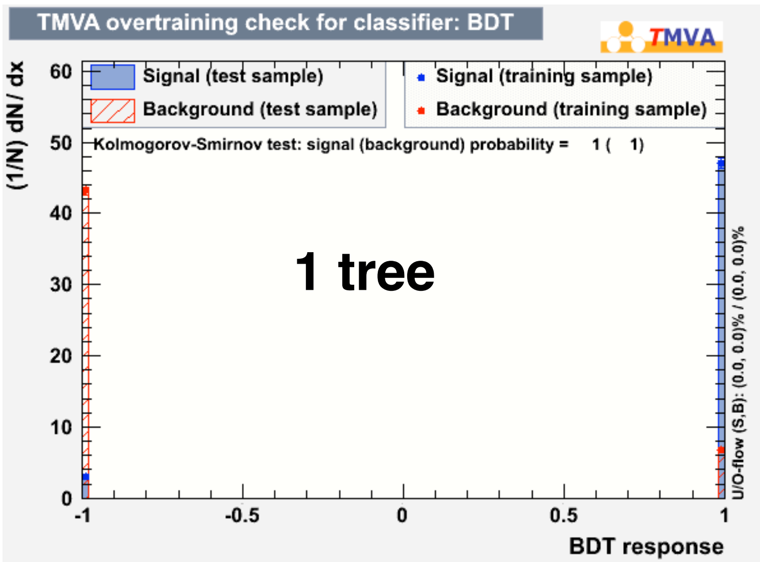
Decision tree : output

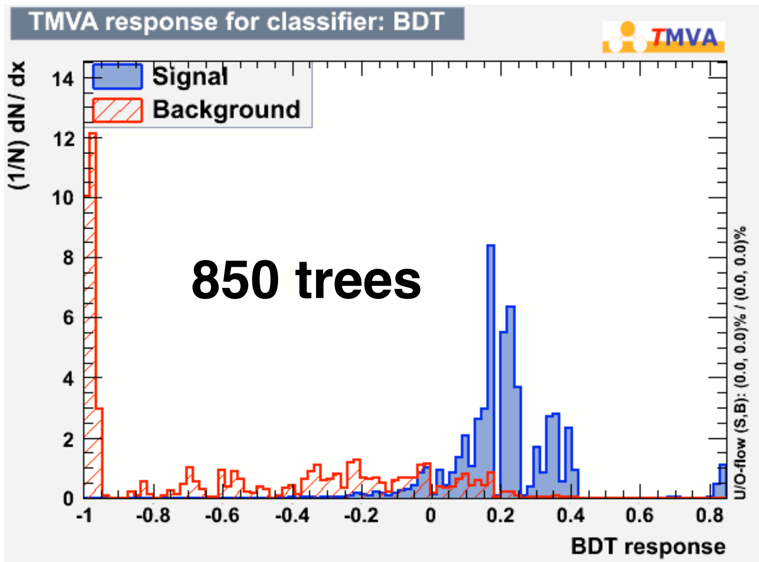
- A single decision tree can be trained to gives always an integer response, : signal (+1) / background (-1)



Boosted decision trees give a Real-valued output :

- The output is a linear combination of +1 and -1, because of the weights over the different training decision trees during boosting
- Output is **quasi-continuous**. The number of classes depends on the number of trees used in the boosting process





Decision tree : bagging, random forests, pruning

- One can also use different techniques such as **bagging** and **random forest**
- Improves the stability against fluctuations, not much the performance
- Both of them makes use of the idea of **randomizing trees**.

Bagging :

- Resampling technique. Training is repeated on “bootstrap” samples (i.e re-sample training data with replacement), then combined

Random forests :

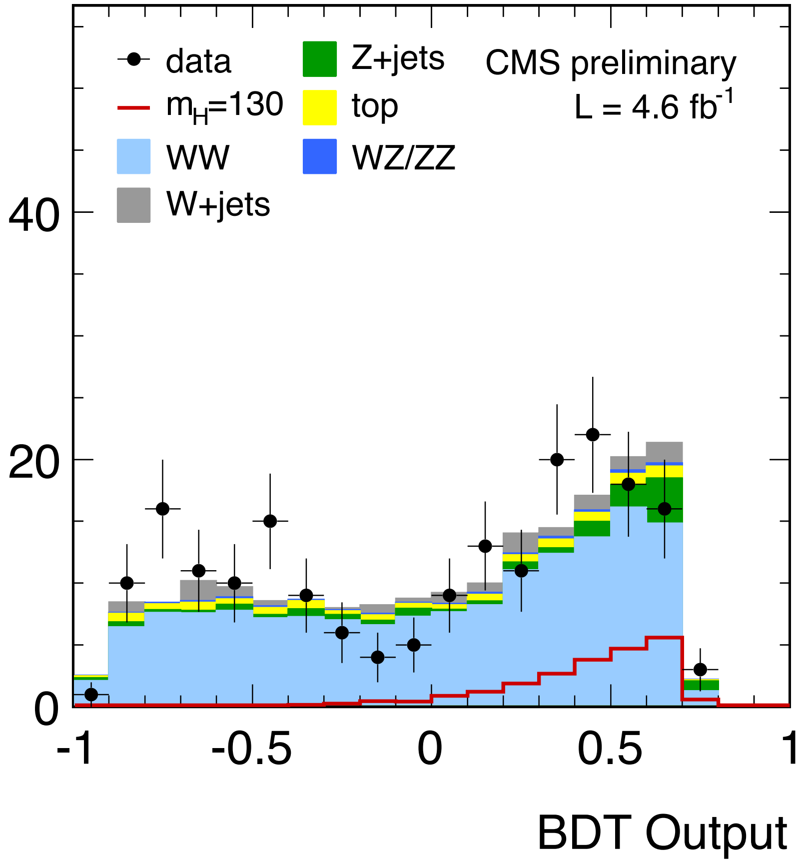
- Training repeated on random bootstrap (or subsets) of the training data only
- Consider at each node only a random subsets of variables for the split

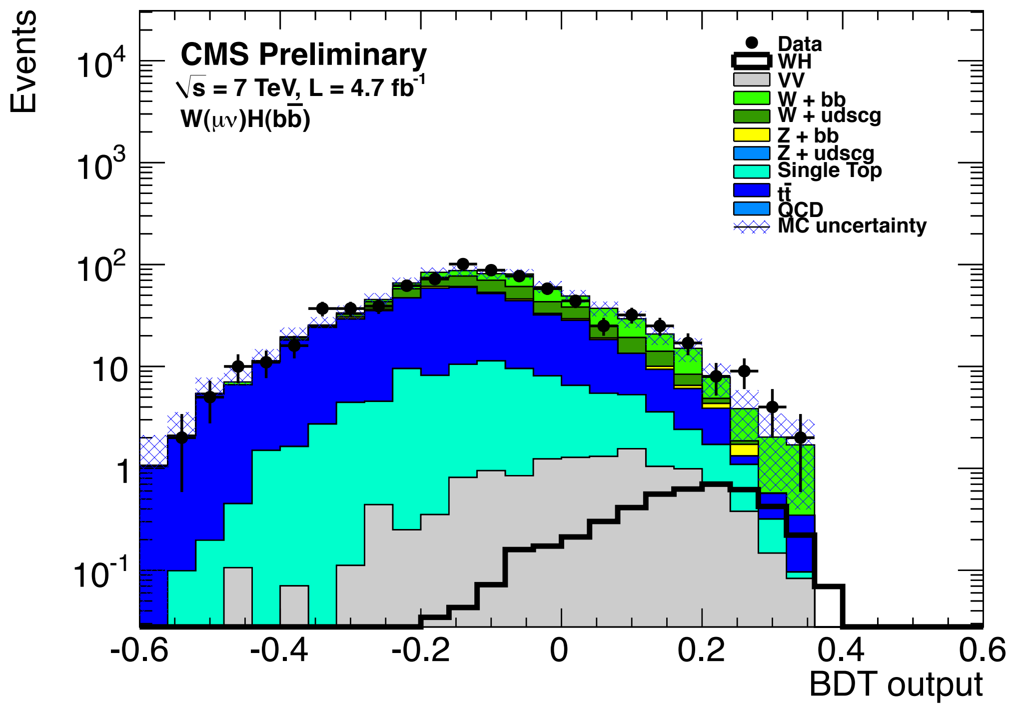
Pruning :

- Grow tree to the end and “cut back”, nodes that seem statistically dominated

Decision tree : example in HEP

Examples in CMS : H→WW, H→bb analyses





The package TMVA

- Package widely used in HEP
- Root-based implementation (included in every recent ROOT release)

TMVA functionalities :

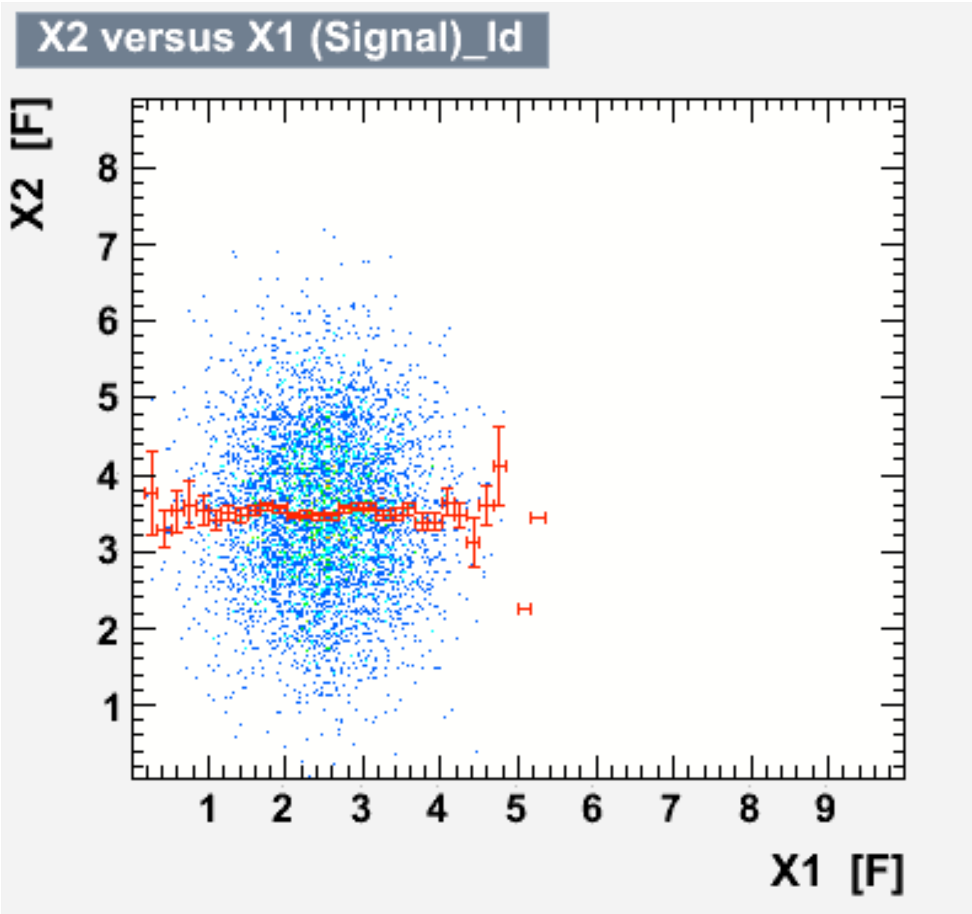
- Allows to check input variables, correlations, overtraining, performance
- **Many multivariate methods** available : rectangular cuts, likelihood, various decision trees, SVM...
- **Classification** and **regression**
- **Tuning of parameters** relatively easy
- **Training is user-friendly** and fast enough to be manageable on a laptop
- Application is less user friendly : basically have to do it by hand in ROOT

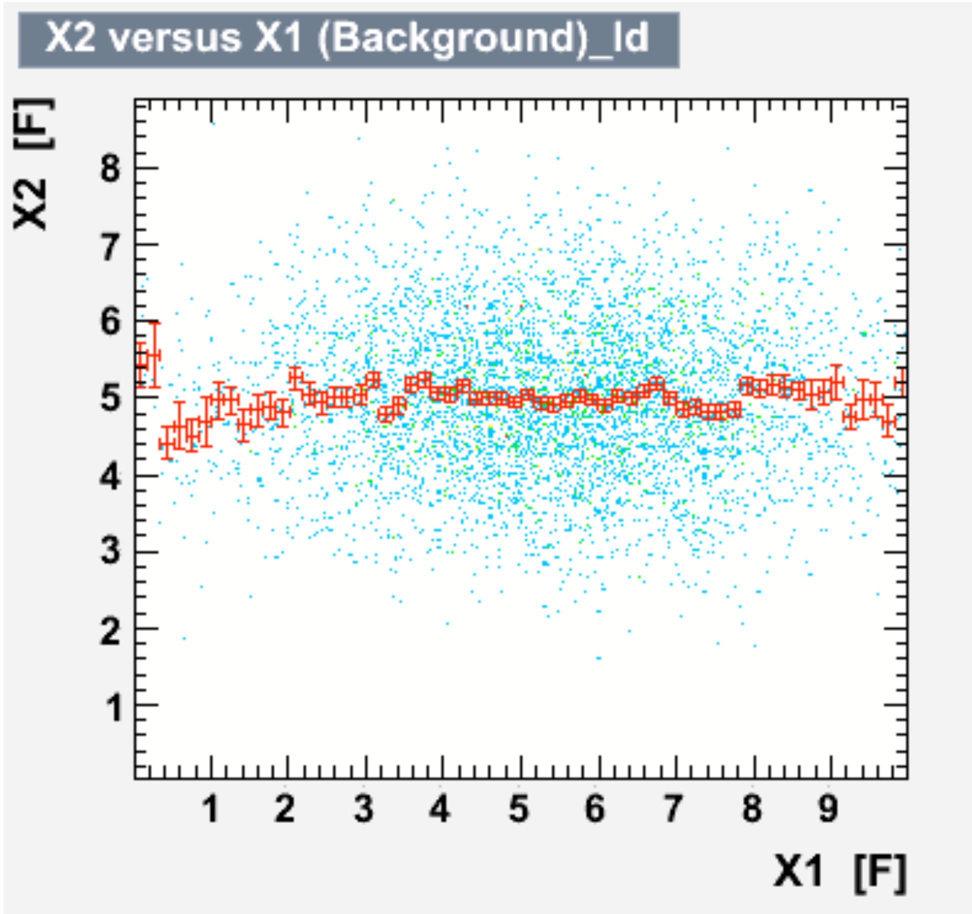
Available classifiers

```
// --- Cut optimisation
Use["Cuts"]            = 1;
Use["CutsD"]           = 0;
Use["CutsPCA"]         = 0;
Use["CutsGA"]          = 0;
Use["CutsSA"]          = 0;
//
// --- 1-dimensional likelihood ("naive Bayes estimator")
Use["Likelihood"]      = 0;
Use["LikelihoodD"]     = 0; // the "D" extension indicates decorrelated input variables (see option strings)
Use["LikelihoodPCA"]   = 0; // the "PCA" extension indicates PCA-transformed input variables (see option strings)
Use["LikelihoodKDE"]   = 0;
Use["LikelihoodMIX"]   = 0;
//
// --- Mutidimensional likelihood and Nearest-Neighbour methods
Use["PDERS"]           = 0;
Use["PDERSD"]          = 0;
Use["PDERSPCA"]        = 0;
Use["PDEFoam"]         = 0;
Use["PDEFoamBoost"]    = 0; // uses generalised MVA method boosting
Use["KNN"]             = 0; // k-nearest neighbour method
//
// --- Linear Discriminant Analysis
Use["LD"]              = 0; // Linear Discriminant identical to Fisher
Use["Fisher"]          = 0;
Use["FisherG"]         = 0;
Use["BoostedFisher"]   = 0; // uses generalised MVA method boosting
Use["HMatrix"]         = 0;
//
// --- Function Discriminant analysis
Use["FDA_GA"]          = 0; // minimisation of user-defined function using Genetics Algorithm
Use["FDA_SA"]          = 0;
Use["FDA_MC"]          = 0;
Use["FDA_MT"]          = 0;
Use["FDA_GAMT"]        = 0;
Use["FDA_MCMT"]        = 0;
//
// --- Neural Networks (all are feed-forward Multilayer Perceptrons)
Use["MLP"]             = 1; // Recommended ANN
Use["MLPBFGS"]         = 0; // Recommended ANN with optional training method
Use["MLPBNN"]          = 0; // Recommended ANN with BFGS training method and bayesian regulator
Use["CFMlpANN"]        = 0; // Depreciated ANN from ALEPH
Use["TMlpANN"]         = 0; // ROOT's own ANN
//
// --- Support Vector Machine
Use["SVM"]             = 0;
//
// --- Boosted Decision Trees
Use["BDT"]             = 0; // uses Adaptive Boost
Use["BDTG"]            = 0; // uses Gradient Boost
Use["BDTB"]            = 0; // uses Bagging
Use["BDTD"]            = 0; // decorrelation + Adaptive Boost
//
// --- Friedman's RuleFit method, ie, an optimised series of cuts ("rules")
Use["RuleFit"]         = 0;
```

Functionalities : correlations

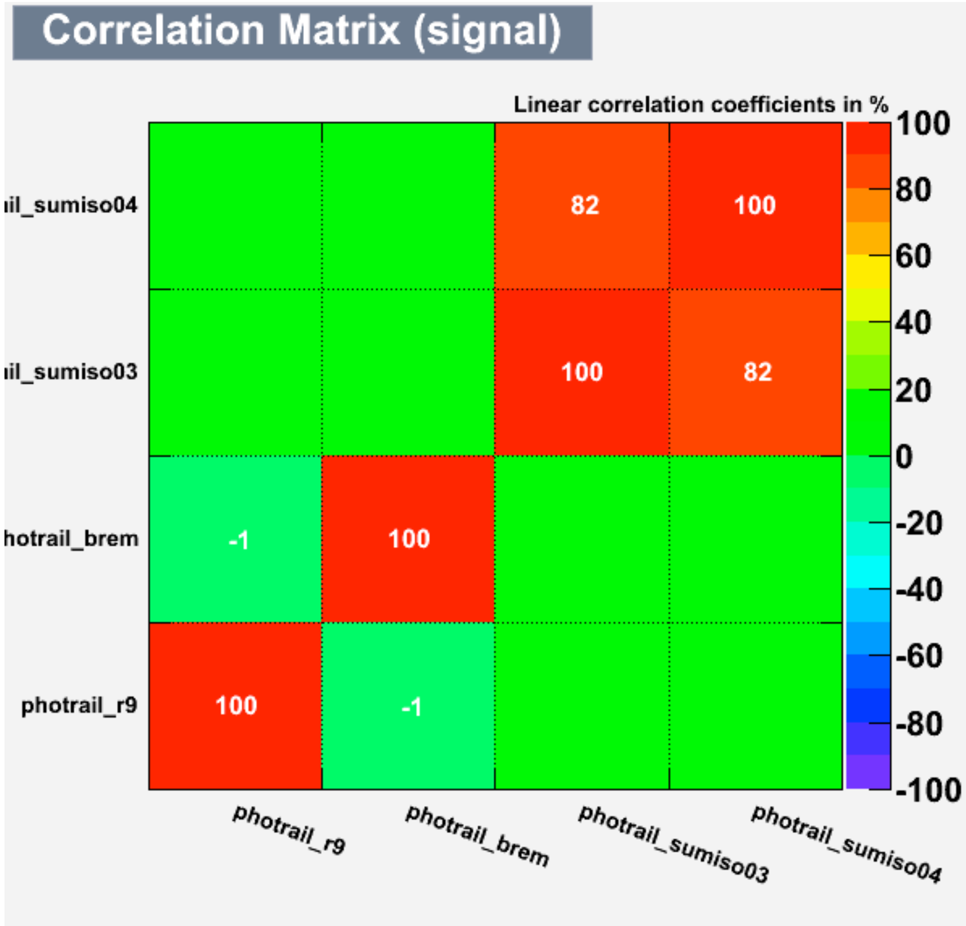
- Linear correlations are easily investigated via the GUI :
- (Here, no correlation)

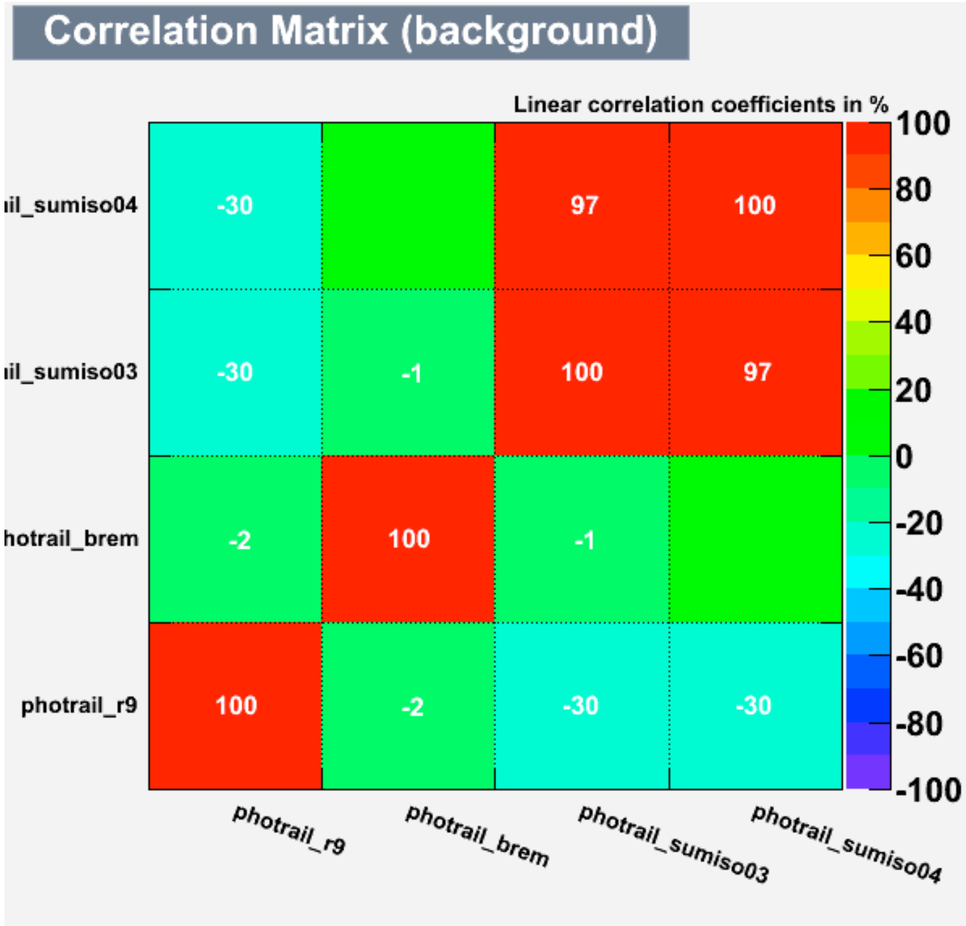




Functionalities : correlations

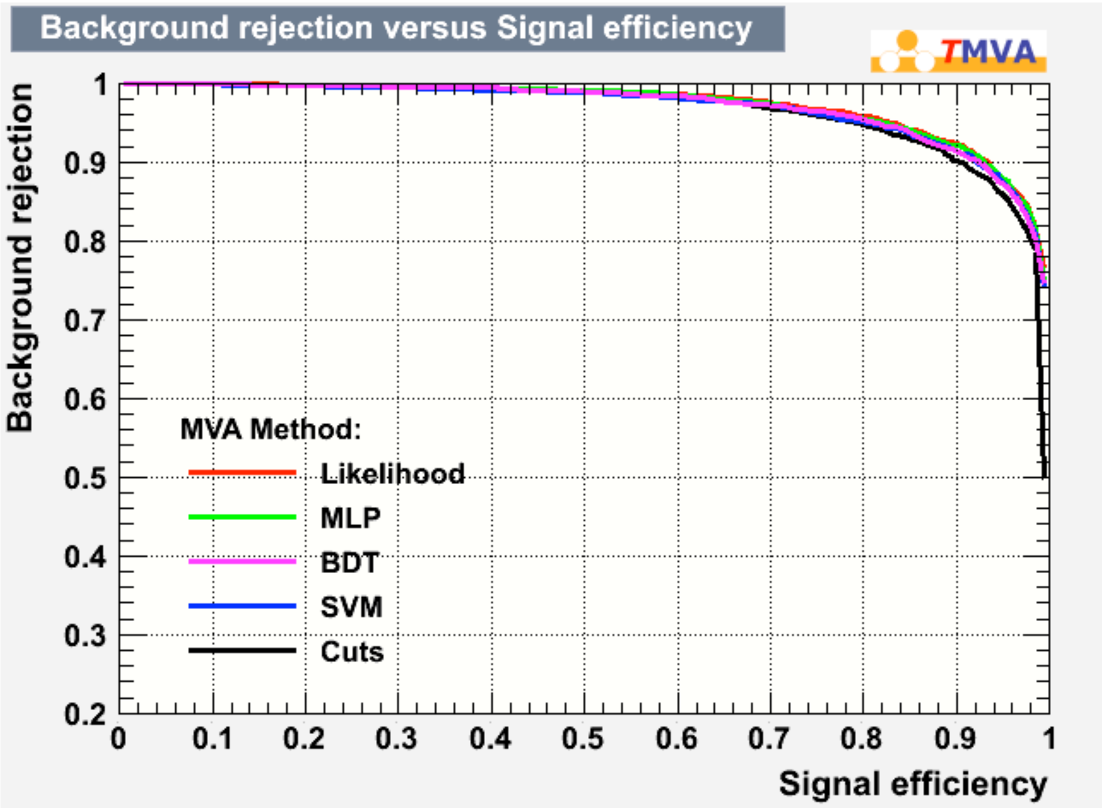
- Linear correlations are easily investigated via the GUI :
- Signal and background input variables can be correlated differently





Functionalities : performance

- Many classifiers can be trained in one shot
- Useful for performance comparison



Advantages and drawbacks of different classifiers

From TMVA manual

		MVA METHOD									
CRITERIA		Cuts	Likelihood	PDE-RS / k-NN	PDE-Foam	H-Matrix	Fisher / LD	MLP	BDT	RuleFit	SVM
Performance	No or linear correlations	⋆	⋆⋆	⋆	⋆	⋆	⋆⋆	⋆⋆	⋆	⋆⋆	⋆
	Nonlinear correlations	○	○	⋆⋆	⋆⋆	○	○	⋆⋆	⋆⋆	⋆⋆	⋆⋆
Speed	Training	○	⋆⋆	⋆⋆	⋆⋆	⋆⋆	⋆⋆	⋆	○	⋆	○
	Response	⋆⋆	⋆⋆	○	⋆	⋆⋆	⋆⋆	⋆⋆	⋆	⋆⋆	⋆
Robustness	Overtraining	⋆⋆	⋆	⋆	⋆	⋆⋆	⋆⋆	⋆	○	⋆	⋆⋆
	Weak variables	⋆⋆	⋆	○	○	⋆⋆	⋆⋆	⋆	⋆⋆	⋆	⋆
Curse of dimensionality		○	⋆⋆	○	○	⋆⋆	⋆⋆	⋆	⋆	⋆	
Transparency		⋆⋆	⋆⋆	⋆	⋆	⋆⋆	⋆⋆	○	○	○	○

Exercises

- Problem inspired by Higgs searches in H->2photons channel at LHC

- **Goal** : be able to estimate the sensitivity of a search for a small peak over a huge background, using multivariate methods

- **3 exercises** :
    - **Setting up Root and TMVA environment, TMVA basics**
    - Using a MVA method inside the analysis
    - Estimation of analysis sensitivity