Machine Learning and Multivariate Techniques in HEP data Analyses

Example of MVA methods :

- **Rectangular cut optimization**
- Fisher
- Likelihood
- **Neural network**
- **Decision tree**
- Support Vector Machine
- ...

Extracted from slides by N. Chanon, ETH Zurich.

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

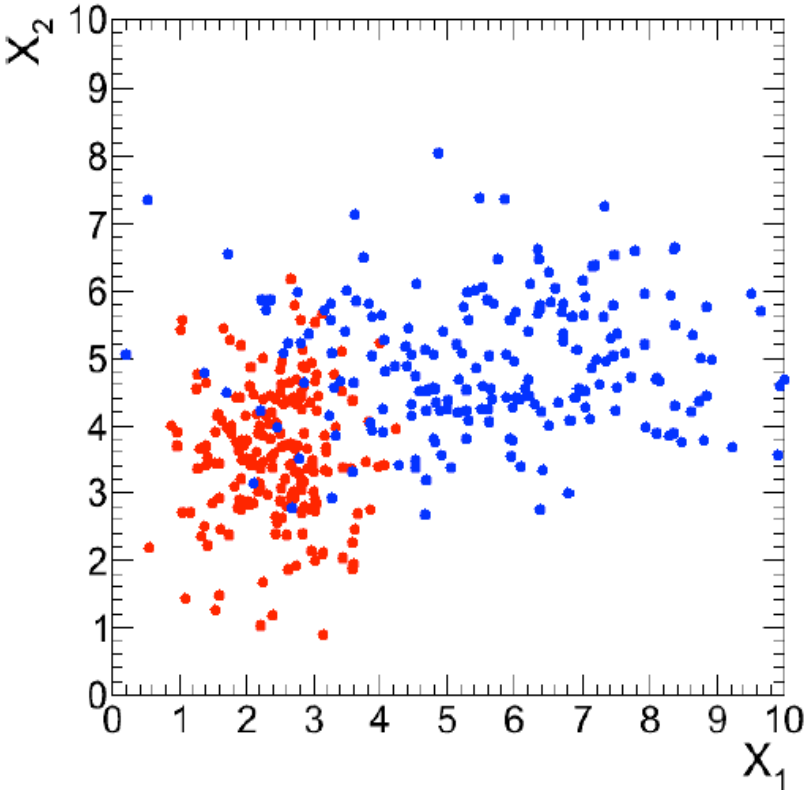
Statistical learning :

- **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**

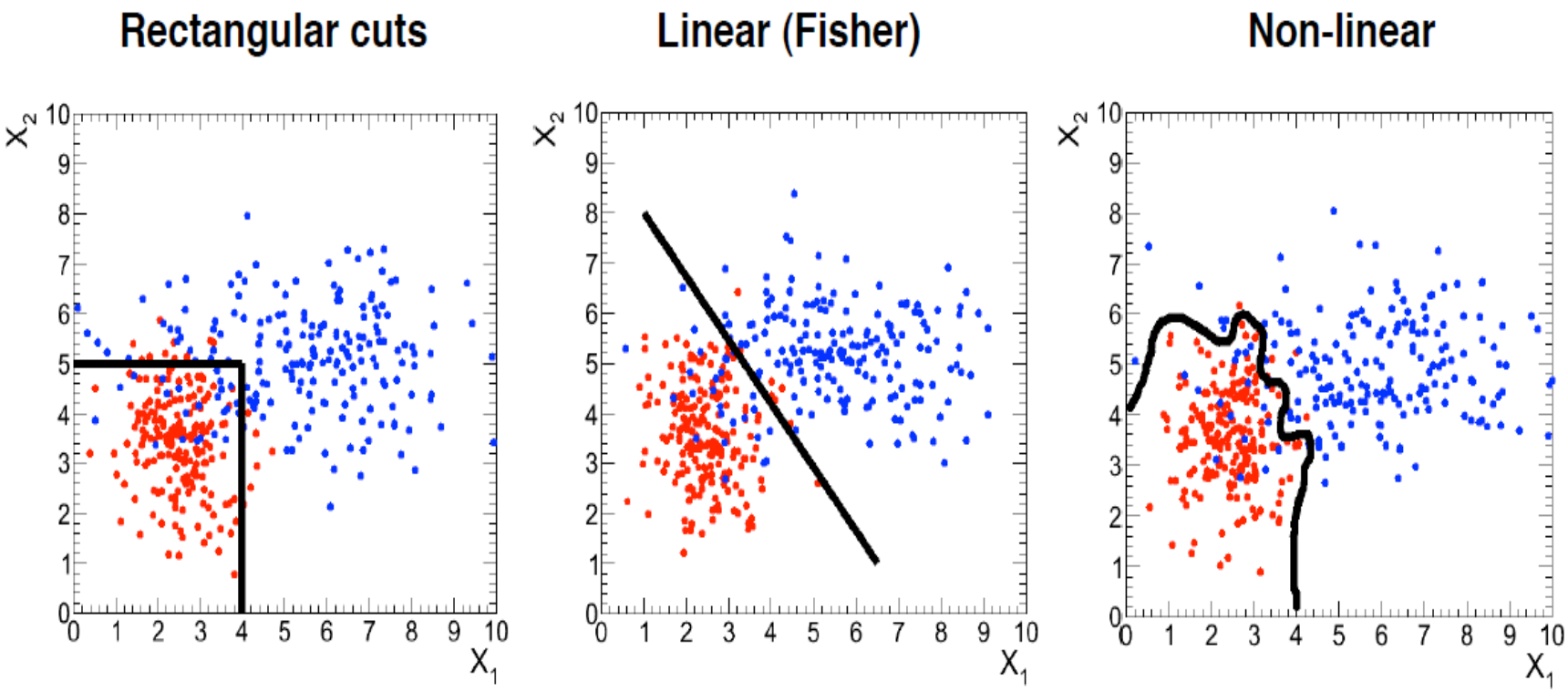
- Assume we have two populations (signal and background) and two variables



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

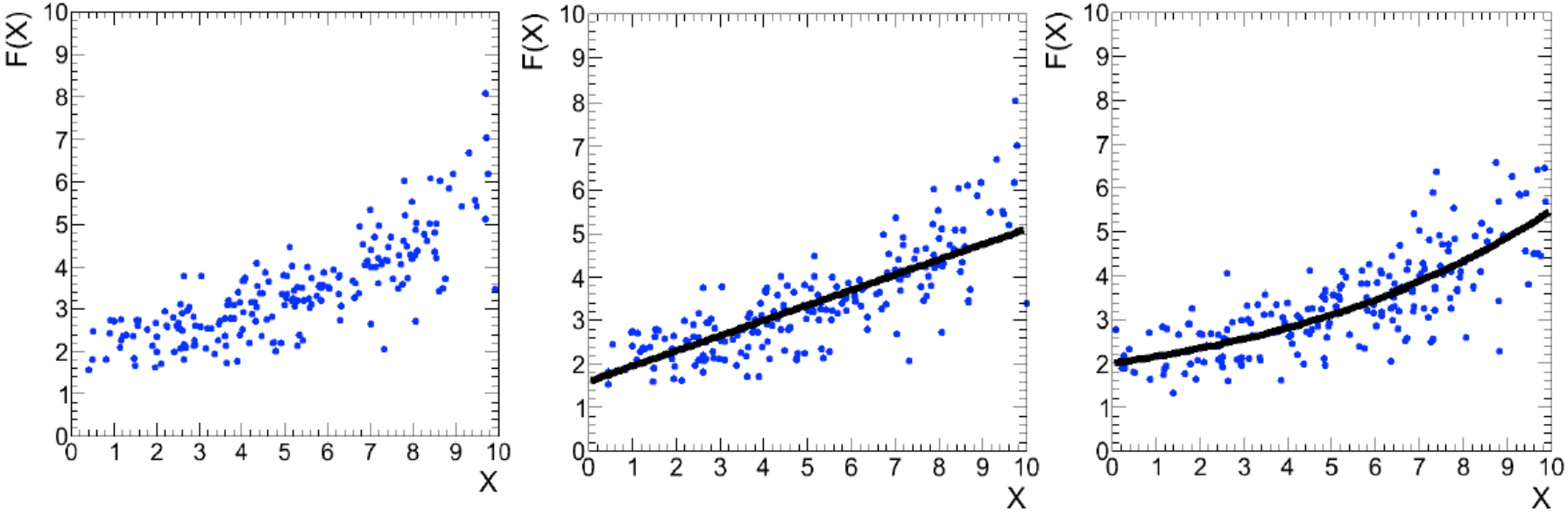
Event classification

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



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**



7

Multi-variate analyses in HEP

- **Signal/background discrimination :**
  - **Object reconstruction :** discriminate against instrumental background (electronic noise...)
  - **Object identification :** e.g. electron, bottom quark identification, to improve the rejection other objects resembling (e.g. jets)
  - **Discriminating physics process against physics backgrounds**. Many examples, e.g. single top against W+jets, H->WW against WW background...

- **Improving the energy measurement**, via regression. Allows to narrow the reconstructed mass peak, improve the resolution.

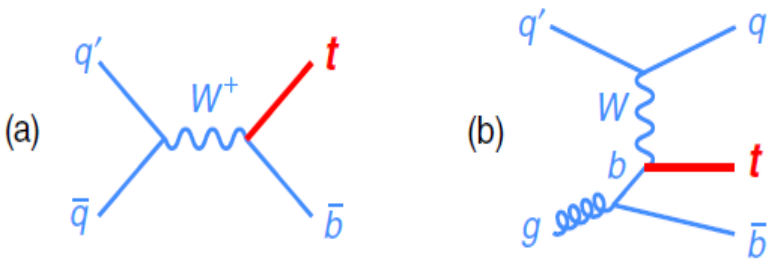
- **Estimate the sensitivity of the analysis :**
  - **Sensitivity to signal exclusion or discoveries :** Likelihood of the data to be consistent with background only or signal+background hypothesis
  - **Combination** of many channels

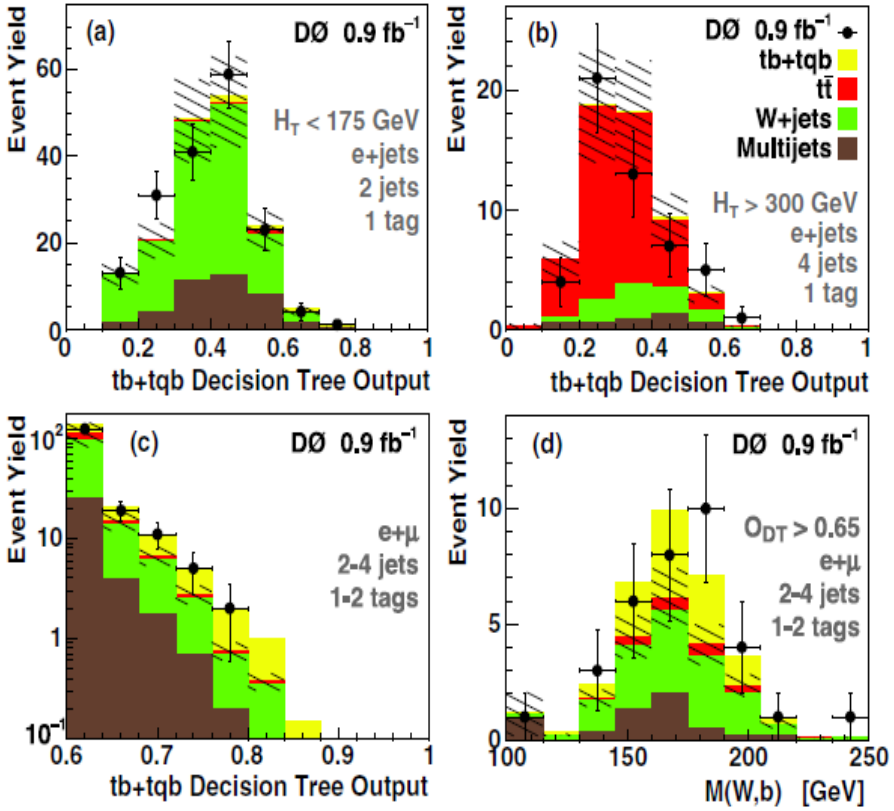
  => exclusion limits or discoveries

Multi-variate examples in HEP: Tevatron

Single top discovery

PhysRevLett.98.181802



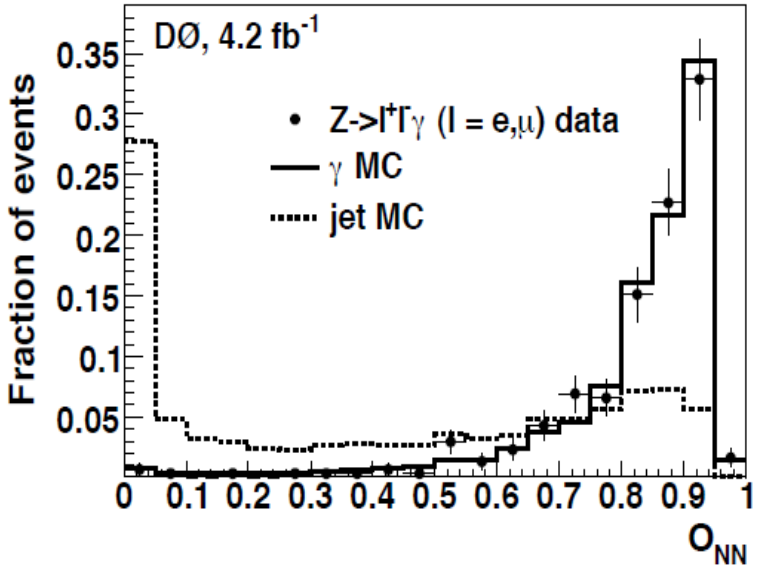


- When published, very controversial
- 36 boosted decision trees used to discriminate signal from background
- First measurement of the single top cross-section, today well established

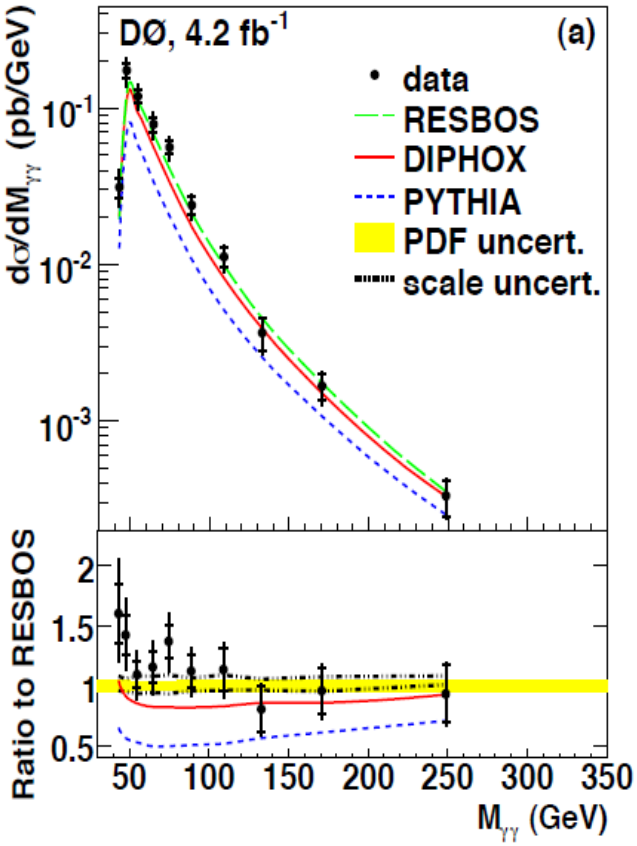
Multi-variate examples in HEP: Tevatron

Photon identification at D0 and applications

arxiv:1002.4917v3



- Neural network for Photon Id based on calorimeter energy deposit and track variables in an isolation cone around the photon
- Used to identify and measure the diphoton+X cross-section



Multi-variate examples in HEP: LHC

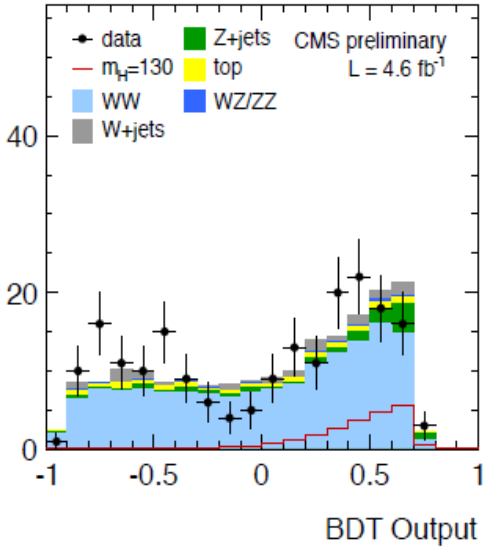
H→WW→llνν searches in CMS

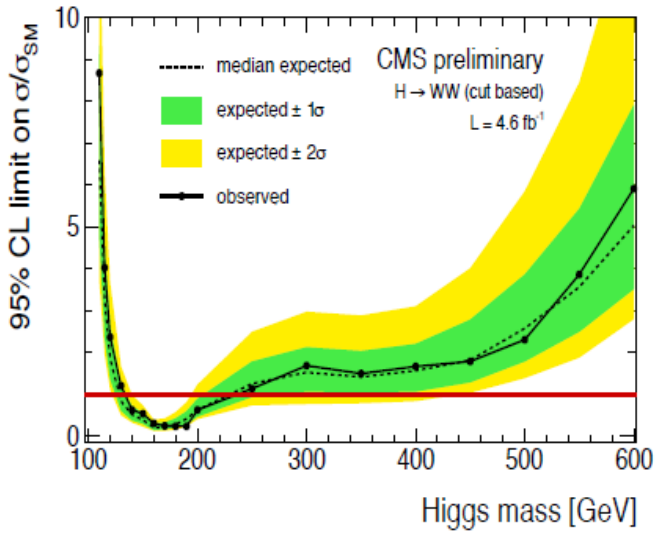
CMS-PAS-HIG-11-024

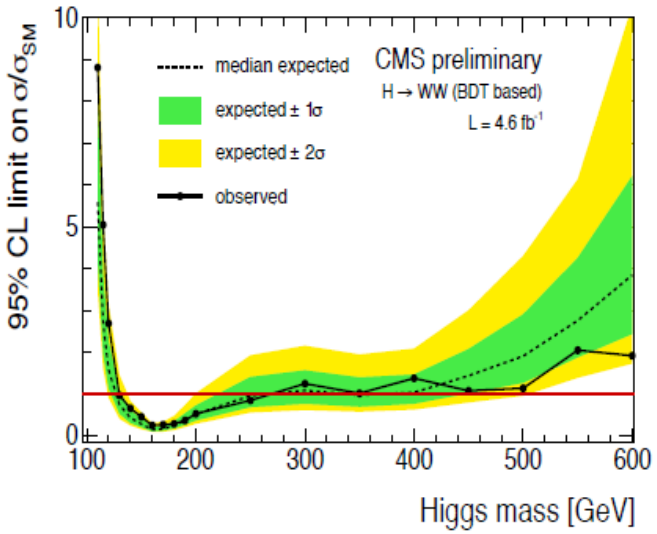
- 3 channels : 0-jet, 1-jet, 2-jet

- Electron identification with a multivariate technique : 50% more background rejection for the same signal efficiency
- Boosted decision tree in 0-jet and 1-jet channels : kinematic variables

- Limits improved by using BDT





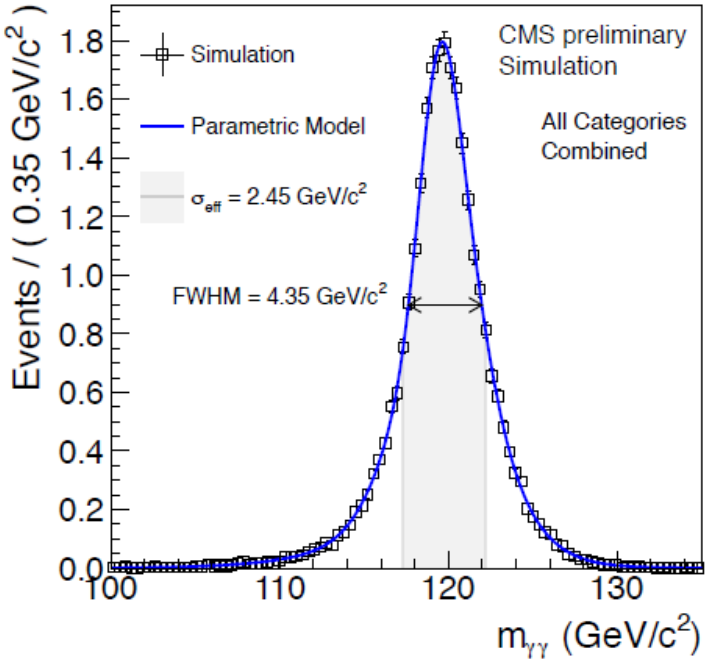


Multi-variate examples in HEP: LHC

H->γγ in CMS: energy resolution

- Higgs natural width is zero from an experimental point of view in the γγ channel
- So the **experimental width** is driven by **how well the photon energy is reconstructed** (once measured the position in the ECAL and the vertex found)

- CMS : PbWO4 crystals calorimeter, subject to **loss of transparency**
- Clustering of the energy deposited is affected by the **tracker material** in front of the ECAL
- Corrections to get back the reconstructed energy to the energy at the vertex might not be optimal
- CMS : energy regression



Plenty of multi-variate 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

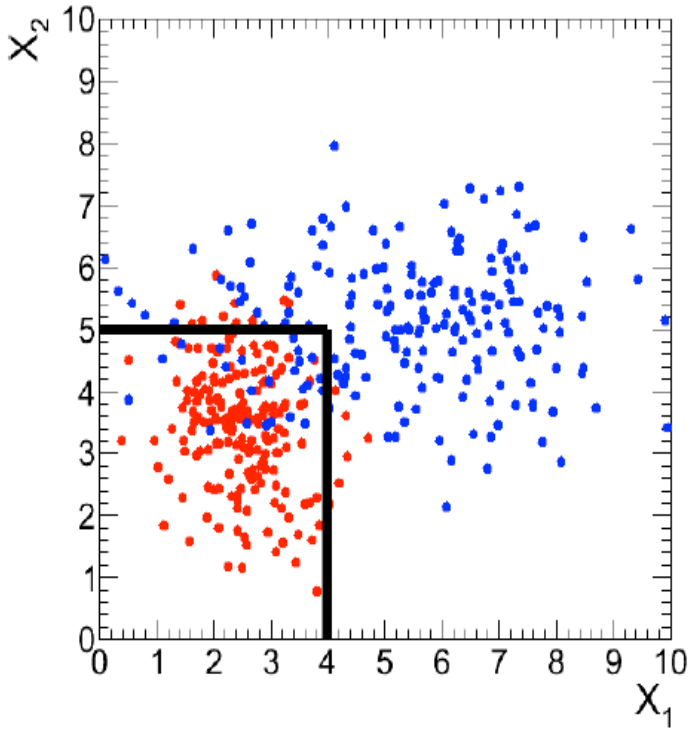
- Simplest multivariate method, very intuitive
- All HEP analyses are using rectangular cuts, not always completely optimized

Rectangular cuts optimization :

- Grid search, Monte-Carlo sampling
- Genetic algorithm
- Simulated annealing

Characteristics :

- Difficult to discriminate signal from background if non-linear correlations
- Optimization difficult to handle with high number of variables



Define the signal region :

$a1 < x1 < a2$,

$b1 < x2 < b2$

...

Fisher discriminant

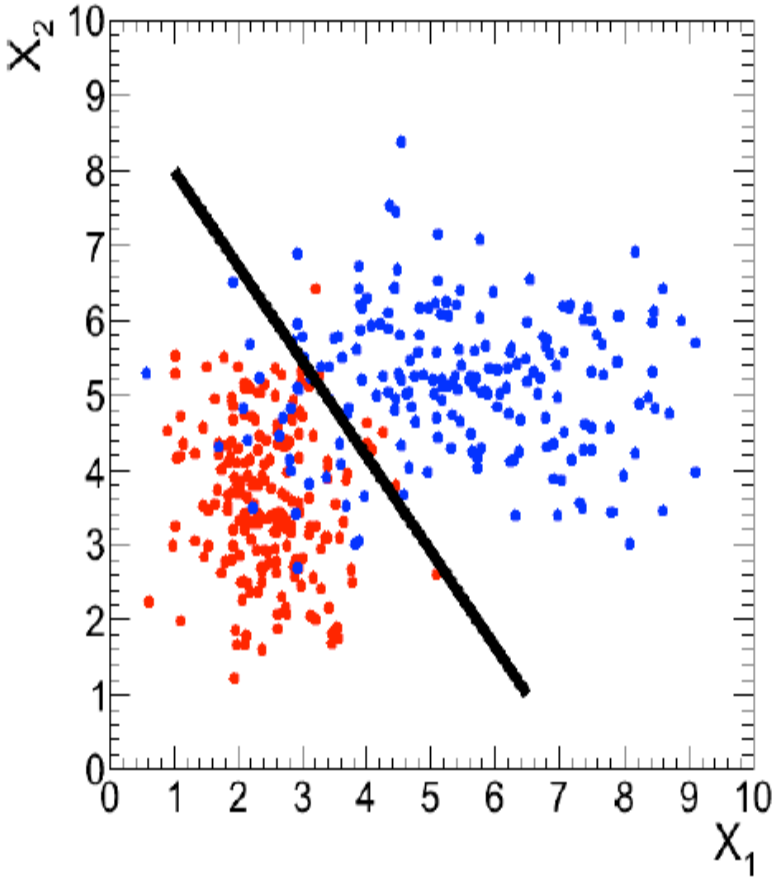
Fisher method :

- Cut on a linear combination of the input variables

  $y < a.x1 + b.x2$

- This corresponds to an hyper-plan in the variable phase-space
- Very efficient if linear correlations

- Again, difficult to handle non-linear correlations
- More easily trained than rectangular cuts



Likelihood estimator

- The likelihood ratio is defined by : $y_{\mathcal{L}}(i) = \frac{\mathcal{L}_S(i)}{\mathcal{L}_S(i) + \mathcal{L}_B(i)}$

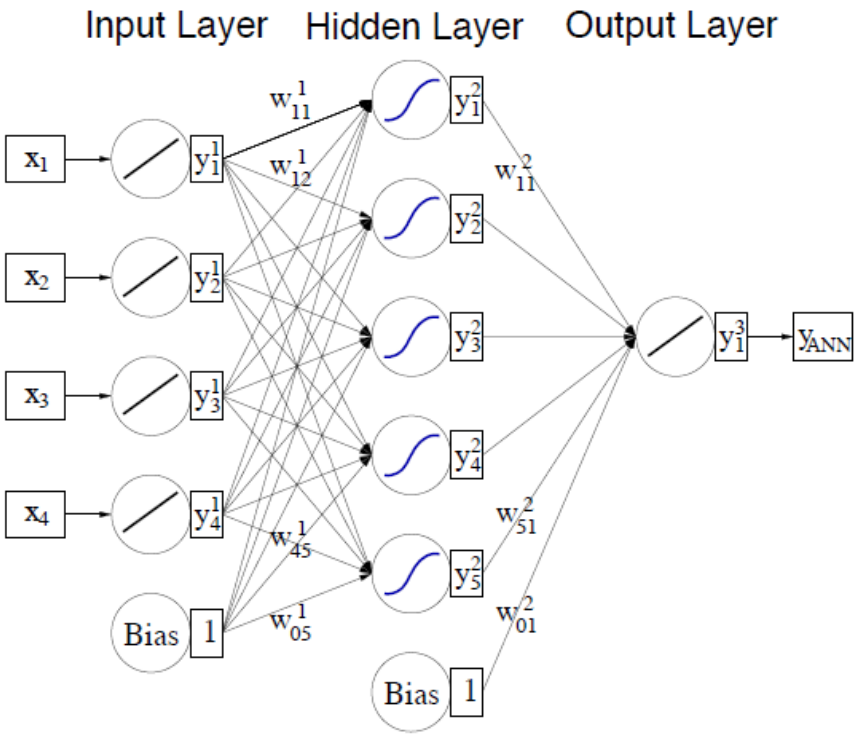
$$\mathcal{L}_{S(B)}(i) = \prod_{k=1}^{n_{\text{var}}} p_{S(B),k}(x_k(i))$$

is the product of the probability function for each variables.

- Optimal when no correlation between the variables
- This likelihood method does not take into account the correlations and is therefore sub-optimal in presence of correlations
- Refinements exist to take into account the correlations

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

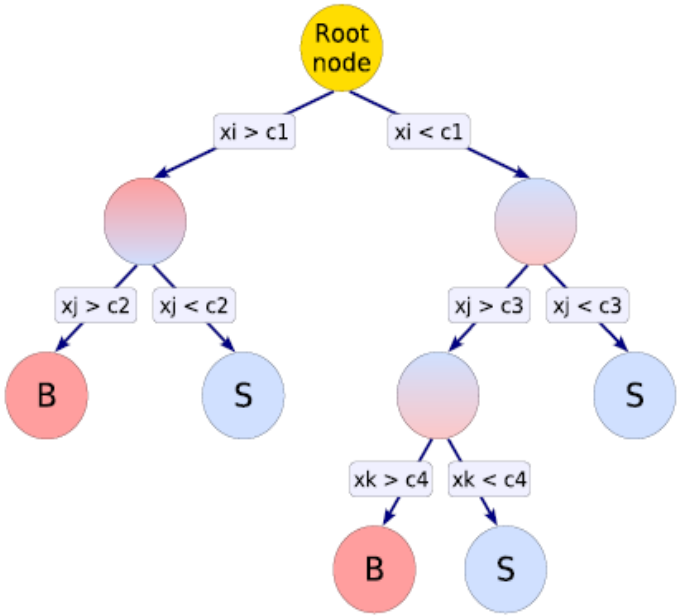


- Neural networks are universal approximators : takes advantage of correlations

- Quite stable against overtraining and against increasing number of variables

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)

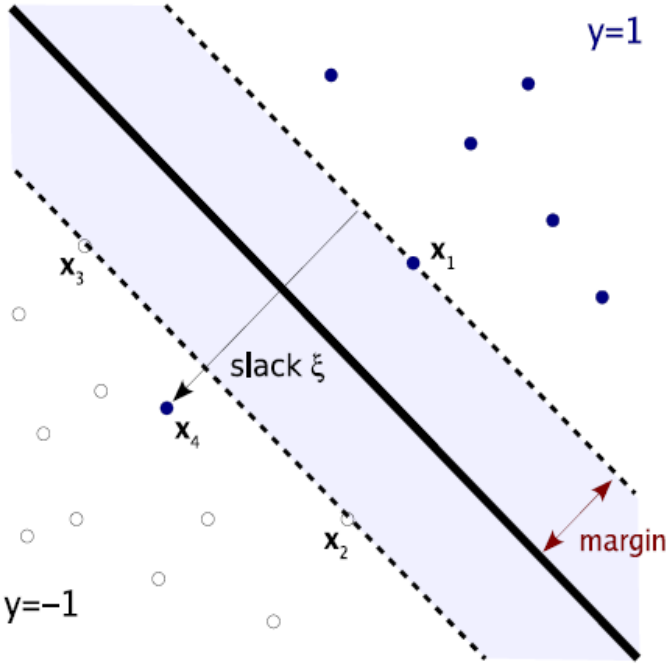


- Decision trees are **extremely sensitive to the training samples**, therefore to overtraining

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

Support Vector Machine

- **Idea** : build a hyperplane that separate signal and background vectors (events) using only a subset of all training vectors (support vectors)
- Position of the hyperplane found by maximizing the margin between it and the support vectors
- Higher dimensions spaces are used by non-linear transformation, using kernel functions such as the gaussian basis



x_1, x_2, x_3, x_4 – support vectors

- SVM can be competitive with NN and BDT but is often less discriminant : often data are non-separable, therefore sensitive to all the SVM parameters

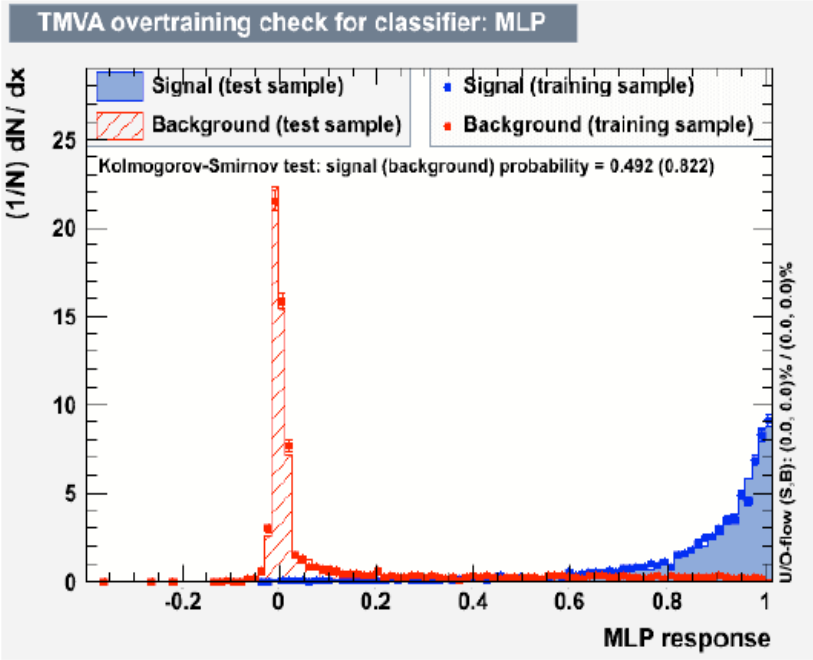
- In some cases this method performs very well

Training and application

Training / test samples

- For all multivariate methods, two samples are used :
  - Training sample
  - Test sample
- This is mandatory to check that the training has converged to a solution which does not depend on the statistical fluctuations of the training sample
- Generally speaking, MVA should be applied (or tested) in events where the response is not known

- Training is time-consuming, especially while increasing the number of variables (and depending on the method)
- Application is usually faster : it uses a set of weights used in the MVA output computation



Which method to choose

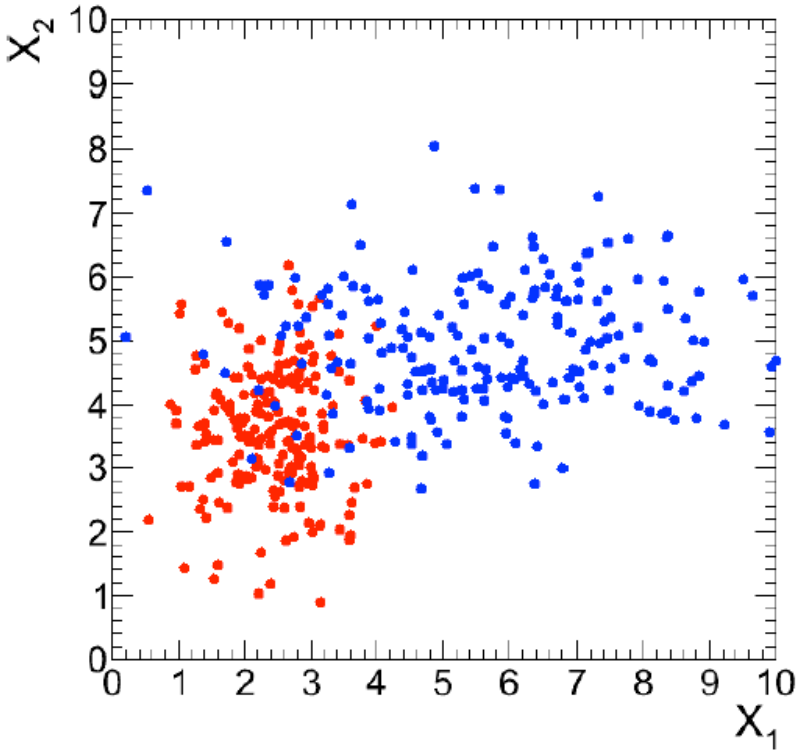
From TMVA manual

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		⋆⋆	⋆⋆	⋆	⋆	⋆⋆	⋆⋆	○	○	○	○

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

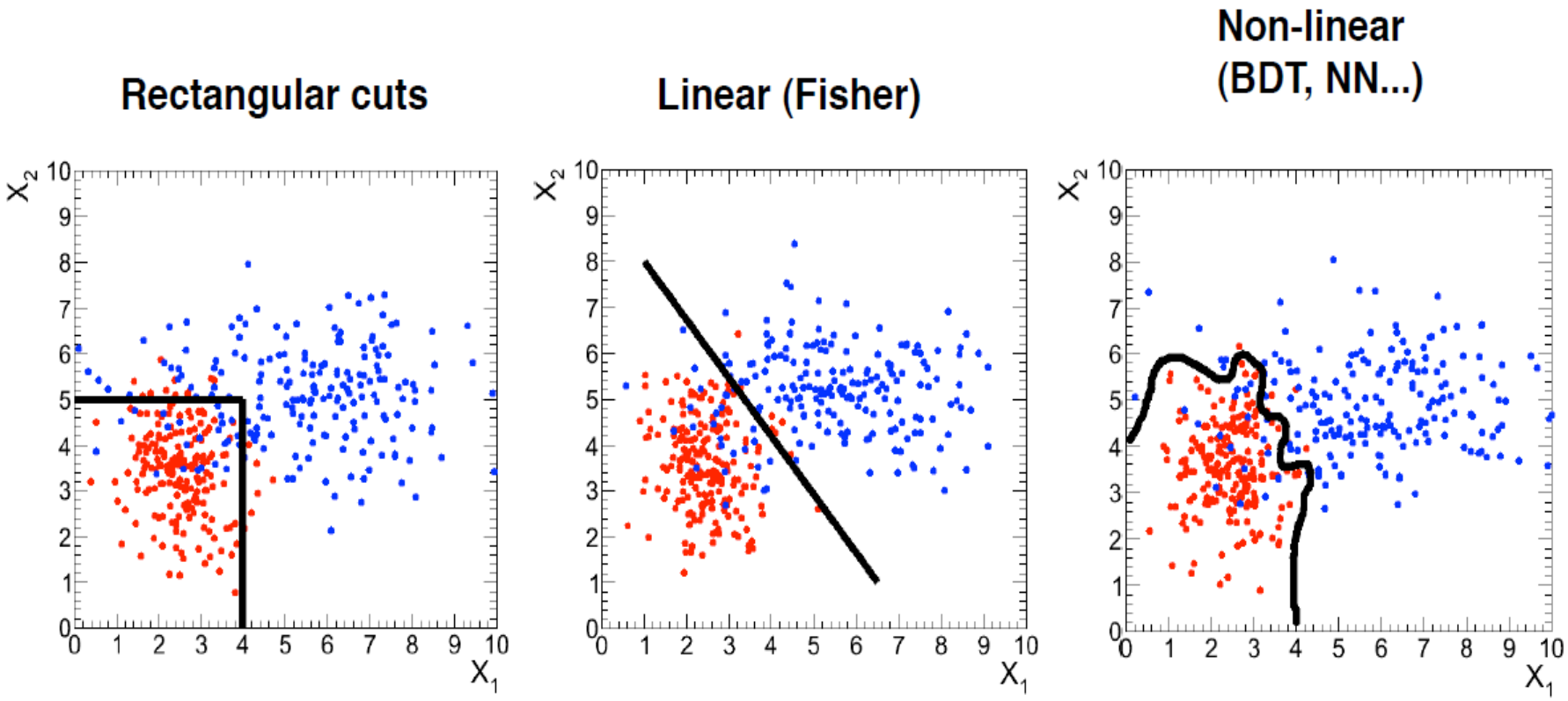


- How to decorrelate, what decision boundary (on X1 and X2) to choose, to decide if an event is signal or background ?

5

Event classification

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



Rectangular cuts

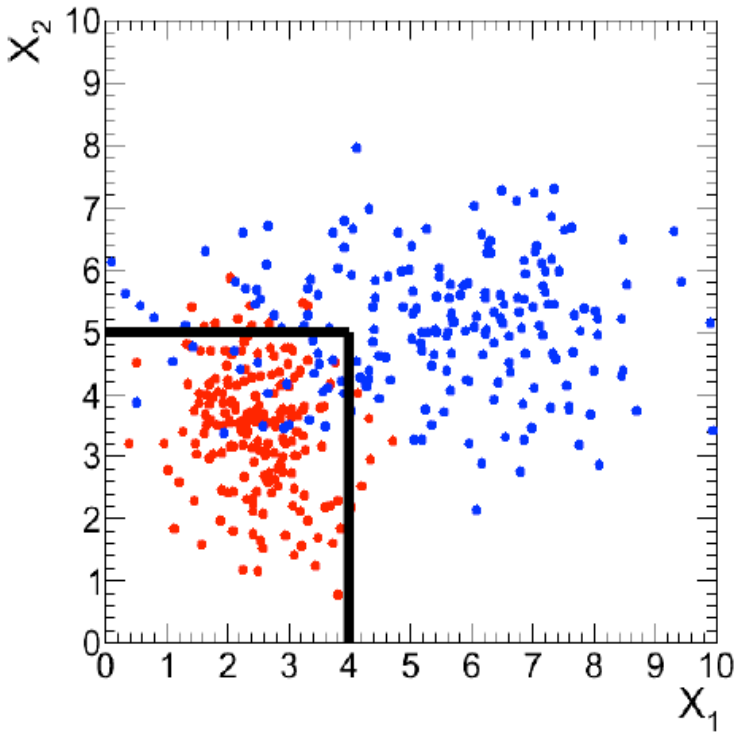
- Simplest multivariate method, very intuitive
- All HEP analyses are using rectangular cuts, not always completely optimized

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 :

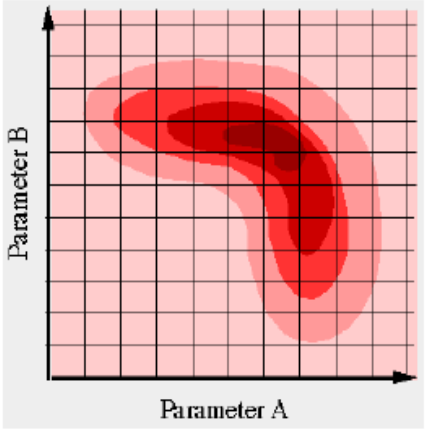
$a1 < x1 < a2$,

$b1 < x2 < b2$

...

Cut optimisation

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

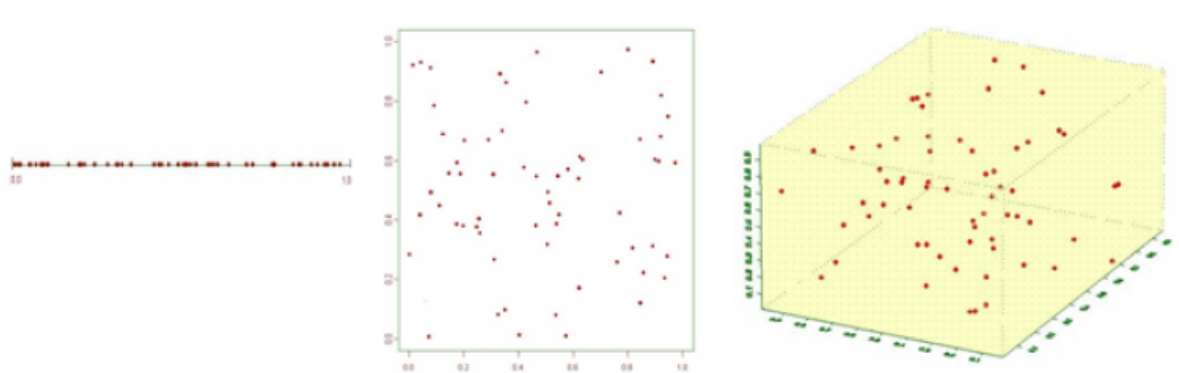
Curse of dimensionality

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

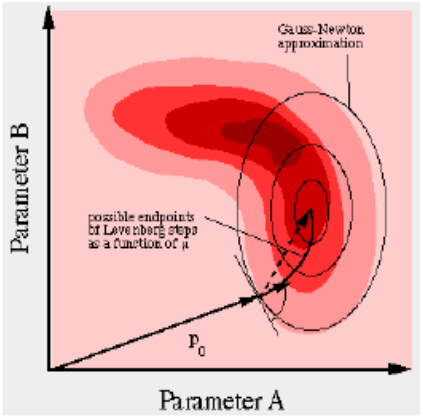


Optimisation methods

Quadratic interpolation

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

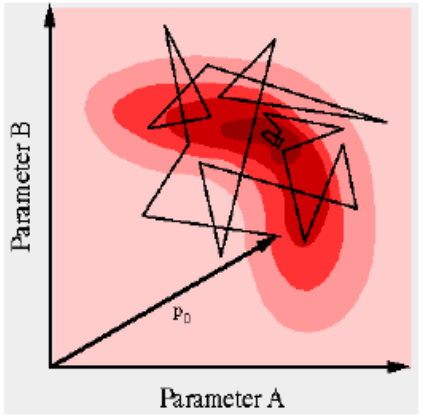
=> Problem if no minimum but a maximum is found (work around exist)



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.

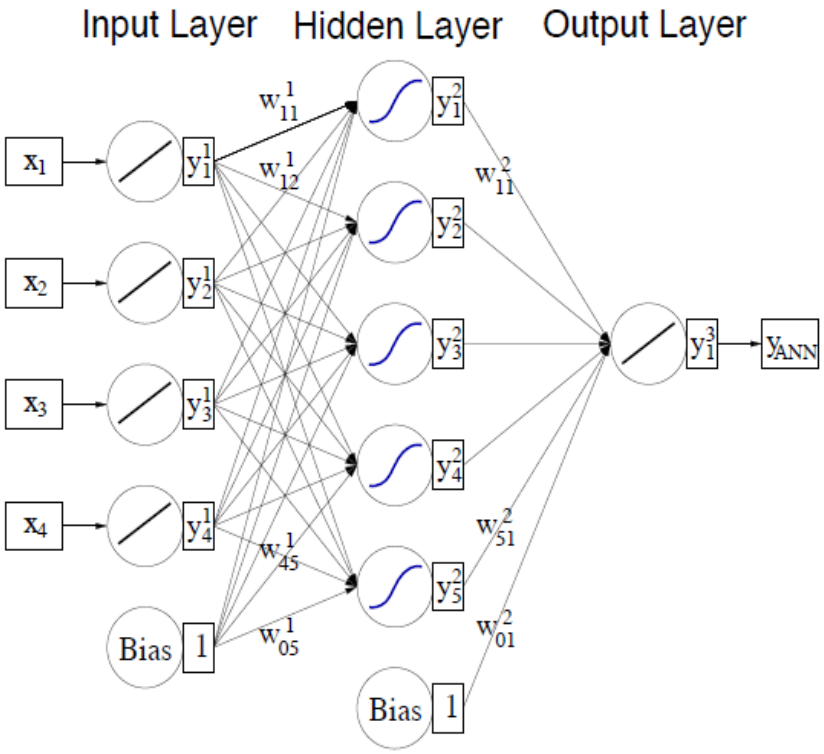


Both methods are good to find local minima

- **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

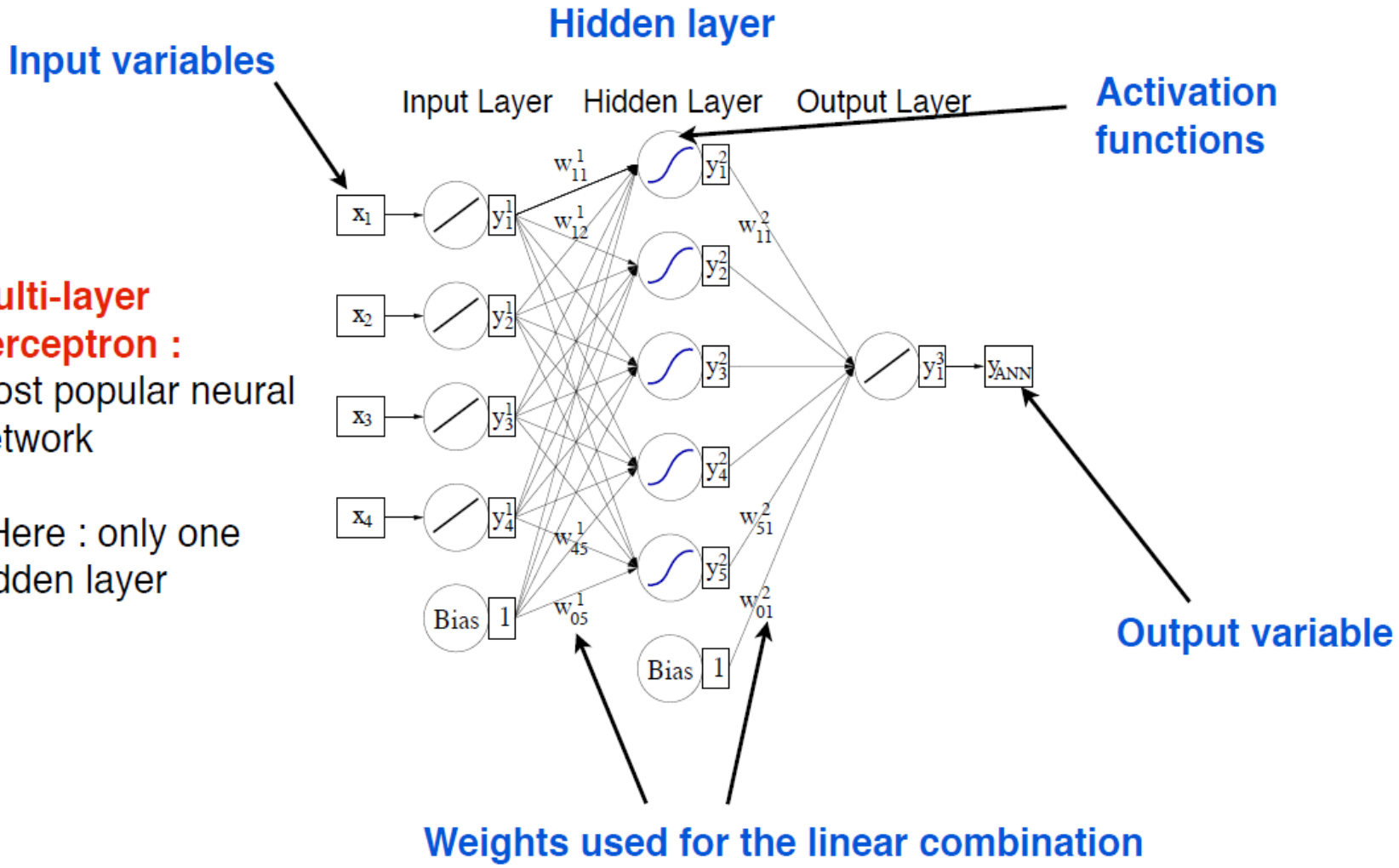
- Most commonly used : the **multi-layer perceptron**
- Composed of **neurons** taking as input a linear combination of the previous neuron outputs
- **Activation function** (usually tanh) transforms the linear combination
- **Weights** for each neurons are found during the training phase by minimizing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

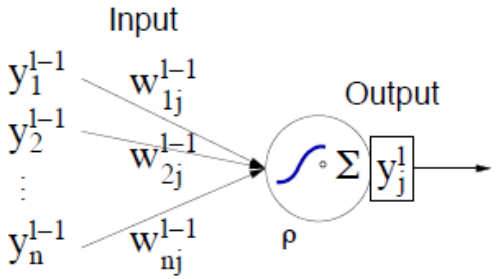


Neural network: structure

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

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

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

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 :

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

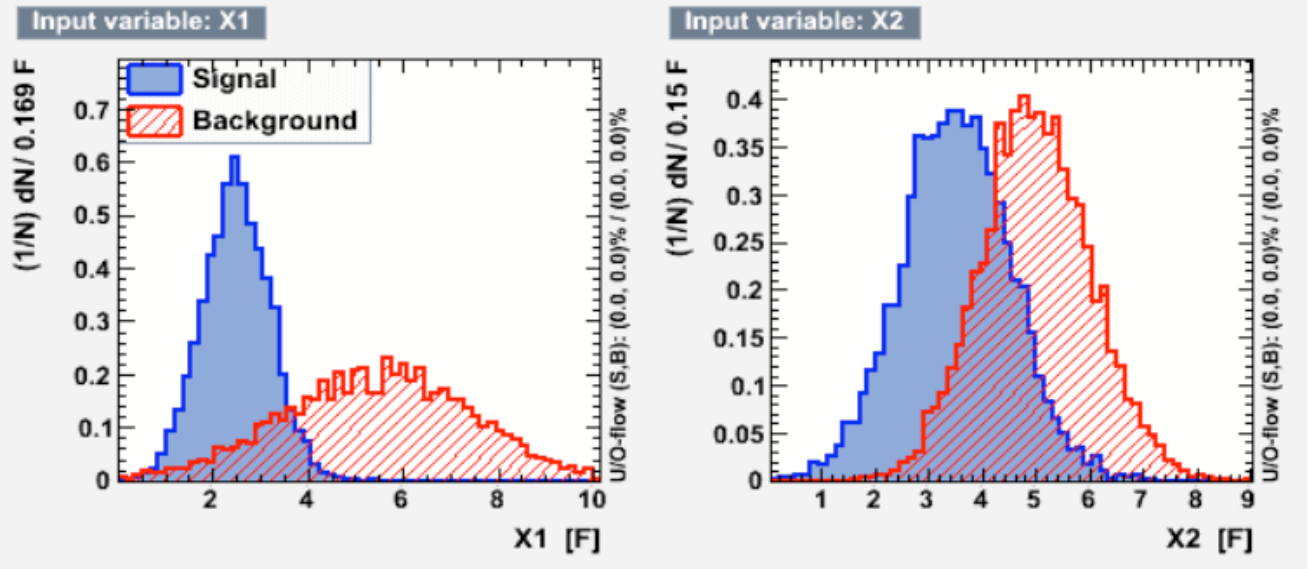
And weights connected to the hidden layer are therefore updated with :

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

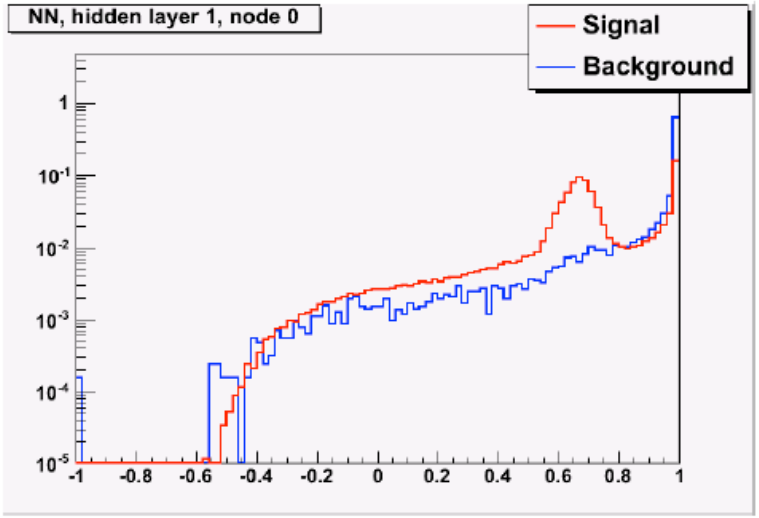
Neural network: input

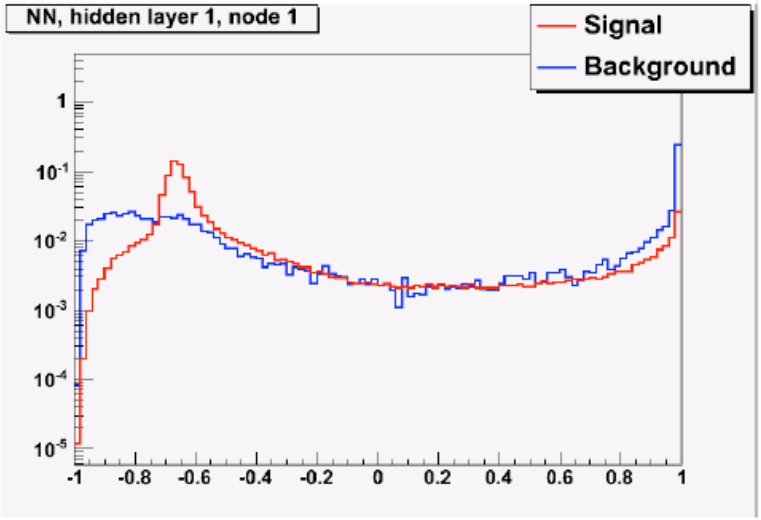
Input variables :

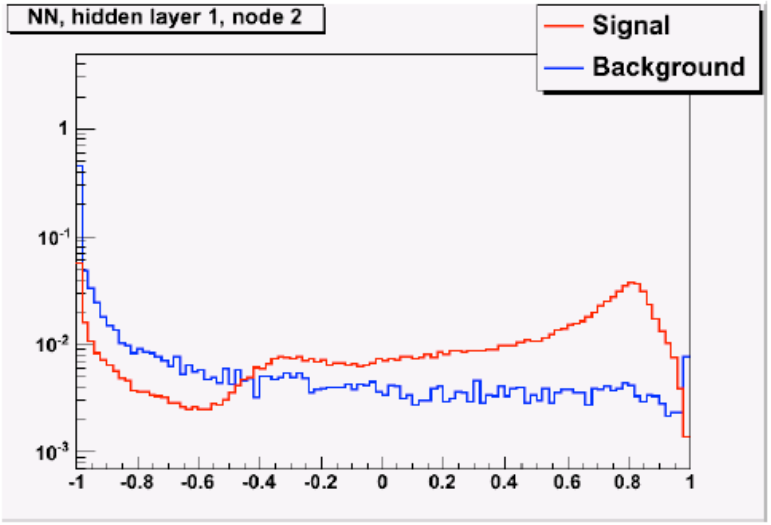
- Can be correlated (NN uses correlations)
- To improve the NN performance, should avoid unuseful variables (too much correlated, too low discrimination power)
- They can be transformed to improve their discrimination power before the training

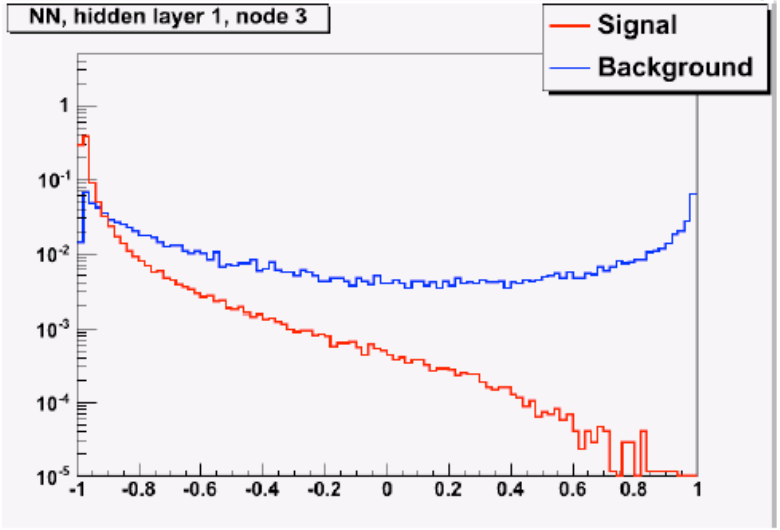


Neural network: neurons

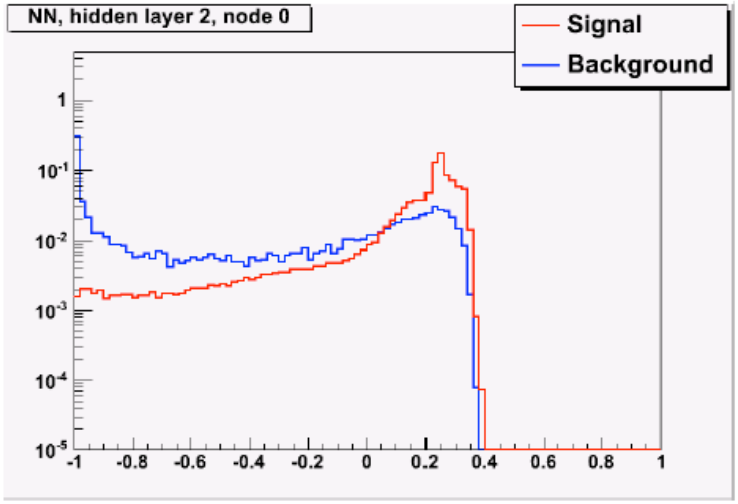


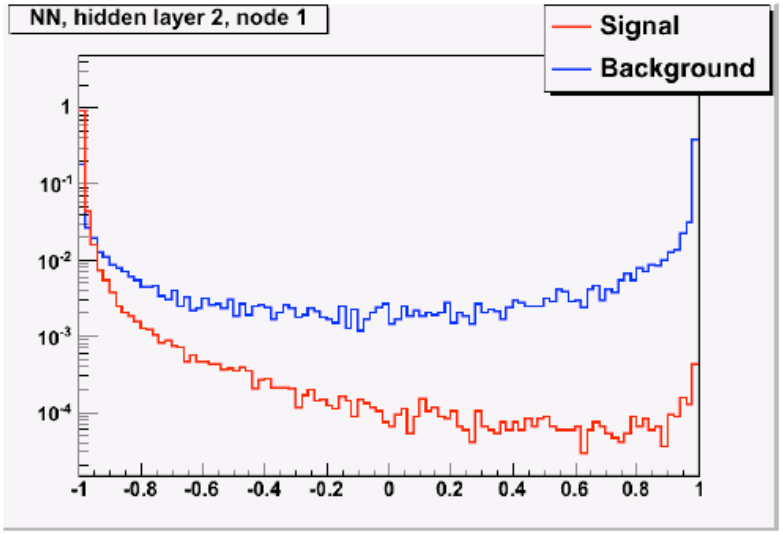


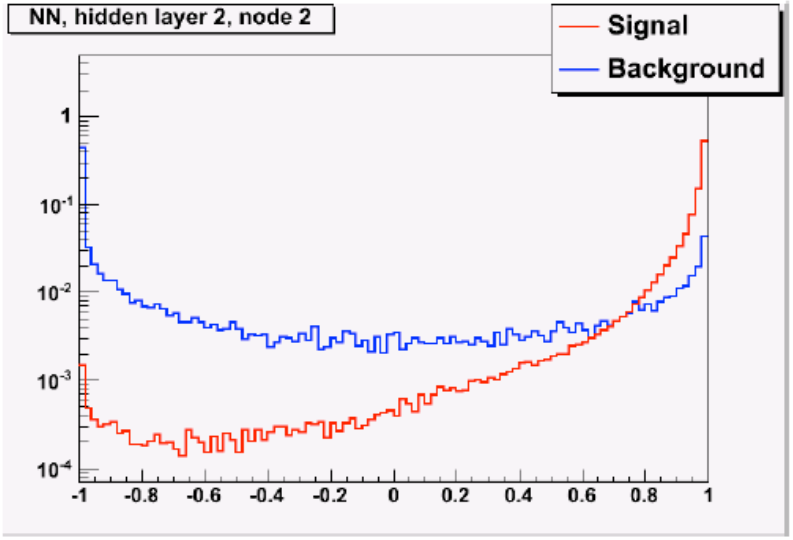




Neural network: neurons

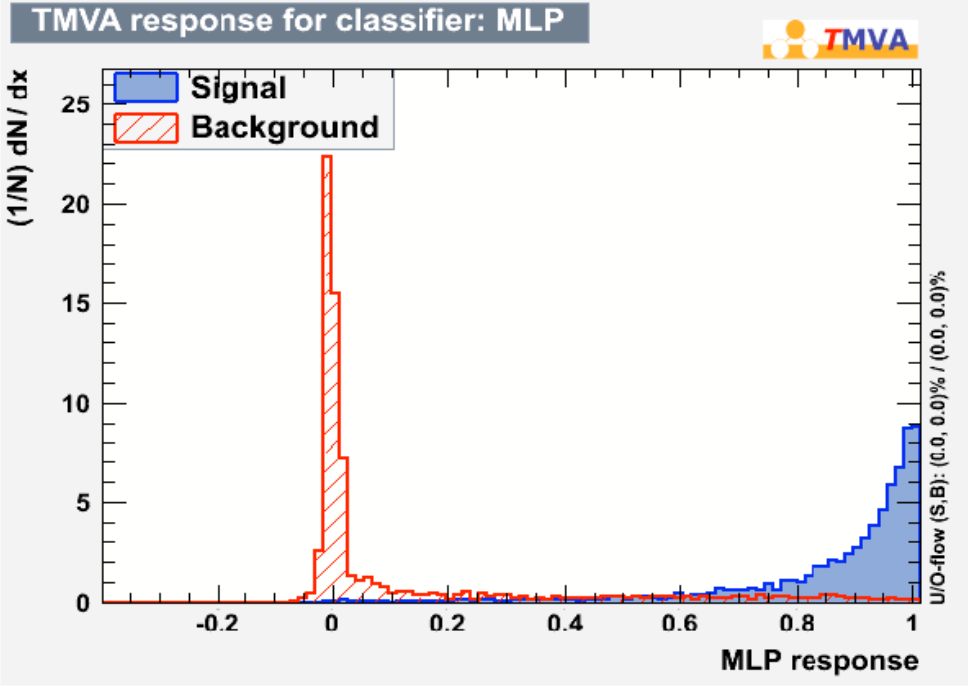






Neural network: output

- 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.

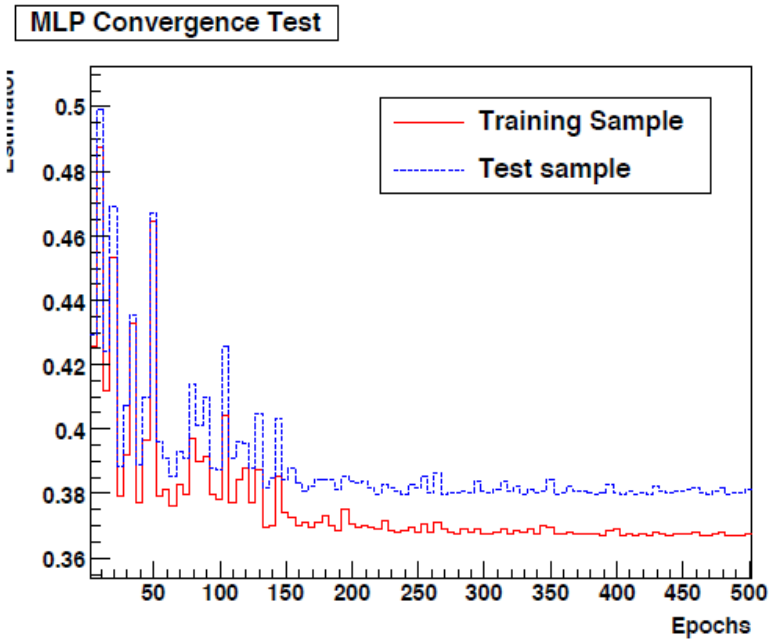


Neural network: error

- **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

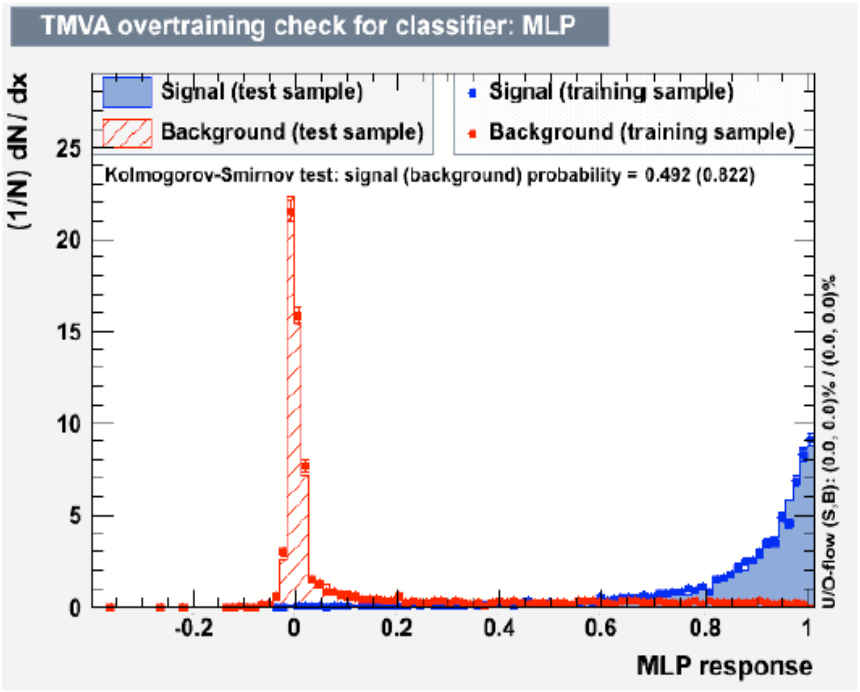


=> **Overtraining** :
- The neural network was trained to recognize even the statistical fluctuations of the training sample and is therefore not suitable for any test sample

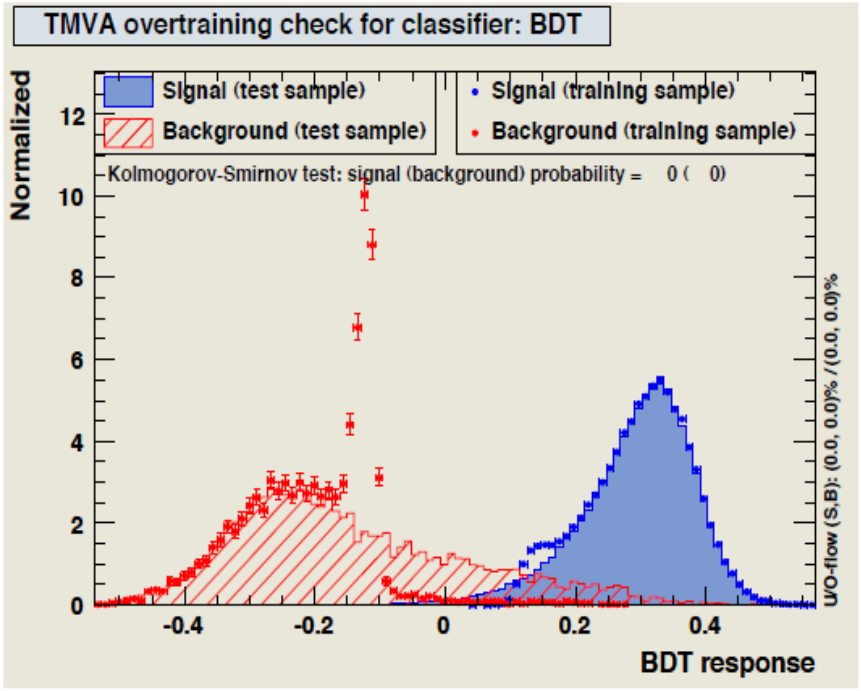
Neural network: overtraining

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

Not overtrained



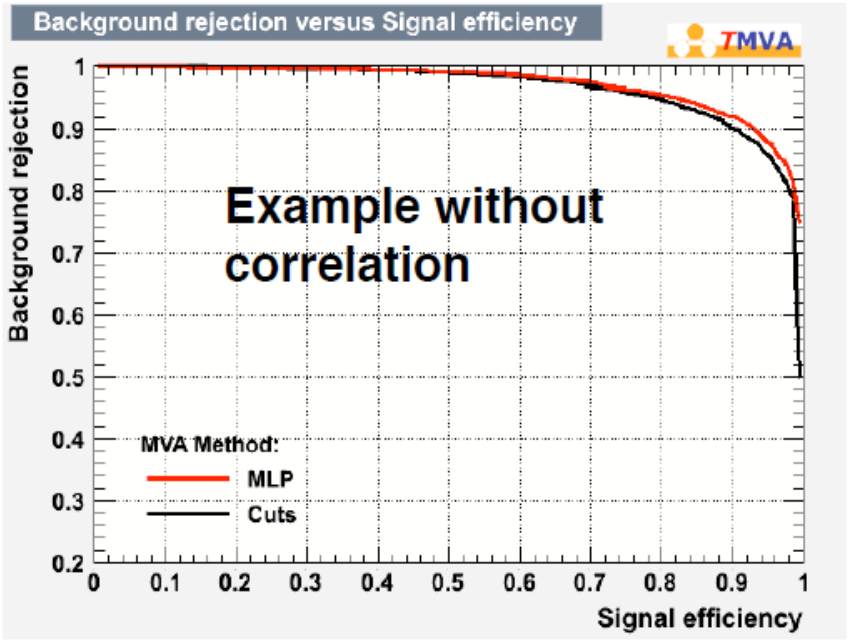
Overtrained

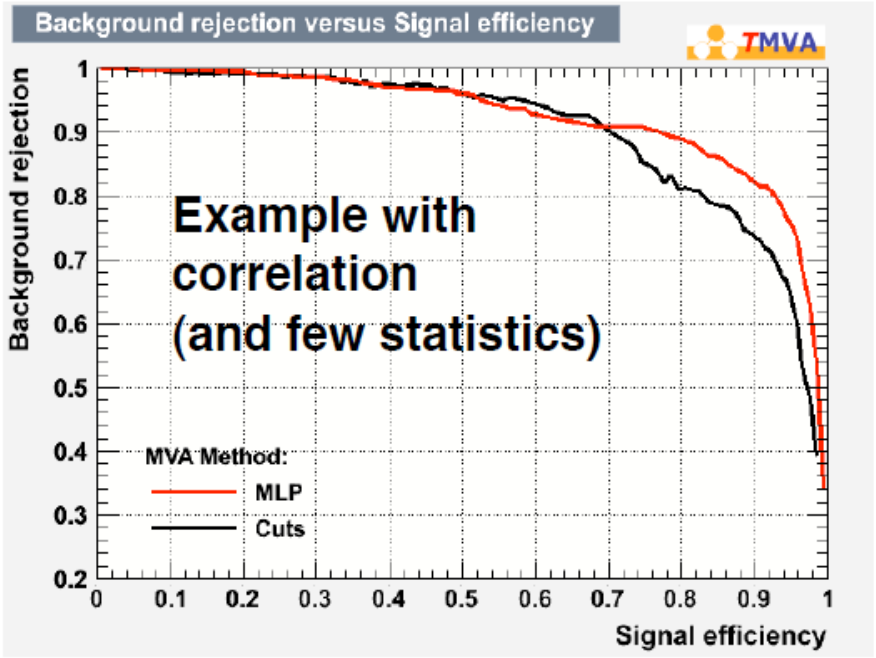


Neural network: performance

Usual figure of merit to check the performance :

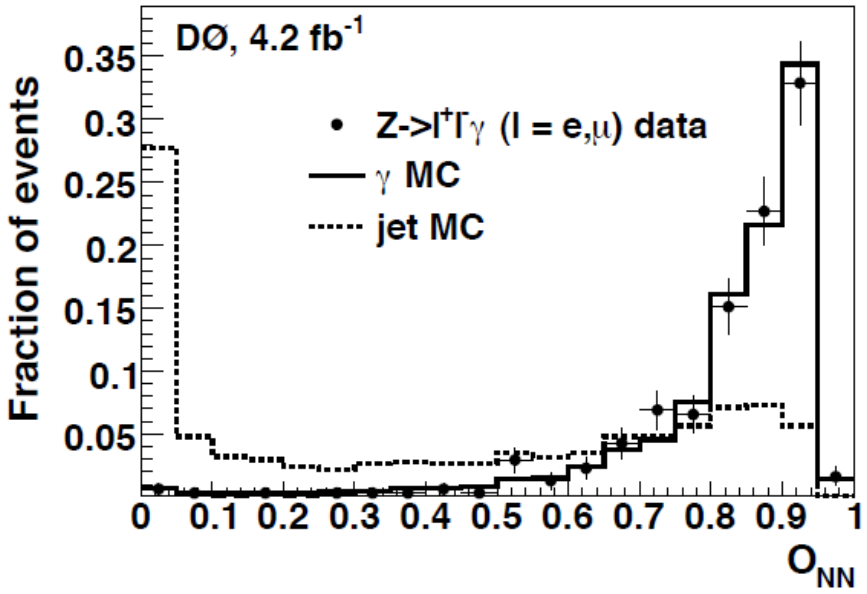
- Scan the performance varying the cut on the network output
- Plot the signal efficiency versus background efficiency (or background rejection). Each cut on the NN output is one point on the figure.
- The NN performs (almost all the time) better than the rectangular cut

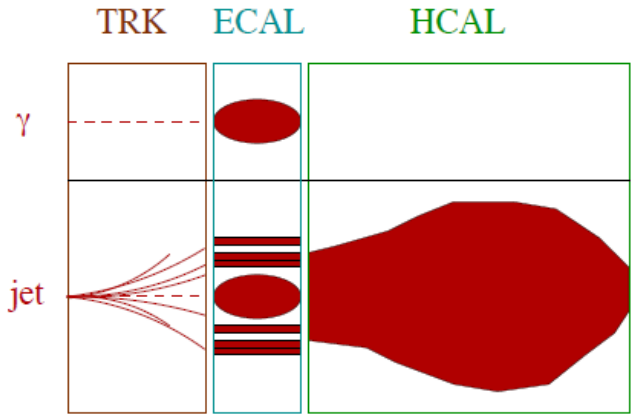


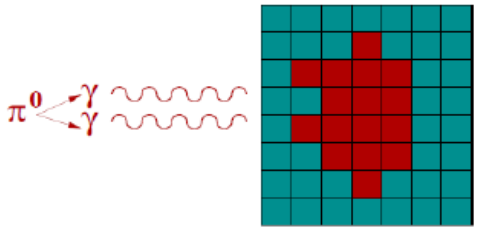


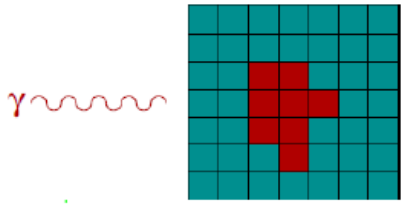
Neural network: example in HEP

Photon identification at D0 and applications









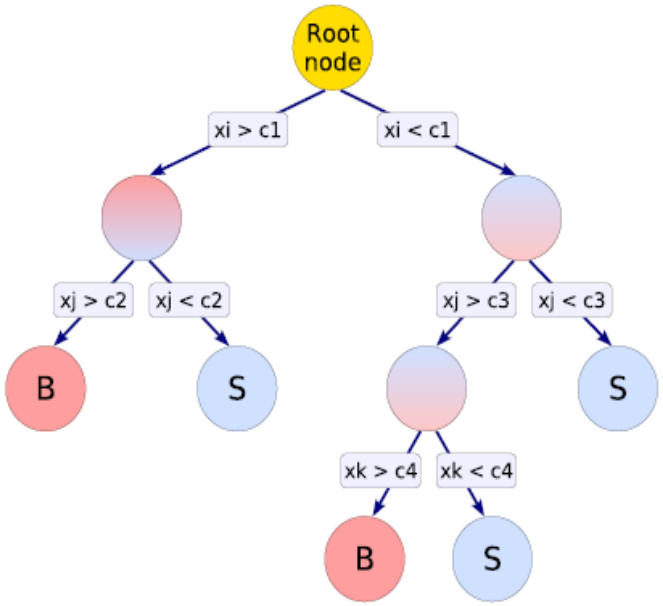
Goal : discriminate photons against neutral mesons in jets

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)

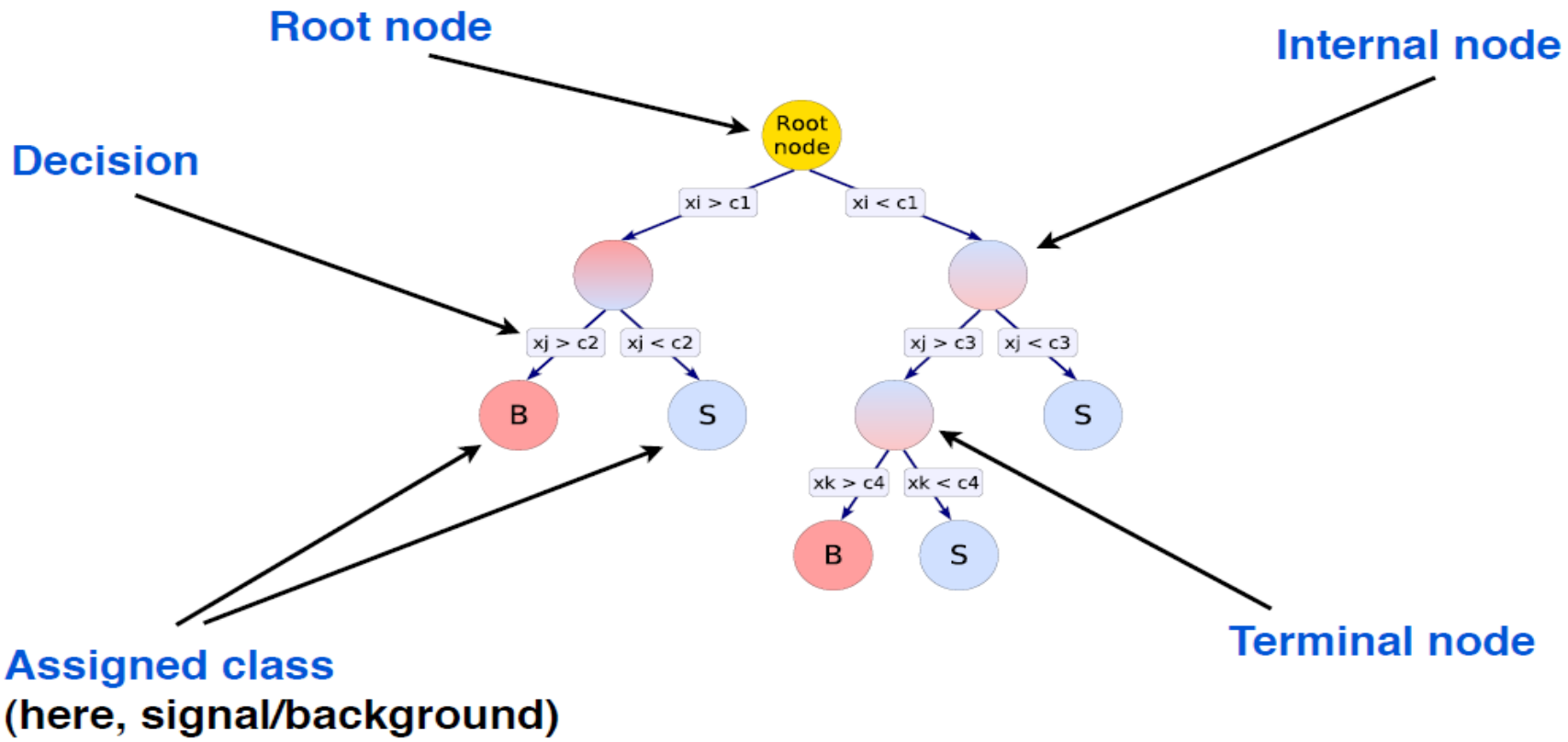


- Decision trees are **extremely sensitive to the training samples**, therefore to overtraining

- 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

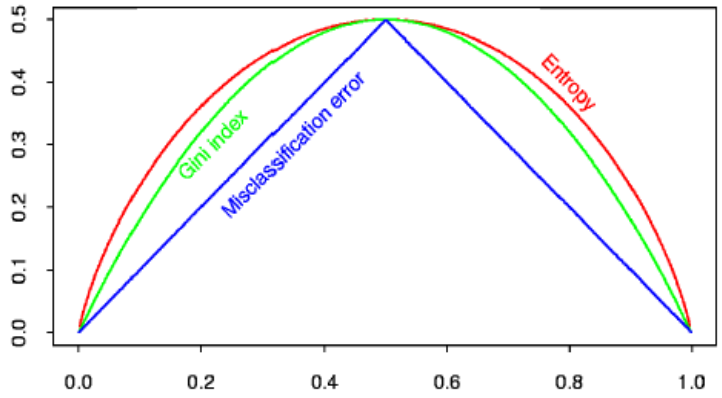


Decision tree: training

- **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...)

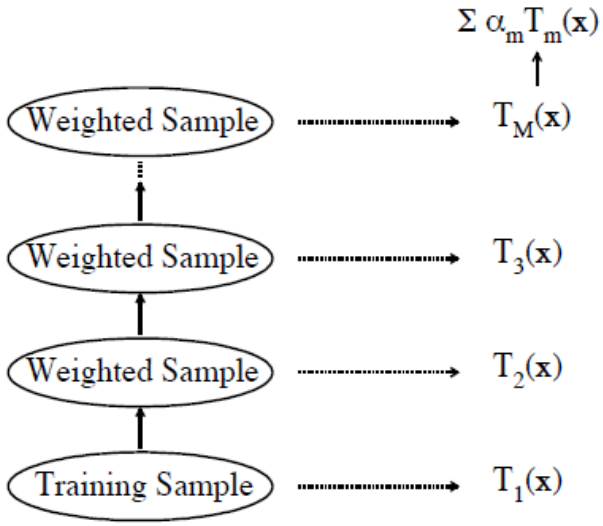


FIG. 2: Schematic of a boosting procedure.

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.

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

Decision tree: AdaBoost

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)

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

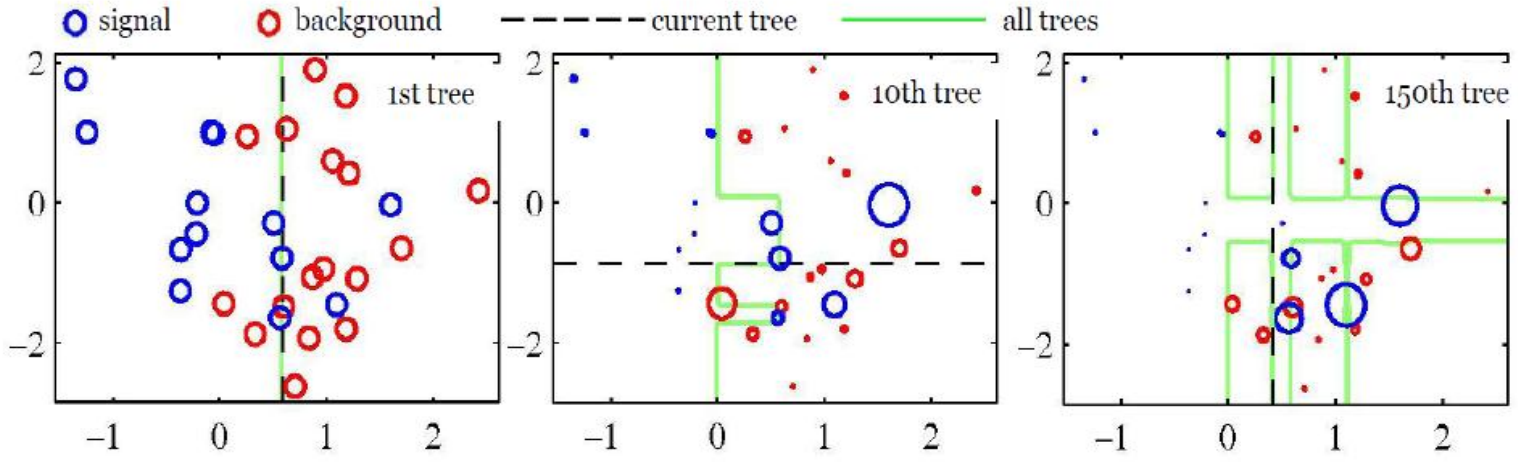
weight of the ith event :

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

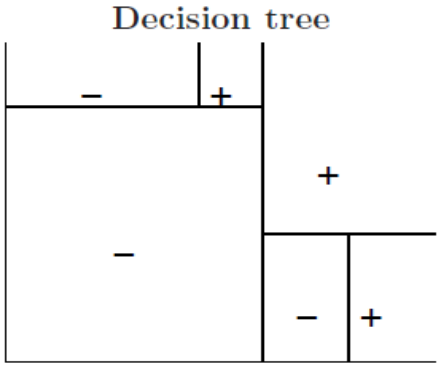
Start here: equal event weights

misclassified events get larger weights

… and so on



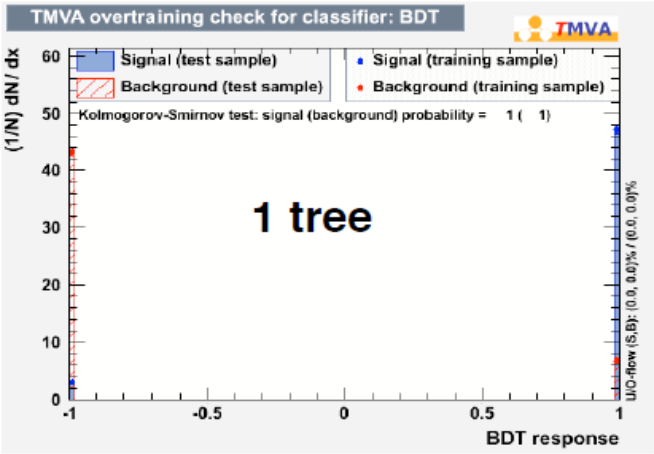
Decision tree: output

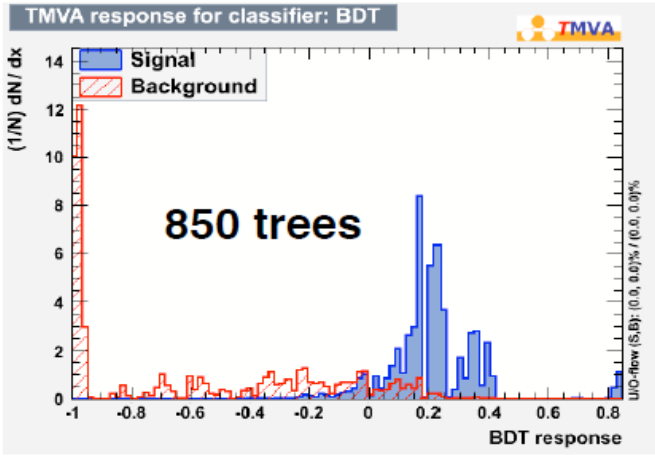


- A single decision tree can be trained to gives always an integer response, : signal (+1) / background (-1)

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 forest, prunning

- 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 :

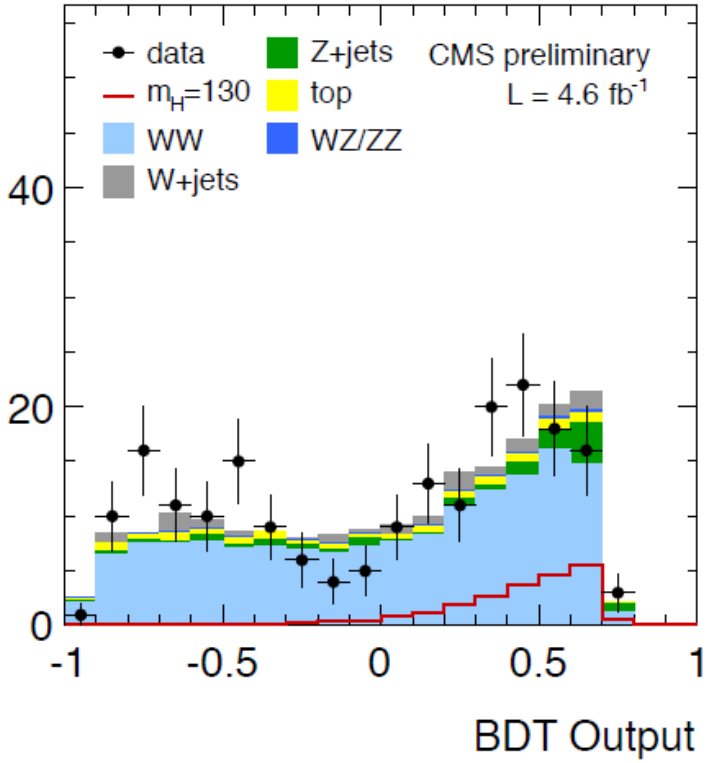
- Training repeated on random bootstrap (or subsets) of the training data only
- Consider at each node only a random subsets of variables for the split

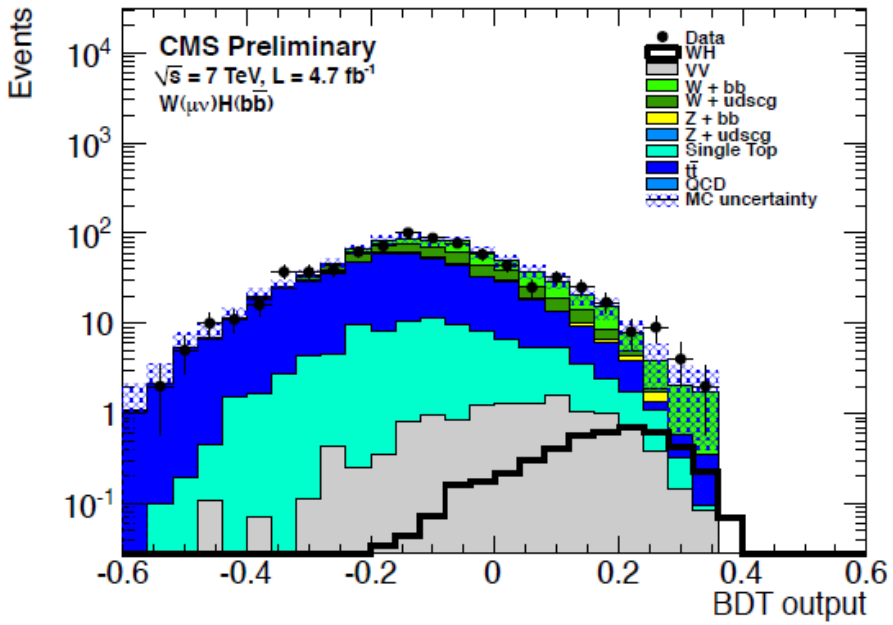
Pruning :

- 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 :

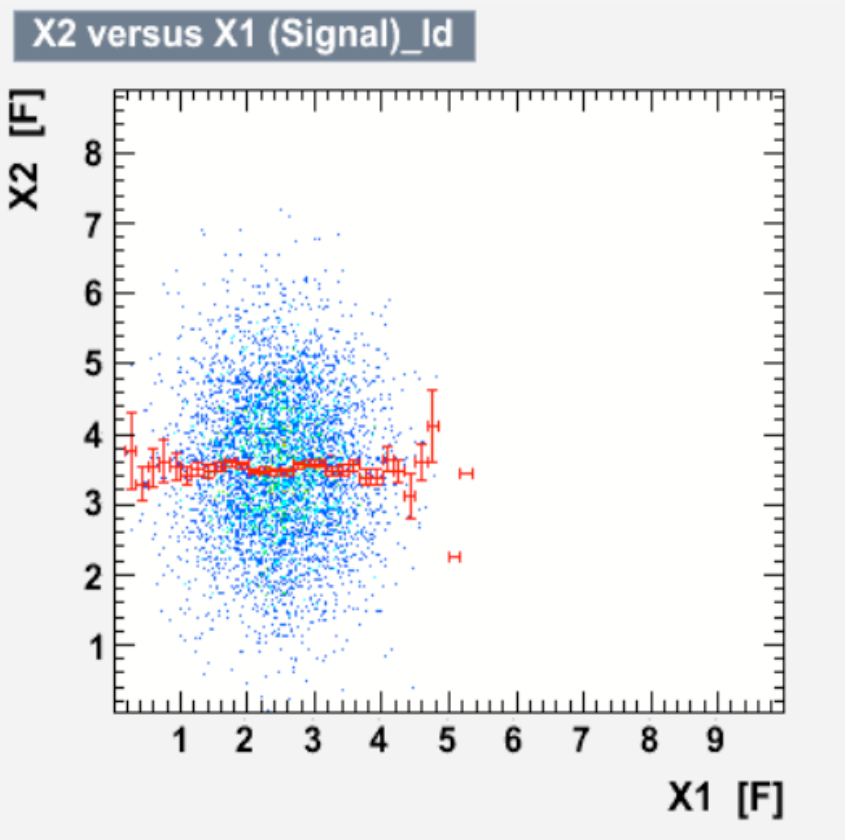
- Allows to check input variables, correlations, overtraining, performance
- **Many multivariate methods** available : rectangular cuts, likelihood, various decision trees, SVM...
- **Classification** and **regression**
- **Tuning of parameters** relatively easy
- **Training is user-friendly** and fast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

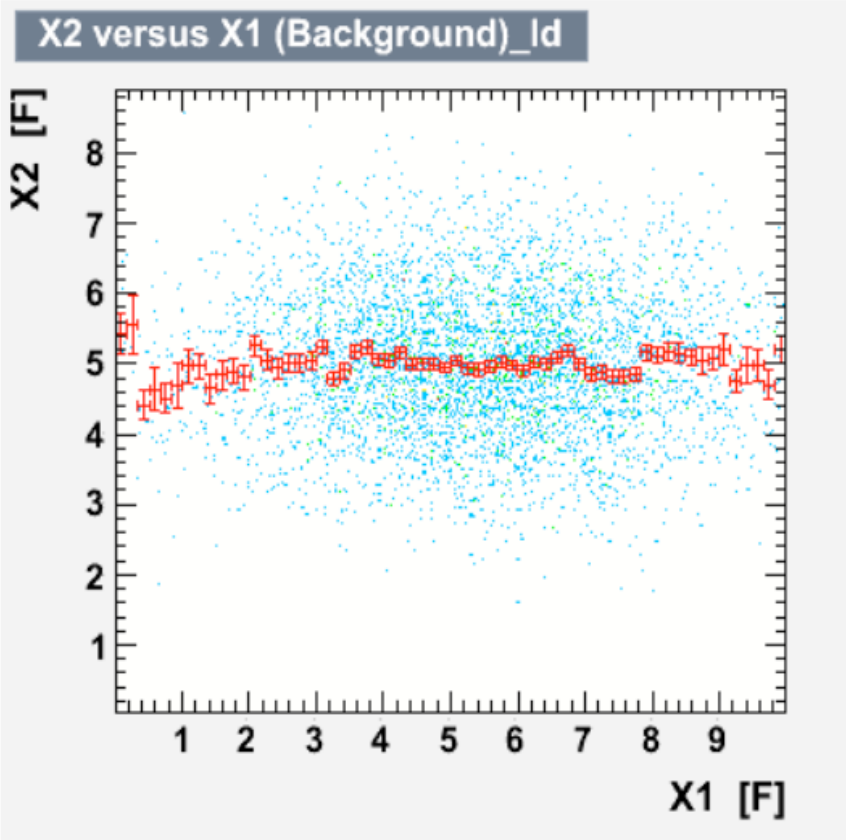
Available classifiers

```
// --- 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

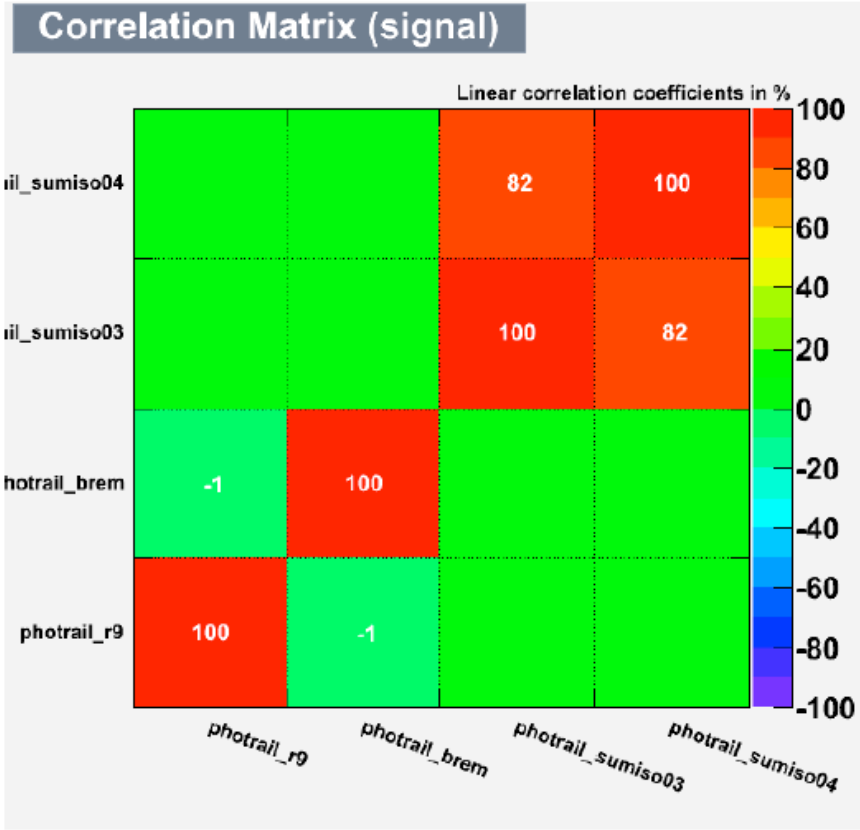
- Linear correlations are easily investigated via the GUI :
- (Here, no correlation)

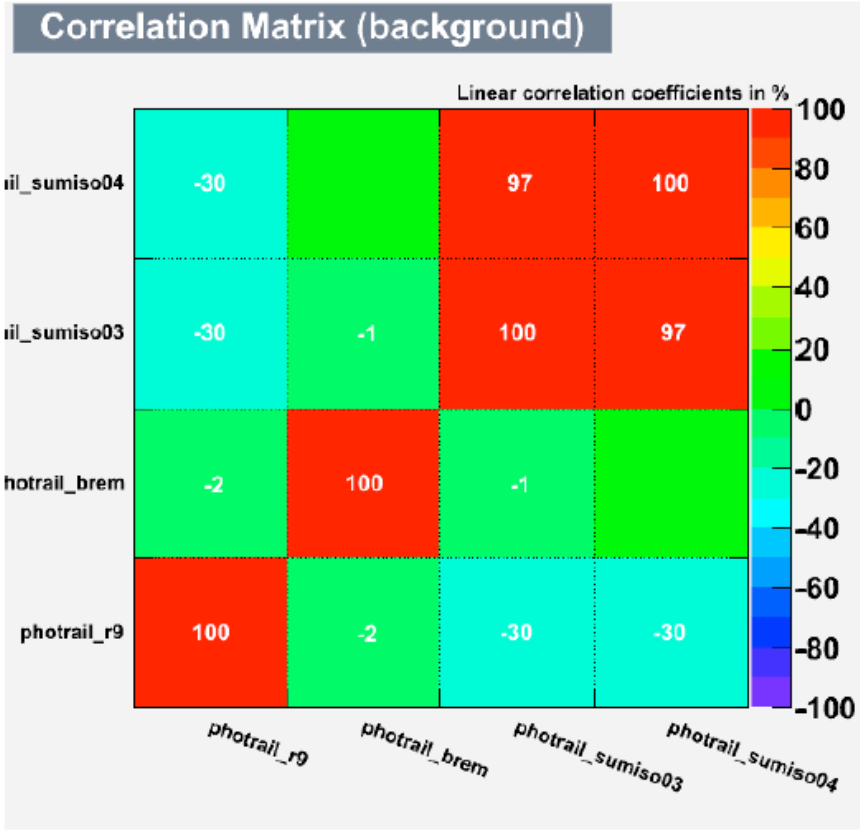




Functionalities: correlations

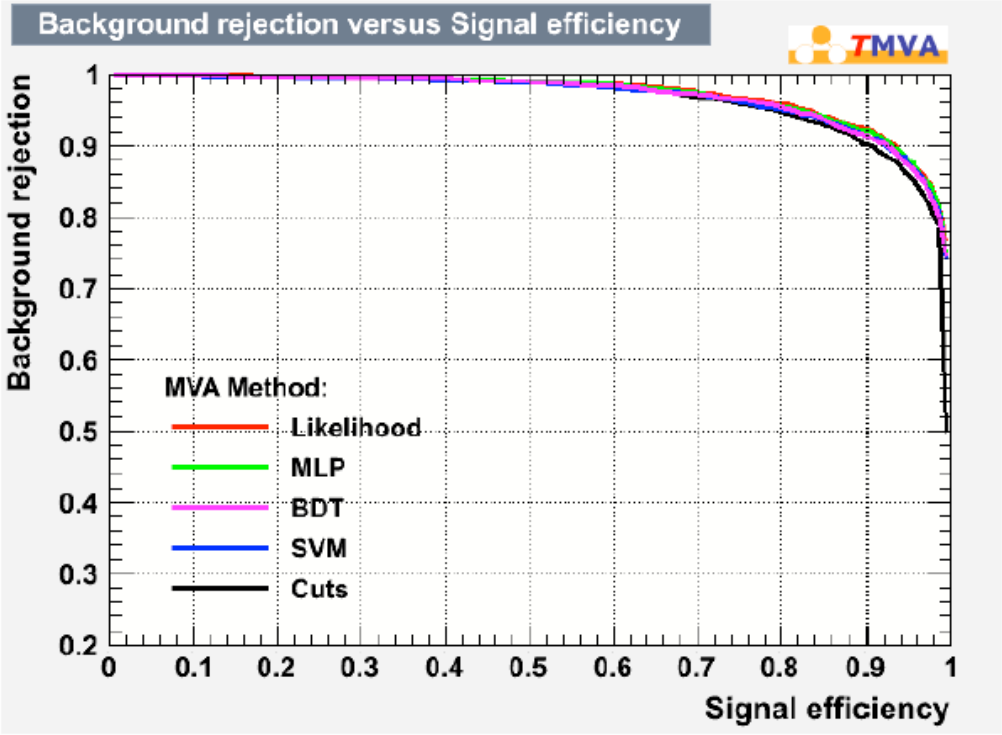
- Linear correlations are easily investigated via the GUI :
- Signal and background input variables can be correlated differently





Functionalities: performance

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



Advantages and draw backs of different classifiers.

From TMVA manual

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		⋆⋆	⋆⋆	⋆	⋆	⋆⋆	⋆⋆	○	○	○	○