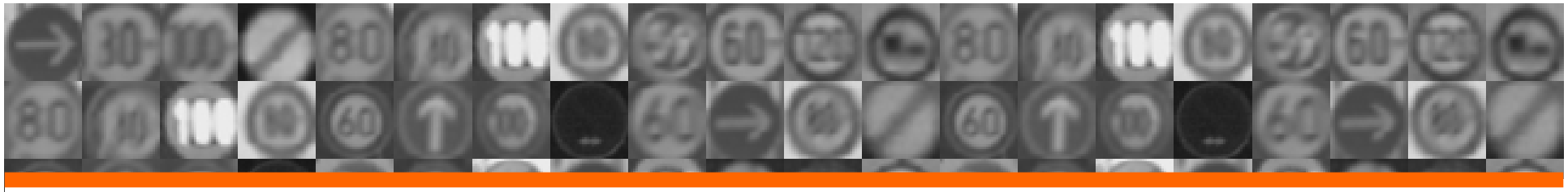
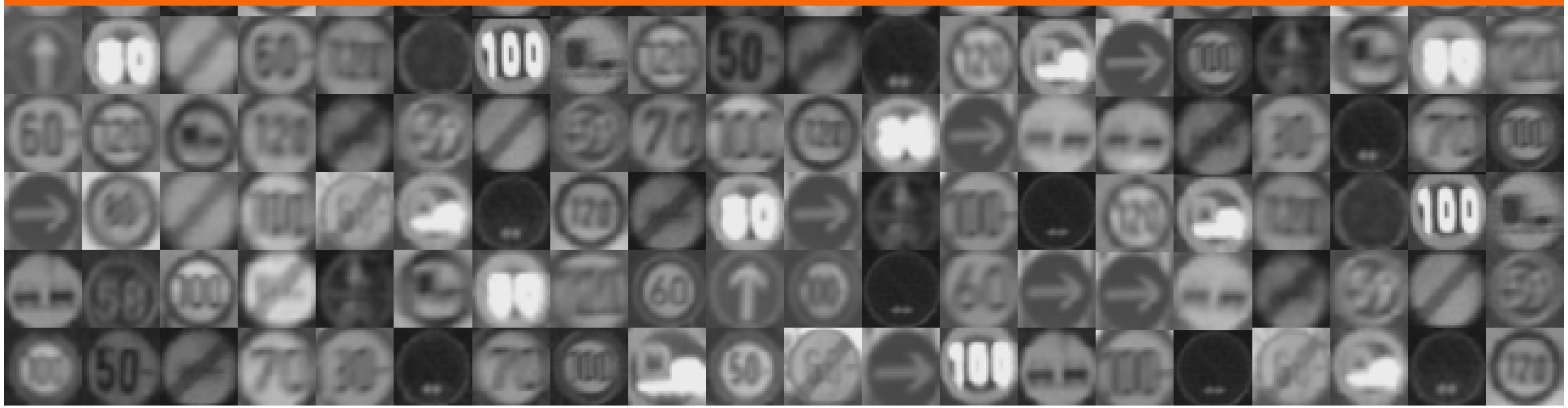


Data Analysis With TMVA

Helge Voss (MPI–K, Heidelberg)

Seminar , Lausanne, 12 April 2010



MVA-Literature /Software Packages... a biased selection

Literature:

- T.Hastie, R.Tibshirani, J.Friedman, "*The Elements of Statistical Learning*", Springer 2001
- C.M.Bishop, "*Pattern Recognition and Machine Learning*", Springer 2006

Software packages for Mulitvariate Data Analysis/Classification

- individual classifier software
  - e.g. "JETNET" C.Peterson, T. Rognvaldsson, L.Loennblad

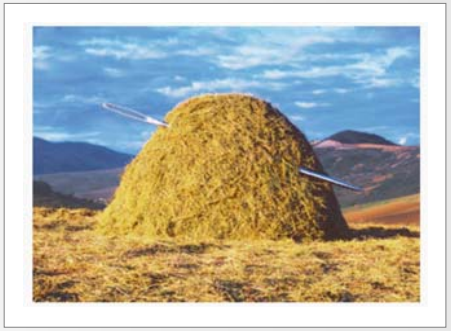
  and many other packages

- attempts to provide "all inclusive" packages
  - StatPatternRecognition: I.Narsky, *arXiv: physics/0507143*
    http://www.hep.caltech.edu/~narsky/spr.html
  - TMVA: Höcker,Speckmayer,Stelzer,Therhaag,von Toerne,Voss, *arXiv: physics/0703039*
    http://tmva.sf.net or every ROOT distribution (development moved from SourceForge to ROOT repository)
  - WEKA: http://www.cs.waikato.ac.nz/ml/weka/
  - "R": a huge data analysis library: http://www.r-project.org/

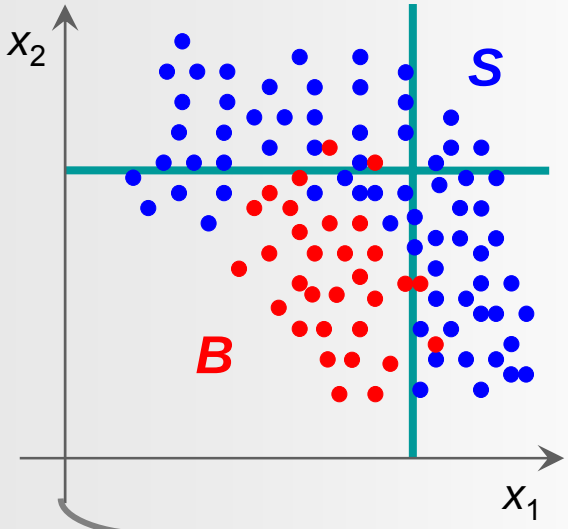
Conferences: PHYSTAT, ACAT,…

Event Classification

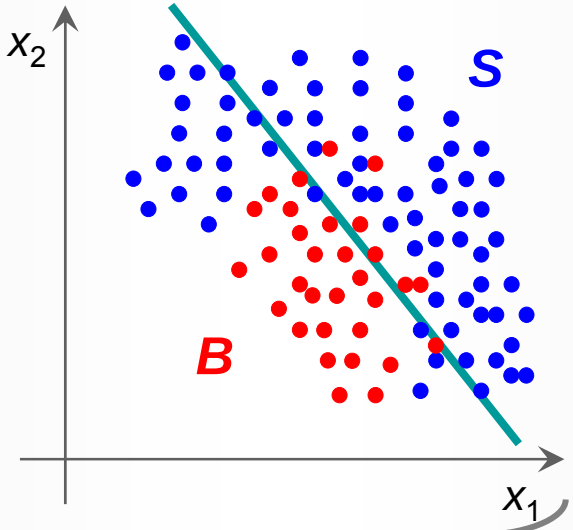
- Suppose data sample of two types of events: with class labels *Signal* and *Background* (will restrict here to two class cases. Many classifiers can in principle be extended to several classes, otherwise, analyses can be staged)
  - how to set the decision boundary to select events of type *S* ?
  - we have discriminating variables x_1, x_2, …



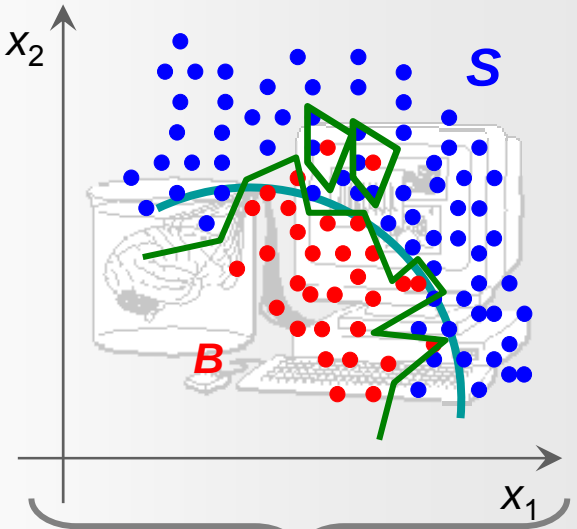
Rectangular cuts?



A linear boundary?



A nonlinear one?



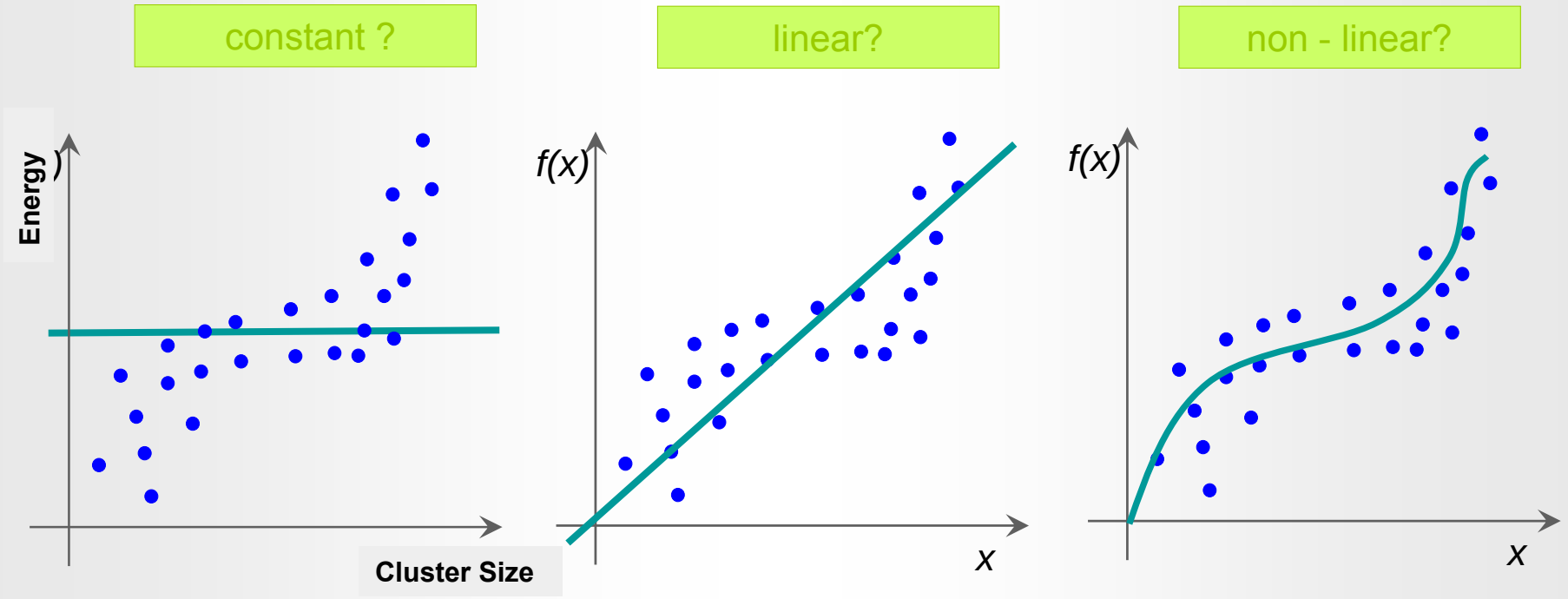
- How can we decide what to uses ?

Low variance (stable), high bias methods — High variance, small bias methods

- Once decided on a class of boundaries, how to find the “optimal” one ?

Regression

- how to estimate a “functional behaviour” from a given set of ‘known measurements” ?
- assume for example “D”-variables that somehow characterize the shower in your calorimeter.



- seems trivial ? → The human brain has very good pattern recognition capabilities!
- what if you have **many** input variables?

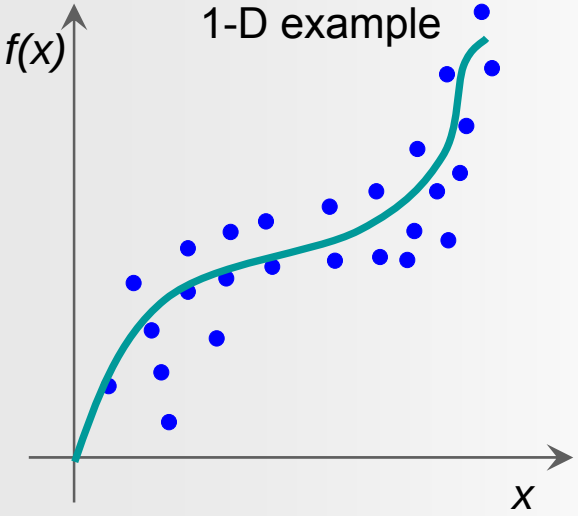
Regression → model functional behaviour

- Assume for example "D"-variables that somehow characterize the shower in your calorimeter.
  - Monte Carlo or testbeam

    → data sample with measured cluster observables

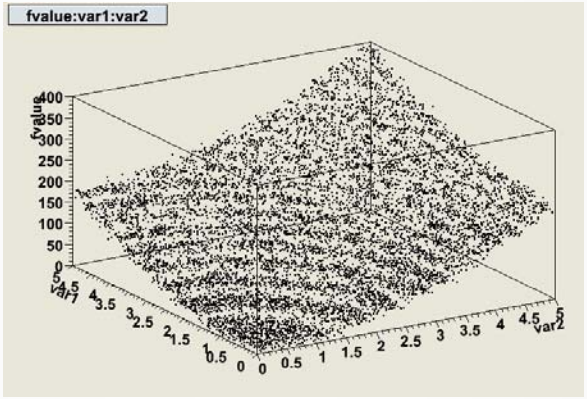
    + known particle energy

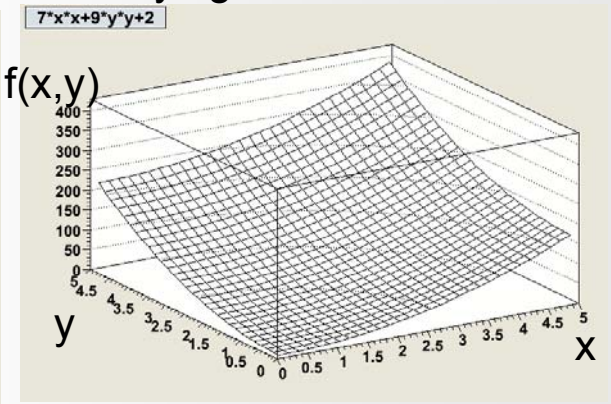
    = calibration function (energy == surface in D+1 dimensional space)



2-D example

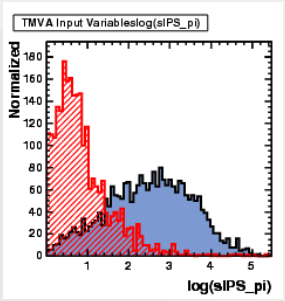
events generated according: underlying distribution

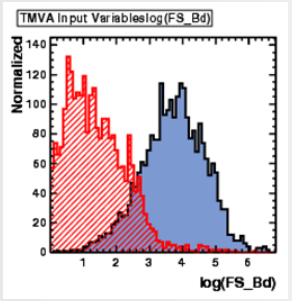


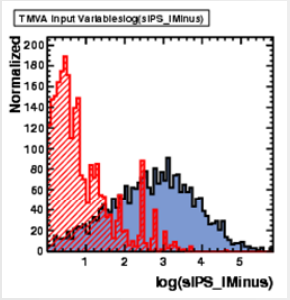


- better known: (linear) regression → fit a known analytic function
  - *e.g.* the above 2-D example → reasonable function would be: $f(x) = ax^2+by^2+c$
- what if we don't have a reasonable "model" ? → need something more general:
  - *e.g.* piecewise defined splines, kernel estimators, decision trees to approximate f(x)

Event Classification





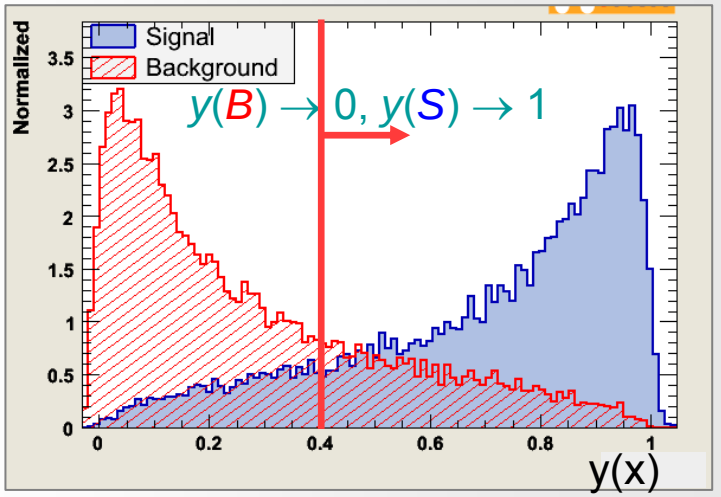


$\mathbb{R}^D$

"feature space"

- Each event, if Signal or Background, has "D" measured variables.
- Find a mapping from D-dimensional input/observable/"feature" space to one dimensional output → class lables

y(x): $R^n \rightarrow R$: → $\mathbb{R}$

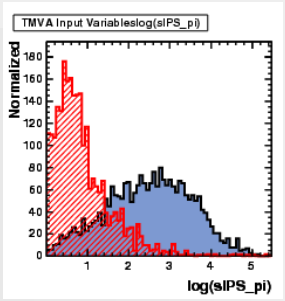


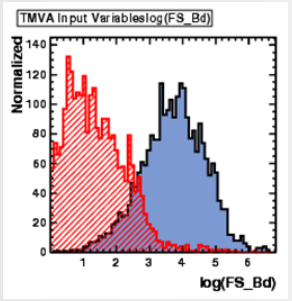
- y(x): "test statistic" in D-dimensional space of input variables
- distributions of y(x): $PDF_S(y)$ and $PDF_B(y)$
- used to set the selection cut!
- → efficiency and purity

y(x): > cut: signal; = cut: decision boundary; < cut: background

- y(x)=const: surface defining the decision boundary.
- overlap of $PDF_S(y)$ and $PDF_B(y)$ → separation power , purity

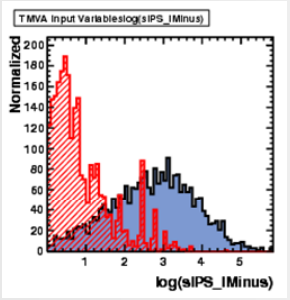
Classification ↔ Regression





$\mathbb{R}^D$

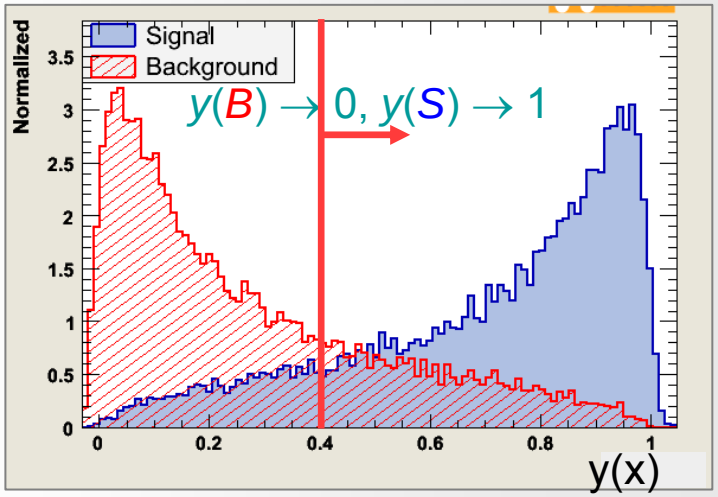
"feature space"



Classification:

- Each event, if Signal or Background, has "D" measured variables.

- y(x): $R^D \rightarrow R$: "test statistic" in D-dimensional space of input variables
- y(x)=const: surface defining the decision boundary.

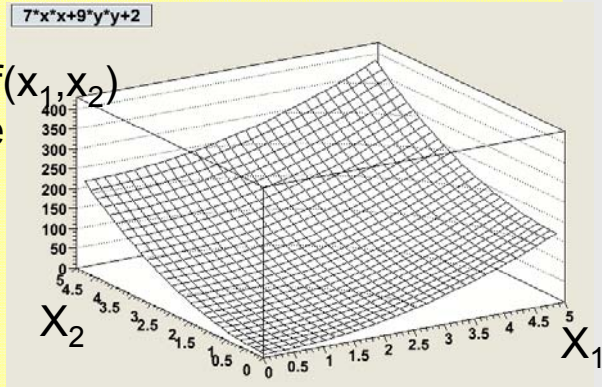


y(x): $R^D \rightarrow R$: $\longrightarrow \mathbb{R}$

Regression:

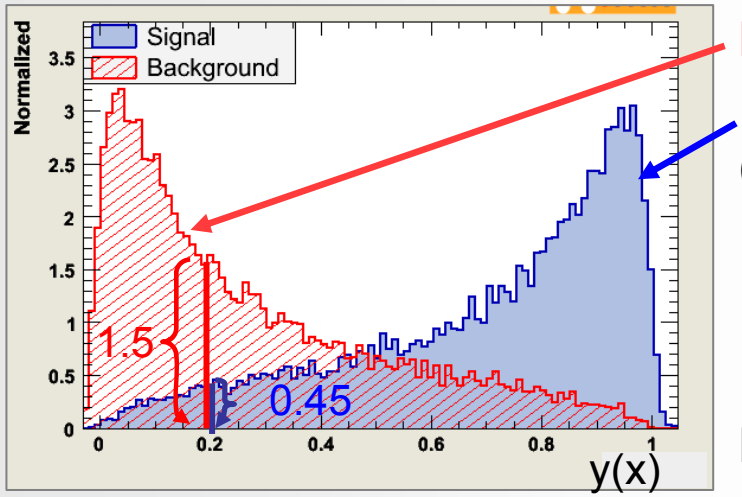
- Each event has "D" measured variables + one function value (*e.g.* cluster shape variables in the ECAL + particles energy)
- y(x): $R^D \rightarrow R$ → find
- y(x)=const → hyperplanes where the target function is constant

Now, y(x) needs to be build such that it best approximates the target, not such that it best separates signal from bkgr.



Event Classification

y(x): $R^n \rightarrow R$: the mapping from the "feature space" (observables) to one output variable



$PDF_B(y)$. $PDF_S(y)$: normalised distribution of y=y(x) for background and signal events (i.e. the "function" that describes the shape of the distribution)

with y=y(x) one can also say $PDF_B(y(x))$, $PDF_S(y(x))$: :

Probability densities for background and signal

now let's assume we have an unknown event from the example above for which y(x) = 0.2

→ $PDF_B(y(x)) = 1.5$ and $PDF_S(y(x)) = 0.45$

let f_S and f_B be the fraction of signal and background events in the sample, then:

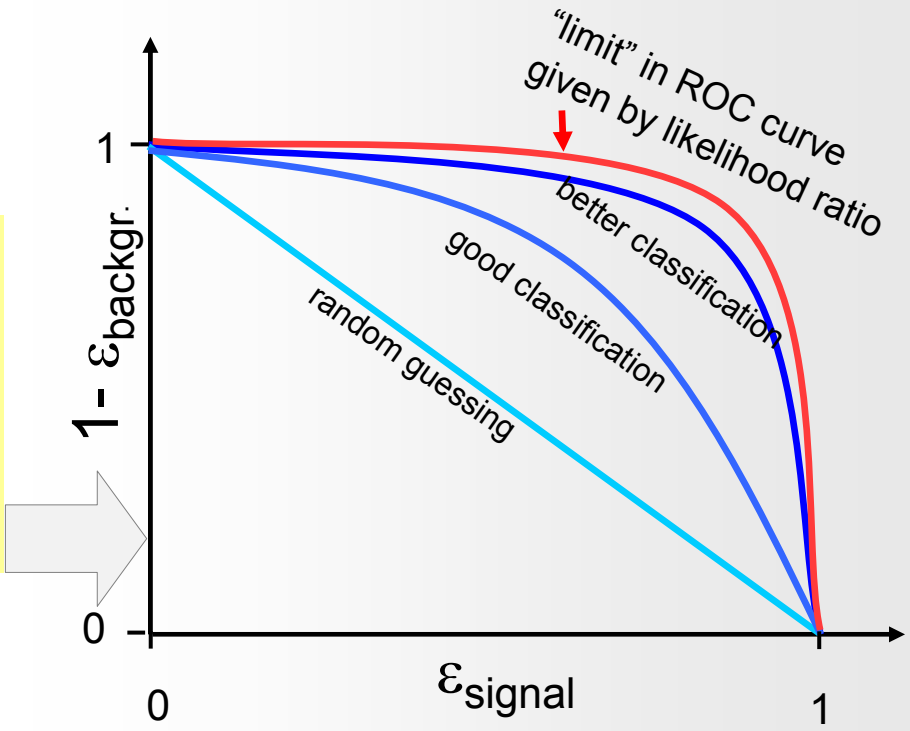
$$\frac{f_S PDF_S(y)}{f_S PDF_S(y) + f_B PDF_B(y)} = P(C = S \mid y)$$

is the probability of an event with measured $\mathbf{x}=\{x_1,\ldots,x_D\}$ that gives y(x) to be of type signal

Neyman-Pearson Lemma

$$\text{Likelihood Ratio}: \quad y(x) = \frac{P(x|S)}{P(x|B)}$$

Neyman-Peason:
The Likelihood ratio used as "selection criterium" y(x) gives for each selection efficiency the best possible background rejection.
i.e. it maximises the area under the "*Receiver Operation Characteristics*" (ROC) curve



varying y(x)>"cut" moves the working point (efficiency and purity) along the ROC curve

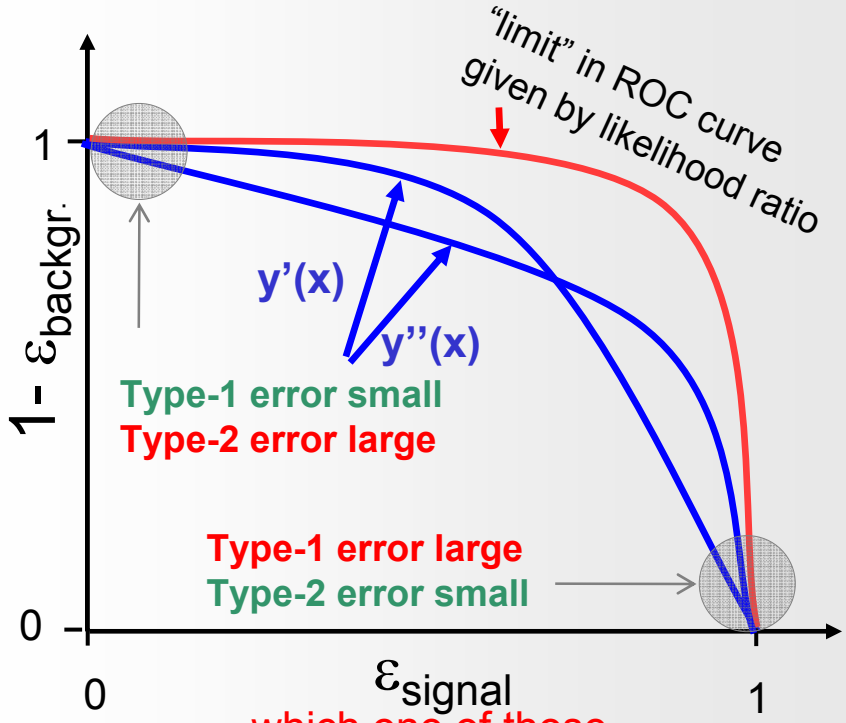
how to choose "cut" → need to know prior probabilities (S, B abundances)

- **measurement of signal cross section:** **maximum of S/√(S+B) or equiv. √(ε·p)**
- **discovery of a signal (typically: S<<B):** **maximum of S/√(B)**
- **precision measurement:** **high purity (p)**
- **trigger selection:** **high efficiency (ε)**

Efficiency or Purity ?

if discriminating function y(x) = “true likelihood ratio”
→ optimal working point for specific analysis lies somewhere on the ROC curve

y(x) ≠ “true likelihood ratio” differently, point
→ **y’(x)** might be better for a specific working point than **y’’(x)** and vice versa



- application of “cost matrix” → favour one over the other in the training process

e.g. cancer diagnostic:

consequences of a Type2 error much more severe
(death due to missing treatment)

compared to Type1 error
(unnecessary distress)

$$\begin{array}{cc} & \text{Decision} \\ & \begin{array}{cc} \text{cancer} & \text{normal} \end{array} \\ \text{Truth} \begin{array}{c} \text{cancer} \\ \text{normal} \end{array} & \begin{pmatrix} 0 & 1000 \\ 1 & 0 \end{pmatrix} \end{array}$$

MVA and Machine Learning

- The previous slide was basically the idea of “Multivariate Analysis” (MVA)
  - rem: What about “standard cuts” ?
- Finding y(x) : $R^n \rightarrow R$
  - given a certain type of model class y(x)
  - in an automatic way using “known” or “previously solved” events
    - i.e. learn from known “patterns”
  - such that the resulting y(x) has good generalization properties when applied to “unknown” events (regression: fits well the target function “in between” the known training events

  → that is what the “machine” is supposed to be doing: supervised machine learning
- Of course… there’s no magic, we still need to:
  - choose the discriminating variables
  - choose the class of models (linear, non-linear, flexible or less flexible)
  - tune the “learning parameters” → bias vs. variance trade off
  - check generalization properties
  - consider trade off between statistical and systematic uncertainties

Event Classification

- Unfortunately, the true probability densities functions are typically unknown:

→ Neyman-Pearsons lemma doesn't really help us directly

- Monte Carlo simulation or in general cases: set of known (already classified) "events"

- 2 different ways: Use these "training" events to:

  - estimate the functional form of p(x|C): (e.g. the differential cross section folded with the detector influences) from which the likelihood ratio can be obtained

    → e.g. D-dimensional histogram, Kernel densitiy estimators, …

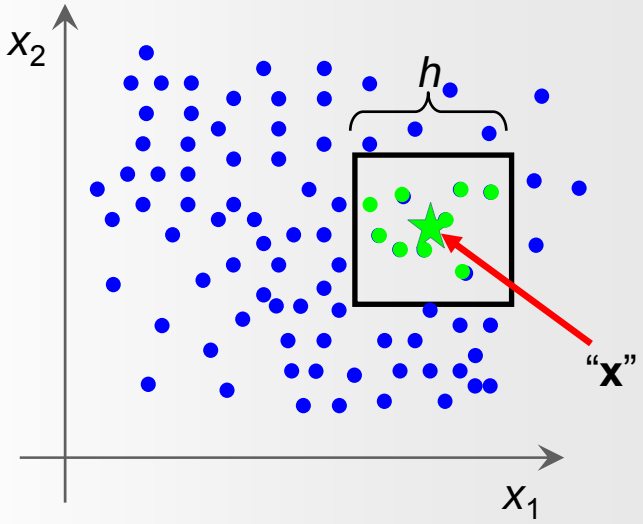
  - find a "discrimination function" y(x) and corresponding decision boundary (i.e. hyperplane* in the "feature space": y(x) = const) that optimially separates signal from background

    → e.g. Linear Discriminator, Neural Networks, …

* hyperplane in the strict sense goes through the origin. Here I mean "affine set" to be precise

Nearest Neighbour and Kernel Density Estimator

- estimate probability density P(x) in D-dimensional space: "events" distributed according to P(x)
- The only thing at our disposal is our "training data"
- Say we want to know P(x) at "this" point "x"
- One expects to find in a volume V around point "**x**"
  $N^* \int_V P(x)dx$ events from a dataset with N events
- For the chosen a rectangular volume
  → K-events:



$$K = \sum_{n=1}^{N} k\left(\frac{x - x_n}{h}\right), \quad \text{with} \quad k(u) = \begin{cases} 1, & |u_i| \le \frac{1}{2}, i = 1...D \\ 0, & \text{otherwise} \end{cases}$$

$k(u)$: is called a Kernel function:
rectangular→ Parzen-Window

- K (from the "training data") → estimate of average P(x) in the volume V: $\int_V P(x)dx = K/N$

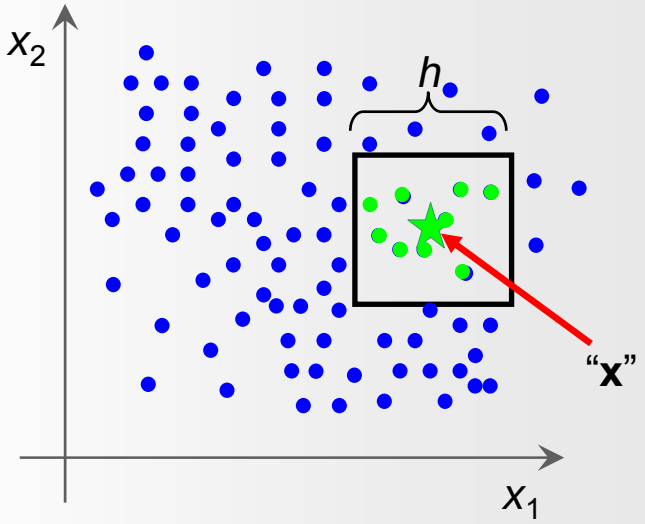
- Classification: Determine
  $PDF_S(x)$ and $PDF_B(x)$
  →likelihood ratio as classifier!

$$P(x) = \frac{1}{N} \sum_{n=1}^{N} \frac{1}{h^D} k\left(\frac{x - x_n}{h}\right)$$

→ Kernel Density estimator of the probability density

Nearest Neighbour and Kernel Density Estimator

- estimate probability density P(x) in D-dimensional space: "events" distributed according to P(x)
- The only thing at our disposal is our "training data"
- Say we want to know P(x) at "this" point "x"
- One expects to find in a volume V around point "**x**"
  $N^* \int_V P(x)dx$ events from a dataset with N events
- For the chosen rectangular volume
  → K-events:



$$K = \sum_{n=1}^{N} k\left(\frac{x - x_n}{h}\right), \quad \text{with} \quad k(u) = \begin{cases} 1, & |u_i| \le \frac{1}{2}, i = 1...D \\ 0, & \text{otherwise} \end{cases}$$

$k(u)$: is called a Kernel function:
rectangular→ Parzen-Window

- K (from the "training data") → estimate of average P(x) in the volume V: $\int_V P(x)dx = K/N$

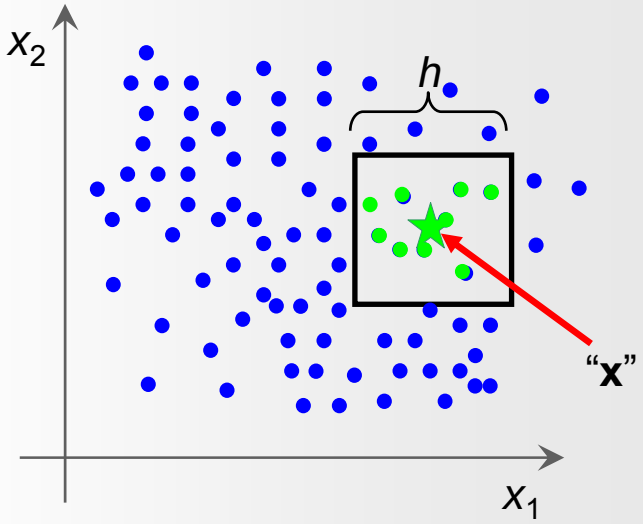
- Regression: If each events with (x_1,x_2) carries a "function value" $f(x_1,x_2)$ (e.g. energy of incident particle) →

$$\frac{1}{N}\sum_{i}^{N} f(x_1^i, x_2^i) = \int_V \hat{f}(x_1, x_2) P(\vec{x}) d\vec{x}$$

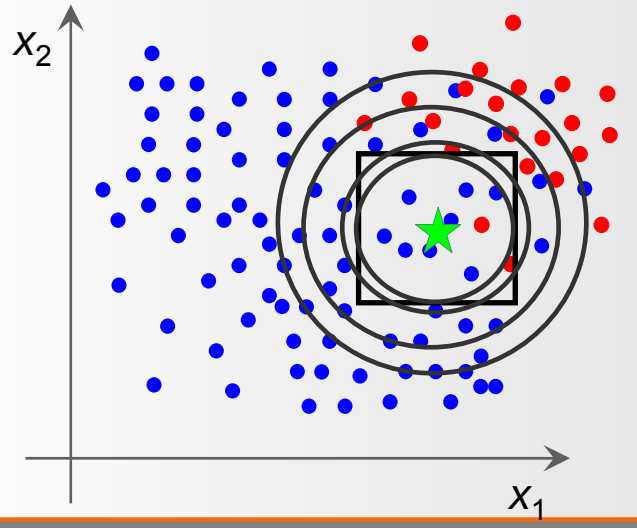
i.e.: the average function value

Nearest Neighbour and Kernel Density Estimator

- estimate probability density P(x) in D-dimensional space: "events" distributed according to P(x)
- The only thing at our disposal is our "training data"
- Say we want to know P(x) at "this" point "x"
- One expects to find in a volume V around point "**x**"
  $N^* \int_V P(x)dx$ events from a dataset with N events
- For the chosen rectangular volume
  → K-events:



$$K = \sum_{n=1}^{N} k\left(\frac{x - x_n}{h}\right), \quad \text{with} \quad k(u) = \begin{cases} 1, & |u_i| \le \frac{1}{2}, i = 1...D \\ 0, & \text{otherwise} \end{cases}$$



- determine K from the "training data" with signal and background mixed together

→kNN : k-Nearest Neighbours
relative number events of the various classes amongst the k-nearest neighbours

$$y(x) = \frac{n_S}{K}$$

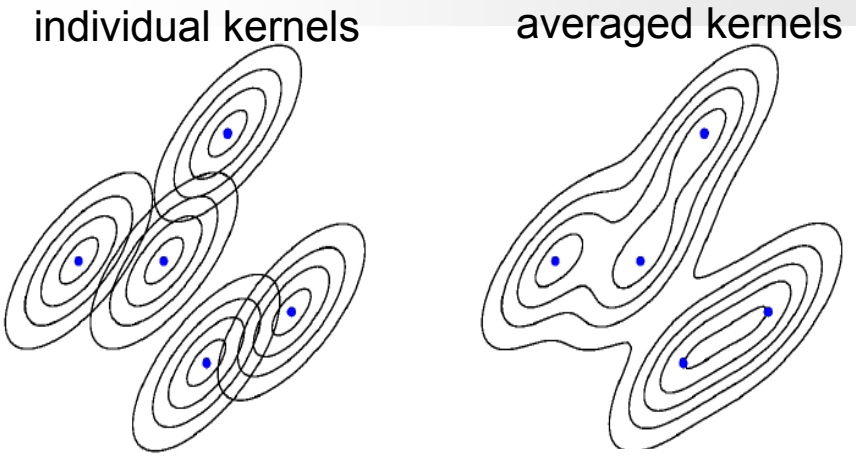
Kernel Density Estimator

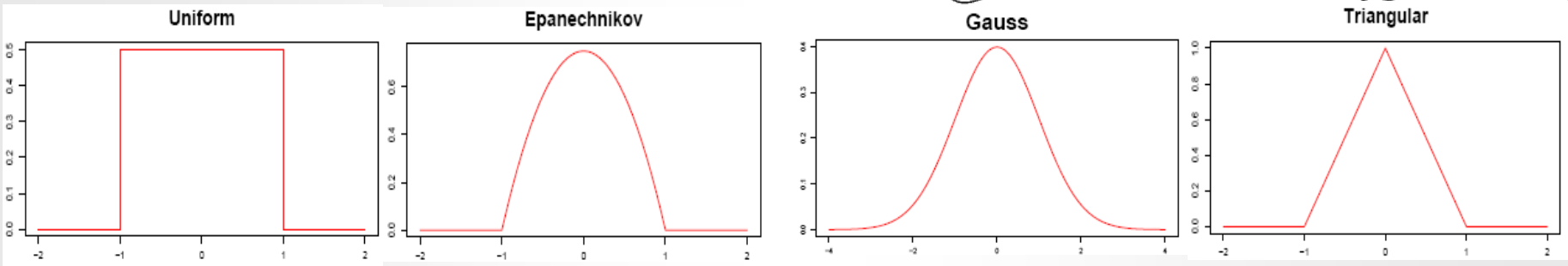
- Parzen Window: "rectangular Kernel" → discontinuities at window edges

→smoother model for P(x) when using smooth Kernel Functions: e.g. Gaussian

$$P(\mathrm{x}) = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{(2\pi h^2)^{1/2}}\exp\left(\frac{\|\mathrm{x}-\mathrm{x}_n\|^2}{2h^2}\right) \quad \leftrightarrow \text{probability density estimator}$$

- place a "Gaussian" around each "training data point" and sum up their contributions at arbitrary points "**x**" → P(**x**)
- h: "size" of the Kernel → "smoothing parameter"
- there is a large variety of possible Kernel functions



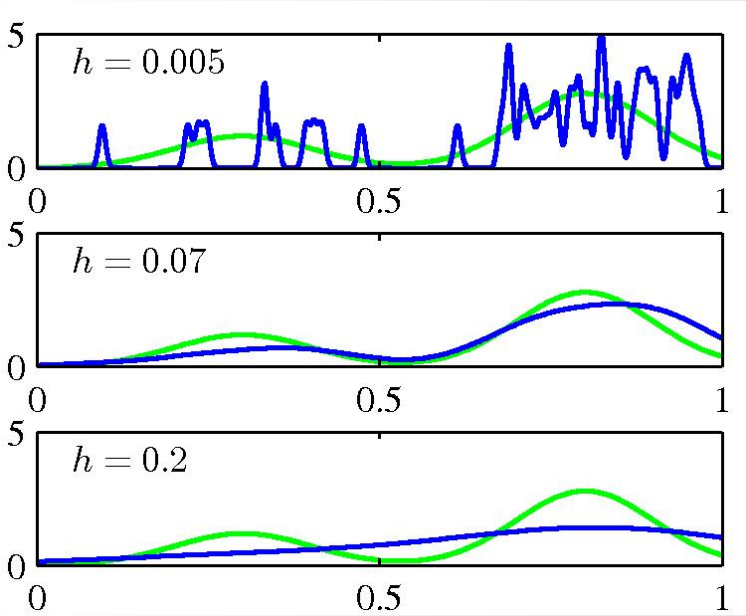


Kernel Density Estimator

$$P(\mathbf{x}) = \frac{1}{N}\sum_{n=1}^{N} K_h(\mathbf{x} - \mathbf{x}_n)$$: a general probability density estimator using kernel K

- h: "size" of the Kernel → "smoothing parameter"
- chosen size of the "smoothing-parameter" → more important than kernel function
- h too small: overtraining
- h too large: not sensitive to features in P(x)

for Gaussian kernels, the optimum in terms of MISE (mean integrated squared error) is given by: $h_{xi}=(4/(3N))^{1/5}\ \sigma_{xi}$ with σ_{xi}=RMS in variable x_i



(Christopher M.Bishop)

- a drawback of Kernel density estimators:

Evaluation for any test events involves ALL TRAINING DATA → typically very time consuming

- binary search trees (i.e. Kd-trees) are typically used in kNN methods to speed up searching

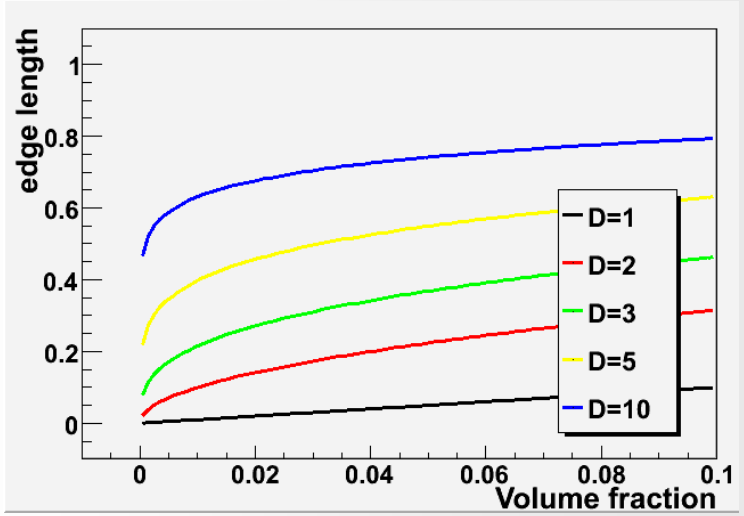
"Curse of Dimensionality"

Bellman, R. (1961), Adaptive Control Processes: A Guided Tour, Princeton University Press.

We all know:

Filling a D-dimensional histogram to get a mapping of the PDF is typically unfeasable due to lack of Monte Carlo events.

Shortcoming of nearest-neighbour strategies:



- in higher dimensional classification/regression cases the idea of looking at "training events" in a reasonably small "vicinity" of the space point to be classified becomes difficult:

consider: total phase space volume $V=1^D$

for a cube of a particular fraction of the volume:

$$\text{edge length} = (\text{fraction of volume})^{1/D}$$

- In 10 dimensions: in order to capture 1% of the phase space
  → 63% of range in each variable necessary → that's not "local" anymore..☹

→Therefore we still need to develop all the alternative classification/regression techniques

Naïve Bayesian Classifier
"often called: (projective) Likelihood"

while Kernel methods or Nearest Neighbour classifiers try to estimate the full D-dimensional joint probability distributions

If correlations between variables are weak: → $P(\mathbf{x}) \cong \prod_{i=0}^{D} P_i(\mathbf{x})$

product of marginal PDFs (1-dim "histograms")

PDFs

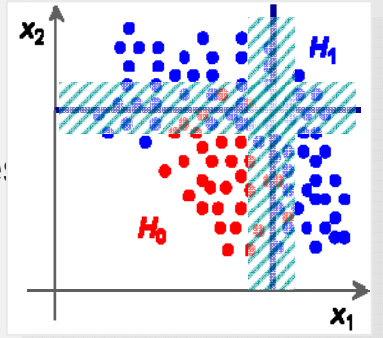
discriminating variables

Likelihood ratio for event *event*

$$y(x_{PDE,k_{event}}) = \frac{\prod_{i\in\{variables\}} P_i^{signal}(x_{i,k_{event}})}{\sum_{C\in\{classes\}} \left(\prod_{i\in\{variables\}} P_i^{C}(x_{i,k_{event}}) \right)}$$

Classes: signal, background types

- One of the first and still very popular MVA-algorithm in HEP
  - rather than making hard cuts on individual variables, allow for some "fuzzyness": one very signal like variable may counterweigh another le signal like variable
- This is about the optimal method in the absence of correlations



PDE introduces fuzzy logic

Naïve Bayesian Classifier
"often called: (projective) Likelihood"

How parameterise the 1-dim PDFs ??

event counting (histogramming)

Automatic, unbiased, but suboptimal

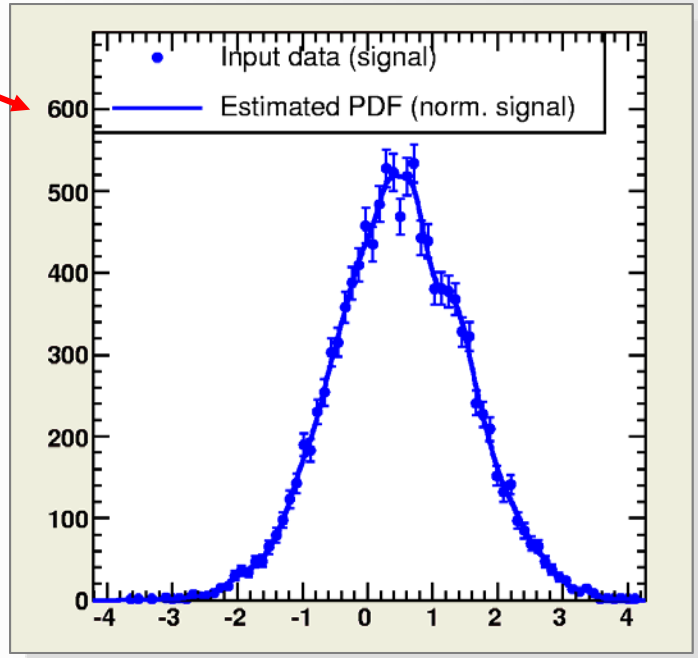
parametric (function) fitting

Difficult to automate for arbitrary PDFs

nonparametric fitting (i.e. splines,kernel)

Easy to automate, can create artefacts/suppress information

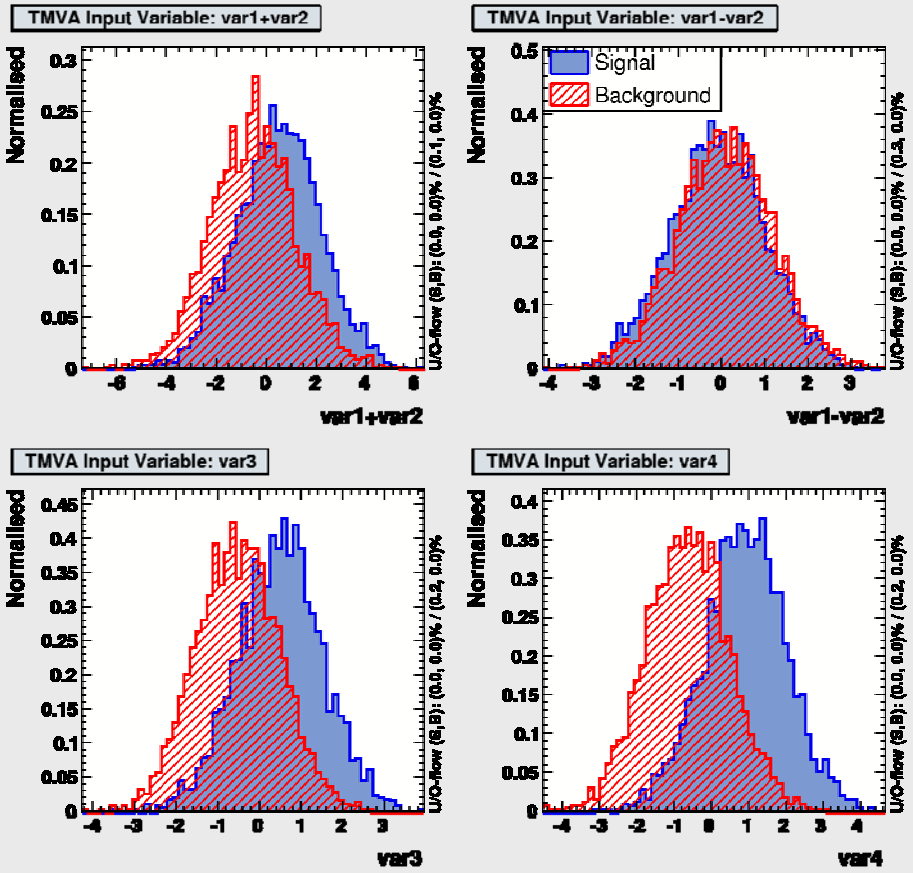
example: original (underlying) distribution is Gaussian

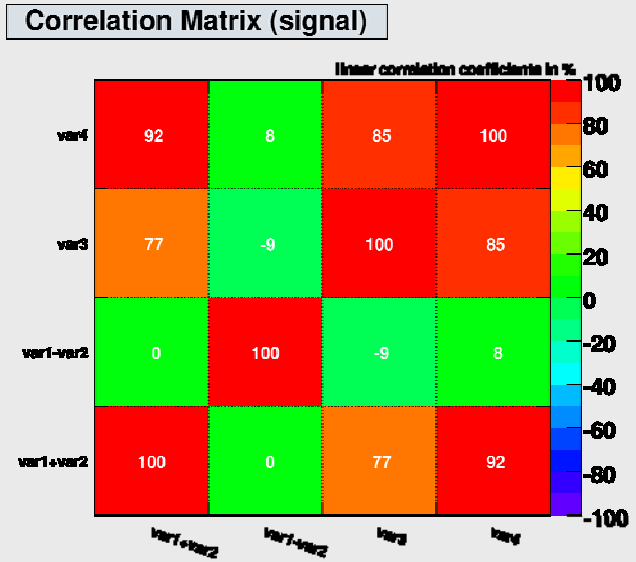


- If the correlations between variables is really negligible, this classifier is "perfect" (simple, robust, performing)
- If not, you seriously loose performance ☹
- How can we "fix" this ?

What if there are correlations?

- Typically correlations are present: $C_{ij}=cov[\,x_i\,,\,x_j\,]=E[\,x_i\,x_j\,]-E[\,x_i\,]E[\,x_j\,]\neq 0 \quad (i\neq j)$

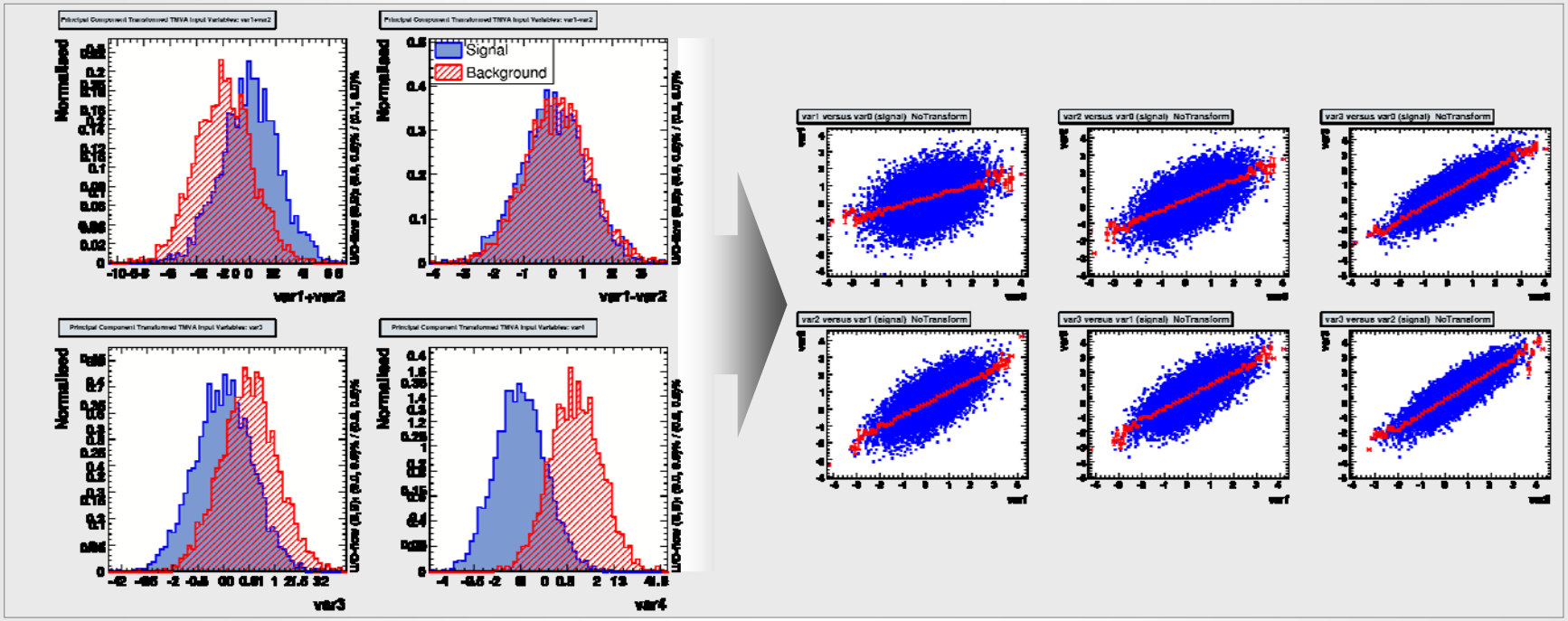




→ pre-processing: choose set of linear transformed input variables for which $C_{ij} = 0$ $(i\neq j)$

Decorrelation

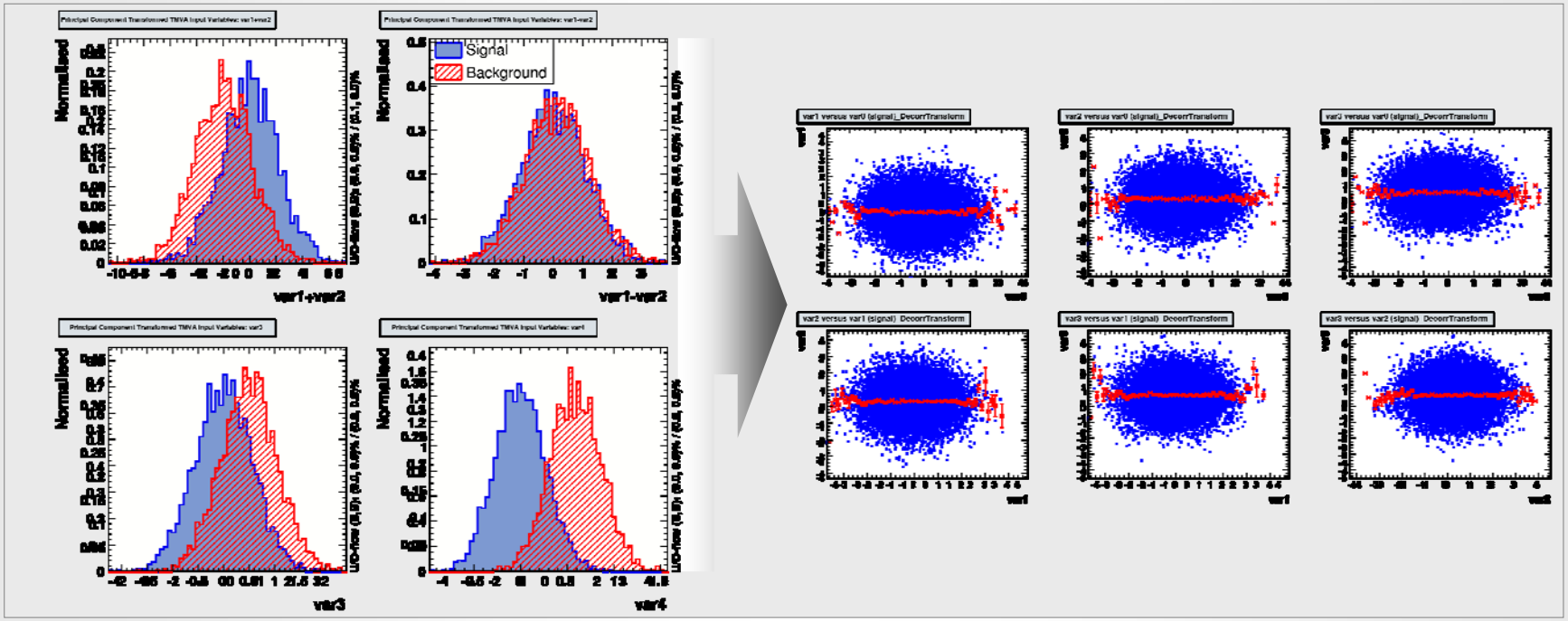
- Determine *square-root C′* of correlation matrix *C*, *i.e.*, $C = C'C'$
  - compute *C′* by diagonalising *C*: $D = S^T C S \;\Rightarrow\; C' = S\sqrt{D}S^T$
  - transformation from original (x) in de-correlated variable space (x′) by: $x' = C'^{-1}x$



Attention. This is able to eliminate only linear correlations!!

Decorrelation

- Determine *square-root C′* of correlation matrix *C*, *i.e.*, $C = C'C'$
  - compute C' by diagonalising C: $D = S^T C S \Rightarrow C' = S\sqrt{D}S^T$
  - transformation from original (x) in de-correlated variable space (x′) by: $x' = C'^{-1}x$



Attention. This is able to eliminate only linear correlations!!

Decorrelation: Principal Component Analysis

- PCA is typically used to:
  - reduce the dimensionality of a problem
  - find the most dominant features in your distribution by transforming
- The eigenvectors of the covariance matrix with the largest eigenvalues correspond to the dimensions that have the strongest correlation in the data set. Along these axis the variance is largest
  - sort the eigenvectors according to their eigenvalues
- Dataset is transformed in variable space along these eigenvectors

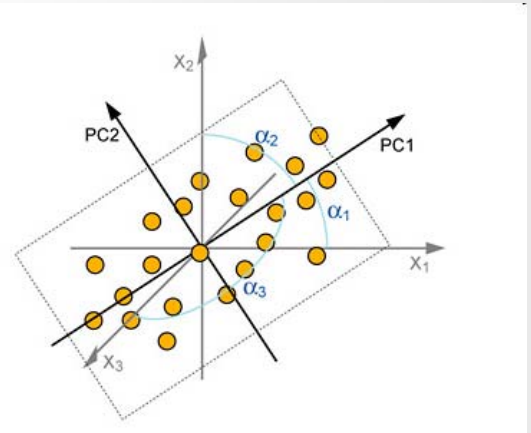
  → Along the "first" dimension the data show the largest "features", the smallest features are found in the "last" dimension.

$$x_k^{PC}(i_{event}) = \sum_{v \in \{variables\}} \left[x_v(i_{event}) - \bar{x}_v \right] \cdot v_v^{(k)} \quad , \forall k \in \{variables\}$$

Principle Component (PC) of variable *k*

sample means

eigenvector



- Matrix of eigenvectors V obey the relation: $C \cdot V = D \cdot V$ → PCA eliminates correlations!

correlation matrix

diagonalised square root of C

How to Apply the Pre-Processing Transformation?

In general: the decorrelation for signal and background variables is different
however: for a "test event" we don't know beforehand if it is signal or background.
? What do we do?

→ for likelihood ratio, decorrelate signal and background independently

$$y_L(i_{\text{event}}) = \frac{\prod_{k\in\{\text{variables}\}} p_k^S\left(x_k(i_{\text{event}})\right)}{\prod_{k\in\{\text{variables}\}} p_k^S\left(x_k(i_{\text{event}})\right) + \prod_{k\in\{\text{variables}\}} p_k^B\left(x_k(i_{\text{event}})\right)}$$

signal transformation

background transformation

$$y_L^{\text{trans}}(i_{\text{event}}) = \frac{\prod_{k\in\{\text{variables}\}} p_k^S\left(\hat{T}^S x_k(i_{\text{event}})\right)}{\prod_{k\in\{\text{variables}\}} p_k^S\left(\hat{T}^S x_k(i_{\text{event}})\right) + \prod_{k\in\{\text{variables}\}} p_k^B\left(\hat{T}^B x_k(i_{\text{event}})\right)}$$

→ for other estimators, one needs to decide on one of the two… (or decorrelate on a mixture of signal and background events)

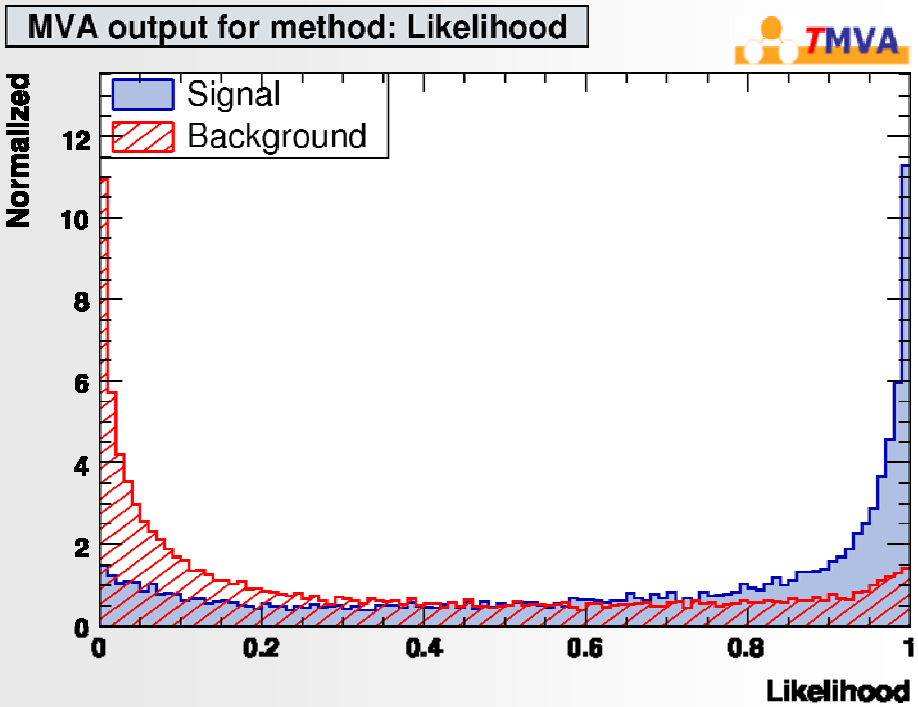
Decorrelation at Work

- Example: linear correlated Gaussians → decorrelation works to 100%

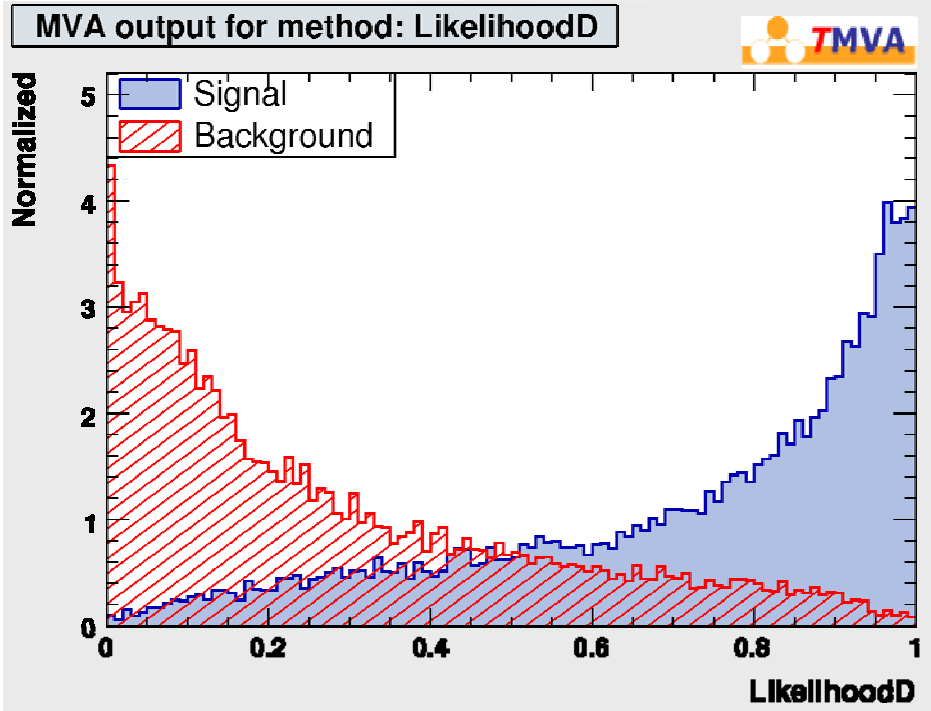
→1-D Likelihood on decorrelated sample give best possible performance

→compare also the effect on the MVA-output variable!

correlated variables:



after decorrelation



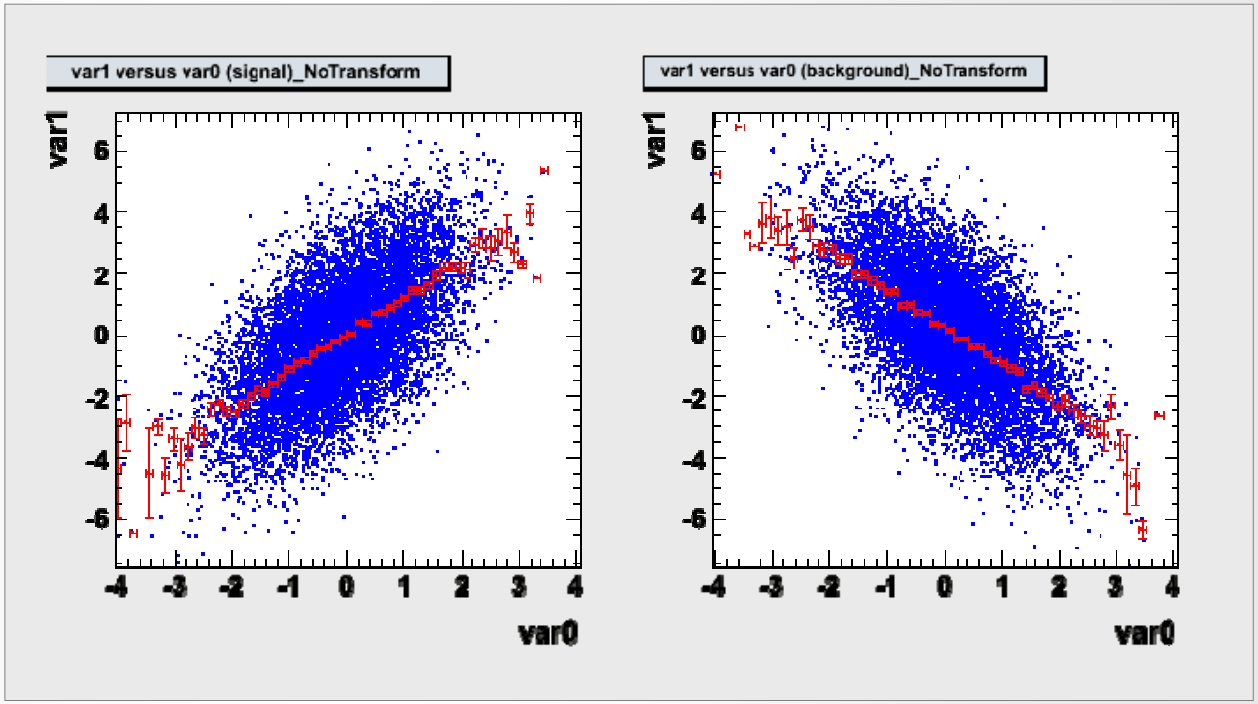
(note the different scale on the y-axis… sorry)

Limitations of the Decorrelation

- in cases with non-Gaussian distributions and/or nonlinear correlations, the decorrelation needs to be treated with care

- How does linear decorrelation affect cases where correlations between signal and background differ?

Original correlations



Signal

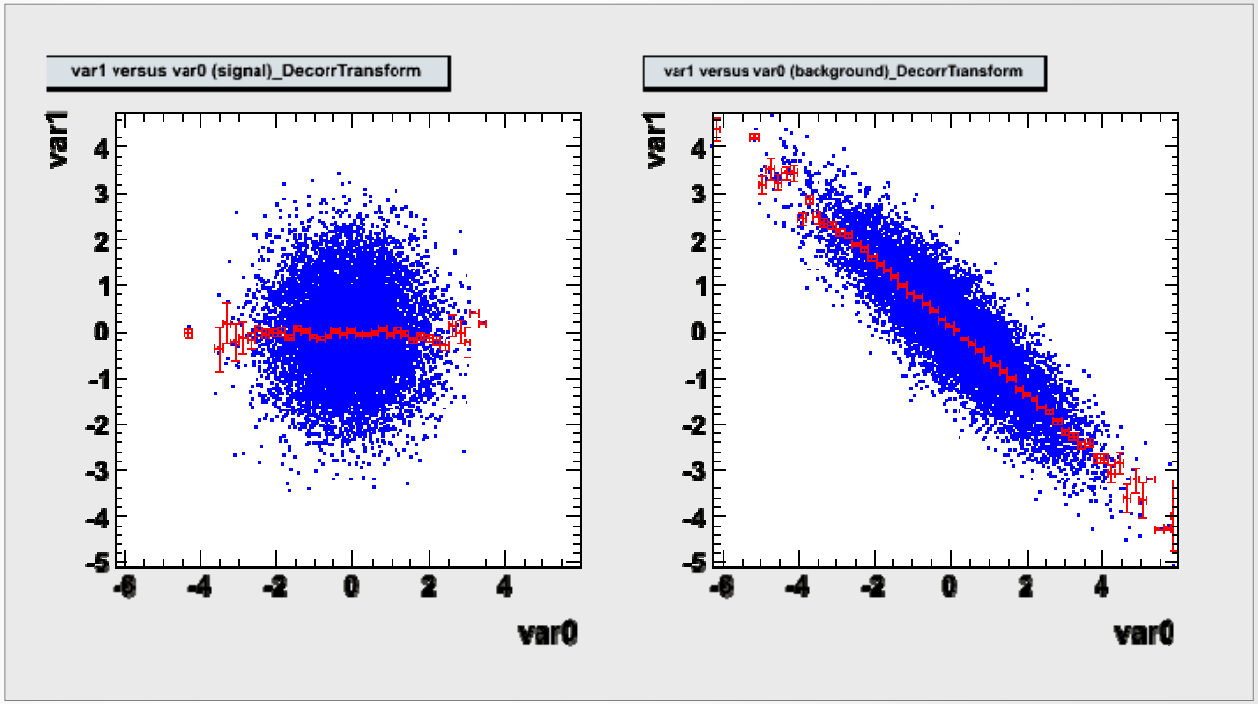
Background

Limitations of the Decorrelation

- in cases with non-Gaussian distributions and/or nonlinear correlations, the decorrelation needs to be treated with care

- How does linear decorrelation affect cases where correlations between signal and background differ?

SQRT decorrelation



Signal

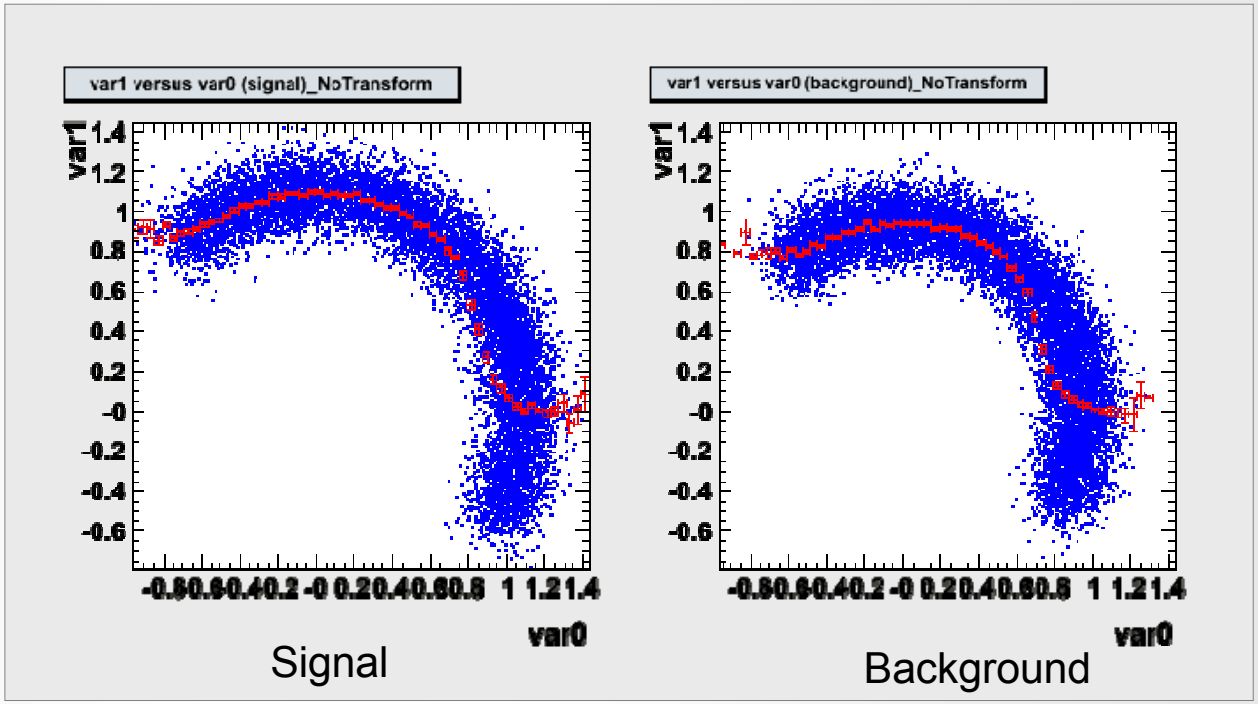
Background

Limitations of the Decorrelation

- in cases with non-Gaussian distributions and/or nonlinear correlations, the decorrelation needs to be treated with care

- How does linear decorrelation affect strongly nonlinear cases ?

Original correlations

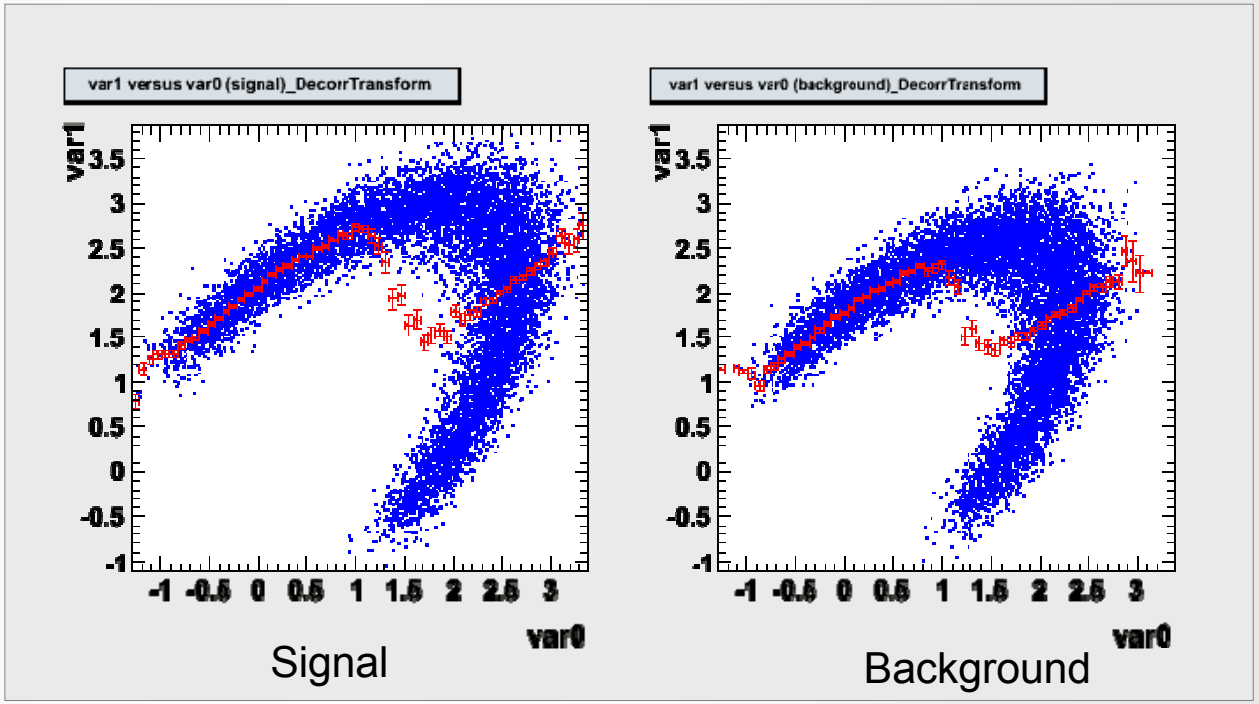


Limitations of the Decorrelation

- in cases with non-Gaussian distributions and/or nonlinear correlations, the decorrelation needs to be treated with care

- How does linear decorrelation affect strongly nonlinear cases ?

SQRT decorrelation



→Watch out before you used decorrelation “blindly”!!

→Perhaps “decorrelate” only a subspace!

“Gaussian-isation”

- Improve decorrelation by pre-Gaussianisation of variables
  - First: transformation to achieve uniform (flat) distribution:

$$x_k^{\text{flat}}(i_{\text{event}}) = \int_{-\infty}^{x_k(i_{\text{event}})} p_k(x'_k)\,dx'_k \quad , \forall k \in \{\text{variables}\}$$

Rarity transform of variable k | Measured value | PDF of variable k

The integral can be solved in an unbinned way by event counting, or by creating non-parametric PDFs (see later for likelihood section)

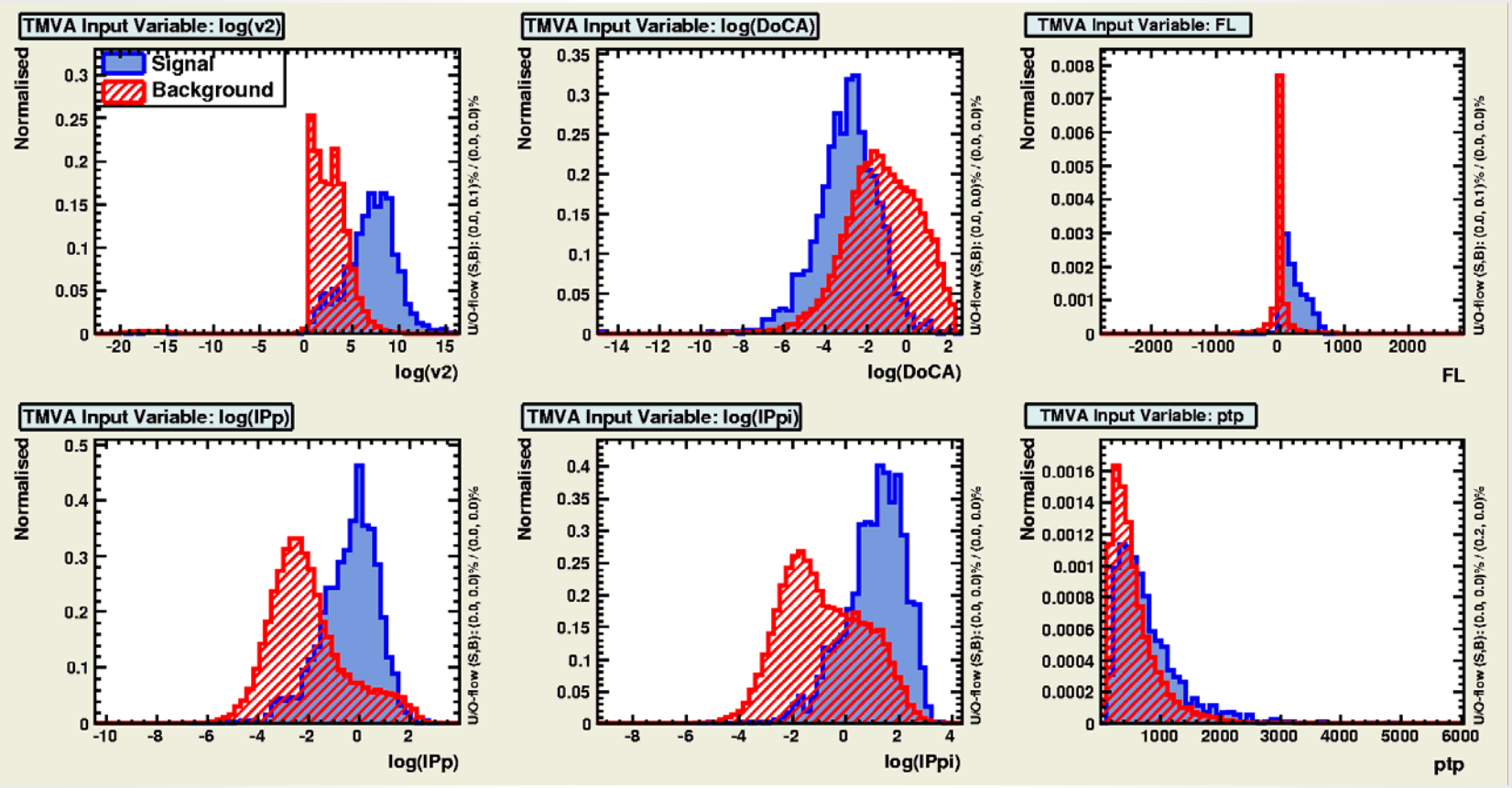
  - Second: make Gaussian via inverse error function: $\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$

$$x_k^{\text{Gauss}}(i_{\text{event}}) = \sqrt{2} \cdot \text{erf}^{-1}\left(2x_k^{\text{flat}}(i_{\text{event}}) - 1\right) \quad , \forall k \in \{\text{variables}\}$$

  - Third: decorrelate (and “iterate” this procedure)

“Gaussian-isation”

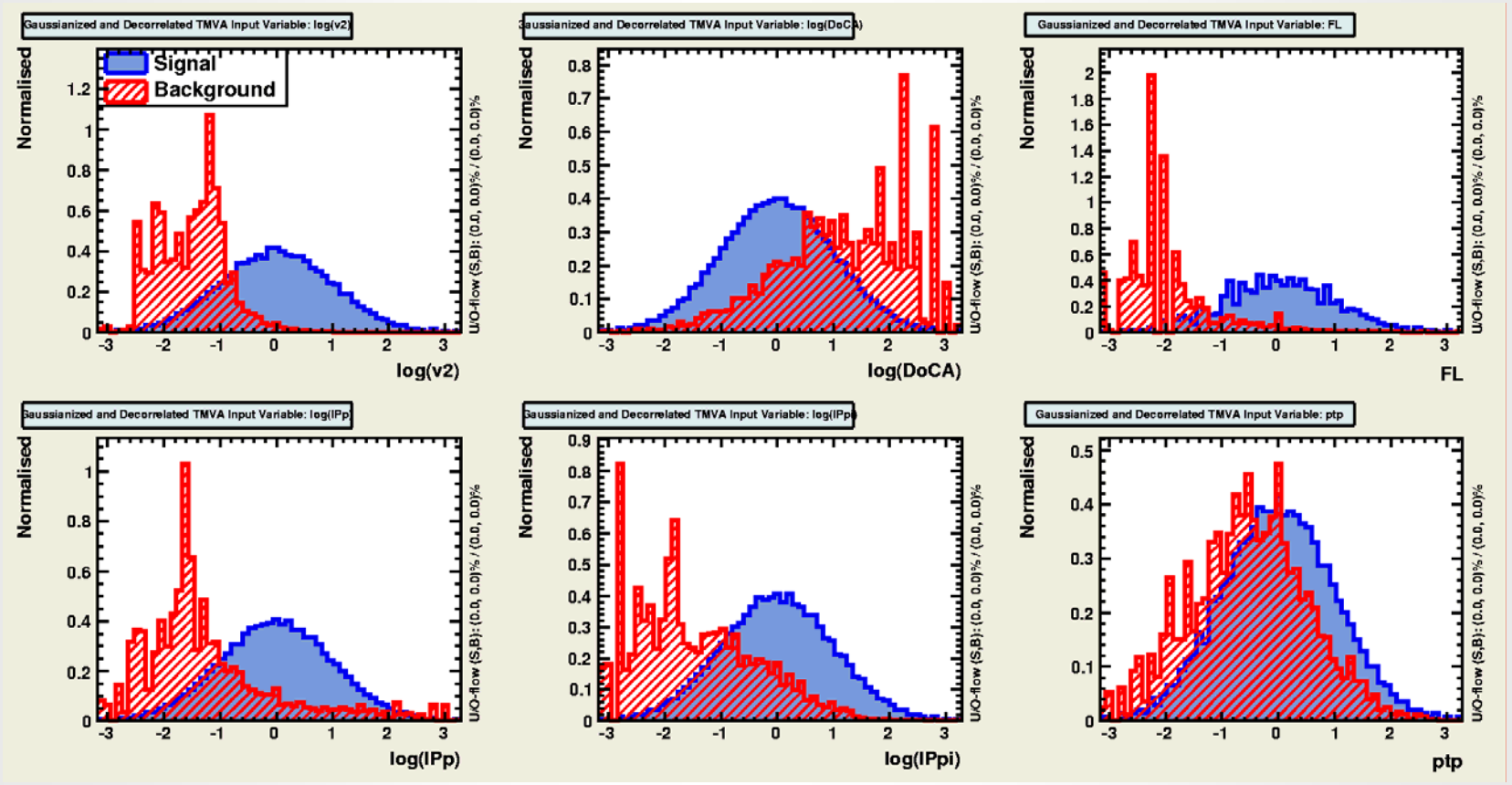
Original



We cannot simultaneously “Gaussianise” both signal and background !

“Gaussian-isation”

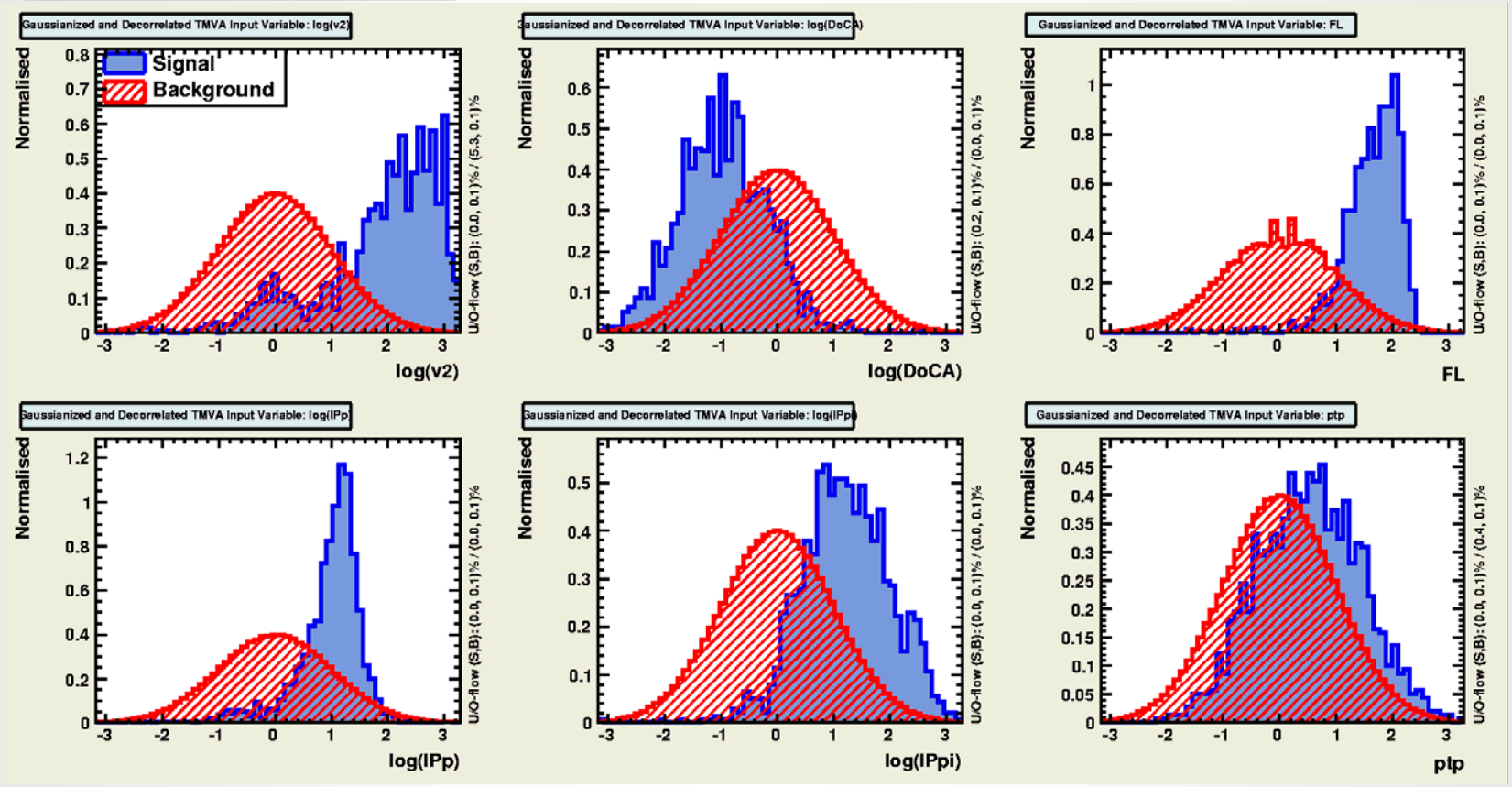
Signal - Gaussianised



We cannot simultaneously “Gaussianise” both signal and background !

“Gaussian-isation”

Background - Gaussianised



We cannot simultaneously “Gaussianise” both signal and background !

Linear Discriminant

If non parametric methods like 'k-Nearest-Neighbour" or "Kernel Density Estimators" suffer from

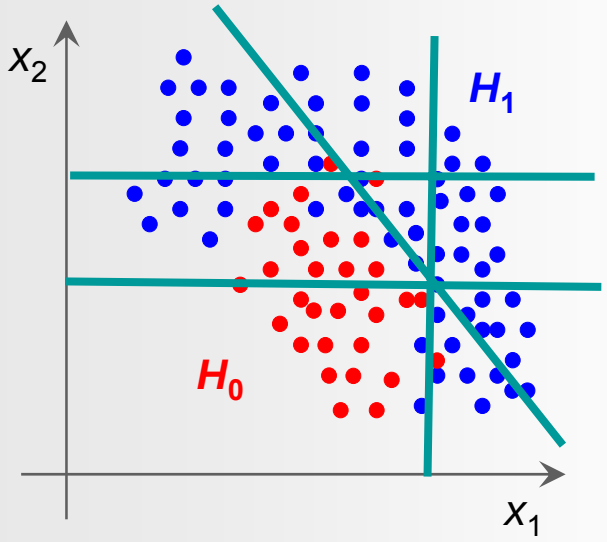
- lack of training data → "curse of dimensionality"
- slow response time → need to evaluate the whole training data for every test event

→ use of parametric models y(x) to fit to the training data $y(x = \{x_1,\ldots,x_D\}) = \sum_{i=0}^{M} w_i h_i(x)$

Linear Discriminant:

i.e. any linear function of the input variables: giving rise to linear decision boundaries

$$y(x = \{x_1,\ldots,x_D\}) = w_0 + \sum_{i=1}^{D} w_i x_i$$

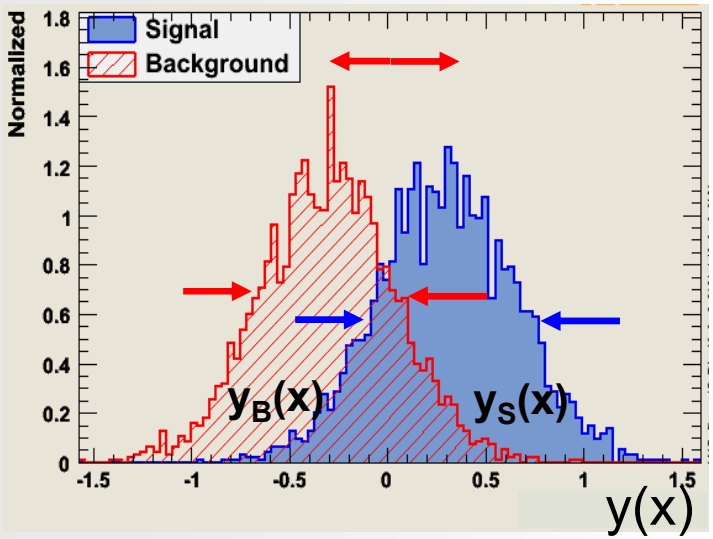


How do we determine the "weights" w that do "best"??

Watch out: these lines are NOT the linear function y(x), but rather the decision boundary given by y(x)=const.

Fisher's Linear Discriminant

$$y(x = \{x_1,\ldots,x_D\}) = y(x,w) = w_0 + \sum_{i=1}^{D} w_i x_i$$



How do we determine the "weights" w that do "best"??

- Maximise the "separation" between the two classes S and B

→ minimise overlap of the distribution $y_S(x)$ and $y_B(x)$

- maximise the distance between the two mean values of the classes
- minimise the variance within each class

→ maximise $$J(\vec{w}) = \frac{(E(y_B) - E(y_S))^2}{\sigma^2_{y_B} + \sigma^2_{y_S}} = \frac{\vec{w}^T B \vec{w}}{\vec{w}^T W \vec{w}} = \frac{\text{"in between" variance}}{\text{"within" variance}}$$

$$\vec{\nabla}_w J(\vec{w}) = 0 \Rightarrow \vec{w} \propto W^{-1}(\langle \vec{x} \rangle_S - \langle \vec{x} \rangle_B)$$ the Fisher coefficients

note: these quantities can be calculated from the training data

Linear Discriminant and non linear correlations

assume the following non-linear correlated data:

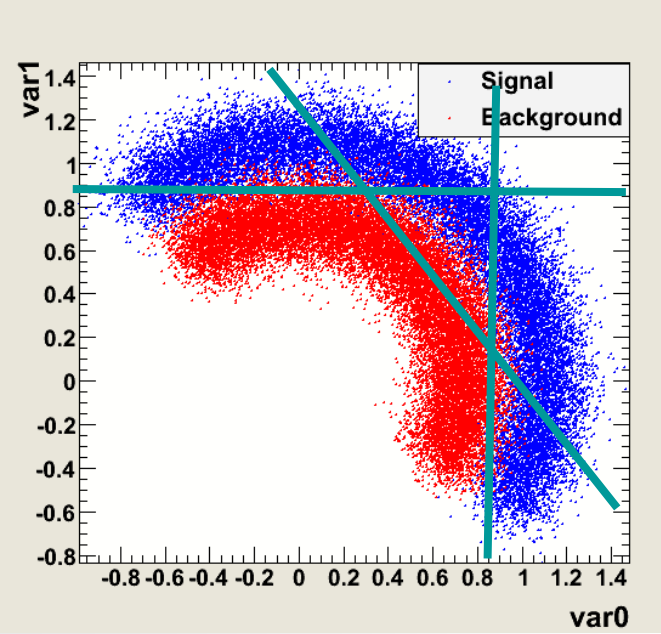
- the Linear discriminant obviously doesn't do a very good job here:

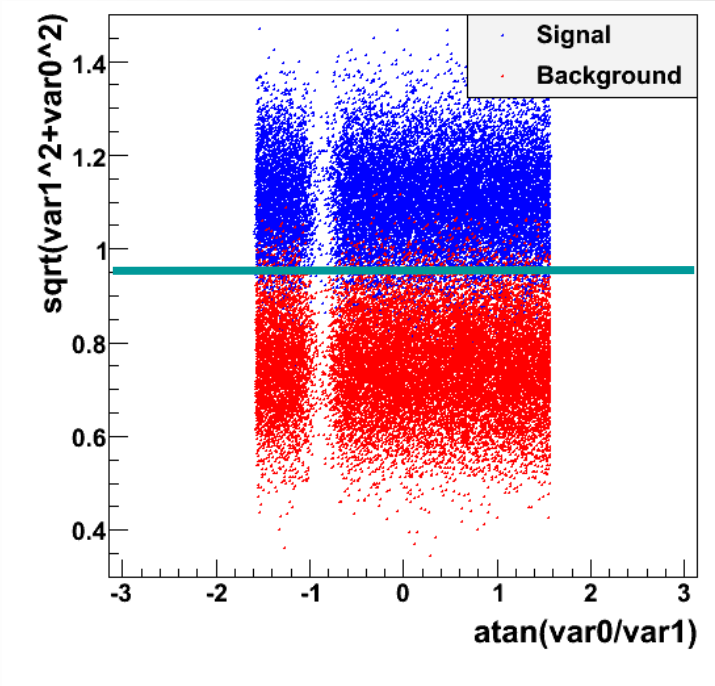
- Of course, these can easily be de-correlated:

  → here: linear discriminator works perfectly on de-correlated data

$$\mathrm{var}0' = \sqrt{\mathrm{var}0^2 + \mathrm{var}1^2}$$

$$\mathrm{var}1' = \mathrm{a}\tan\left(\frac{\mathrm{var}0}{\mathrm{var}1}\right)$$





Linear Discriminant with Quadratic input:

- A simple extension of the linear decision boundary to “quadratic” decision boundary:

while:
- var0
- var1

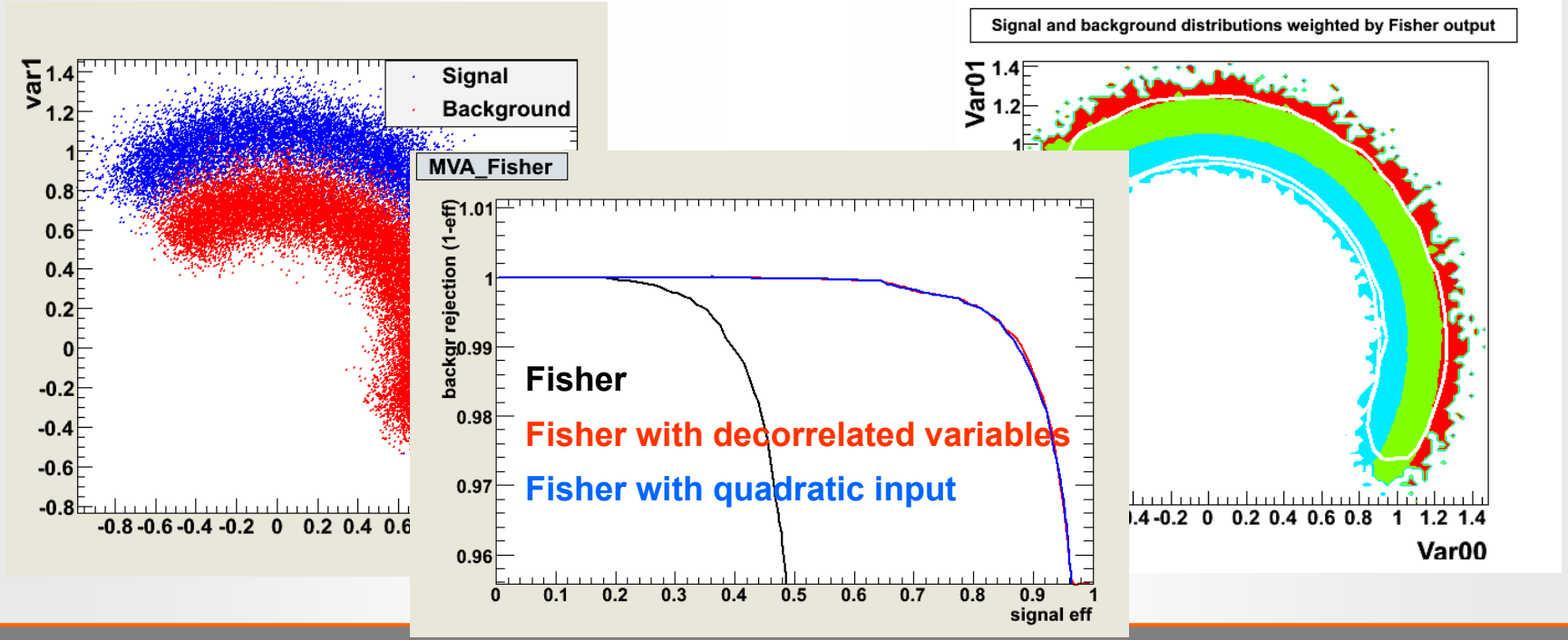
→ linear decision boundaries in var0,var1

- var0 * var0
- var1 * var1
- var0 * var1

→ quadratic decision boundaries in var0,var1

Performance of Fisher Discriminant:

with quadratic input:



Linear Discriminant with Quadratic input:

- A simple extension of the linear decision boundary to “quadratic” decision boundary:

while:
- var0
- var1

→ linear decision boundaries in var0,var1

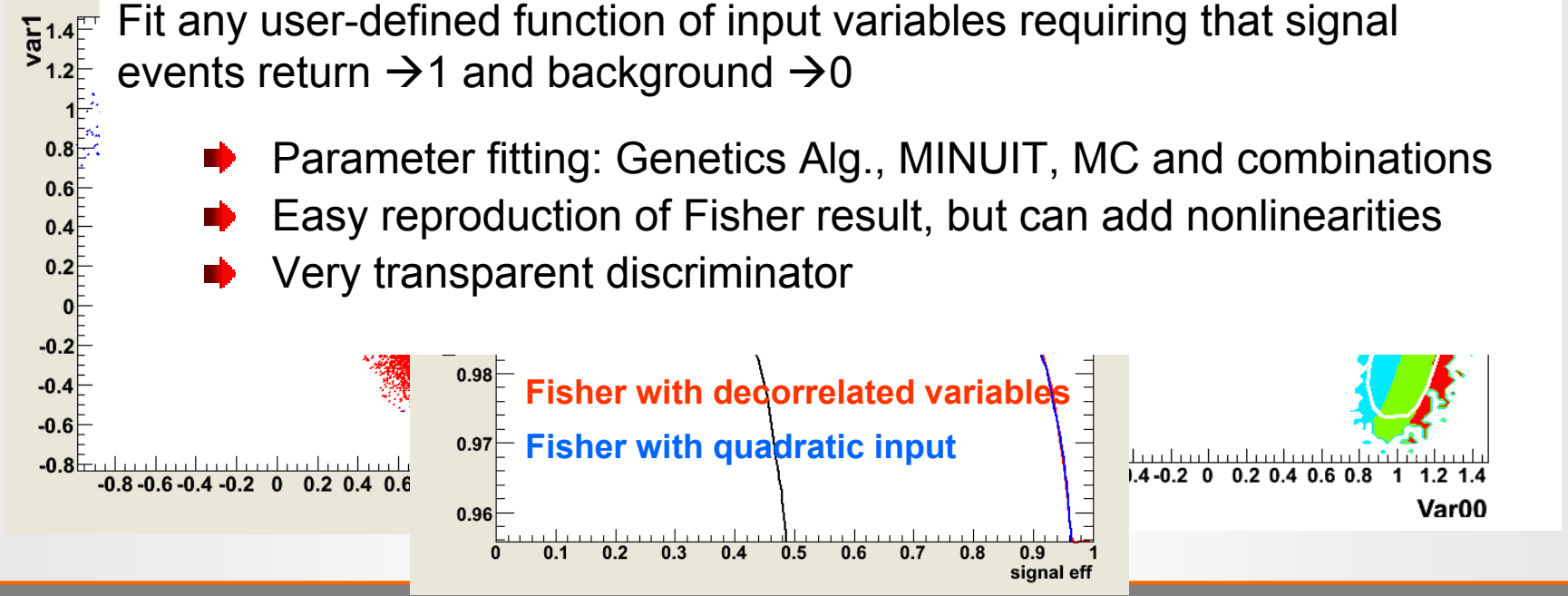
- Of course, if one “finds out”/”knows” correlations they are best treated explicitly!
- explicit decorrelation
- or *e.g*:

ant:

Function discriminant analysis (FDA)

Fit any user-defined function of input variables requiring that signal events return →1 and background →0

- Parameter fitting: Genetics Alg., MINUIT, MC and combinations
- Easy reproduction of Fisher result, but can add nonlinearities
- Very transparent discriminator



Training Classifiers and Loss-Function

- All classifiers seen so far (KNN,Likelihood,Fisher) could be calculated and didn't require parameter fitting → the class that "estimate" the underlying PDF rather than a decision boundary

- Other classifiers provide a set of "basis" functions (or model) that need to be optimally adjusted to find the appropriate separating hyperplane(surface)

- Let $x \in R^D$ be a random variable (i.e. our observables) and y(x)
  - y a real valued output variable with joint distribution Pr(x,y) → regression
  - y value → 1 for signal, y→0 for background → classification

- we are looking for a function y(x) that predicts y given certain input variables: $y(x):R^D \rightarrow R$:

- Loss function: L(y,y(x)) penalizes errors made in the prediction:
  - $EPE(y(x)) = E(y - y(x))^2$ squared error loss, typical "loss function" used in regression
    - by conditioning on "x" we can write:
  - EPE(y(x)) = E(|y-y(x)|) misclassification error, typical "loss function" for classification problems

Neural Networks

naturally, if we want to go to “arbitrary” non-linear decision boundaries, y(x) needs to be constructed in “any” non-linear fashion

$$y(\vec{x}) = \sum_{i}^{M} \left(w_i \, h_i(\vec{x}) \right)$$

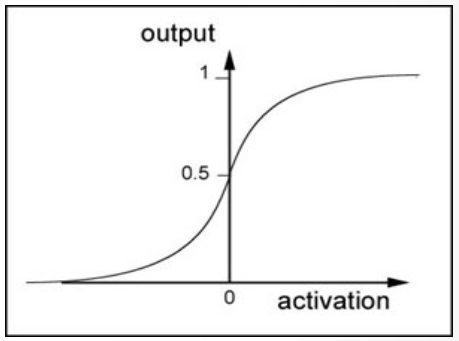
- Think of $h_i(x)$ as a set of “basis” functions
- If h(x) is sufficiently general (i.e. non linear), a linear combination of “enough” basis function should allow to describe any possible discriminating function y(x)

K.Weierstrass Theorem: proves just that previous statement.

$h_i(\mathbf{x})$

Imagine you chose do the following:

$$y(x) = \sum_{i}^{M} w_{0i} \, \mathbb{A}\left(w_{i0} + \sum_{j=1}^{D} w_{ij} \cdot x_j \right)$$



$$A(x) = \frac{1}{1+e^{-x}}:$$

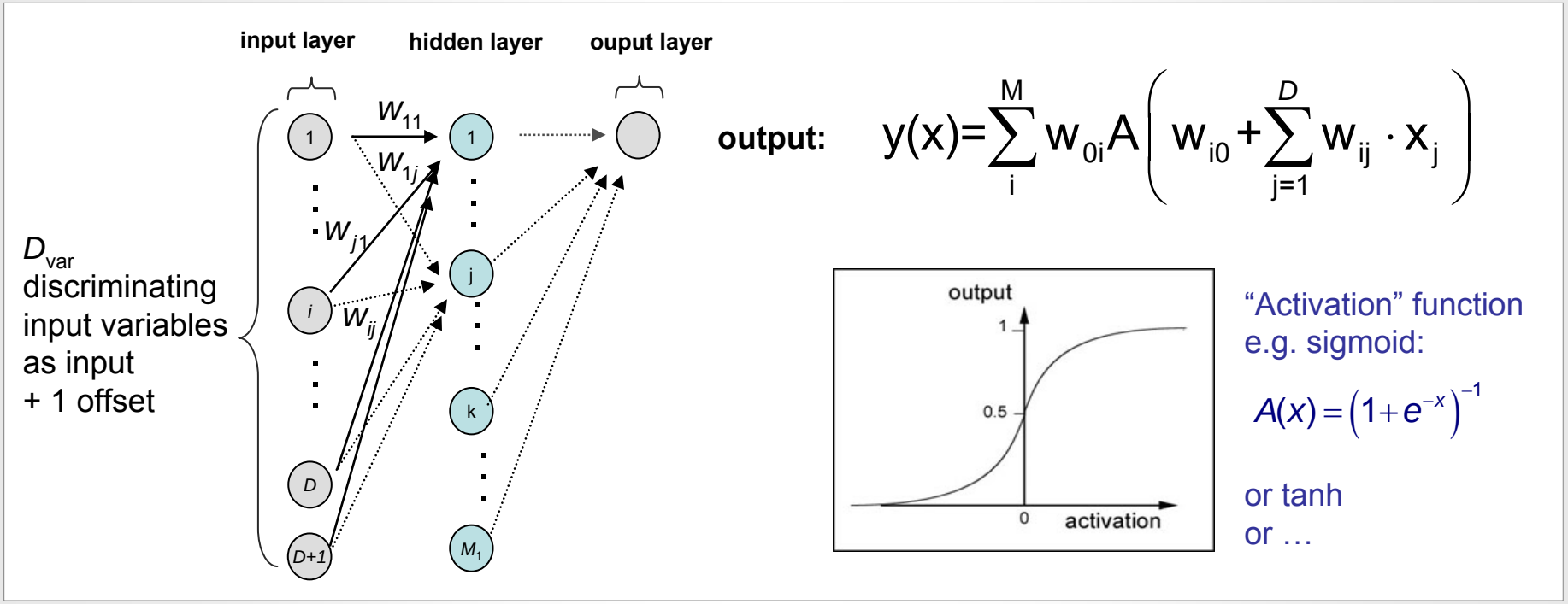
the sigmoid function

y(x) =
a linear combination of
non linear functions of
linear combination of
the input data

Ready is the Neural Network
Now we “only” need to find the appropriate “weights” w

Neural Networks: Multilayer Perceptron MLP

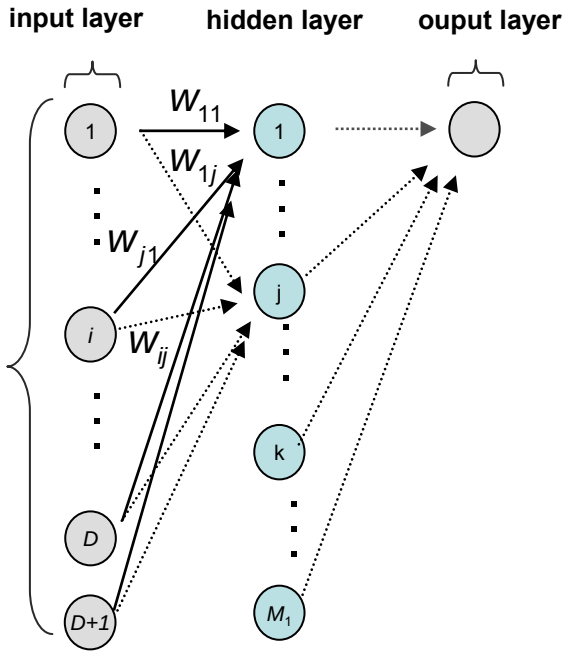
But before talking about the weights, let's try to "interpret" the formula as a Neural Network:



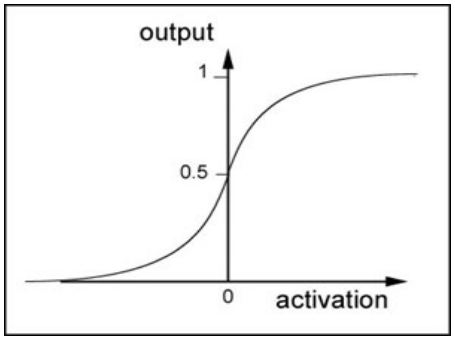
- Nodes in hidden layer represent the "activation functions" whose arguments are linear combinations of input variables → non-linear response to the input
- The output is a linear combination of the output of the activation functions at the internal nodes
- Input to the layers from preceding nodes only → feed forward network (no backward loops)
- It is straightforward to extend this to "several" input layers

Neural Networks: Multilayer Perceptron MLP

But before talking about the weights, let's try to "interpret" the formula as a Neural Network:



output: $$y(x)=\sum_{i}^{M} w_{0i} A\left(w_{i0} + \sum_{j=1}^{D} w_{ij} \cdot x_j \right)$$



"Activation" function e.g. sigmoid:

$$A(x) = \left(1 + e^{-x}\right)^{-1}$$

or tanh
or …

nodes→neurons
links(weights)→synapses

→ Neural network: try to simulate reactions of a brain to certain stimulus (input data)

Neural Network Training

idea: using the "training events" adjust the weights such, that

- y(x)→0 for background events
- y(x)→1 for signal events

how do we adjust ?

- minimize Loss function:

$$L(w) = \sum_{i}^{events} \underbrace{(y(x_i)}_{\text{predicted event type}} - \underbrace{y(C))^2}_{\text{true event type}}$$

i.e. use usual "sum of squares" or misclassification error

where $y(C) = \begin{cases} 1 \text{ for } C = \text{signal} \\ 0 \text{ for } C = \text{backgr.} \end{cases}$

- y(x) is a very "wiggly" function with many local minima. A global overall fit in the many parameters is possible but not the most efficient method to train neural networks

→back propagation (learn from experience, gradually adjust your perception to match reality

→online learning (learn event by event -- not only at the end of your life from all experience)

Neural Network Training
back-propagation and online learning

- start with random weights
- adjust weights in each step a bit in the direction of the steepest descend of the "Loss"-function L

$$w^{n+1} = w^n + \eta \cdot \vec{\nabla}_w L(w) \quad \eta = \text{learning rate} \qquad L(w) = (y(x_i) - y(C))^2$$

- for weights connected to output nodes

$$y(x) = \sum_{i}^{M} w_{0i} A\left(w_{i0} + \sum_{j=1}^{D} w_{ij} \cdot x_j \right)$$

$$\frac{\partial L}{\partial w_{0i}} = (y(x) - y(C)) A\left(w_{i0} + \sum_{j=1}^{D} w_{ij} \cdot x_j \right)$$

- for weights connected to output nodes

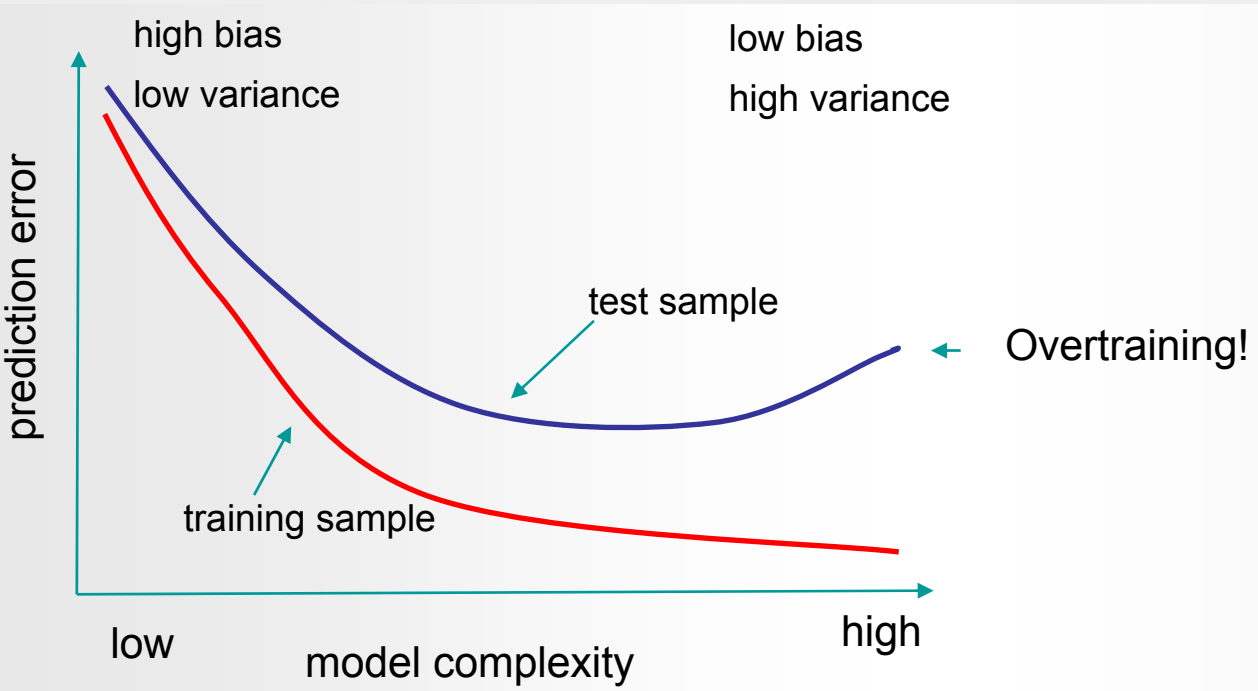
  … a bit more complicated formula

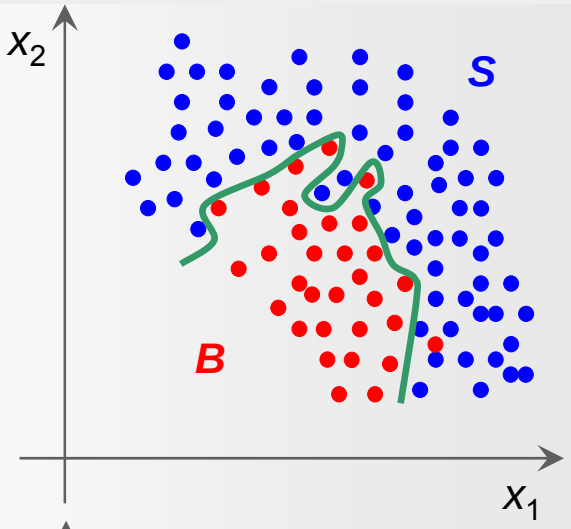
- note: all these gradients are easily calculated from the training event

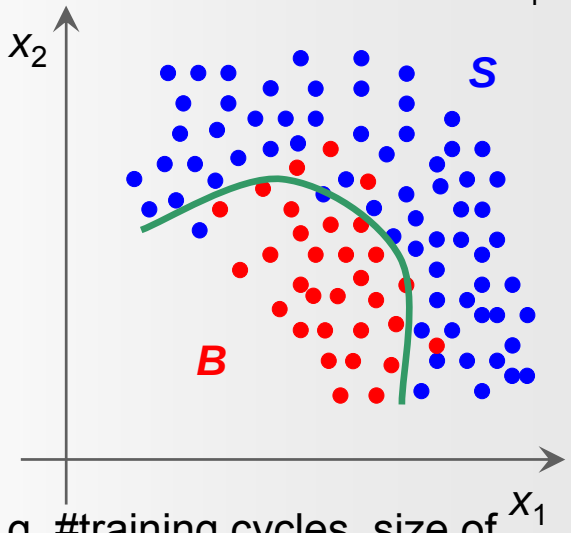
- training is repeated n-times over the whole training data sample. how often ??

for online learning, the training events should be mixed randomly, otherwise you first steer in a wrong direction from which it is afterwards hard to get out again!

Bias ↔ Variance Trade off







- Save Monte Carlo training events using "Cross Validation"

to optimize tuning parameters (α) in the algorithm:

→ divide the data sample into say 5 sub-sets

Train	Train	Train	Train	Test

- train 5 times the same classifier (for a given tune parameter α=e.g. #training cycles, size of kernel) → for each event in the training you have a statistically independent classifier
- repeat with varied α to find optimal value of α → train classifier on full sample using this α

Too bad ☹, cross validation it is still NOT implemented in TMVA

Support Vector Machines

- If Neural Networks are complicated by finding the proper optimum “weights” for best separation power by “wiggly” functional behaviour of the piecewise defined separation hyperplane
- If KNN (multidimensional likelihood) suffers disadvantage that calculating the MVA-output of each test event involves evaluation of ALL training events
- If Boosted Decision Trees in theory are always weaker than a perfect Neural Network
- → Try to get the best of all worlds…

Support Vector Machine

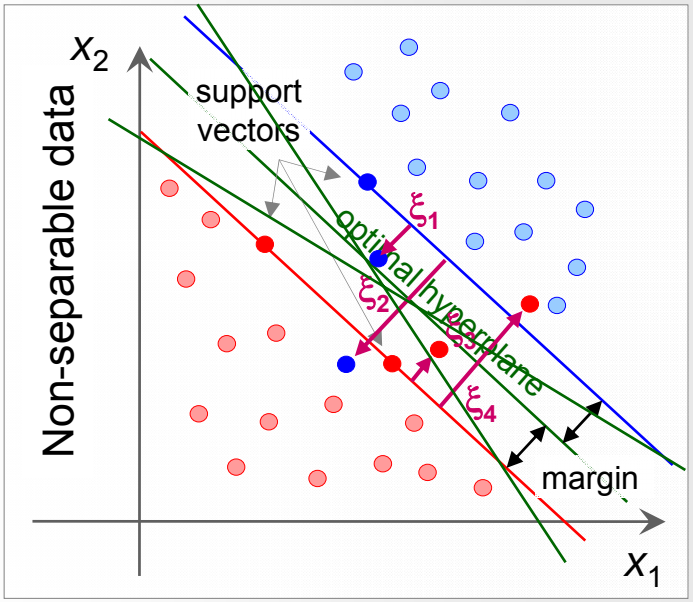
- There are methods to create linear decision boundaries using only measures of distances (= inner (scalar) products)
  - → leads to quadratic optimisation problem
- The decision boundary in the end is defined only by training events that are closest to the boundary
- We've seen that variable transformations, i.e moving into a higher dimensional space (i.e. using var1*var1 in Fisher Discriminant) can allow you to separate with linear decision boundaries non linear problems
- →Support Vector Machine

Support Vector Machines

- Find hyperplane that best separates signal from background
  - Linear decision boundary
  - Best separation: maximum distance (margin) between closest events (*support*) to hyperplane
  - If data non-separable add *misclassification cost* parameter $C \cdot \Sigma_i \xi_i$ to minimisation function

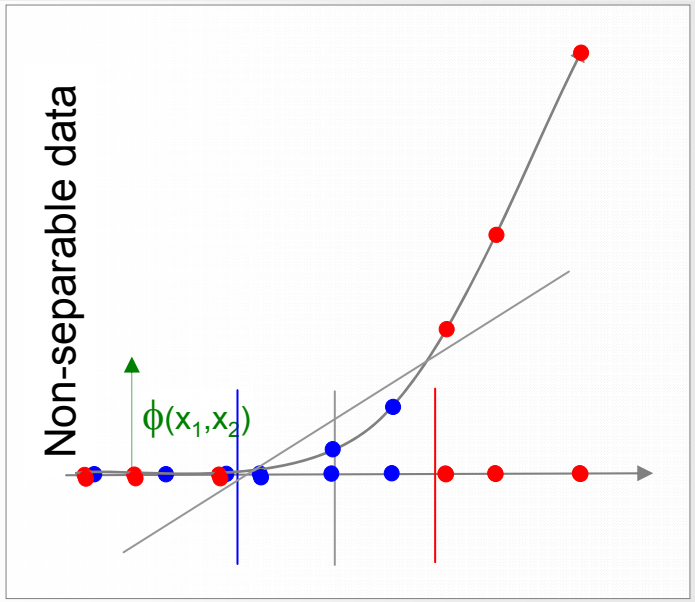
- **Solution of largest margin depends only on inner product of support vectors (distances)**

→ **quadratic minimisation problem**



Support Vector Machines

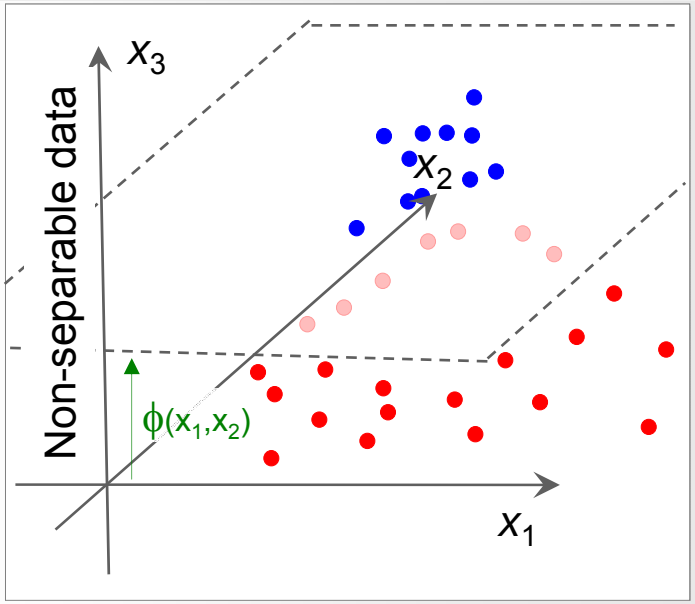
- Find hyperplane that best separates signal from background
  - Best separation: maximum distance (margin) between closest events (*support*) to hyperplane
  - Linear decision boundary
  - If data non-separable add *misclassification cost* parameter $C \cdot \Sigma_i \xi_i$ to minimisation function
  - **Solution depends only on inner product of support vectors → quadratic minimisation problem**



- Non-linear cases:
  - Transform variables into higher dimensional feature space where again a linear boundary (hyperplane) can separate the data

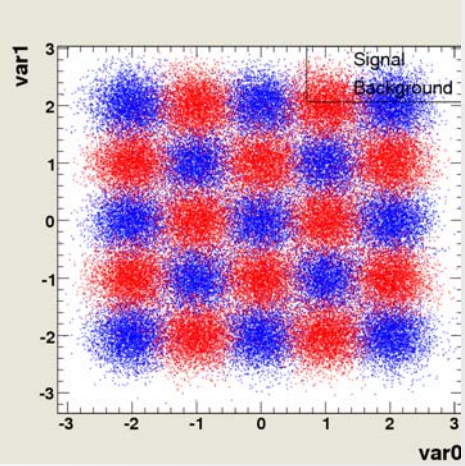
Support Vector Machines

- Find hyperplane that best separates signal from background
  - Best separation: maximum distance (margin) between closest events (*support*) to hyperplane
  - Linear decision boundary
  - If data non-separable add *misclassification cost* parameter $C \cdot \Sigma_i \xi_i$ to minimisation function
  - **Solution depends only on inner product of support vectors → quadratic minimisation problem**



- Non-linear cases:
  - Transform variables into higher dimensional feature space where again a linear boundary (hyperplane) can separate the data
  - Explicit transformation doesn't need to be specified. Only need the "scalar product" (inner product) $x \cdot x \rightarrow \Phi(x) \cdot \Phi(x)$.
    - certain *Kernel Functions* can be interpreted as scalar products between transformed vectors in the higher dimensional feature space. e.g.: Gaussian, Polynomial, Sigmoid
  - Choose Kernel and fit the hyperplane using the linear techniques developed above
  - Kernel size paramter typically needs careful tuning! (Overtraining!)

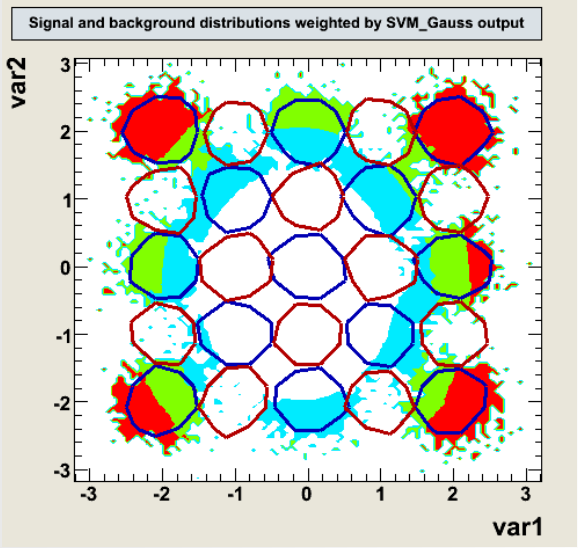
Support Vector Machines



SVM: the Kernel size parameter:

example: Gaussian Kernels

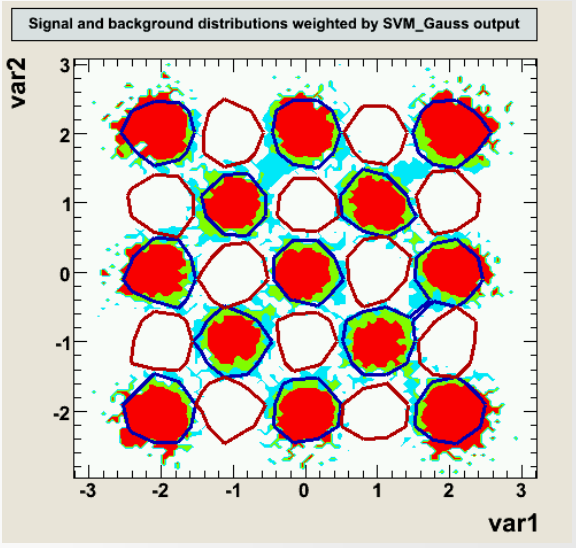
- Kernel size (σ of the Gaussian) choosen too large: → not enough "flexibility" in the underlying transformation



colour code:

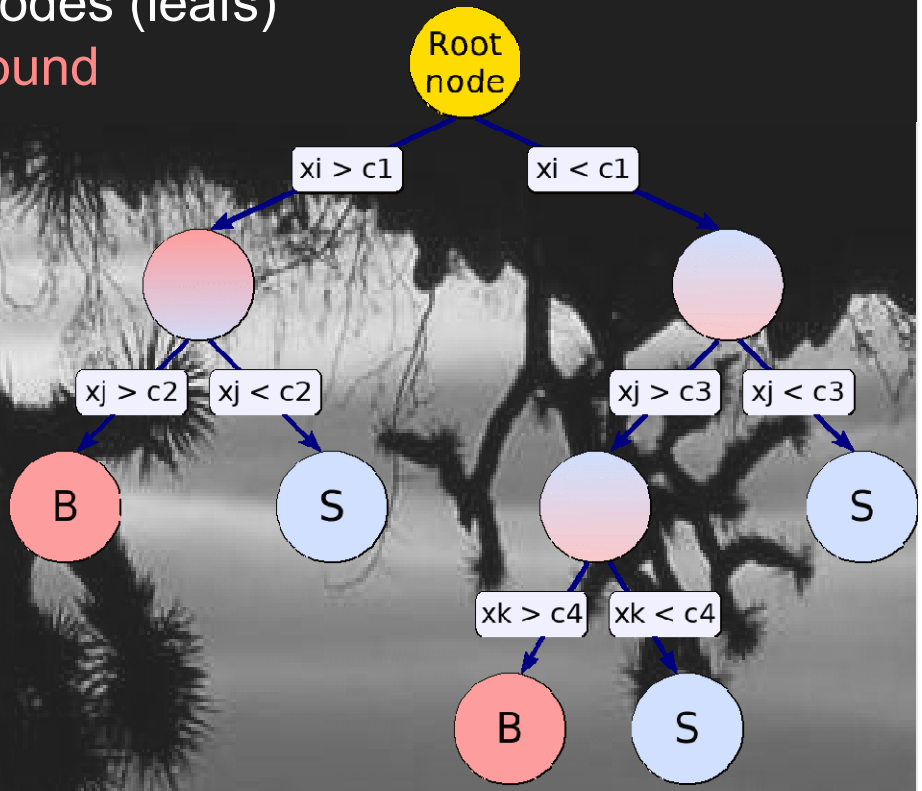
Red → large signal probability:

- Kernel size (σ of the Gaussian) choosen propperly for the given problem



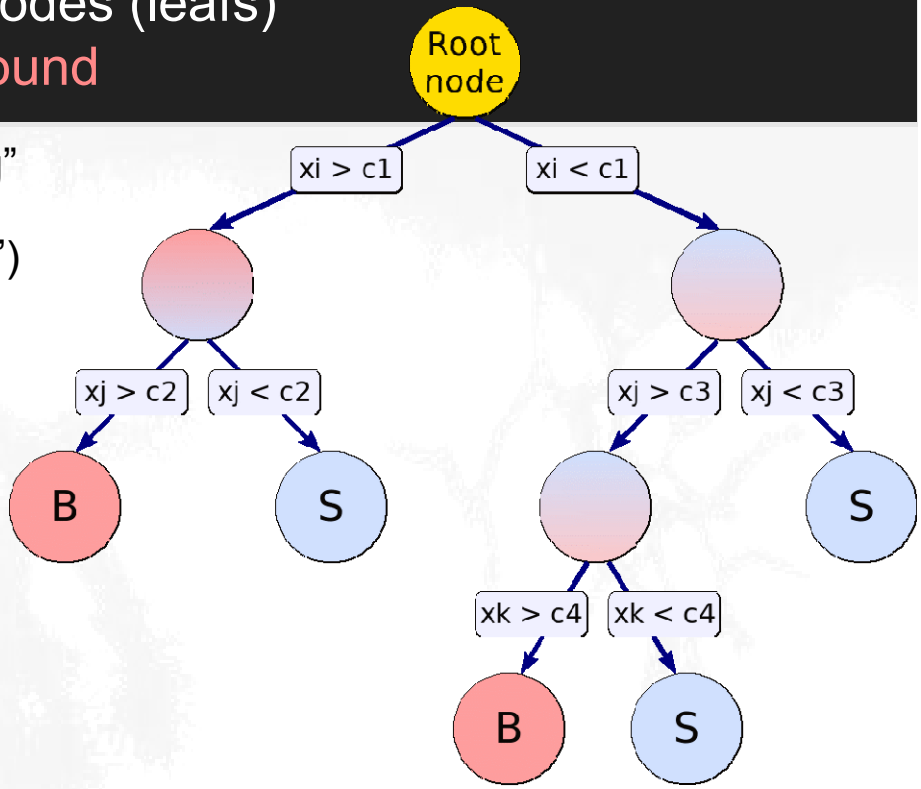
Boosted Decision Trees

- Decision Tree: Sequential application of cuts splits the data into nodes, where the final nodes (leafs) classify an event as signal or background



Boosted Decision Trees

- Decision Tree: Sequential application of cuts splits the data into nodes, where the final nodes (leafs) classify an event as signal or background



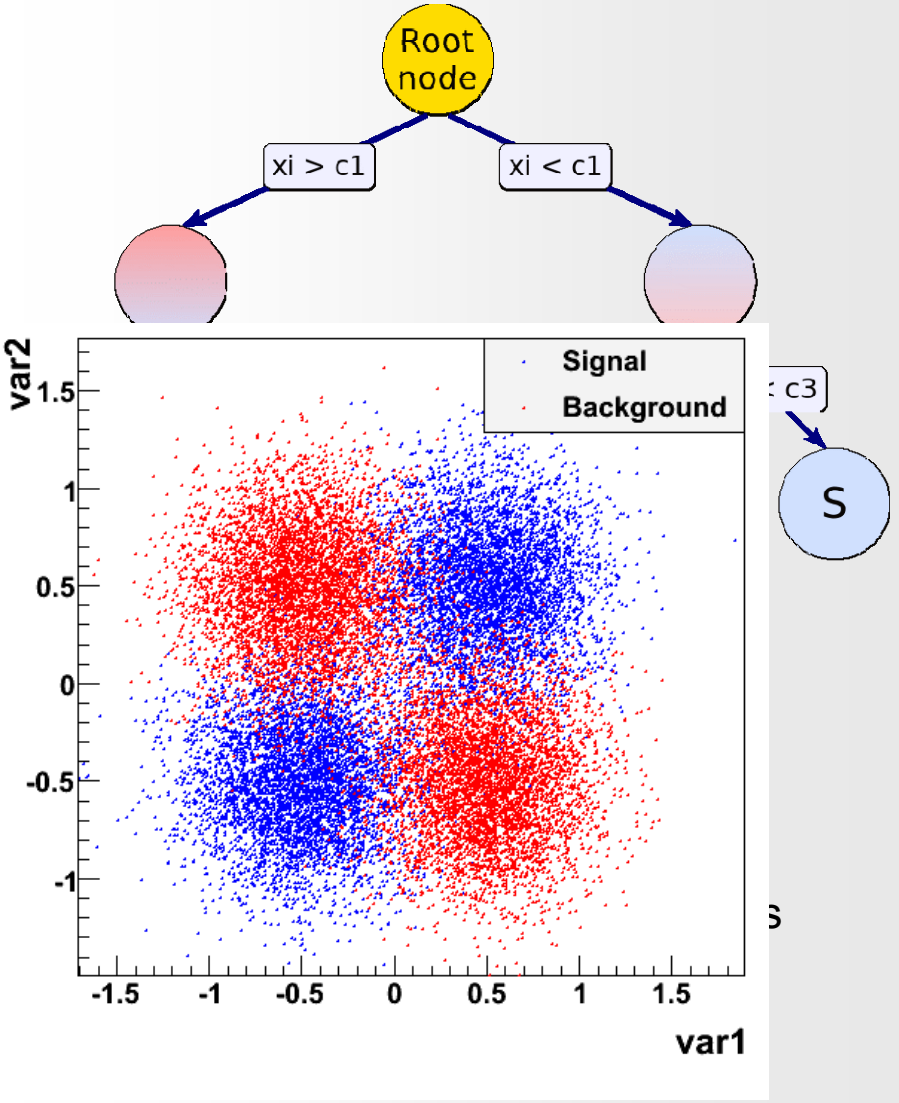
- used since a long time in general "data-mining" applications, less known in (High Energy) Physics (although very similar to "simple Cuts")
  - easy to interpret, visualised
  - independent of monotonous variable transformations, immune against outliers
  - weak variables are ignored (and don't (much) deteriorate performance)
- Disadvatage → very sensitive to statistical fluctuations in training data

- Boosted Decision Trees (1996): combine a whole forest of Decision Trees, derived from the same sample, e.g. using different event weights.
  - overcomes the stability problem

→ became popular in HEP since MiniBooNE, B.Roe et.a., NIM 543(2005)

Growing a Decision Tree

- start with training sample at the root node
- split training sample at node into two, using a cut in the one variable that gives best separation gain
- continue splitting until:
  - minimal #events per node
  - maximum number of nodes
  - maximum depth specified
  - (a split doesn't give a minimum separation g → not a good idea → see "chessboard"
- leaf-nodes classify S,B according to the majority of events or give a S/B probability
- Why no multiple branches (splits) per node ?
  - → Fragments data too quickly; also: multiple s
- What about multivariate splits?
  - → time consuming
  - → other methods more adapted for such correlations



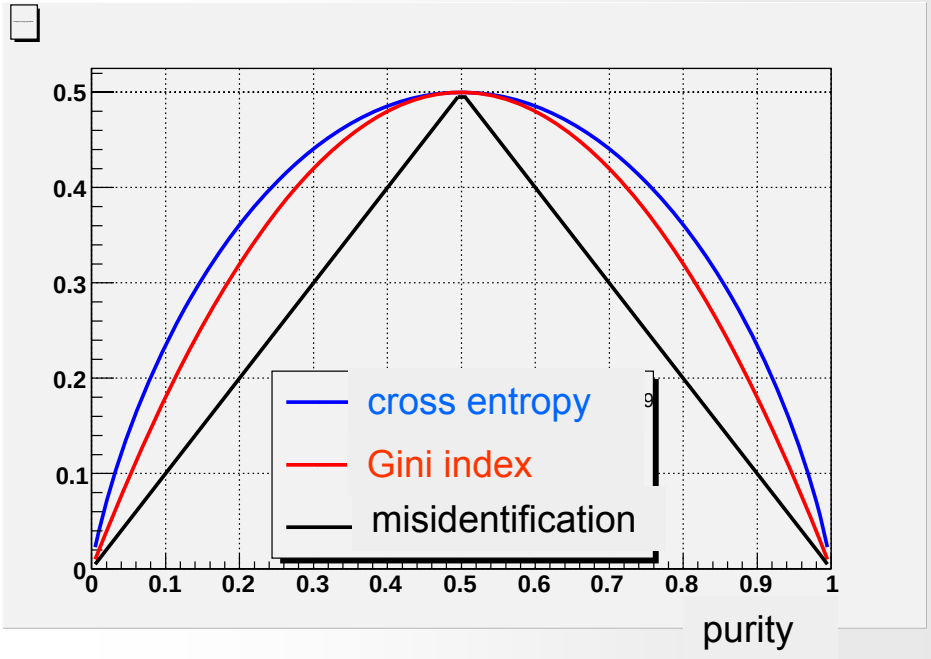
Separation Gain

- What do we mean by “best separation gain”?
- define a measure on how mixed S and B in a node are:
  - Gini-index: (Corrado Gini 1912, typically used to measure income inequality)

    p (1-p) : p=purity
  - Cross Entropy:

    -(plnp + (1-p)ln(1-p))
  - Misidentification:

    1-max(p,1-p)



- difference in the various indices are small, most commonly used: Gini-index

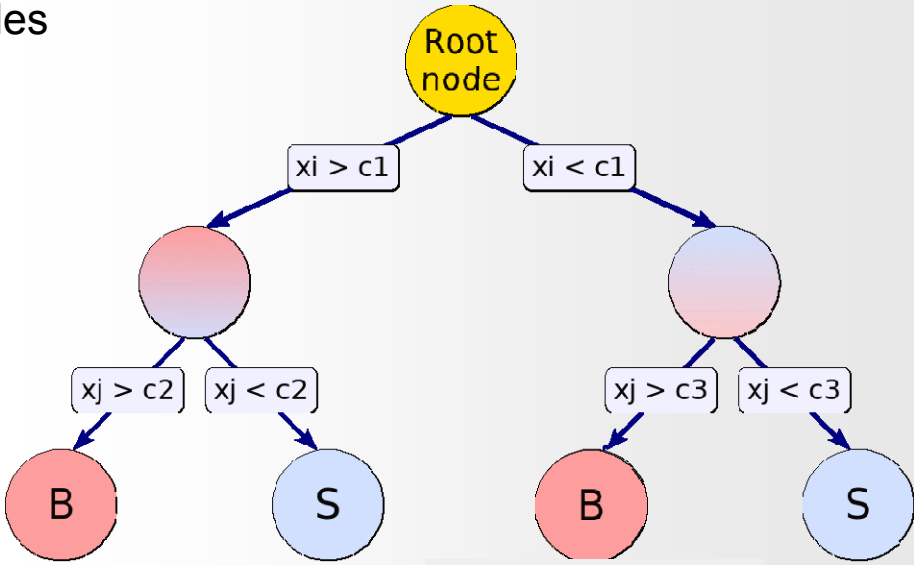
separation gain: e.g. $N_{Parent}*Gini_{Parent} - N_{left}*Gini_{LeftNode} - N_{right}*Gini_{RightNode}$

- Choose amongst all possible variables and cut values the one that maximised the this.

Decision Tree Pruning

- One can continue node splitting until all leaf nodes are basically pure (using the training sample)

  → obviously: that's overtraining



- Two possibilities:
  - stop growing earlier

    generally not a good idea, useless splits might open up subsequent usefull splits
  - grow tree to the end and "cut back", nodes that seem statistically dominated:

    → pruning

- e.g. Cost Complexity pruning:
  - assign to every sub-tree, T $C(T,\alpha)$:
  - find subtree T with minmal $C(T,\alpha)$ for given α
  - prune up to value of a that does not show overtraining in the test sample

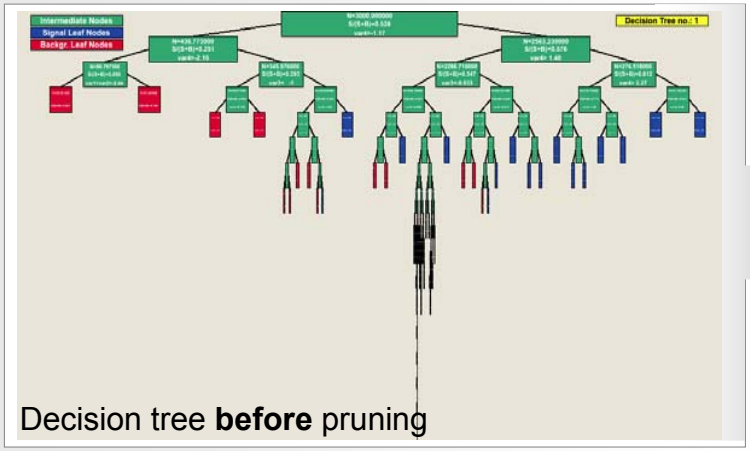
$$C(T,\alpha) = \sum_{\text{leafs of T}} \sum_{\text{events in leaf}} |\, y(x) - y(C) \,| + \alpha N_{\text{leaf nodes}}$$

Loss function

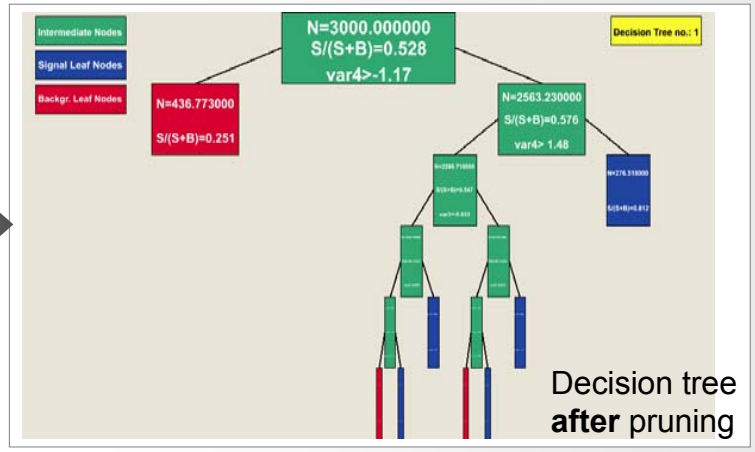
regularisaion/ cost parameter

Decision Tree Pruning

- "Real life" example of an optimally pruned Decision Tree:



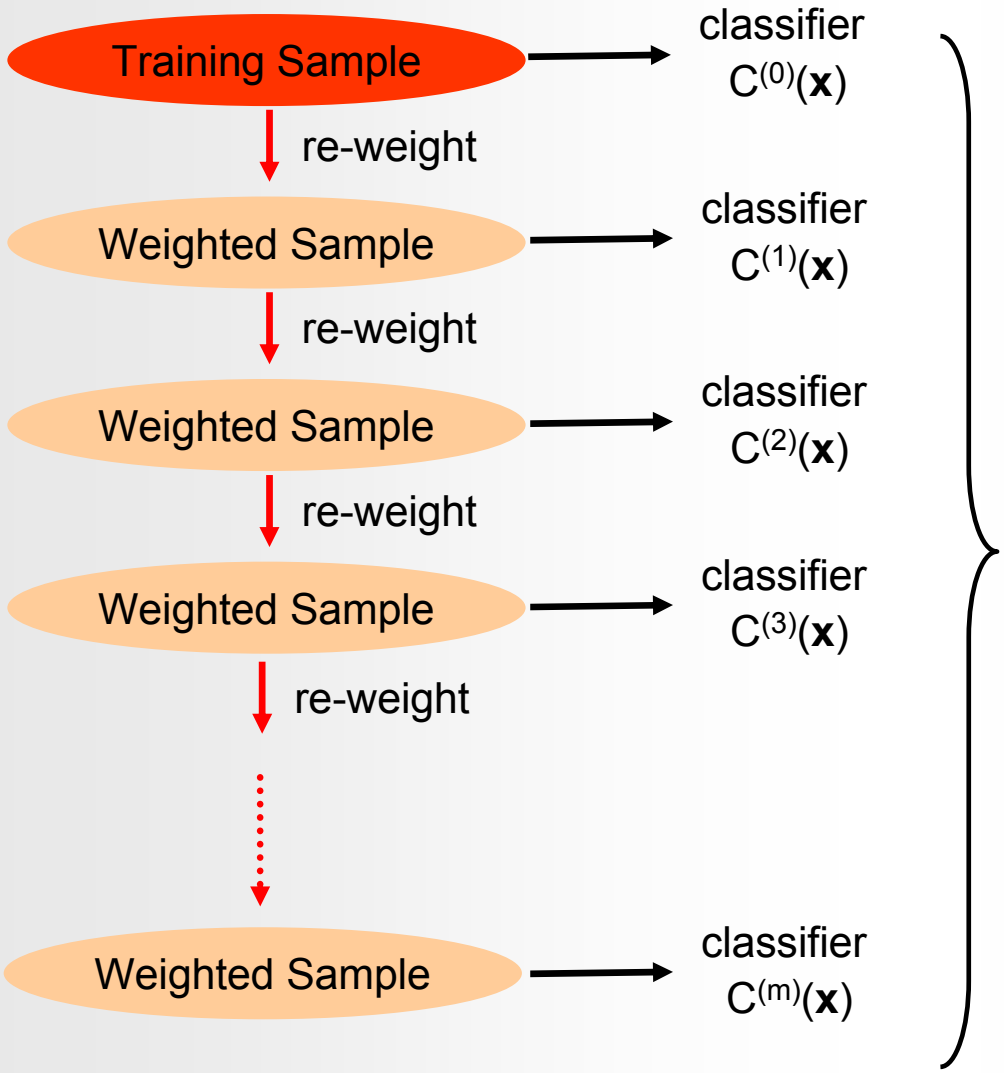
Decision tree **before** pruning



Decision tree **after** pruning

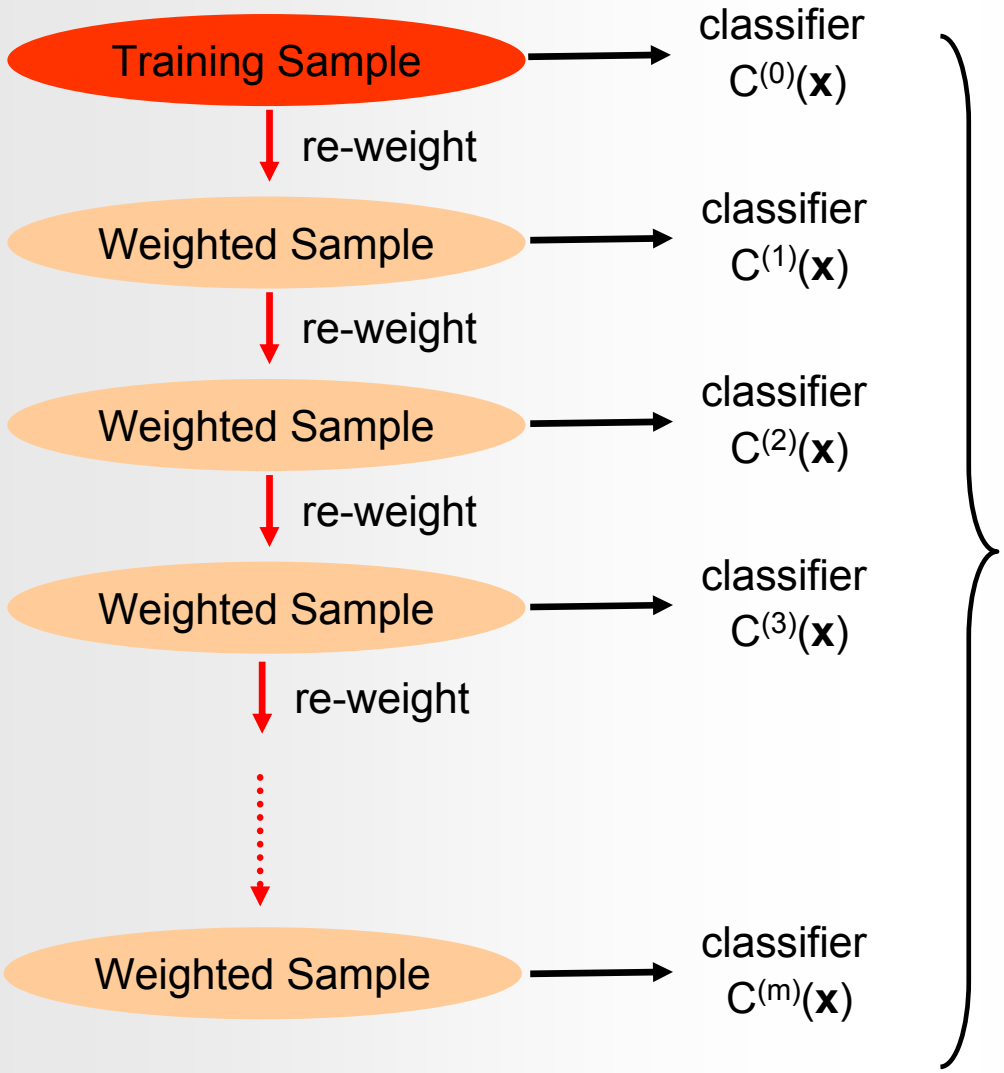
- Pruning algorithms are developed and applied on individual trees
  - optimally pruned single trees are not necessarily optimal in a forest !

Boosting



$$y(x) = \sum_{i}^{N_{Classifier}} w_i C^{(i)}(x)$$

Adaptive Boosting (AdaBoost)



- AdaBoost re-weights events misclassified by previous classifier by:

$$\frac{1-f_{err}}{f_{err}} \quad \text{with :}$$

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

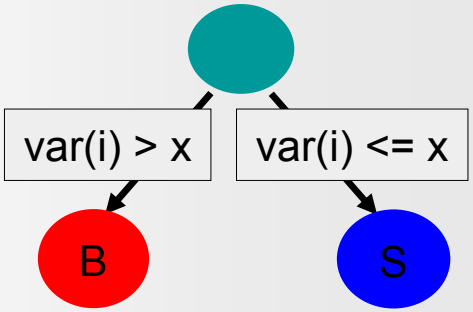
- AdaBoost weights the classifiers also using the error rate of the individual classifier according to:

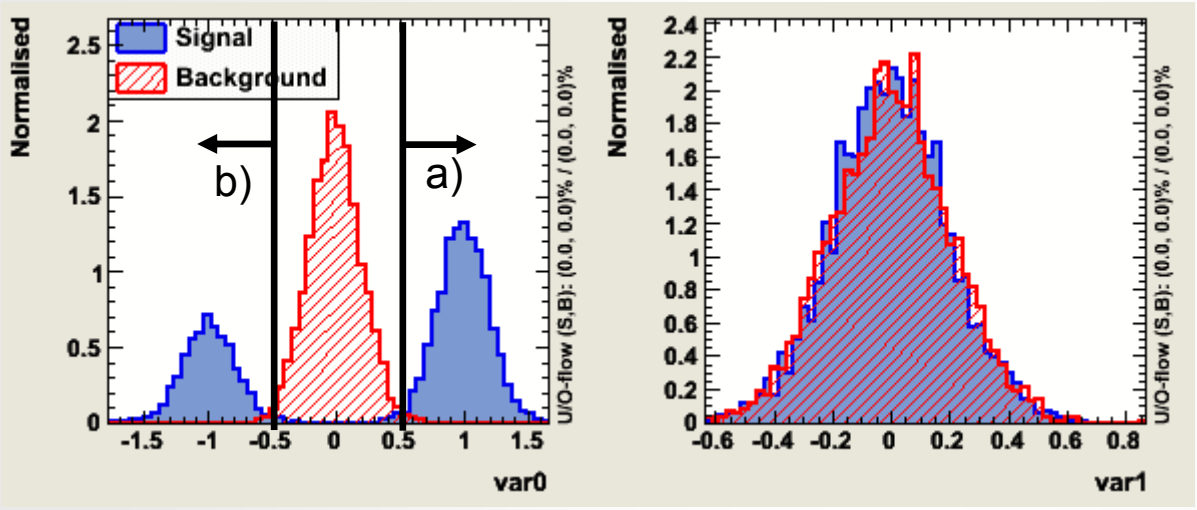
$$y(x) = \sum_{i}^{N_{Classifier}} \log\left(\frac{1-f_{err}^{(i)}}{f_{err}^{(i)}}\right) C^{(i)}(x)$$

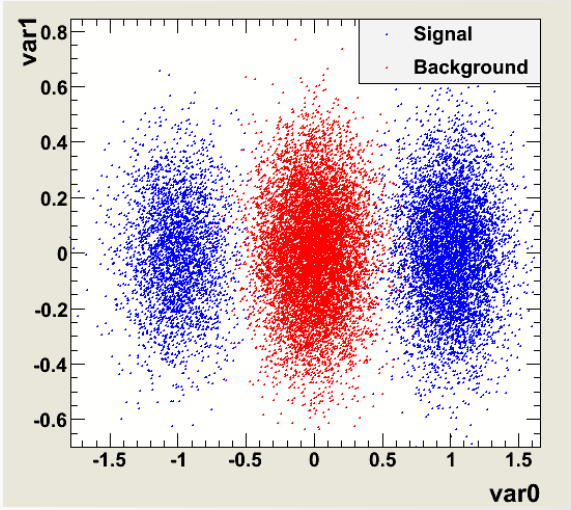
AdaBoost: A simple demonstration

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%

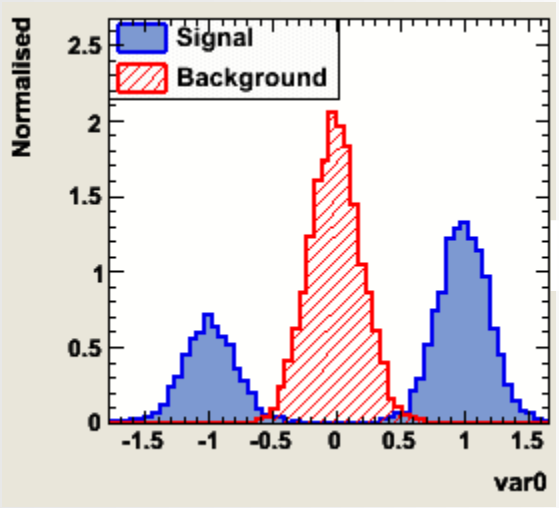
the training of a single decision tree stump will find “cut a)”

AdaBoost: A simple demonstration

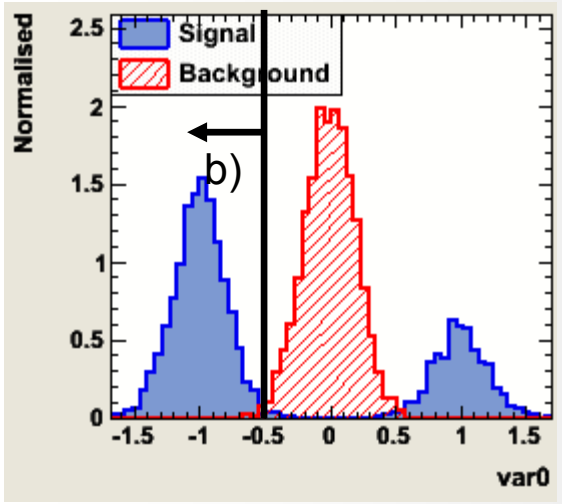
The first “tree”, choosing cut a) will give an error fraction: err = 0.165

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

➔ the next “tree” sees essentially the following data sample:

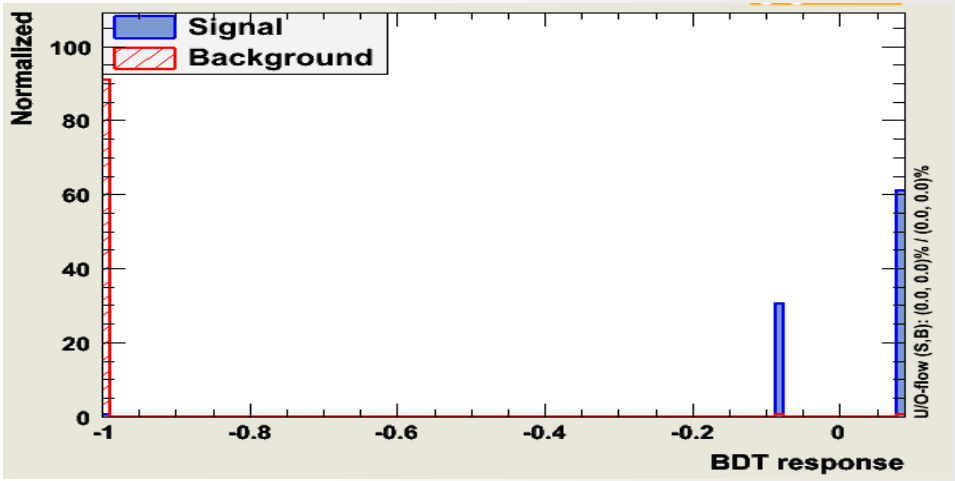


re-weight



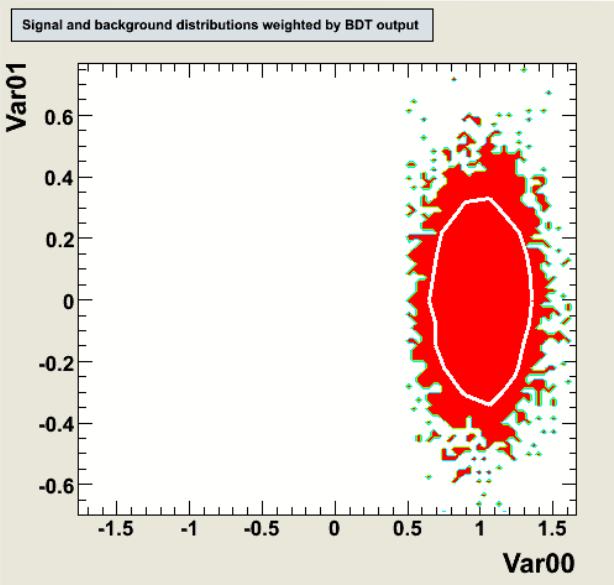
.. and hence will chose: “cut b)”: Var0 < -0.5

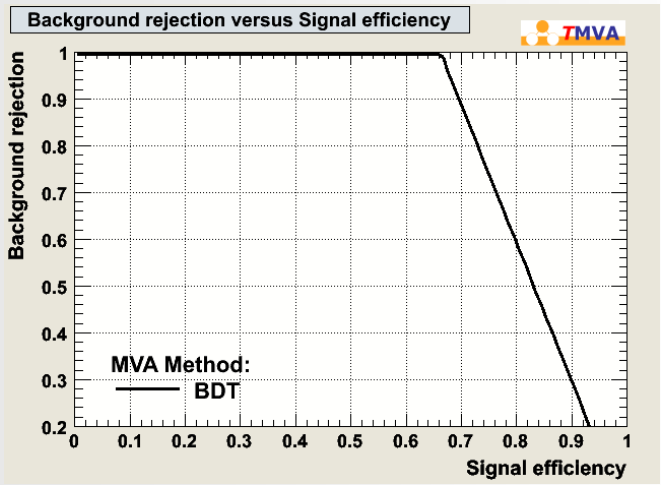
The combined classifier: Tree1 + Tree2
the (weighted) average of the response to a test event from both trees is able to separate signal from background as good as one would expect from the most powerful classifier



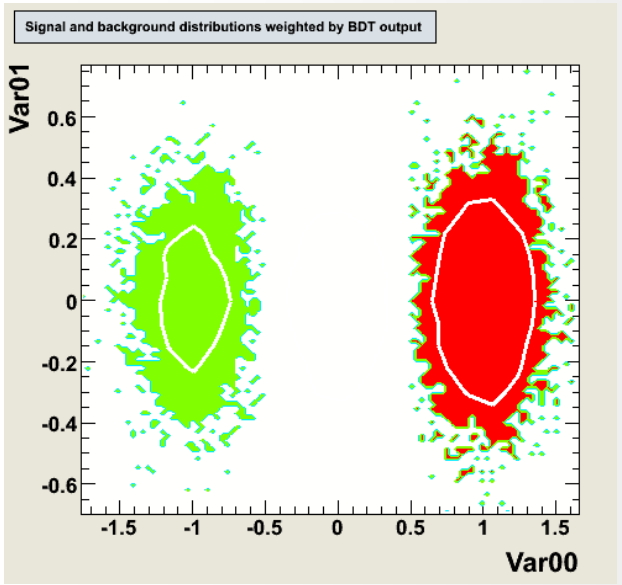
AdaBoost: A simple demonstration

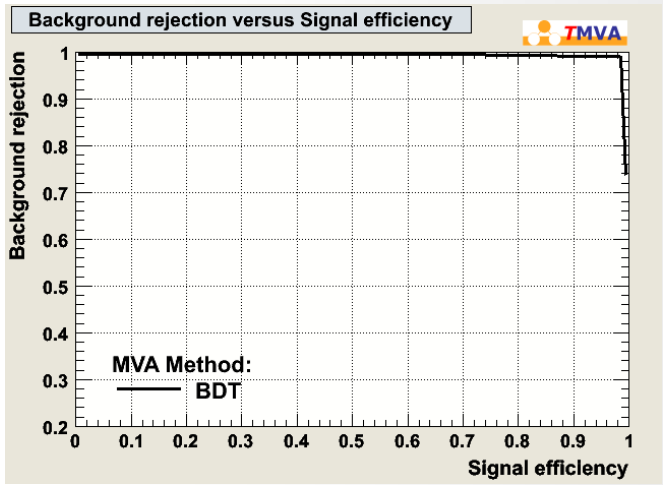
Only 1 tree “stump”





Only 2 tree “stumps” with AdaBoost





Gradient Boost

- Gradient Boost is a way to implement “boosting” with arbitrary “loss functions” by approximating “somehow” the gradient of the loss function
- AdaBoost: Exponential loss $\exp(-y_0 y(\alpha,x))$ → theoretically sensitive to outliers
- Binomial log-likelihood loss $\ln(1 + \exp(-2y_0 y(\alpha,x))$ → more well behaved loss function,
  (the corresponding “GradientBoost” is implmented in TMVA)

Bagging and Randomised Trees

other classifier combinations:

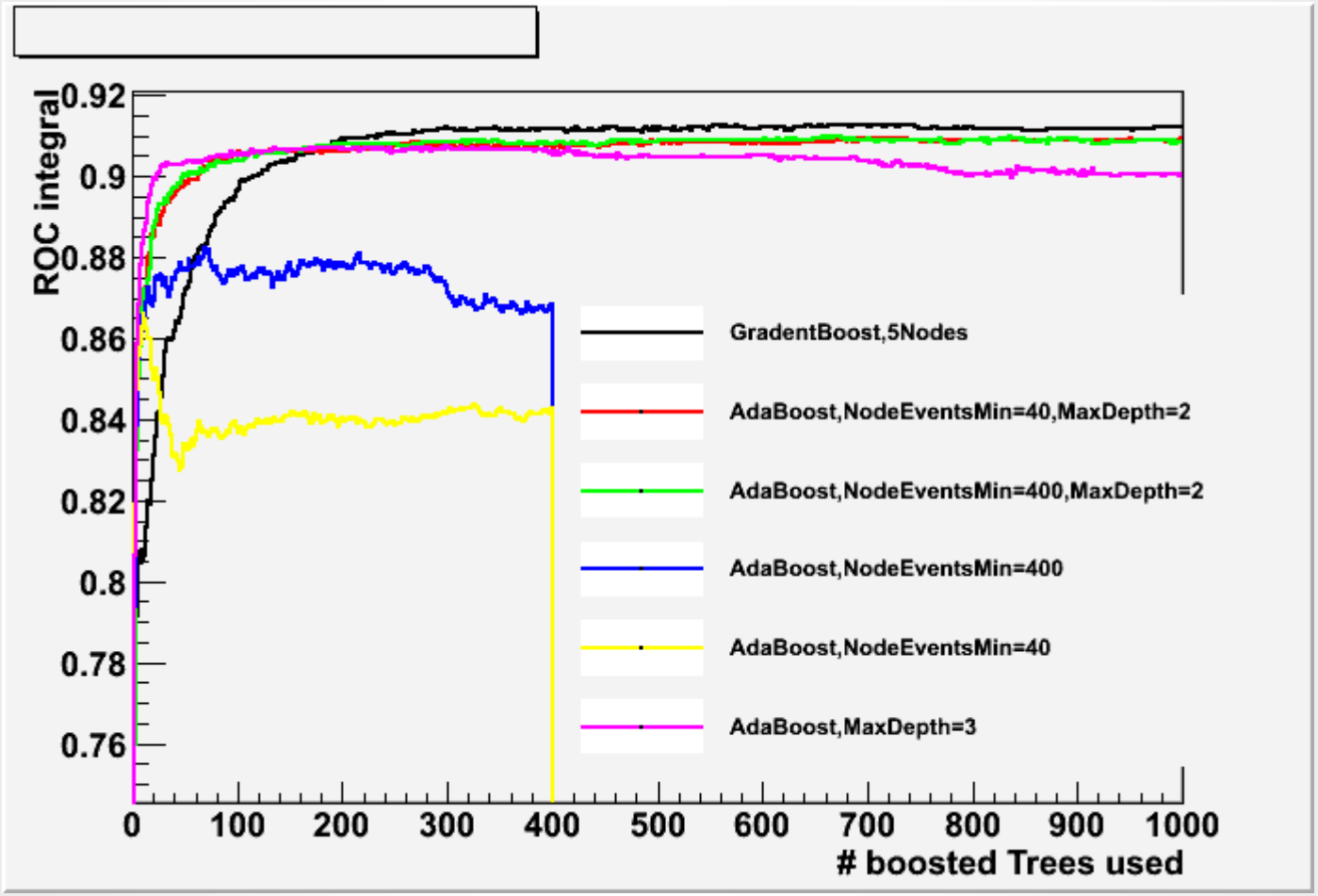
- Bagging:
  - combine trees grown from “bootstrap” samples

  (i.e re-sample training data with replacement)

- Randomised Trees: **(Random Forest: trademark L.Breiman, A.Cutler)**
  - combine trees grown with:
    - random bootstrap (or subsets) of the training data only
    - consider at each node only a random subsets of variables for the split
    - NO Pruning!

- These combined classifiers work surprisingly well, are very stable and almost perfect “out of the box” classifiers

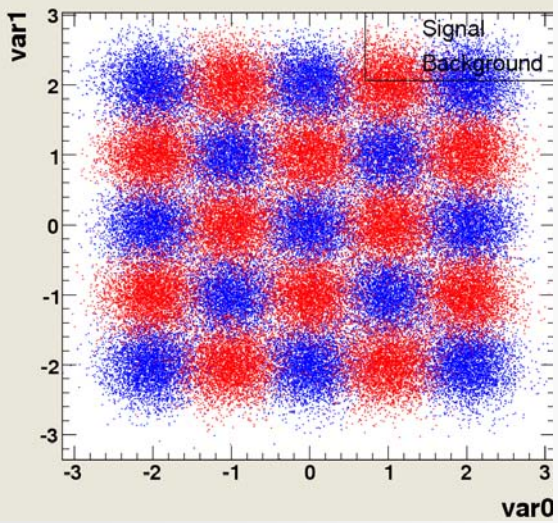
Boosting at Work



- Another plot showing how important it is to choose good “tuning parameters”

The *Schachbrett* Toy (chess board)

Event Distribution

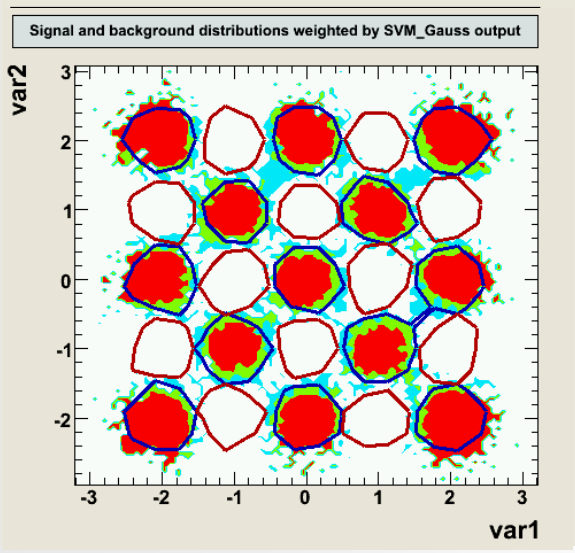


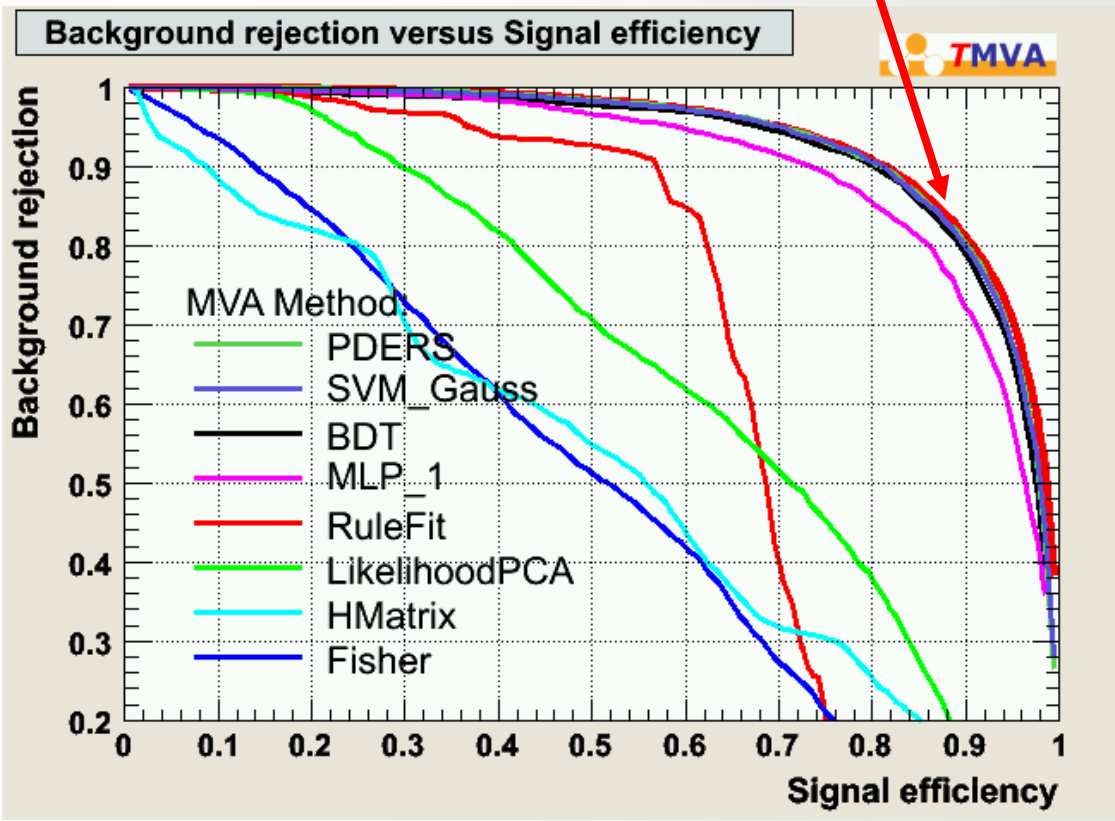
- **Performance achieved without parameter adjustments:**

nearest Neighbour and BDTs are best "out of the box"

- **After some parameter tuning, also SVM und ANN(MLP) perform**

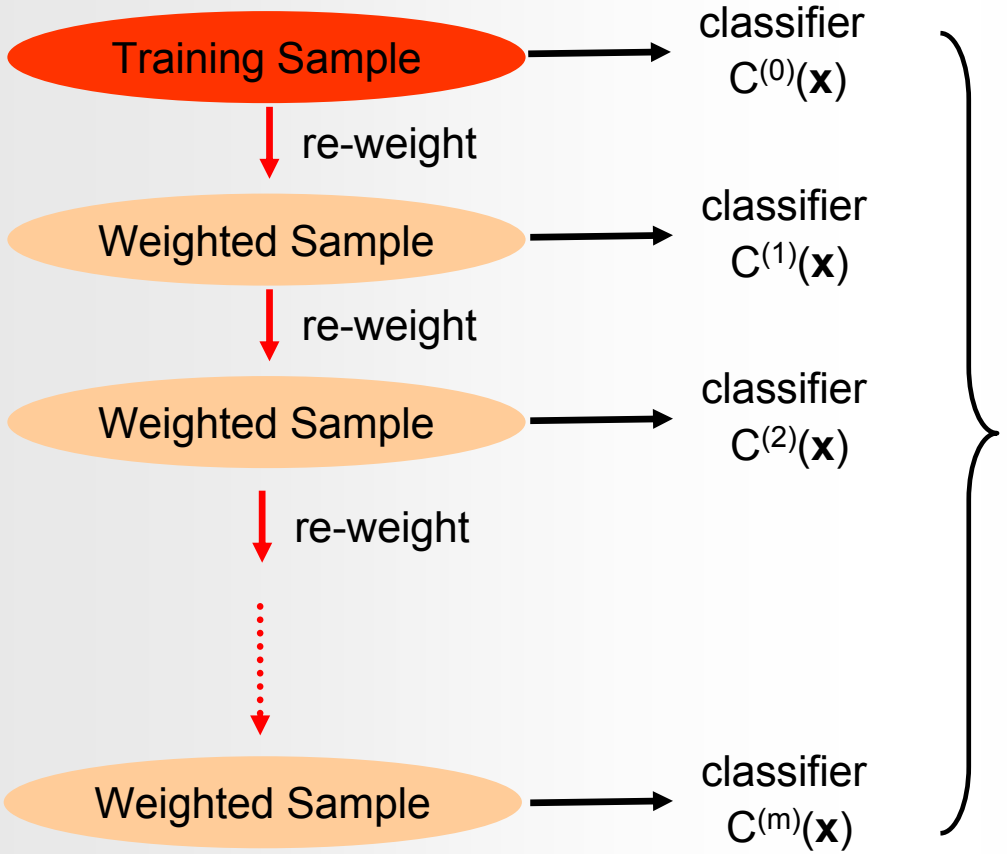
Events weighted by SVM response





Generalised Classifier Boosting

- Principle (just as in BDT): multiple training cycles, each time wrongly classified events get a higher event weight



$$y(x) = \sum_{i}^{N_{Classifier}} \log\left(\frac{1 - f_{err}^{(i)}}{f_{err}^{(i)}}\right) C^{(i)}(x)$$

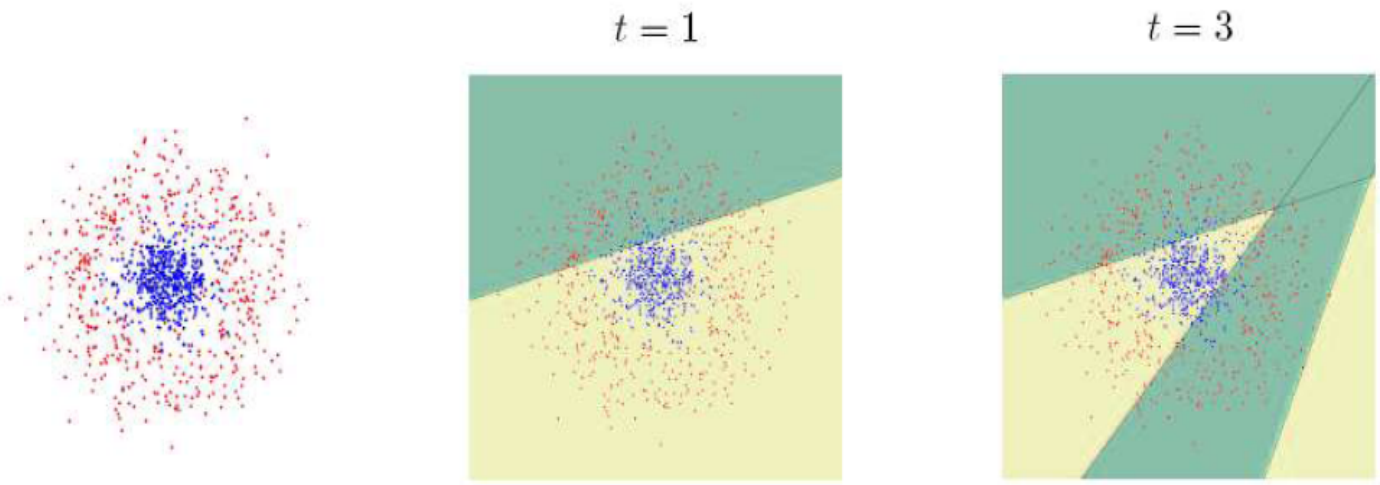
Response is weighted sum of each classifier response

Boosting will be interesting especially for Methods like Cuts, MLP, and SVM

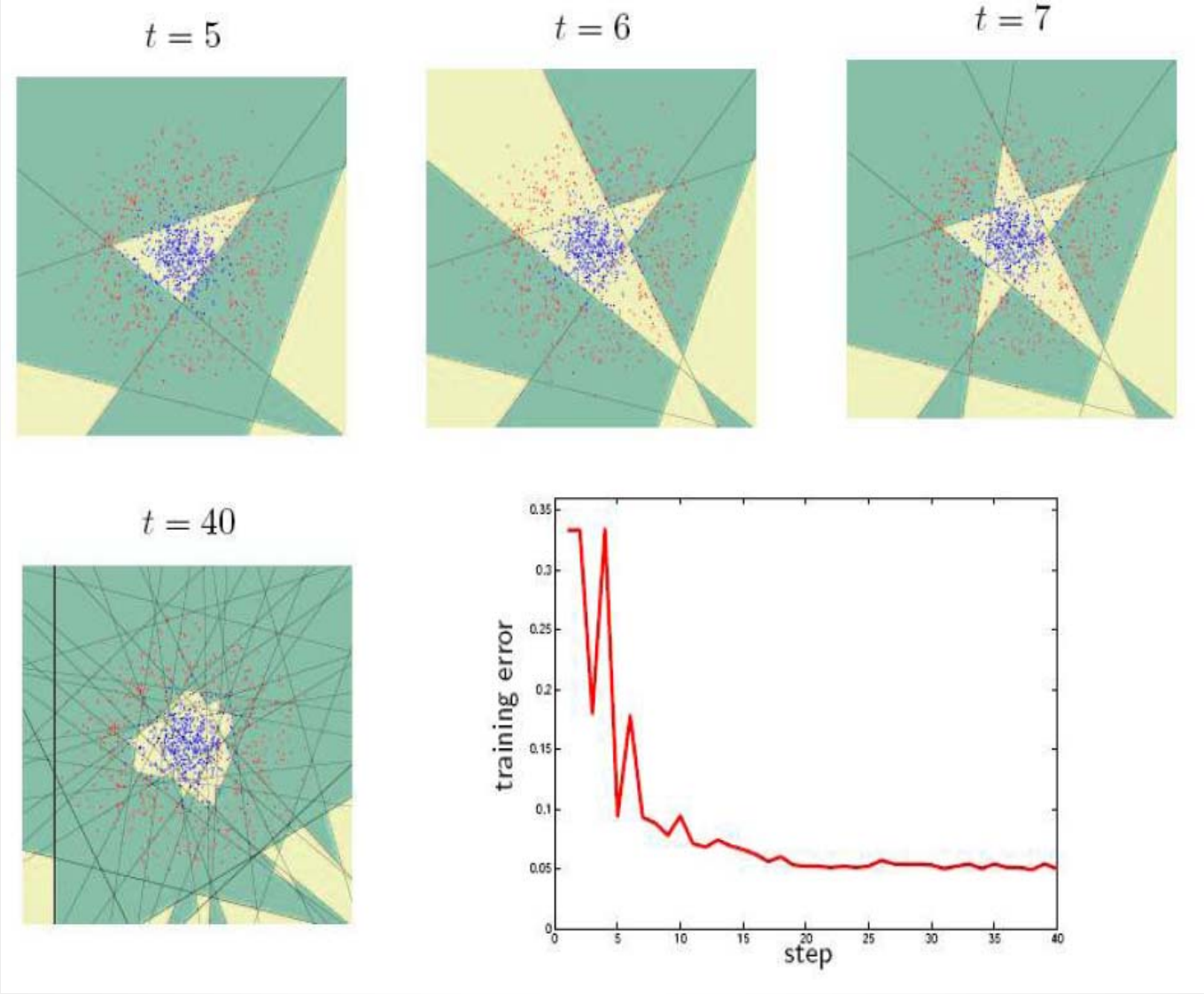
AdaBoost On a linear Classifier (e.g. Fisher)

J. Sochman, J. Matas, `cmp.felk.cvut.cz`

Start with a problem for which a linear classifier is weak:



AdaBoost On a linear Classifier (e.g. Fisher)



Learning with Rule Ensembles

- Following RuleFit approach by Friedman-Popescu

Friedman-Popescu, Tech Rep, Stat. Dpt, Stanford U., 2003

- Model is linear combination of *rules*, where a rule is a sequence of cuts

RuleFit classifier

rules (cut sequence → r_m=1 if all cuts satisfied, =0 otherwise)

normalised discriminating event variables

$$y_{RF}(\vec{x}) = a_0 + \sum_{m=1}^{M_R} a_m r_m(\vec{\hat{x}}) + \sum_{k=1}^{n_R} b_k \hat{x}_k$$

Sum of rules Linear Fisher term

- The problem to solve is
  - Create rule ensemble: use forest of decision trees
  - Fit coefficients a_m, b_k: gradient direct regularization minimising *Risk* (Friedman et al.)
- Pruning removes topologically equal rules" (same variables in cut sequence)

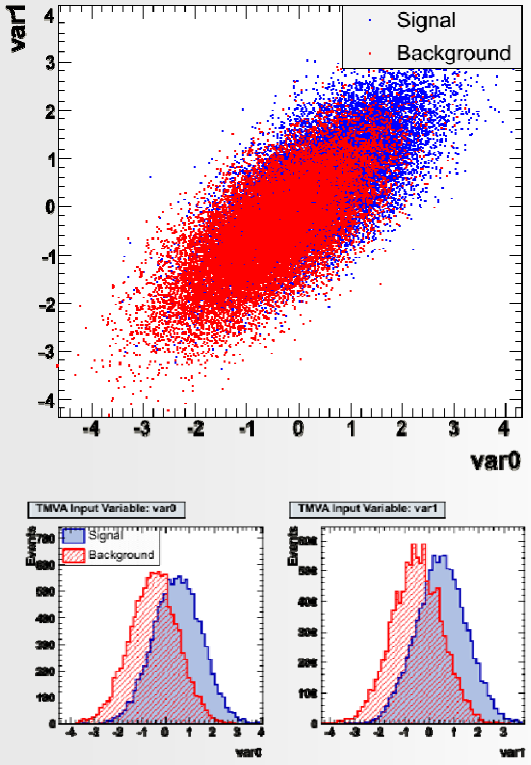
One of the elementary cellular automaton rules (Wolfram 1983, 2002). It specifies the next color in a cell, depending on its color and its immediate neighbors. Its rule outcomes are encoded in the binary representation $30=00011110_2$.

See some Classifiers at Work

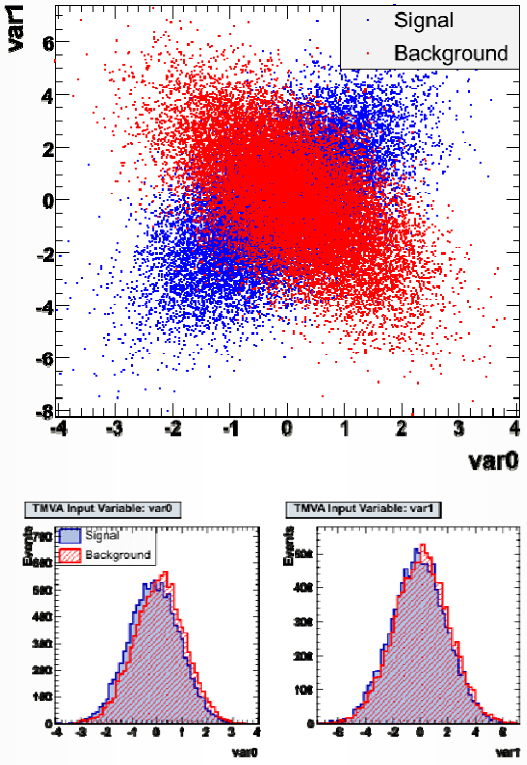
Toy Examples: Linear-, Cross-, Circular Correlations

- Illustrate the behaviour of linear and nonlinear classifiers

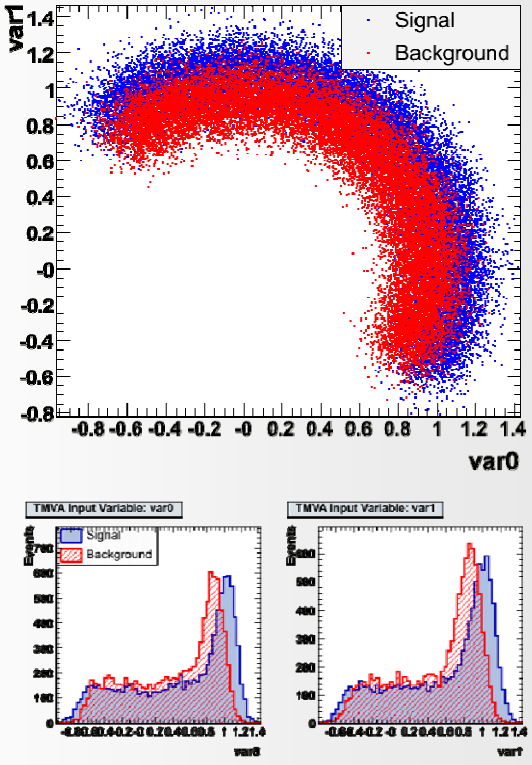
Linear correlations
(same for signal and background)



Linear correlations
(opposite for signal and background)



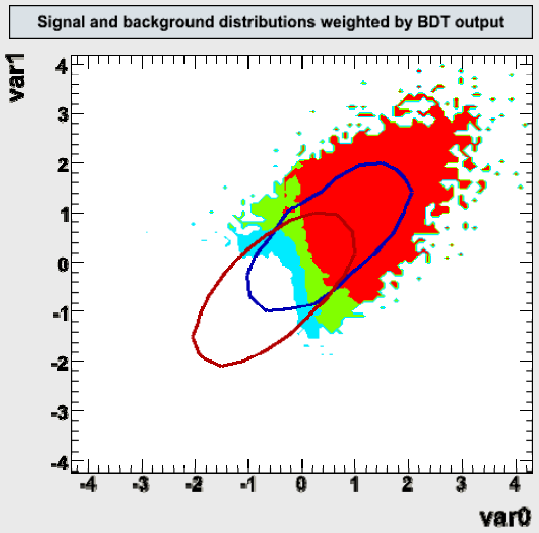
Circular correlations
(same for signal and background)



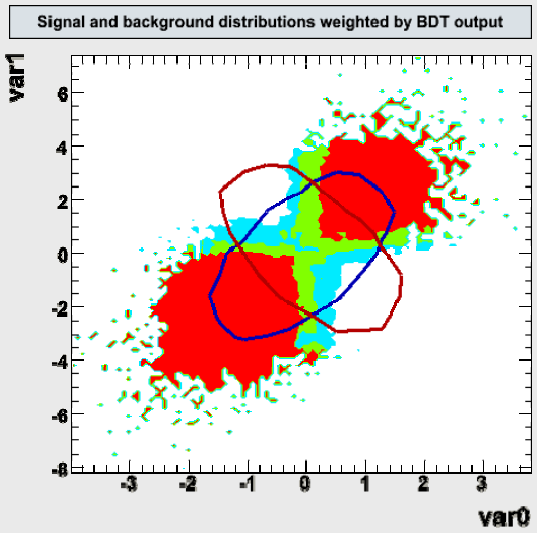
Weight Variables by Classifier Output

- How well do the classifier resolve the various correlation patterns ?

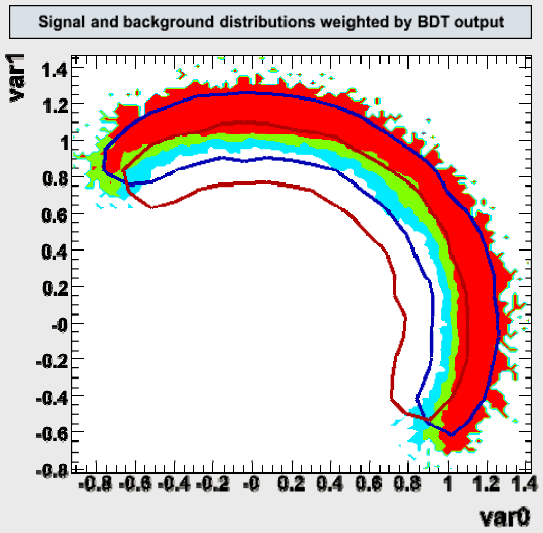
Linear correlations
(same for signal and background)



Cross-linear correlations
(opposite for signal and background)



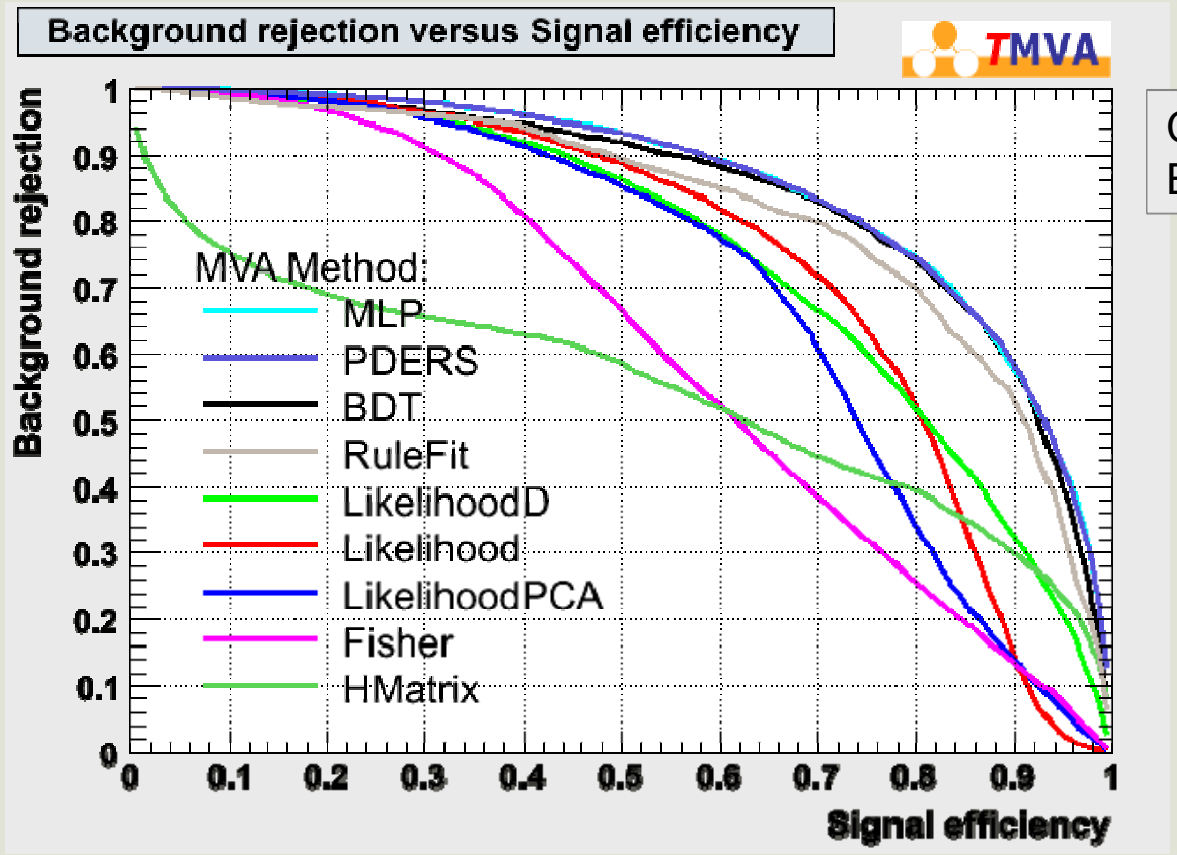
Circular correlations
(same for signal and background)



BDT

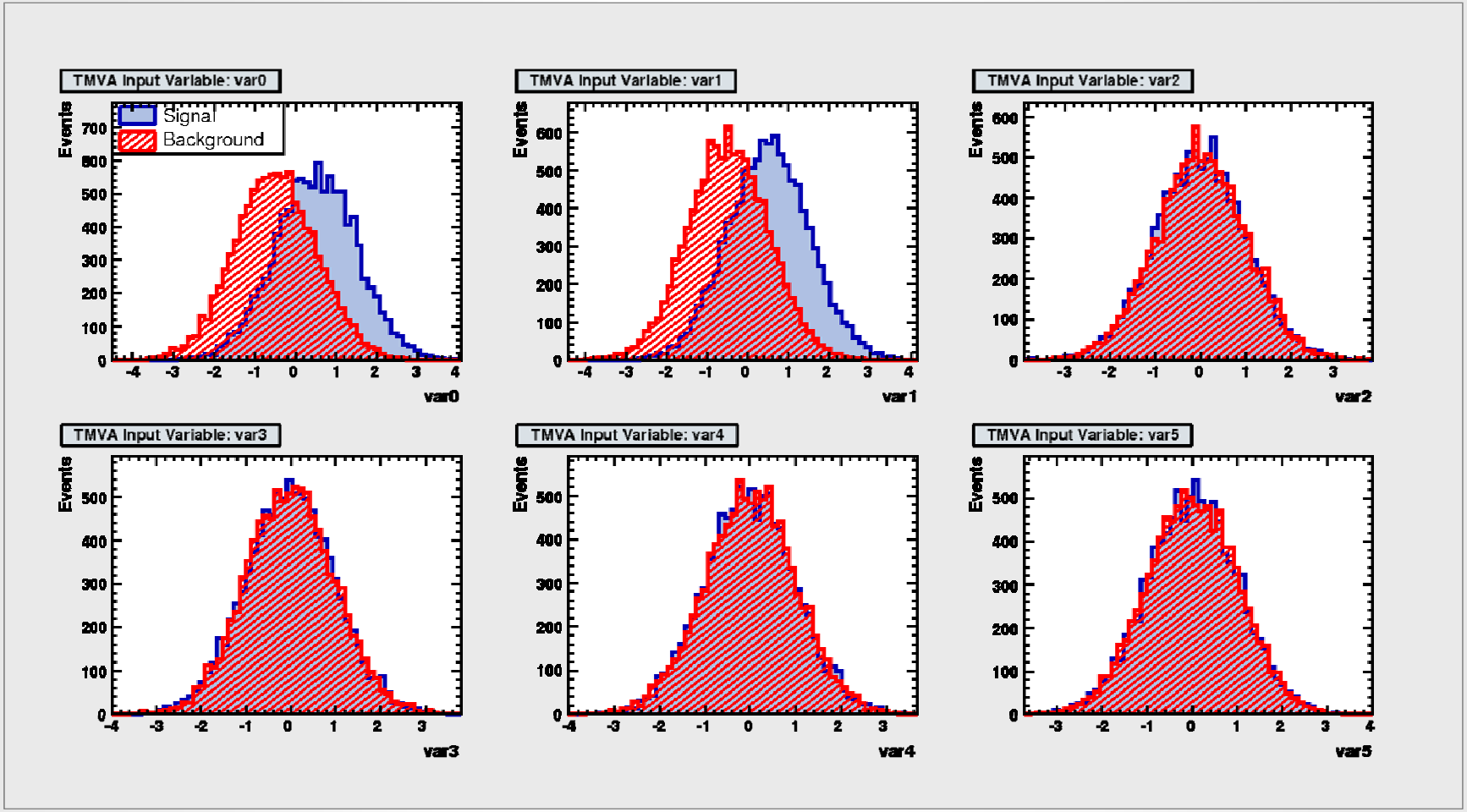
Final Classifier Performance

- Background rejection versus signal efficiency curve:



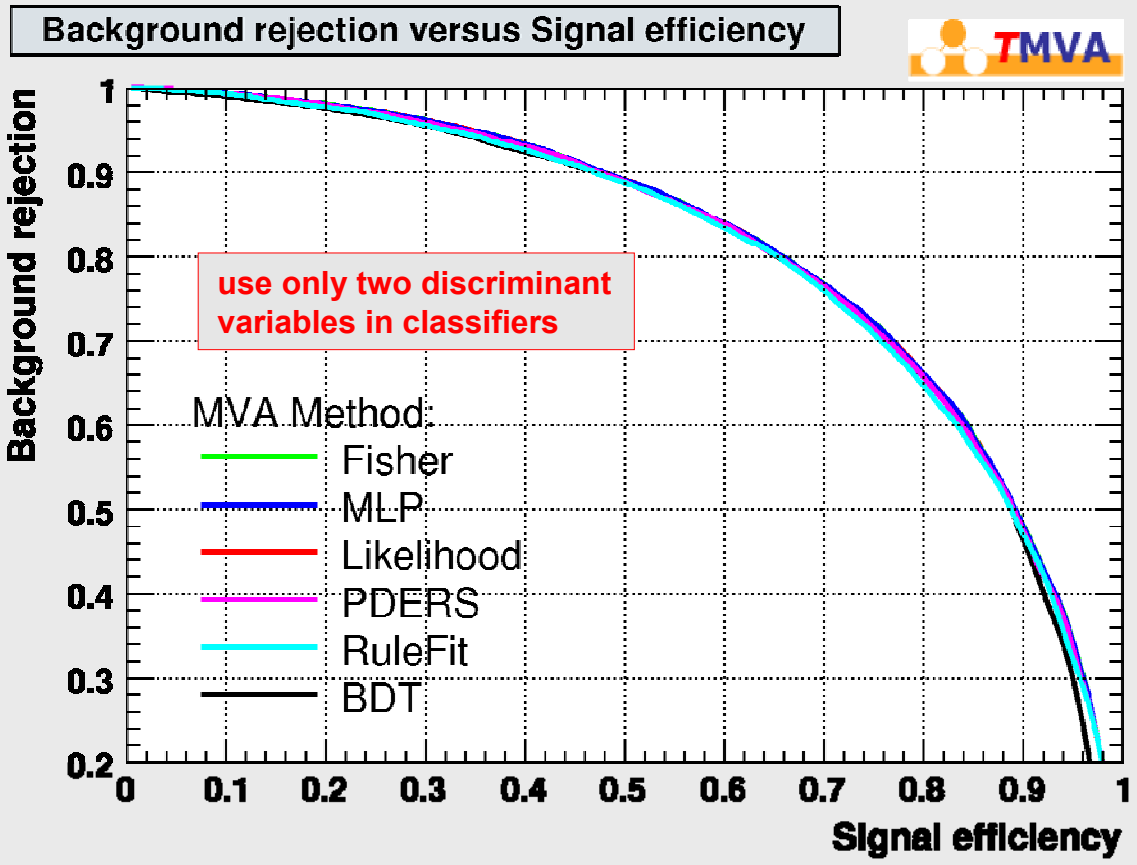
Stability with Respect to Irrelevant Variables

- Toy example with 2 discriminating and 4 non-discriminating variables ?



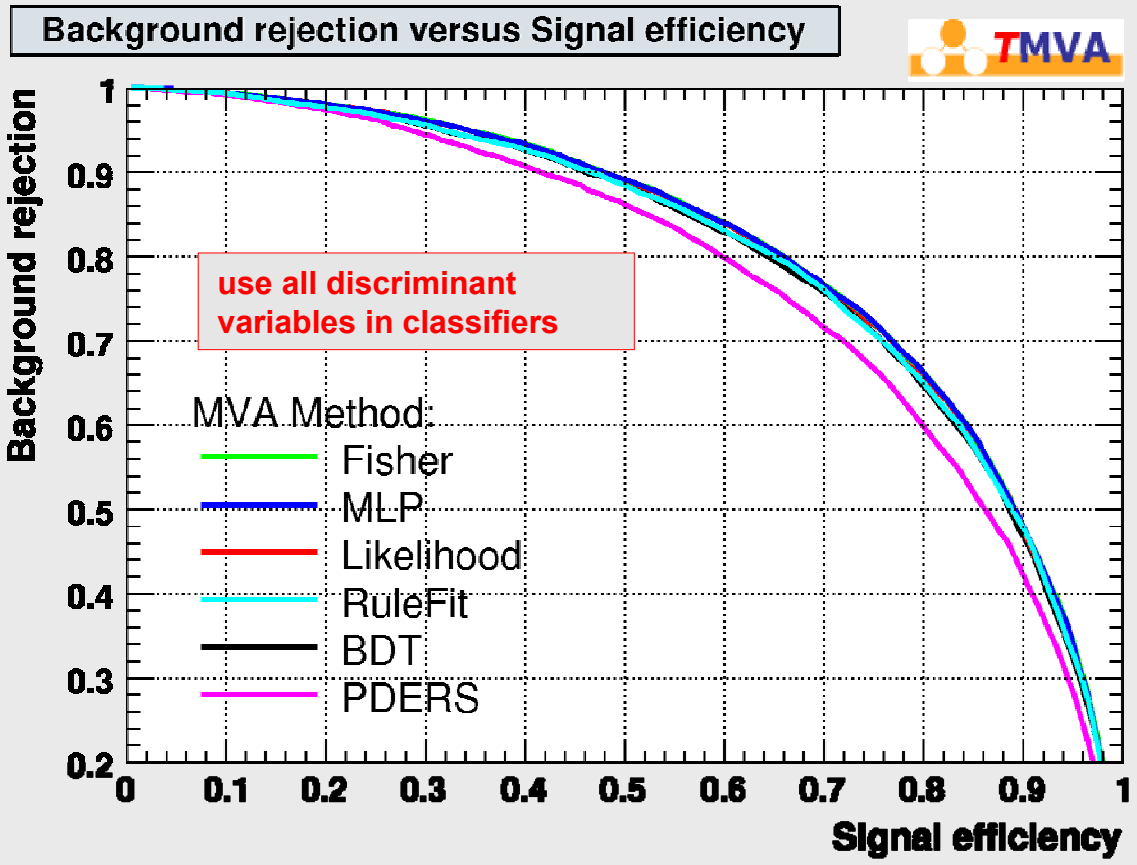
Stability with Respect to Irrelevant Variables

- Toy example with 2 discriminating and 4 non-discriminating variables ?



Stability with Respect to Irrelevant Variables

- Toy example with 2 discriminating and 4 non-discriminating variables ?

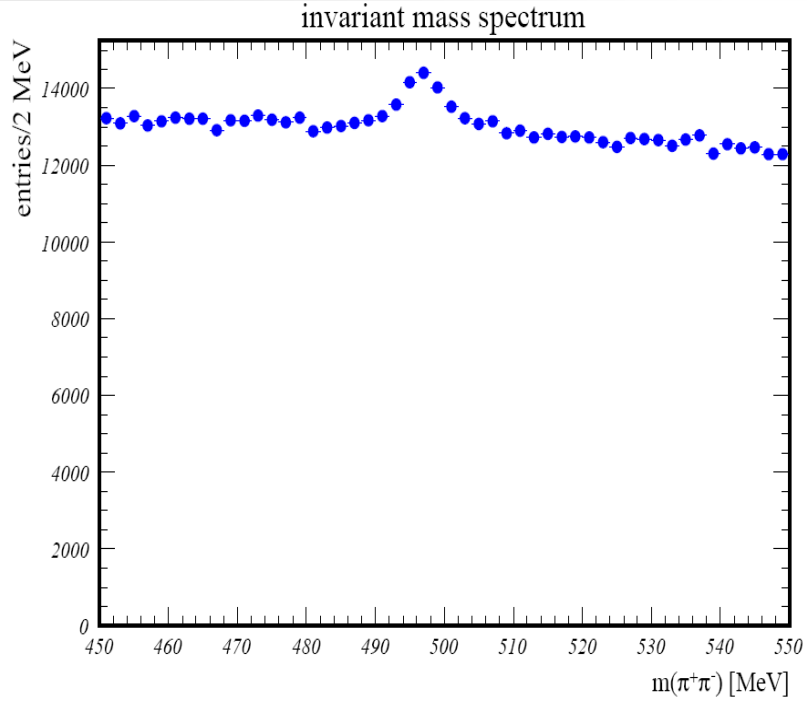


More Visualisation

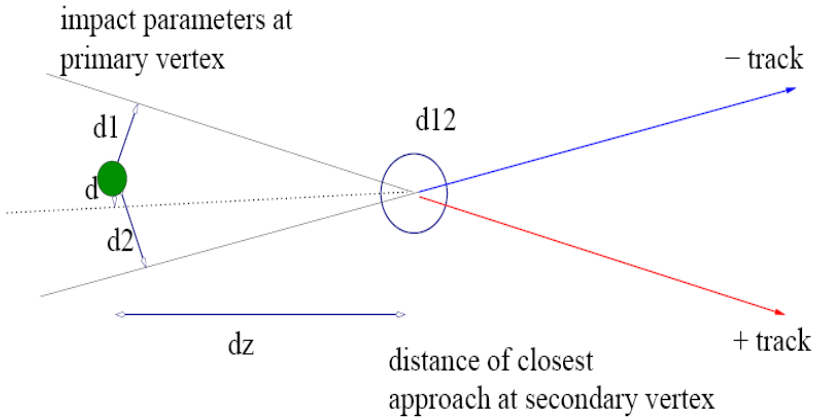
(following example taken from M.Schmelling)

A simple K-short selection:

- consider all opposite tracks with 450GeV < $M_{\pi\text{-}\pi+}$ 550GeV
- reconstruct secondary vertex for thoses tracks
- require the secondary vertex to lie "downstream" $Z_{svtx} - Z_{pvtx} > 0$



- d_1 : impact parameter of the positive track at the primary vertex
- d_2 : impact parameter of the negative track at the primary vertex
- d : impact parameter of the K_s at the primary vertex
- d_{12} : distance of closest approach between the tracks
- dz : z-distanve between secondary and primary vertex



More Visualisation

(following example taken from M.Schmelling)

➜ criteria

- informative: good separation between signal and background
- well behaved: no "singularities" in the PDFs
  - ➜ avoid finetuning in placement of cuts
- independent: every variable contributes new information

❖ variables used in the following

A measure for impact parameter reduction

$$v_1 = \log_{10} \frac{d_1 d_2}{d}$$

Quality of the secondary vertex

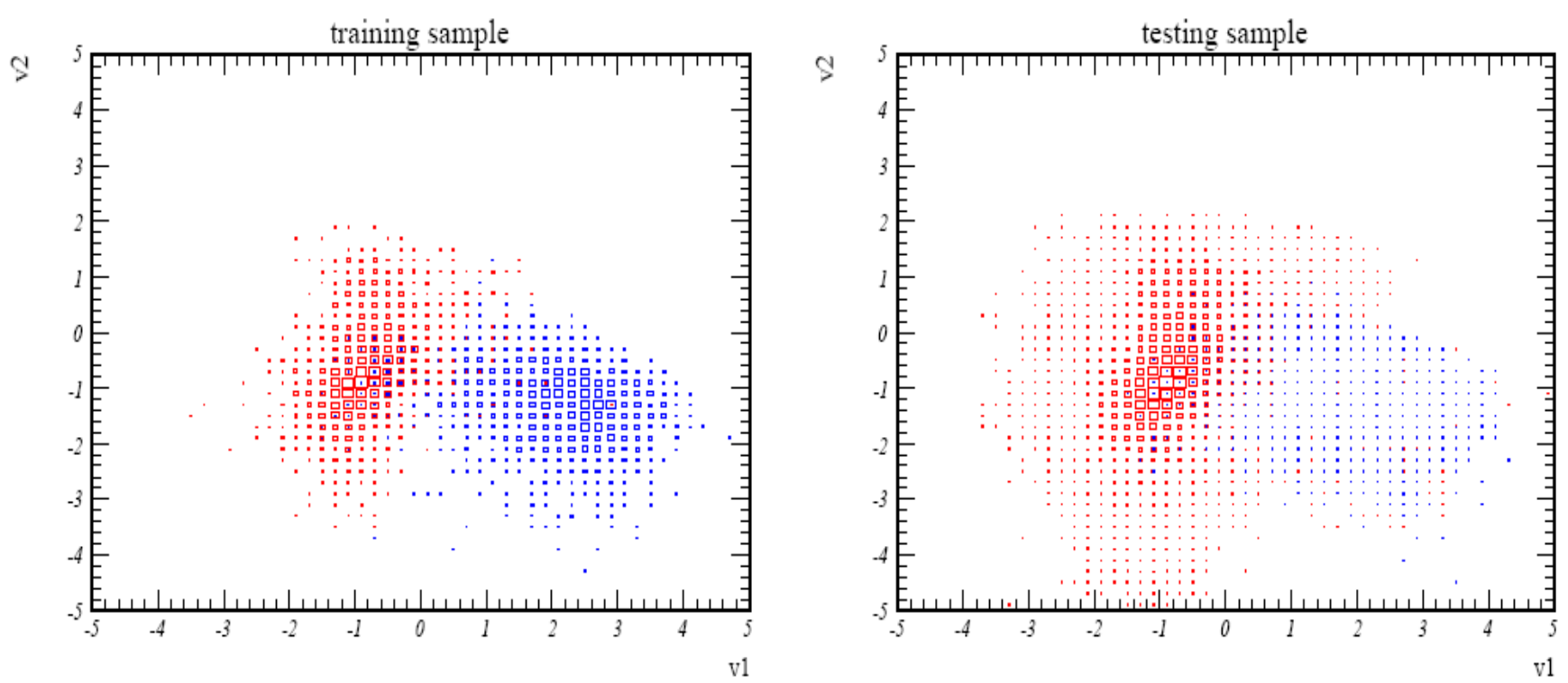
$$v_2 = \log_{10}(d_{12})$$

Lifetime of the particle (correlated with d_1 and d_2, hence taken 3rd)

$$v_3 = \log_{10}(dz)$$

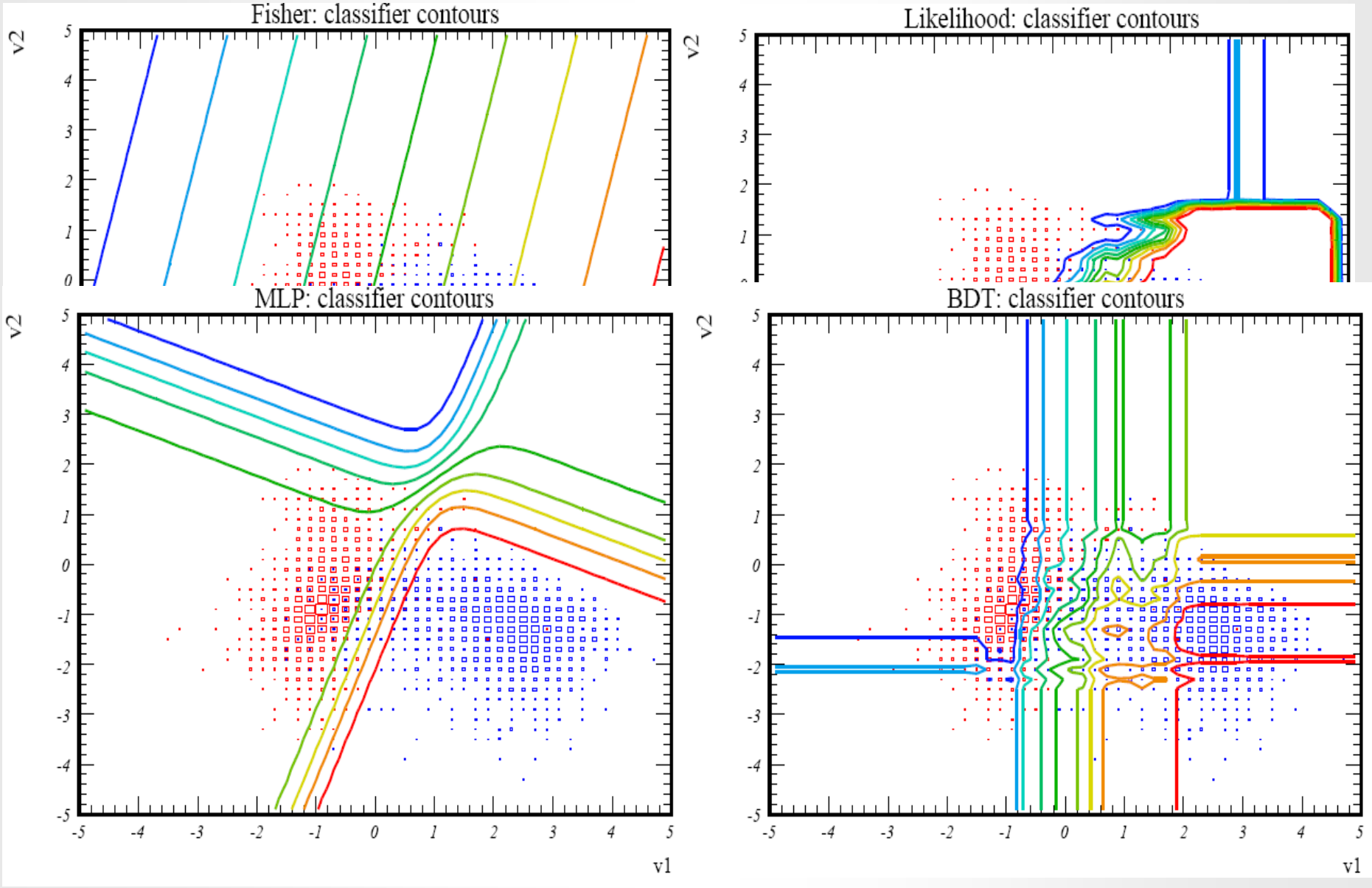
More Visualisation

(following example taken from M.Schmelling)

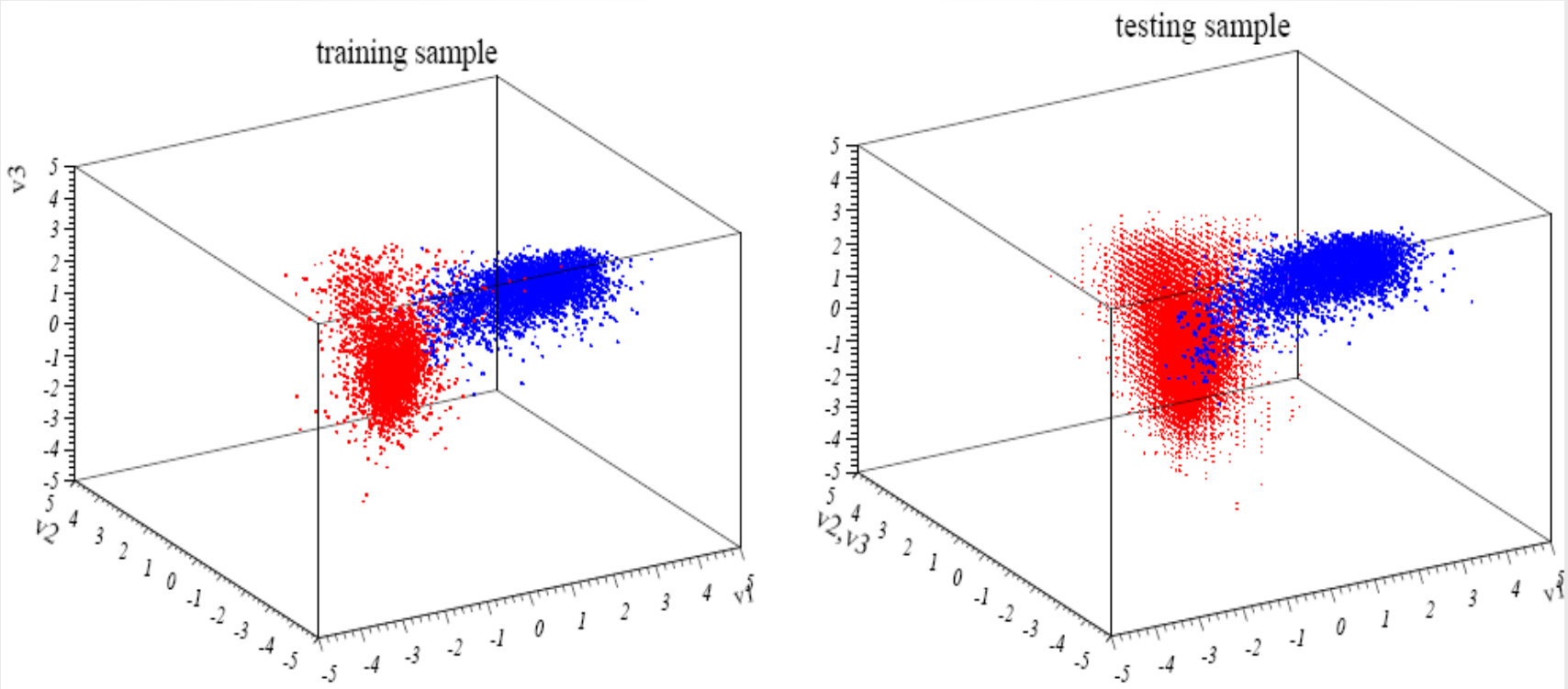


- some number of signal and background candidates in training
  - ➜ same statistical power to estimate PDFs
- imbalanced samples in testing
  - ➜ gauge performance under realistic conditions

Visualisation of Decision Boundary

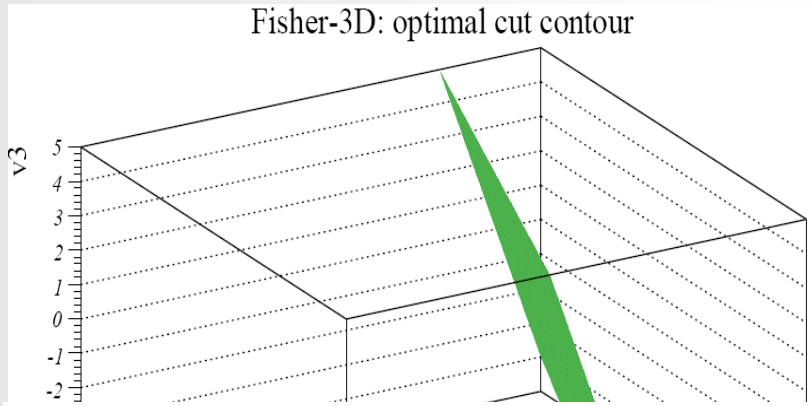


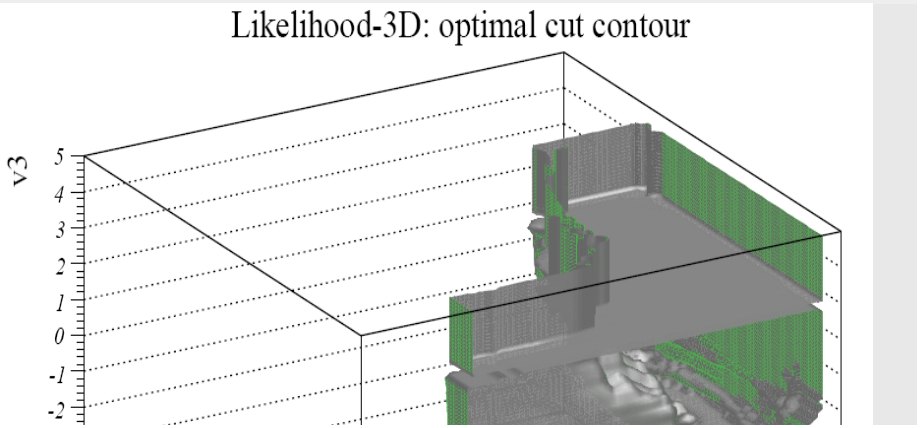
Visualisation in 3 Variables

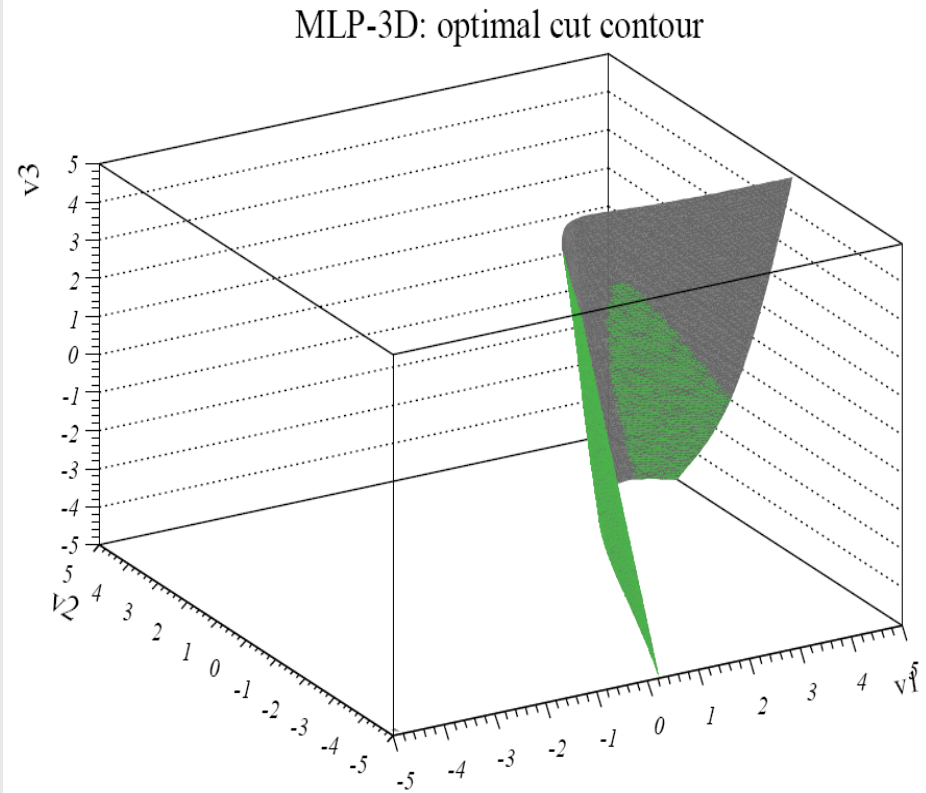


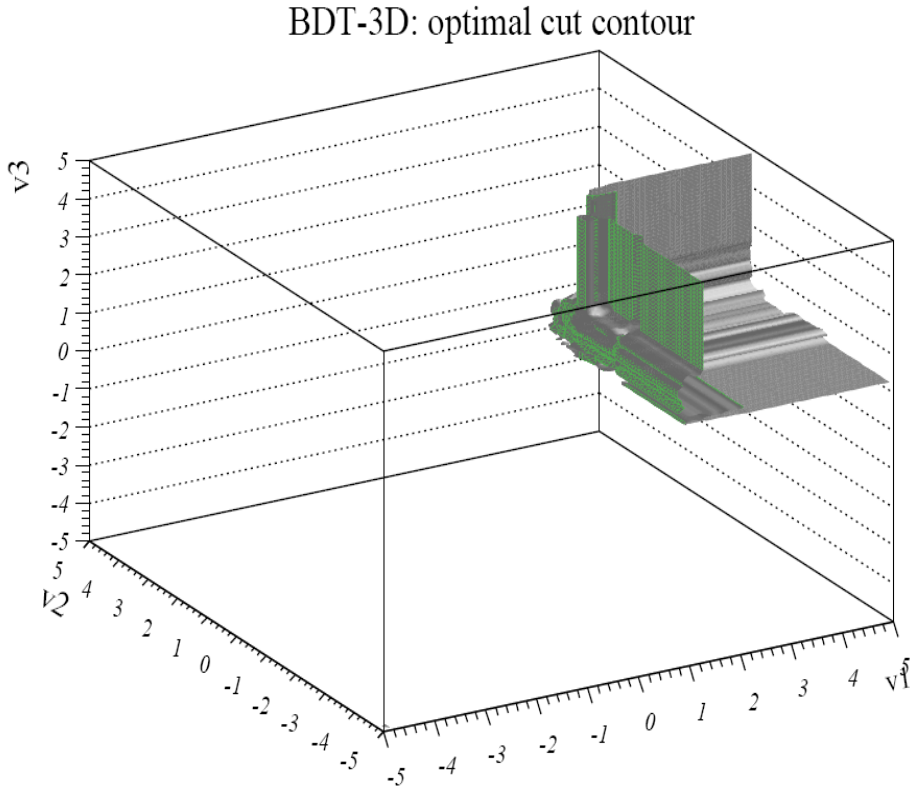
- some number of signal and background candidates in training
  - ➜ same statistical power to estimate PDFs
- imbalanced samples in testing
  - ➜ gauge performance under realistic conditions

Visualisation of Decision Boundary









General Advice for (MVA) Analyses

- There is no magic in MVA-Methods:
  - no need to be too afraid of “black boxes”
  - you typically still need to make careful tuning and do some “hard work”
- The most important thing at the start is finding good observables
  - good separation power between S and B
  - little correlations amongst each other
  - no correlation with the parameters you try to measure in your signal sample!
- Think also about possible combination of variables
  - this may allow you to eliminate correlations
    - rem.: all intelligence that you already put into the system is MUCH better than what the machine will do
- Always apply straightforward preselection cuts and let the MVA only do the rest.
- “Sharp features should be avoided” → numerical problems, loss of information when binning is applied
  - simple variable transformations (i.e. log(variable)) can often smooth out these areas and allow signal and background differences to appear in a clearer way
- Treat regions in the detector that have different features “independent”
  - can introduce correlations where otherwise the variables would be uncorrelated!

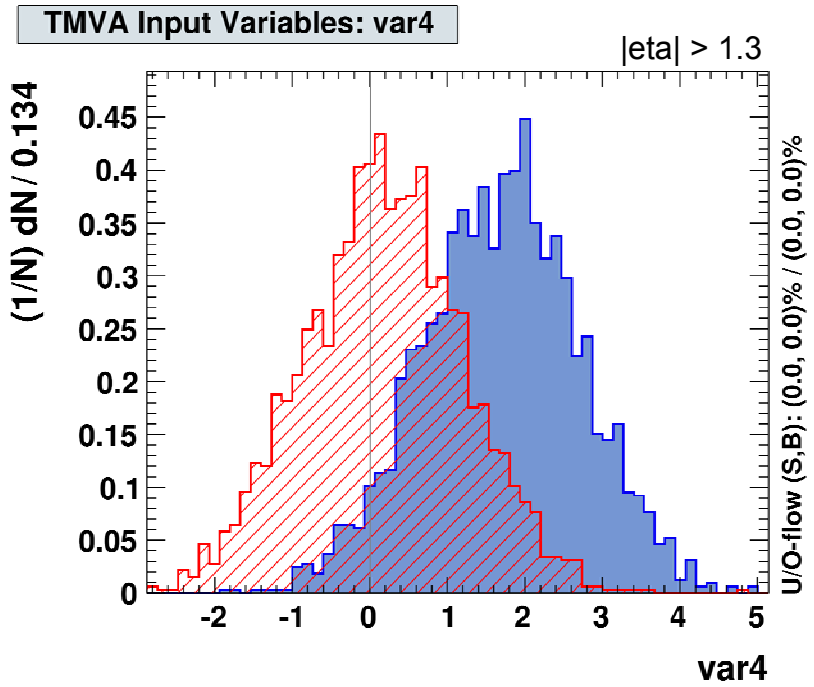
“Categorising” Classifiers

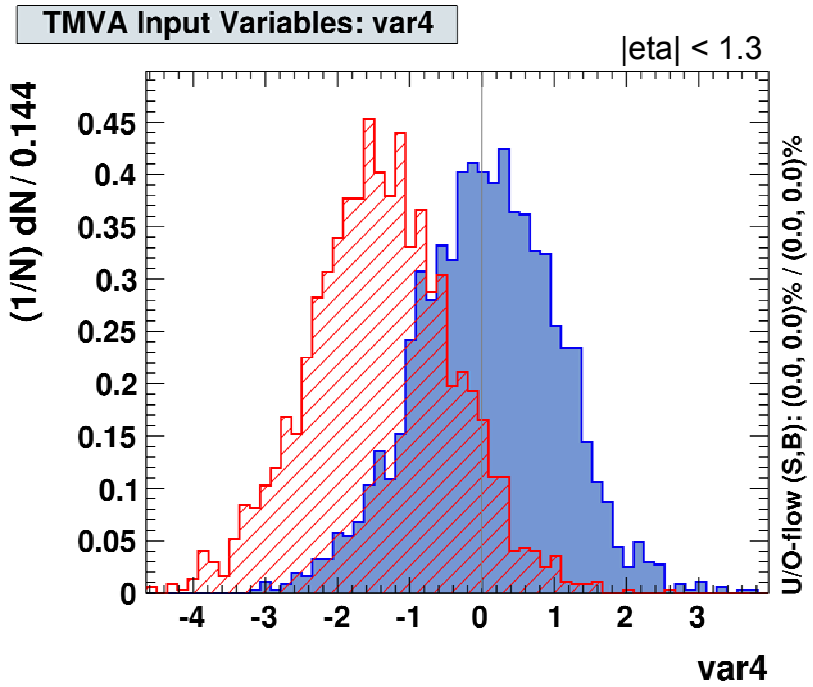
- Multivariate training samples often have distinct sub-populations of data
  - A detector element may only exist in the barrel, but not in the endcaps
  - A variable may have different distributions in barrel, overlap, endcap regions
- Ignoring this dependence creates correlations between variables, which must be learned by the classifier
  - Classifiers such as the projective likelihood, which do not account for correlations, significantly loose performance if the sub-populations are not separated
- Categorisation means splitting the data sample into categories defining disjoint data samples with the following (idealised) properties:
  - Events belonging to the same category are statistically indistinguishable
  - Events belonging to different categories have different properties
- In TMVA: All categories are treated independently for training and application (transparent for user), but evaluation is done for the whole data sample

“Categorising” Classifiers

- Let’s try our standard example of 4 Gaussian-distributed input variables:
  - Now, “var4” depends on a new variable “eta” (which may not be used for classification)

→ for |eta| > 1.3 the Signal and Background Gaussian means are shifted w.r.t. |eta| < 1.3

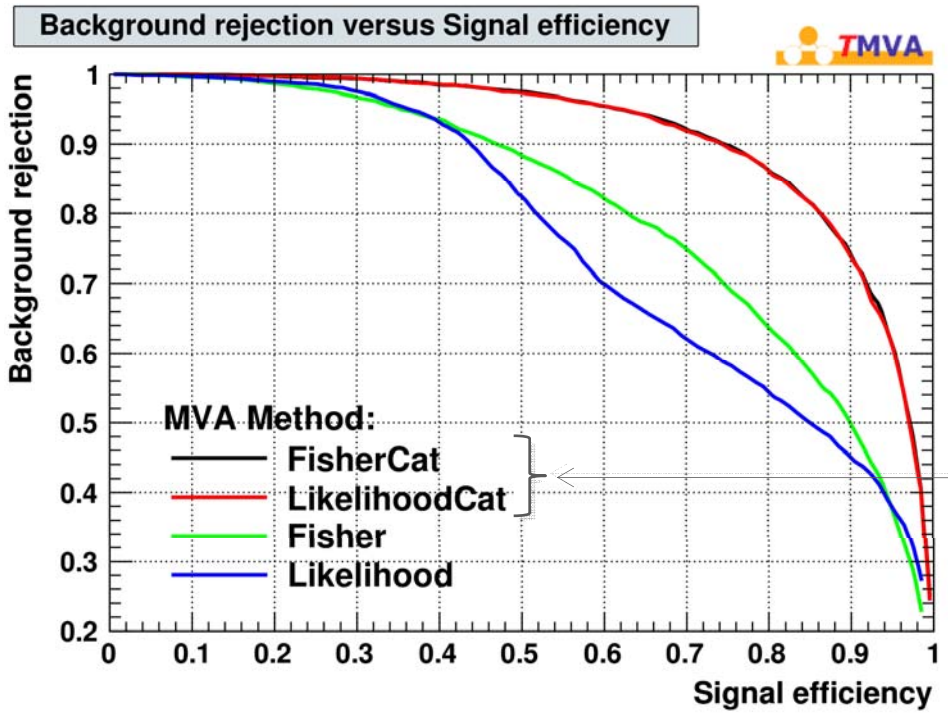




“Categorising” Classifiers

- Let’s try our standard example of 4 Gaussian-distributed input variables:
  - Now, “var4” depends on a new variable “eta” (which may not be used for classification)

→ for |eta| > 1.3 the Signal and Background Gaussian means are shifted w.r.t. |eta| < 1.3

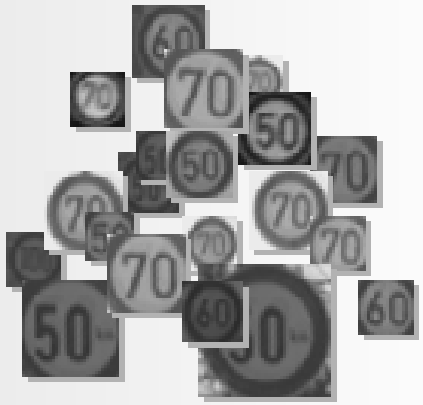


Recover optimal performance after splitting into categories

The category technique is heavily used in multivariate likelihood fits, eg, RooFit (RooSimultaneousPdf)

Multi-Class Classification

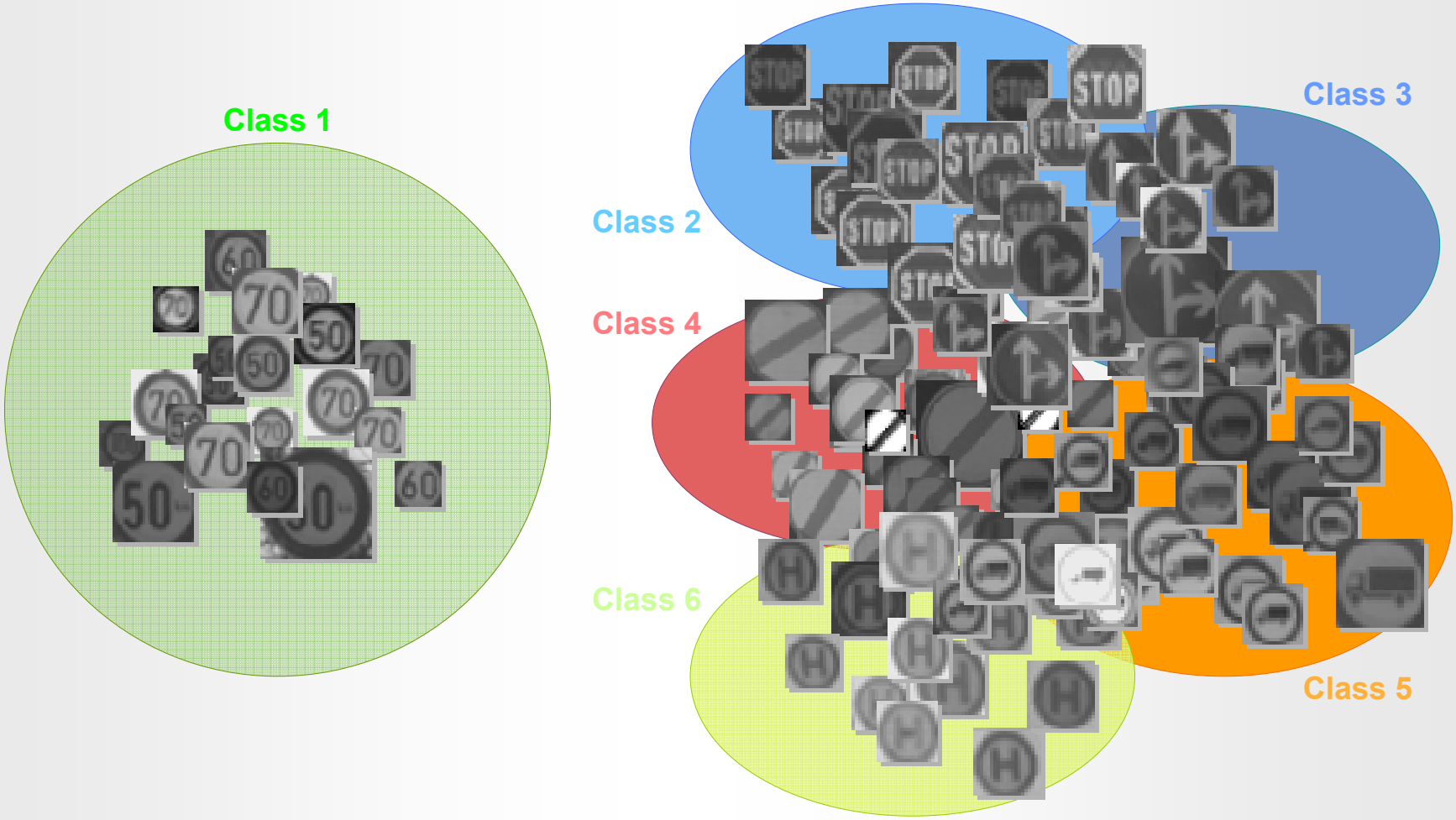
Signal



Background

Binary classification: two classes, “signal” and “background”

Multi-Class Classification



Multi-class classification – natural extension for many classifiers

Systematic Errors/Uncertainties

Some Words about Systematic Errors

- Typical worries are:
  - What happens if the estimated "Probability Density" is wrong ?
  - Can the Classifier, i.e. the discrimination function y(x), introduce systematic uncertainties?
  - What happens if the training data do not match "reality"

→Any wrong PDF leads to imperfect discrimination function $y(x) = \frac{P(x|S)}{P(x|B)}$

→Imperfect (calling it "wrong" isn't "right") y(x) → loss of discrimination power

that's all!

→classical cuts face exactly the same problem, however:

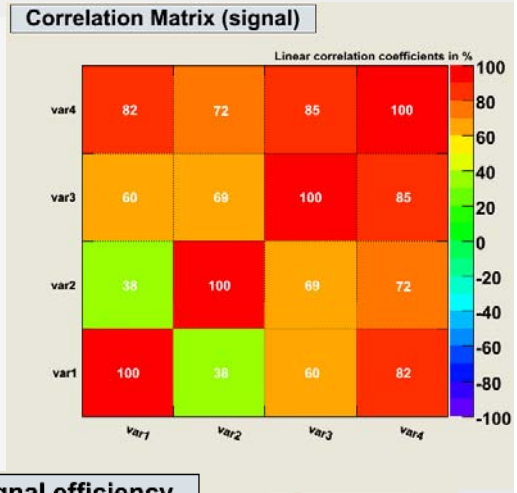
in addition to cutting on features that are not correct, now you can also "exploit" correlations that are in fact not correct

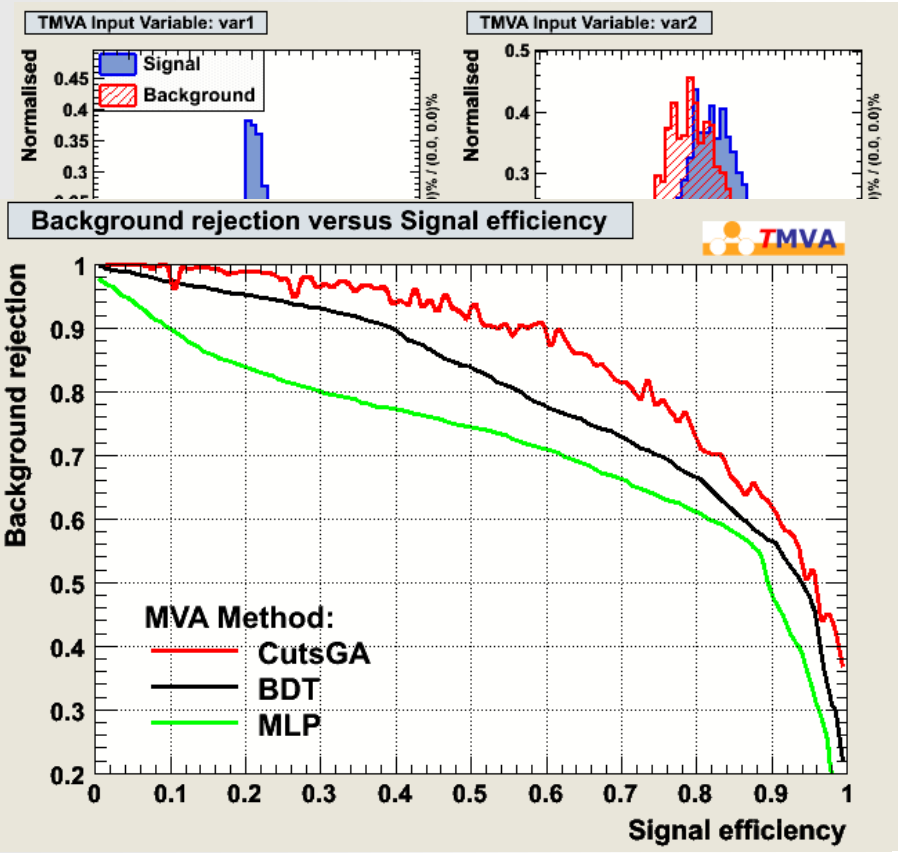
- Systematic error are only introduced once "Monte Carlo events" with imperfect modeling are used for
  - efficiency; purity
  - #expected events
- same problem with classical "cut" analysis
- use control samples to test MVA-output distribution (y(x))

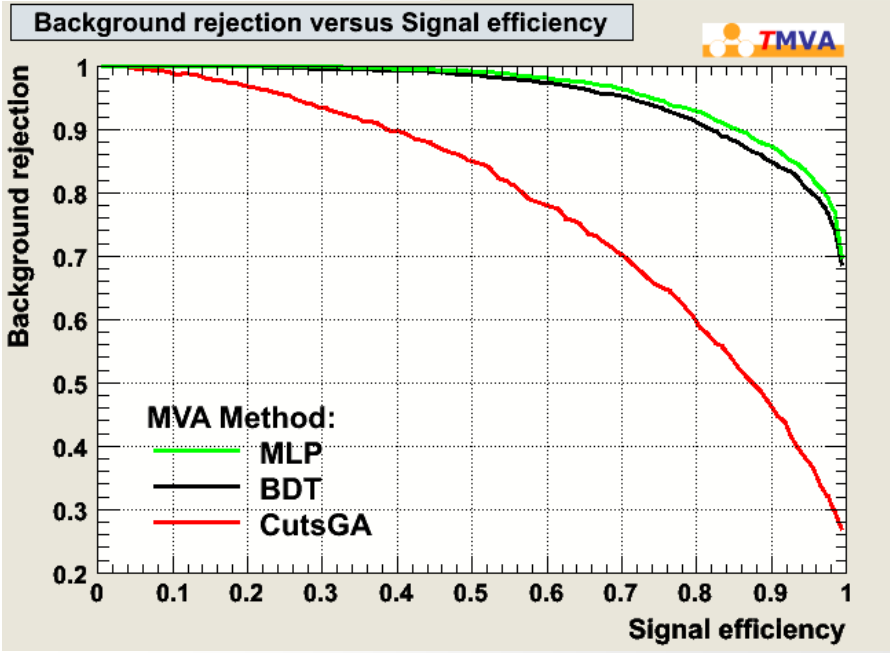
- Combined variable (MVA-output, y(x)) might "hide" problems in ONE individual variable more than if looked at alone → train classifier with few variables only and compare with data

Systematic “Error” in Correlations

- Use as training sample events that have correlatetions
  - optimize CUTs
  - train an propper MVA (e.g. Likelihood, BDT)



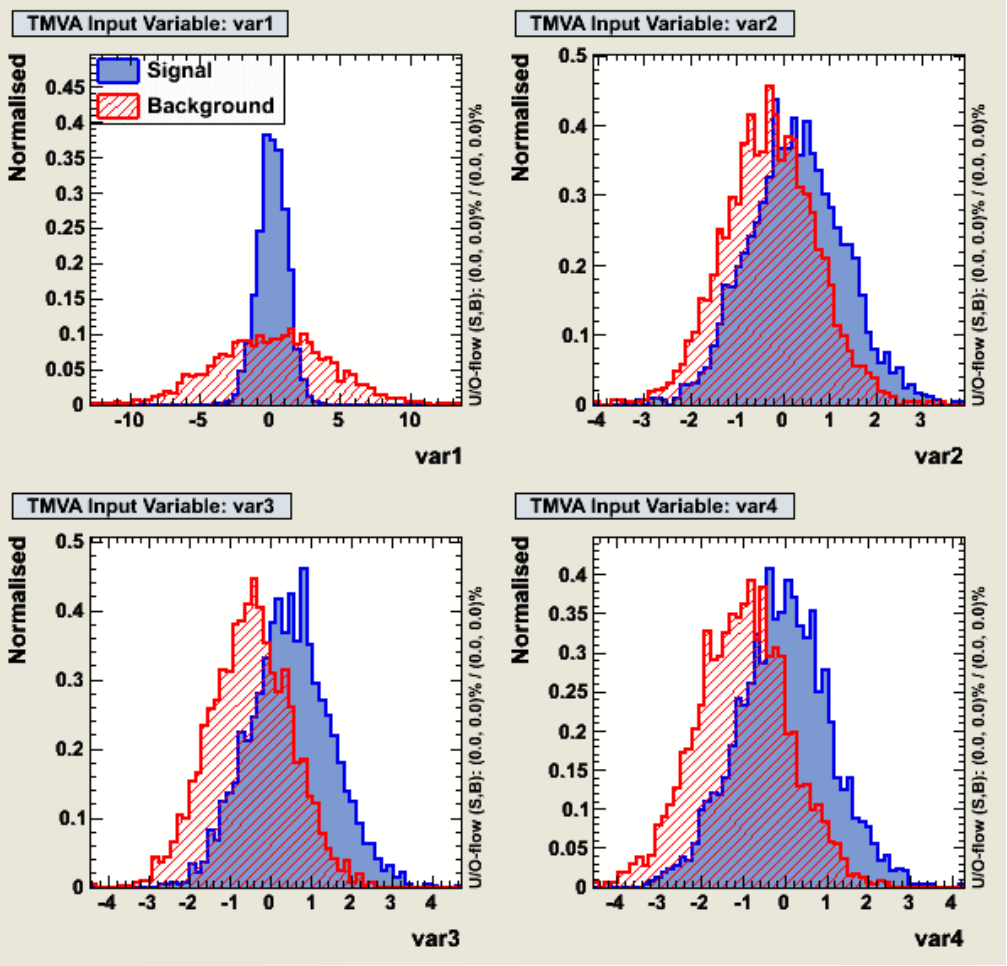


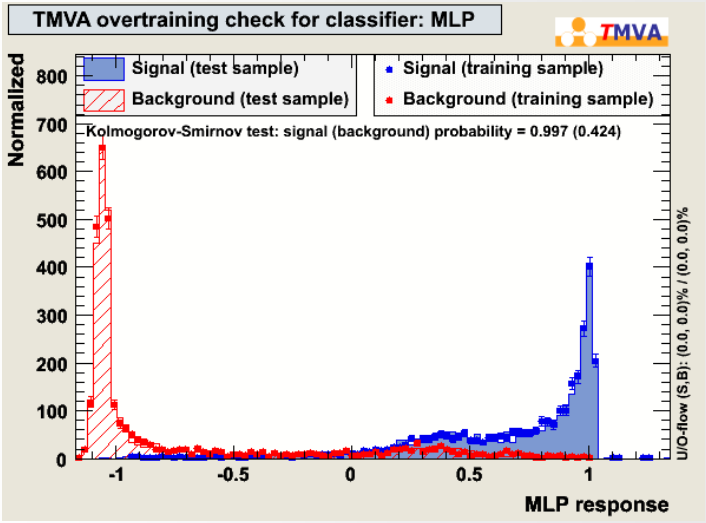


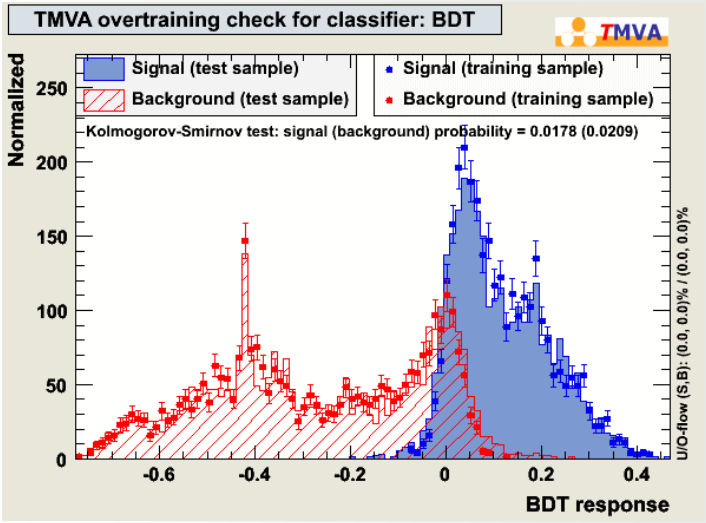
- Assume in “real data” there are NO correlations → SEE what happens!!

Systematic "Error" in Correlations

- Compare "Data" (TestSample) and Monte-Carlo (both taken from the same underlying distribution)

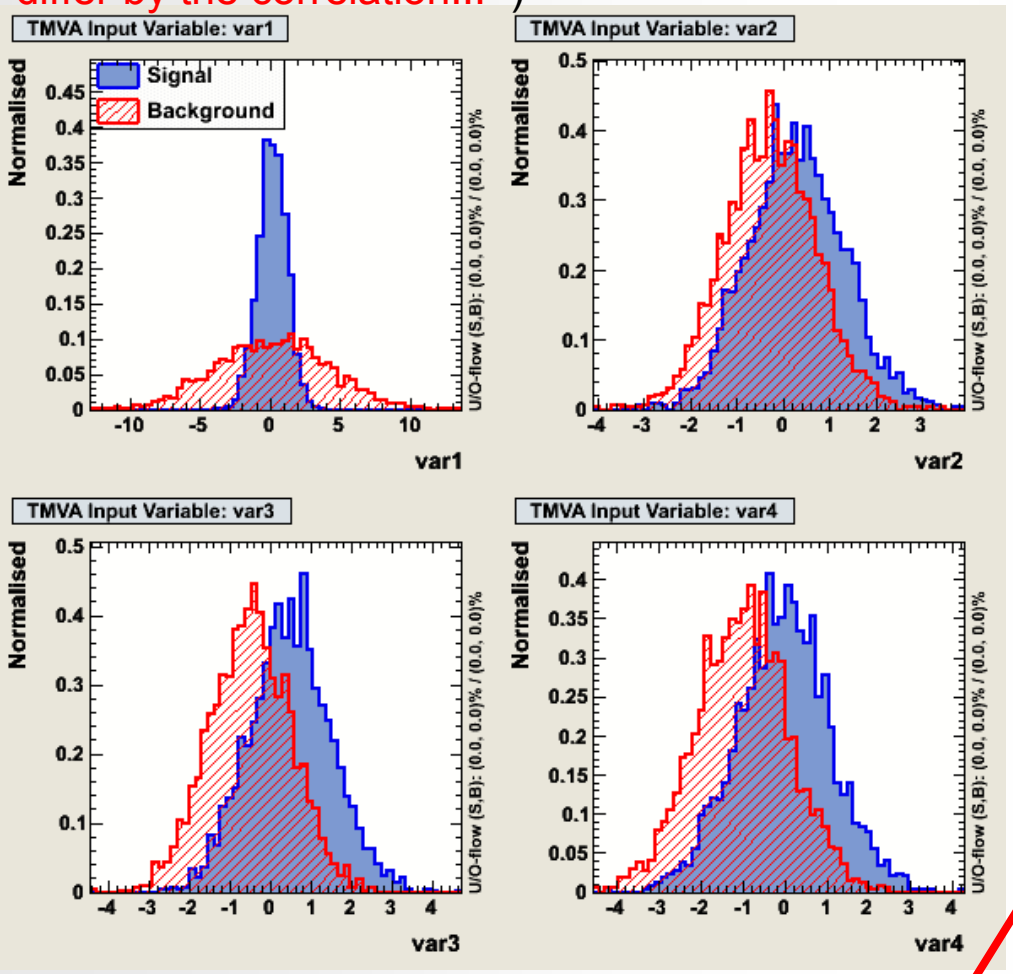


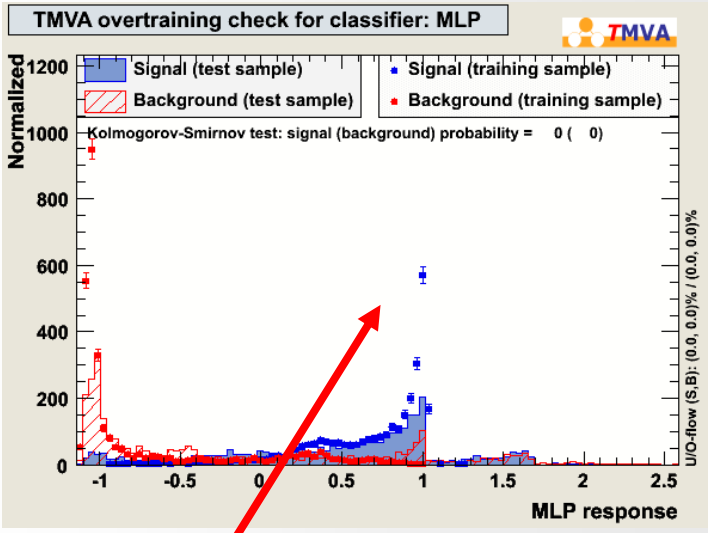


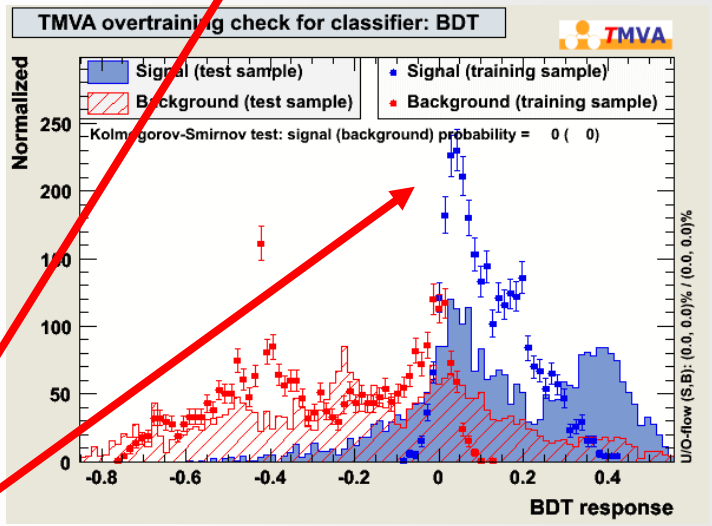


Systematic “Error” in Correlations

- Compare “Data” (TestSample) and Monte-Carlo (both taken from the same underlying distributions that differ by the correlation!!!)







Differences are ONLY visible in the MVA-output plots (and if you’d look at cut sequences….)

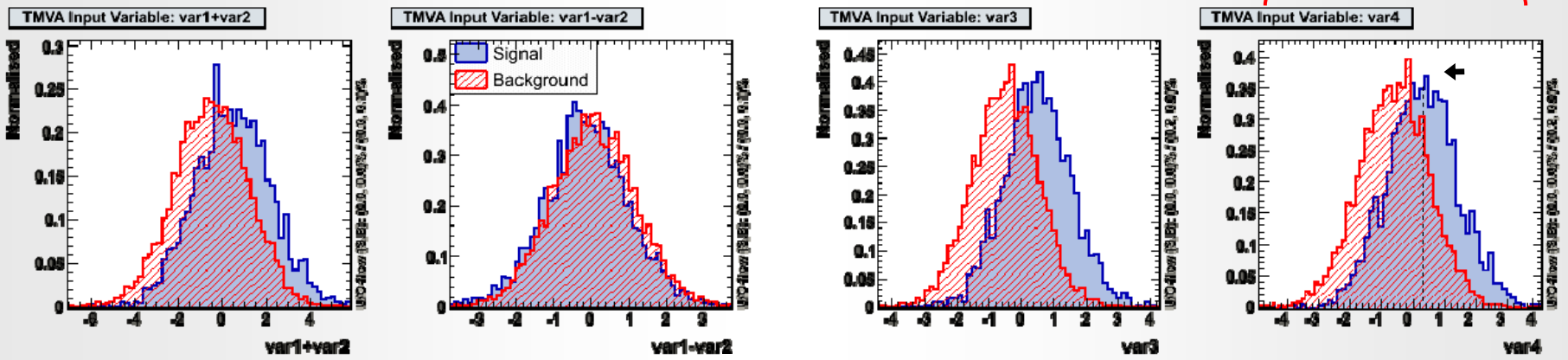
Treatment of Systematic Uncertainties

- Is there a strategy however to become 'less sensitive' to possible systematic uncertainties
  - i.e. classically: variable that is prone to uncertainties → do not cut in the region of steepest gradient
  - classically one would not choose the most important cut on an uncertain variable

- Try to make classifier less sensitive to "uncertain variables"
  - i.e. re-weight events in training to decrease separation in variables with large systematic uncertainty

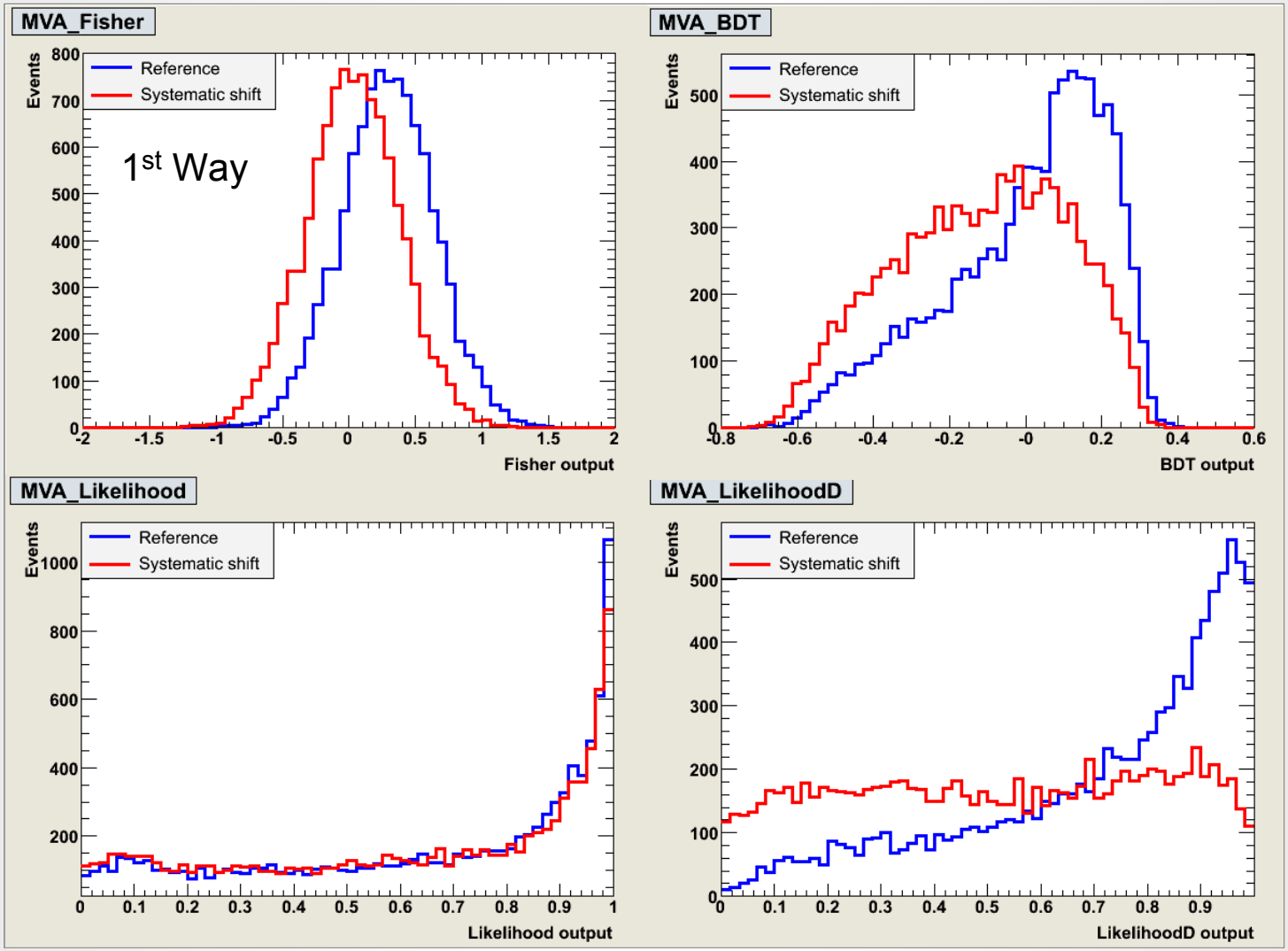
(certainly not yet a recipe that can strictly be followed, more an idea of what could perhaps be done)

"Calibration uncertainty" may shift the central value and hence worsen (or increase) the discrimination power of "var4"



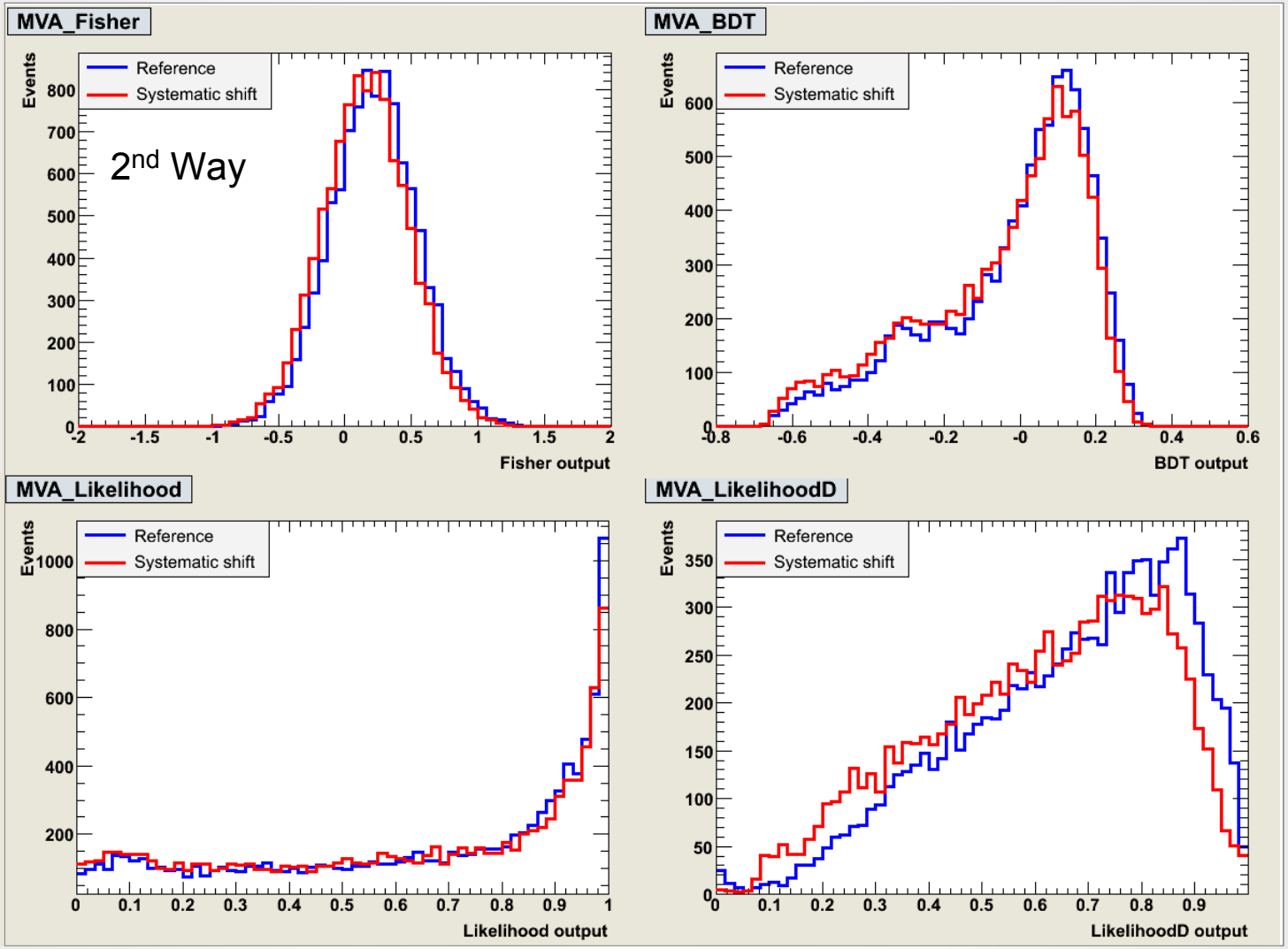
Treatment of Systematic Uncertainties

Classifier output distributions for signal only



Treatment of Systematic Uncertainties

Classifier output distributions for signal only



What is *T*MVA

- One framework for "all" MVA-techniques, available in ROOT
  - Have a common platform/interface for all MVA classification and regression-methods:
  - Have common data pre-processing capabilities
  - Train and test all classifiers on same data sample and evaluate consistently
  - Provide common analysis (ROOT scripts) and application framework
  - Provide access with and without ROOT, through macros, C++ executables or python
- *T*MVA is a sourceforge (SF) package for world-wide access
  - Home page ..... http://tmva.sf.net/
  - SF project page ..... http://sf.net/projects/tmva
  - Mailing list ..... http://sf.net/mail/?group_id=152074
  - Tutorial TWiki ..... https://twiki.cern.ch/twiki/bin/view/TMVA/WebHome
- Integrated and distributed with ROOT since ROOT v5.11/03
  - now fully developed within ROOT SVN → same release cycles

TMVA Content

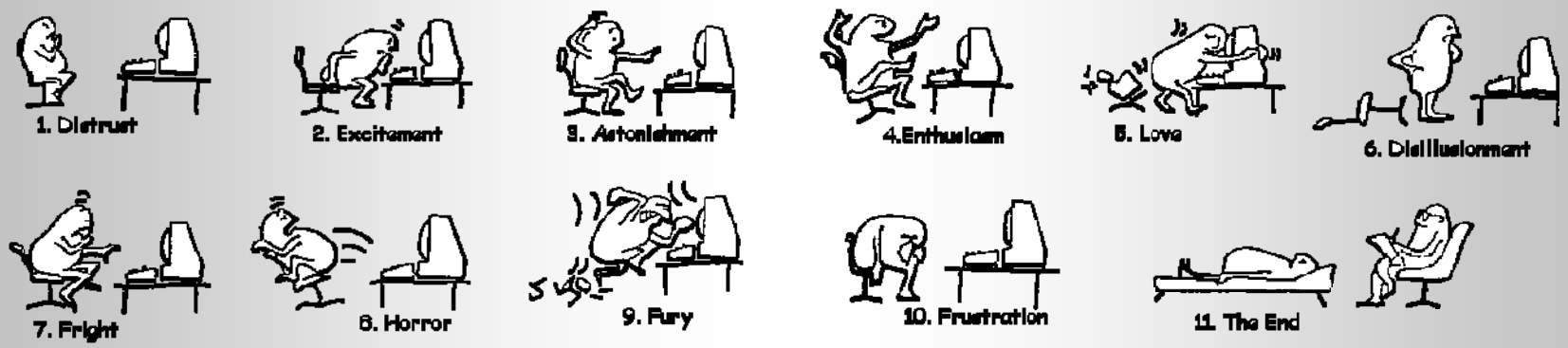
- Currently implemented classifiers and regression methods
  - Rectangular cut optimisation
  - Projective and multidimensional likelihood estimator (incl. regression)
  - k-Nearest Neighbor algorithm (incl. regression)
  - Fisher and H-Matrix discriminants
  - Function discriminant
  - Artificial neural networks (3 *multilayer perceptron* implementations) (incl. regression)
  - Boosted/bagged decision trees (incl. regression)
  - Rule Fitting
  - Support Vector Machine (incl. regression)
- Currently implemented data preprocessing stages:
  - Decorrelation, Principal Value Decomposition, “Gaussianisation”
- Examples for combination methods:
  - Boosting, Categorisation, MVA Committees

Using *T*MVA

A typical *T*MVA analysis consists of two main steps:

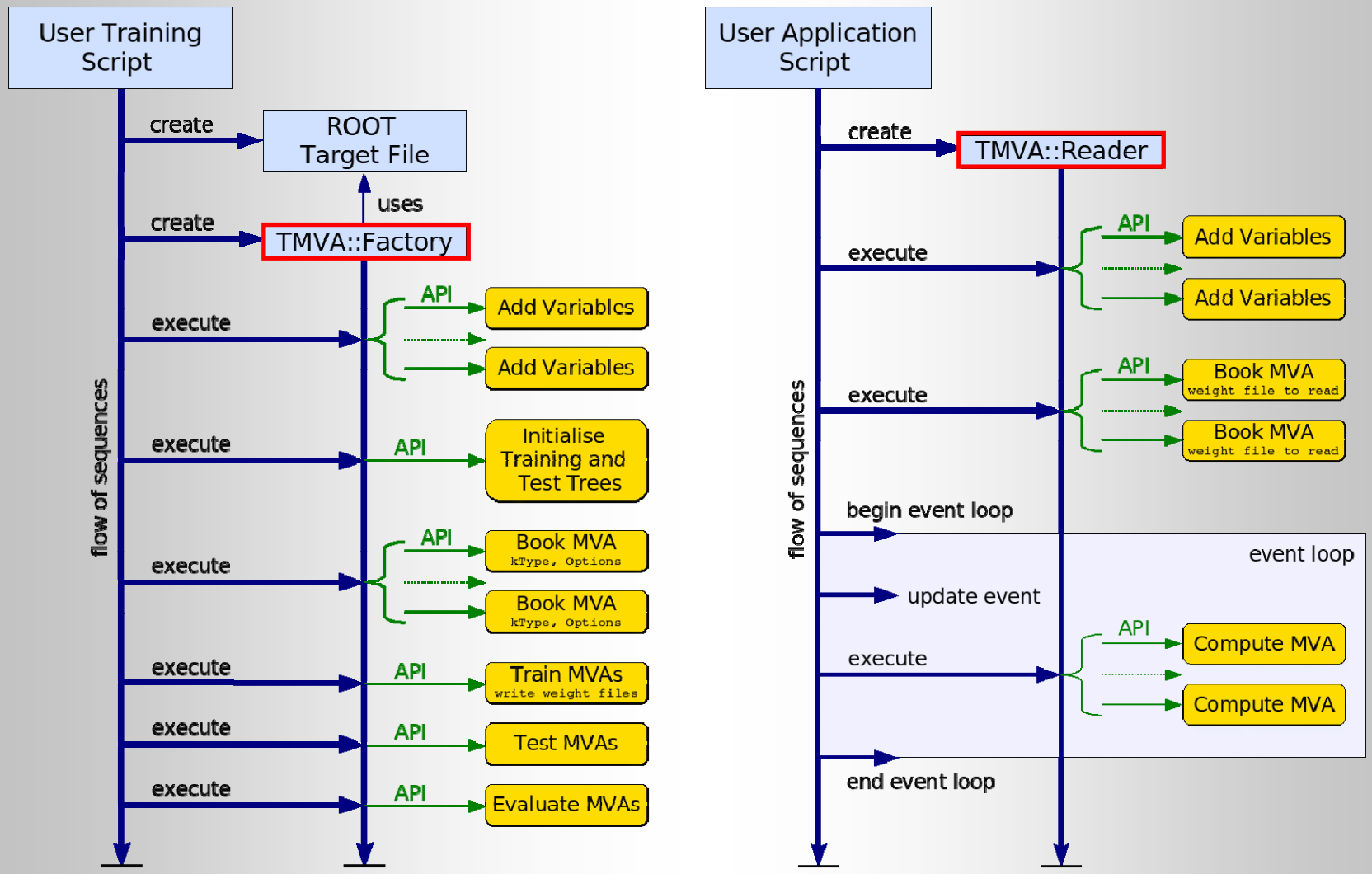
1. *Training phase*: training, testing and evaluation of classifiers using data samples with known signal and background composition
2. *Application phase*: using selected trained classifiers to classify unknown data samples

➡ Illustration of these steps with toy data samples



→ *T*MVA tutorial

Code Flow for *Training* and *Application* Phases



→ TMVA tutorial

A Simple Example for *Training*

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

   TMVA::Factory *factory = new TMVA::Factory( "MVAnalysis", outputFile,"!V");

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

   factory->AddSignalTree        ( (TTree*)input->Get("TreeS"), 1.0 );
   factory->AddBackgroundTree ( (TTree*)input->Get("TreeB"), 1.0 );

   factory->AddVariable("var1+var2", 'F');
   factory->AddVariable("var1-var2", 'F');
   factory->AddVariable("var3", 'F');
   factory->AddVariable("var4", 'F');

   factory->PrepareTrainingAndTestTree("", "NSigTrain=3000:NBkgTrain=3000:SplitMode=Random:!V" );

   factory->BookMethod( TMVA::Types::kLikelihood, "Likelihood",
                        "!V:!TransformOutput:Spline=2:NSmooth=5:NAvEvtPerBin=50" );

   factory->BookMethod( TMVA::Types::kMLP, "MLP", "!V:NCycles=200:HiddenLayers=N+1,N:TestRate=5" );

   factory->TrainAllMethods();
   factory->TestAllMethods();
   factory->EvaluateAllMethods();

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

← create *Factory*

← give training/test trees

← register input variables

← select MVA methods

← train, test and evaluate

→ TMVA tutorial

A Simple Example for an *Application*

```
void TMVClassificationApplication( )
{
   TMVA::Reader *reader = new TMVA::Reader("!Color");

   Float_t var1, var2, var3, var4;
   reader->AddVariable( "var1+var2", &var1 );
   reader->AddVariable( "var1-var2",  &var2 );
   reader->AddVariable( "var3", &var3 );
   reader->AddVariable( "var4", &var4 );

   reader->BookMVA( "MLP classifier",  "weights/MVAnalysis_MLP.weights.txt" );

   TFile *input = TFile::Open("tmva_example.root");
   TTree* theTree = (TTree*)input->Get("TreeS");

   // … set branch addresses for user TTree
   for (Long64_t ievt=3000; ievt<theTree->GetEntries();ievt++) {
      theTree->GetEntry(ievt);

      var1 = userVar1 + userVar2;
      var2 = userVar1 - userVar2;
      var3 = userVar3;
      var4 = userVar4;

      Double_t out =  reader->EvaluateMVA(  "MLP classifier"  );

      // do something with it …
   }
   delete reader;
}
```

← create *Reader*

← register the variables

← book classifier(s)

← prepare event loop

← compute input variables

← calculate classifier output

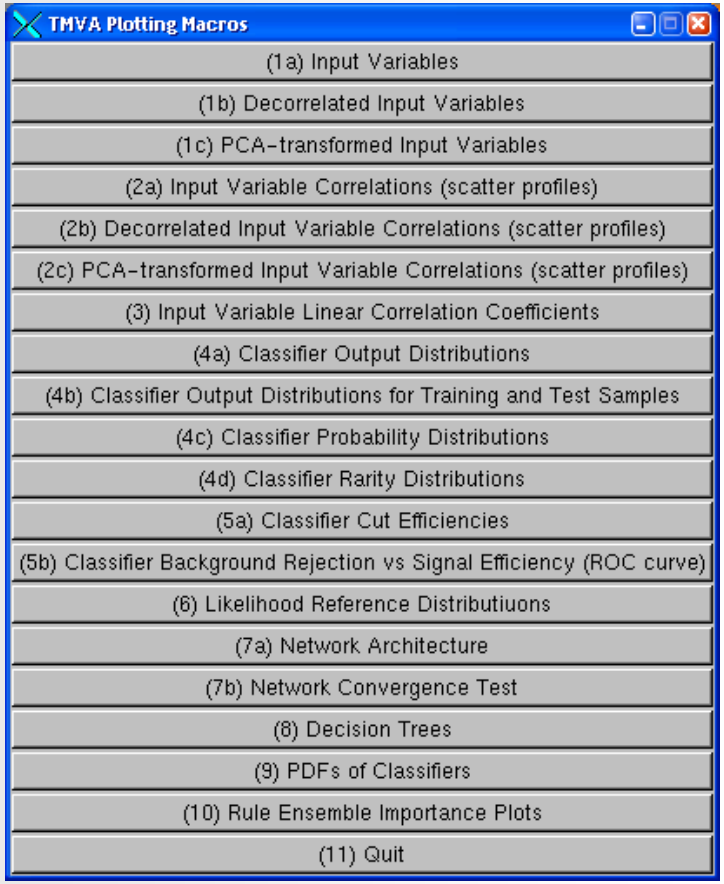
→ TMVA tutorial

Data Preparation

- Data input format: ROOT TTree or ASCII
- Selection any subset or combination or function of available variables
- Apply pre-selection cuts (possibly independent for signal and bkg)
- Define global event weights for signal or background input files
- Define individual event weight (use of any input variable present in training data)
- Choose one out of various methods for splitting into training and test samples:
  - Block wise
  - Randomly
  - Periodically (*i.e.* periodically 3 testing ev., 2 training ev., 3 testing ev, 2 training ev. ….)
  - User defined training and test trees
- Choose preprocessing of input variables (*e.g.*, decorrelation)

MVA Evaluation Framework

- TMVA is not only a collection of classifiers, but an MVA framework
- After training, TMVA provides ROOT evaluation scripts (through GUI)



Plot all signal (S) and background (B) input variables with and without pre-processing

Correlation scatters and linear coefficients for S & B

Classifier outputs (S & B) for test and training samples (spot overtraining)

Classifier *Rarity* distribution

Classifier significance with optimal cuts

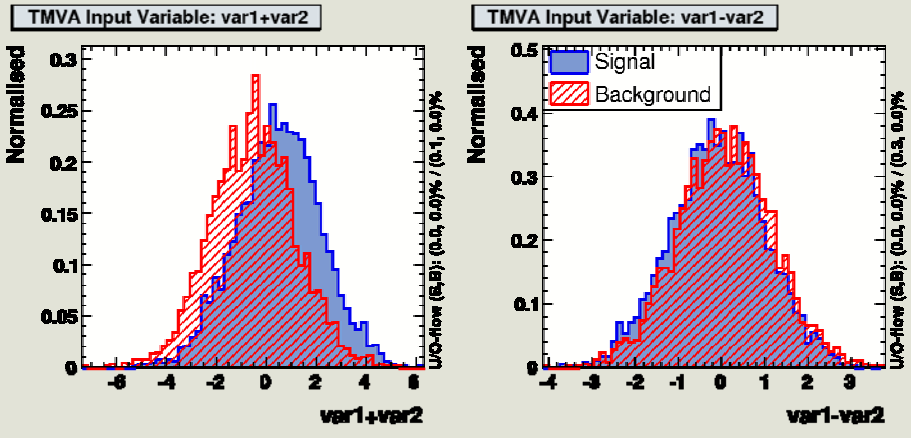
B rejection versus S efficiency

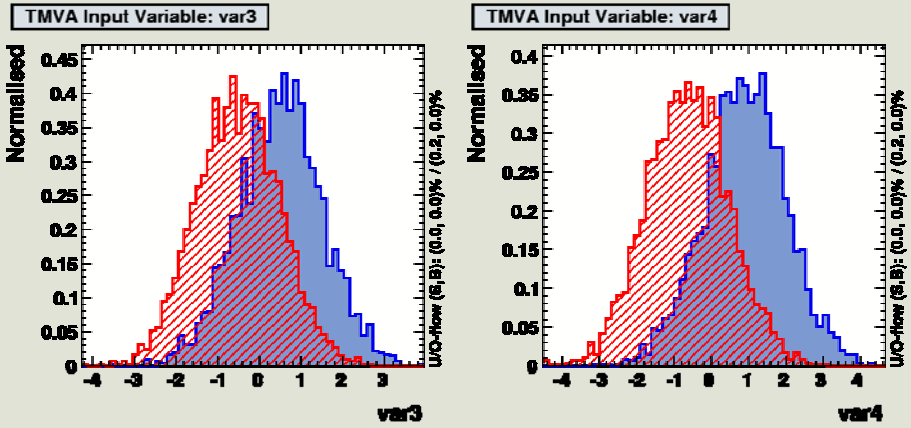
Classifier-specific plots:
- Likelihood reference distributions
- Classifier PDFs (for probability output and Rarity)
- Network architecture, weights and convergence
- Rule Fitting analysis plots

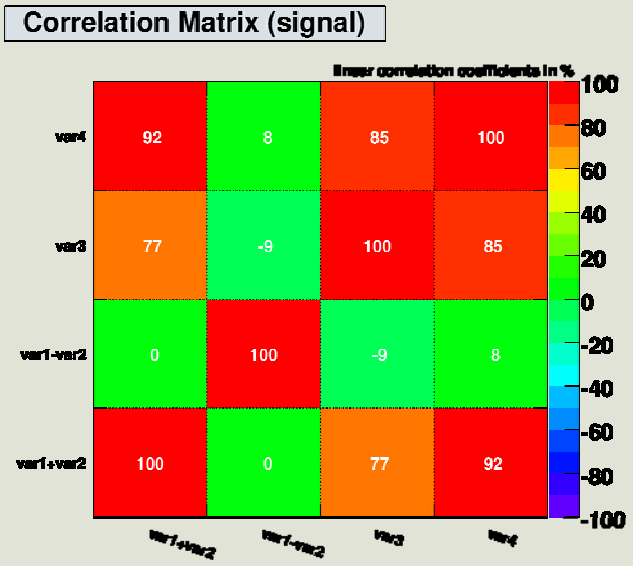
- Visualise decision trees

Example: Toy Monte Carlo

- Data set with 4 linearly correlated Gaussian distributed variables:



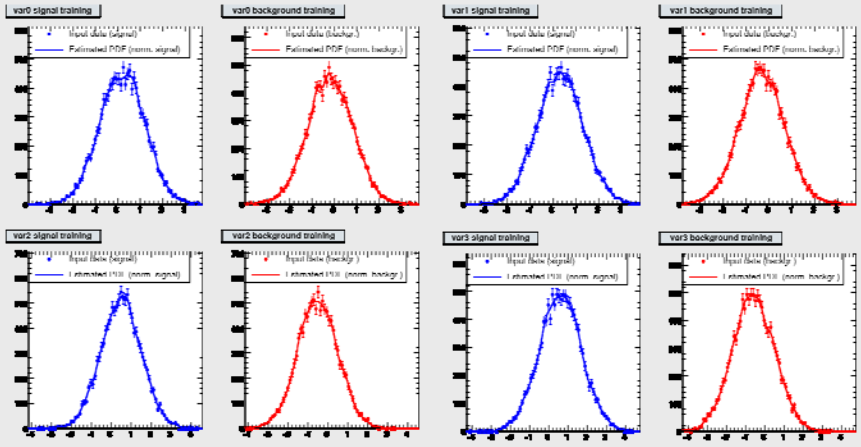


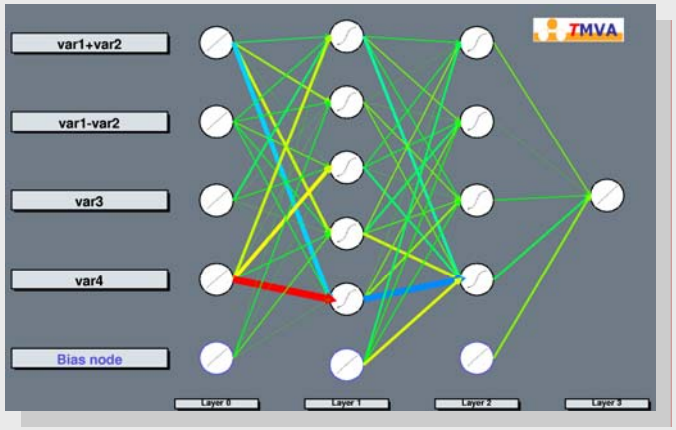


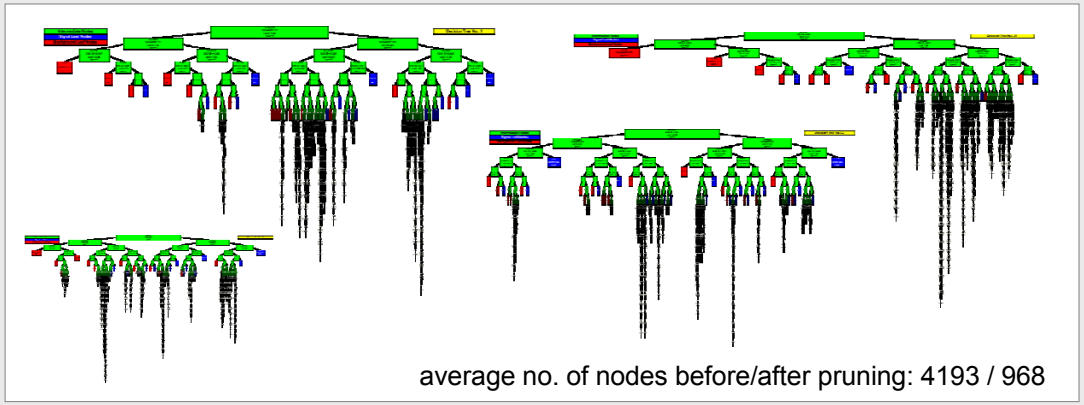
Rank	Variable	Separation
1	var4	0.606
2	var1+var2	0.182
3	var3	0.173
4	var1-var2	0.014

Evaluating the Classifier Training (I)

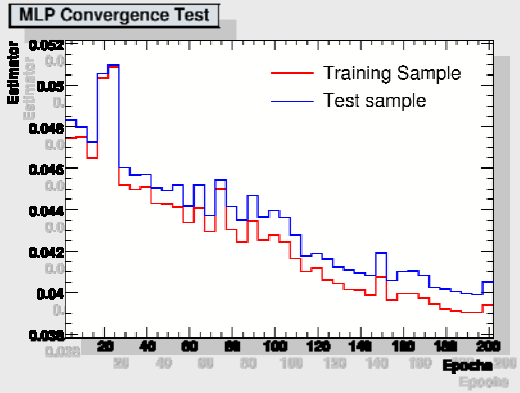
- Projective likelihood PDFs, MLP training, BDTs, …





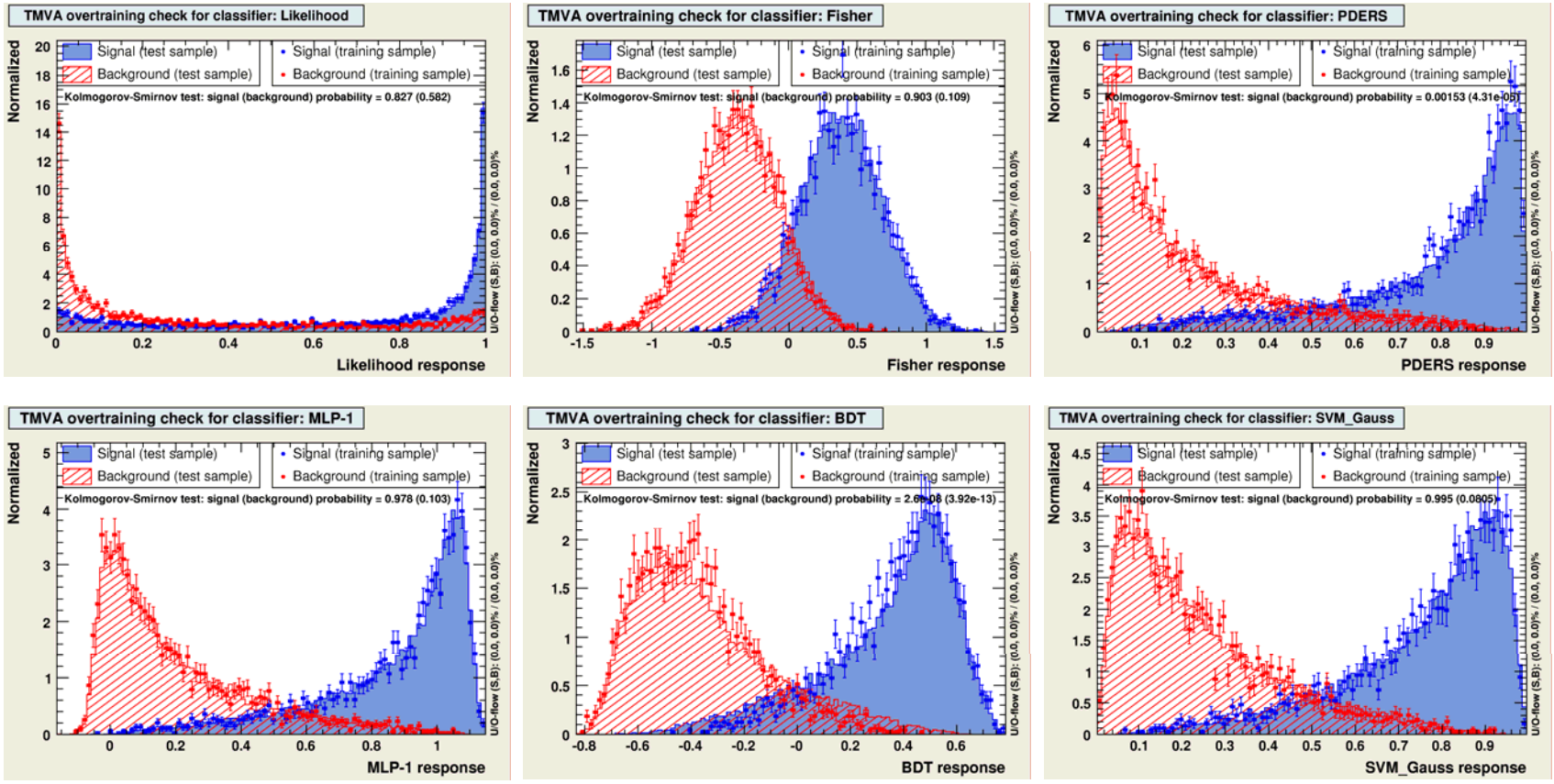


average no. of nodes before/after pruning: 4193 / 968



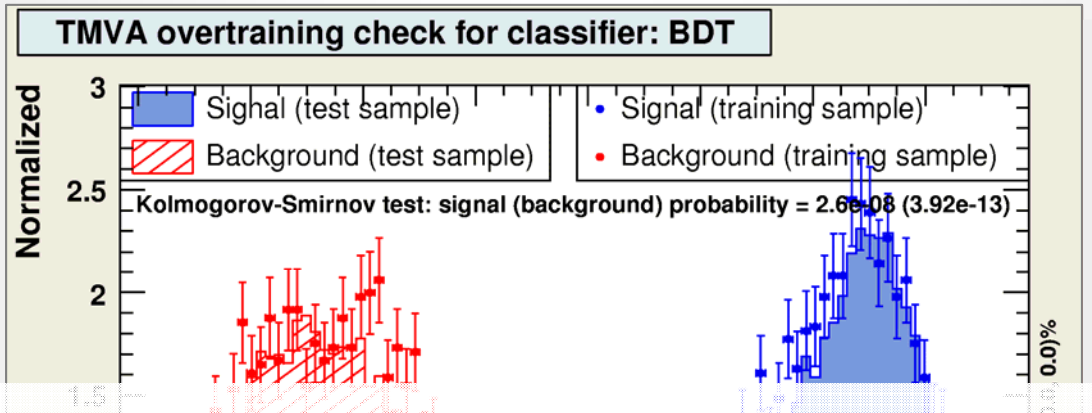
Testing the Classifiers

- Classifier output distributions for independent test sample:



Evaluating the Classifier Training

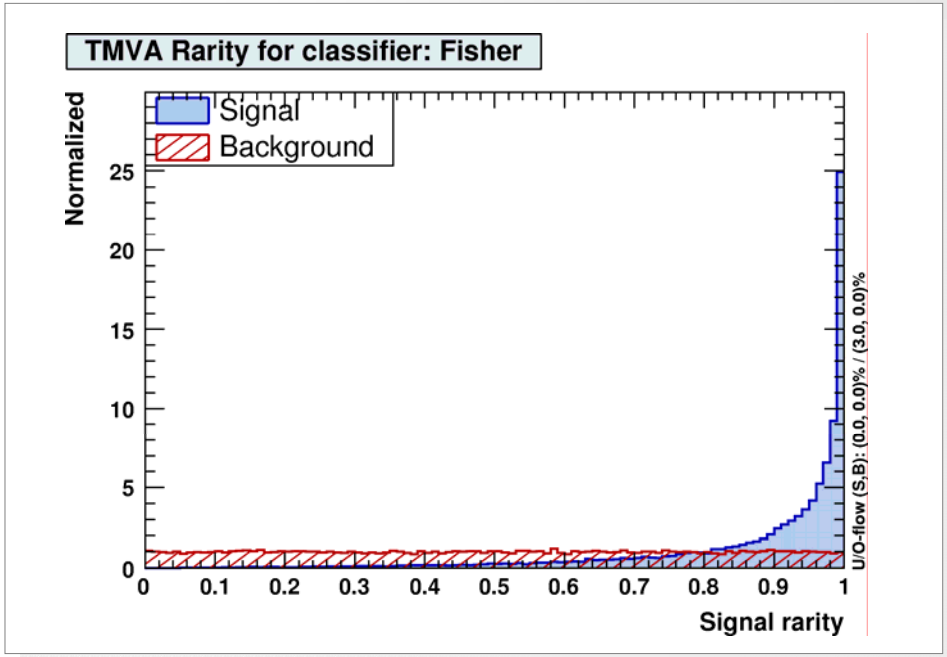
- Check for overtraining: classifier output for test *and* training samples …



- Remark on **overtraining**
  - Occurs when classifier training has too few degrees of freedom because the classifier has too many adjustable parameters for too few training events
  - Sensitivity to overtraining depends on classifier: *e.g.*, Fisher weak, BDT strong
  - Compare performance between training and test sample to detect overtraining
  - Actively counteract overtraining: *e.g.*, smooth likelihood PDFs, prune decision trees, …

Evaluating the Classifier Training (IV)

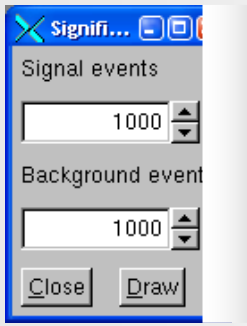
- There is no unique way to express the performance of a classifier
  → several benchmark quantities computed by *T*MVA
  - Signal eff. at various background effs. (= 1 – rejection) when cutting on classifier output
  - The *Separation*: $\frac{1}{2}\int \frac{(\hat{y}_S(y)-\hat{y}_B(y))^2}{\hat{y}_S(y)+\hat{y}_B(y)}dy$
  - "Rarity" implemented (background flat): $R(y)=\int_{-\infty}^{y}\hat{y}(y')dy'$

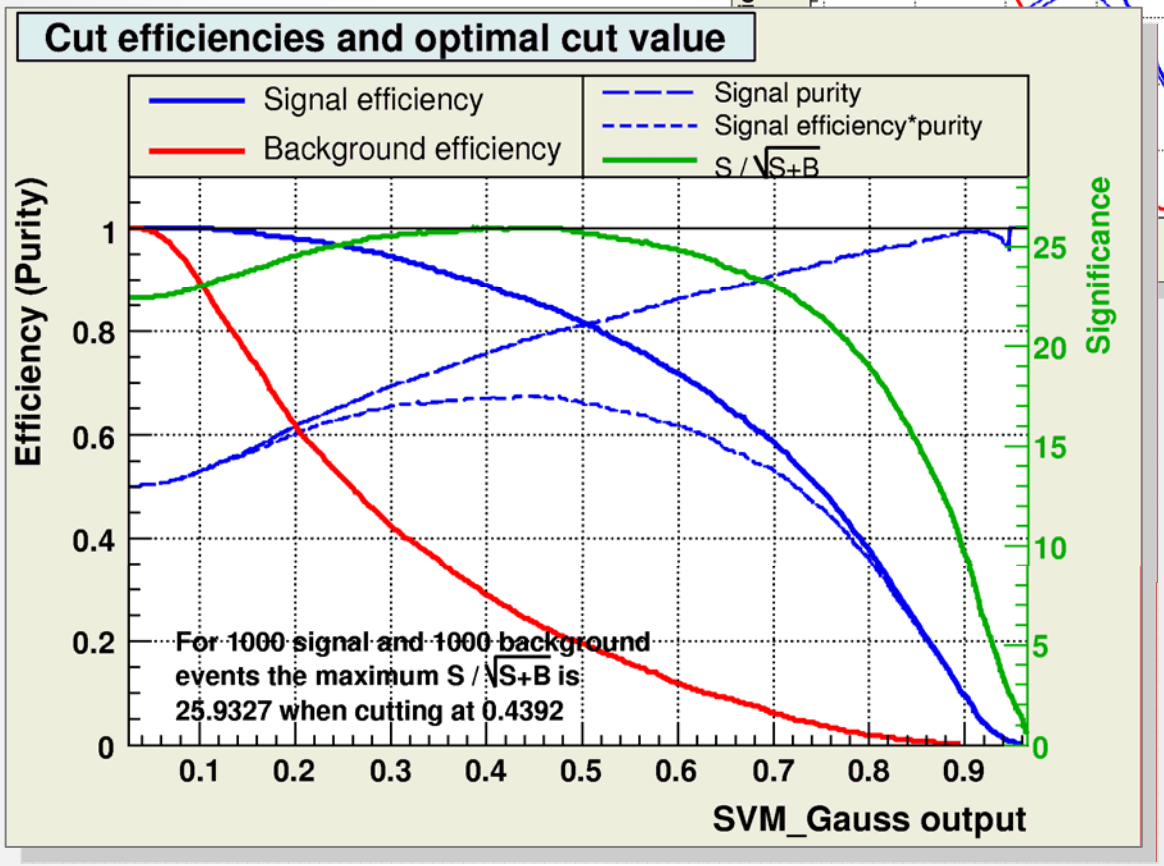


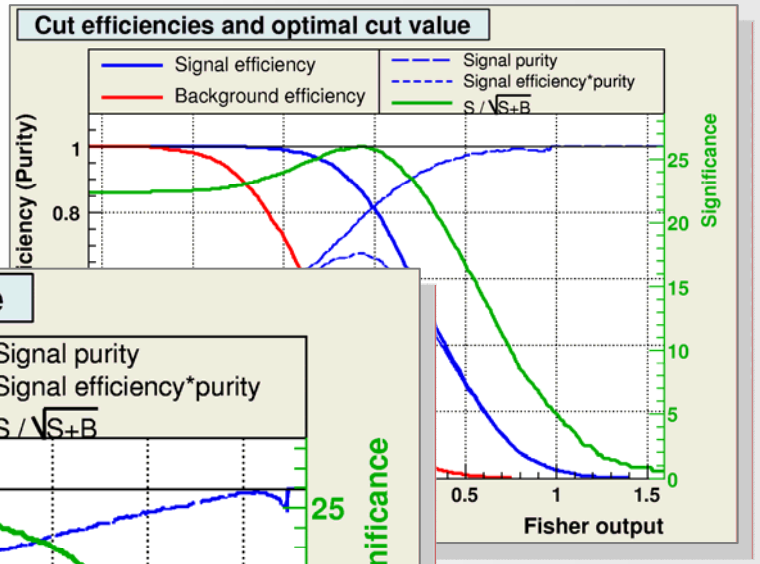
Evaluating the Classifier Training

- Optimal cut for each classifiers …

  Determine the optimal cut (working point)
  on a classifier output







Evaluating the Classifiers Training (taken from *T*MVA output…)

Input Variable Ranking

```
--- Fisher       : Ranking result (top variable is best ranked)
--- Fisher       : ----------------------------------------
--- Fisher       : Rank : Variable  : Discr. power
--- Fisher       : ----------------------------------------
--- Fisher       :    1 : var4      : 2.175e-01
--- Fisher       :    2 : var3      : 1.718e-01
--- Fisher       :    3 : var1      : 9.549e-02
--- Fisher       :    4 : var2      : 2.841e-02
--- Fisher       : ----------------------------------------
```

- How discriminating is a variable ?

Classifier correlation and overlap

```
--- Factory      : Inter-MVA overlap matrix (signal):
--- Factory      : ------------------------------
--- Factory      :             Likelihood  Fisher
--- Factory      : Likelihood:     +1.000  +0.667
--- Factory      :     Fisher:     +0.667  +1.000
--- Factory      : ------------------------------
```

- Do classifiers select the same events as signal and background ?
  If not, there is something to gain !

Evaluating the Classifiers Training (taken from *T*MVA output…)

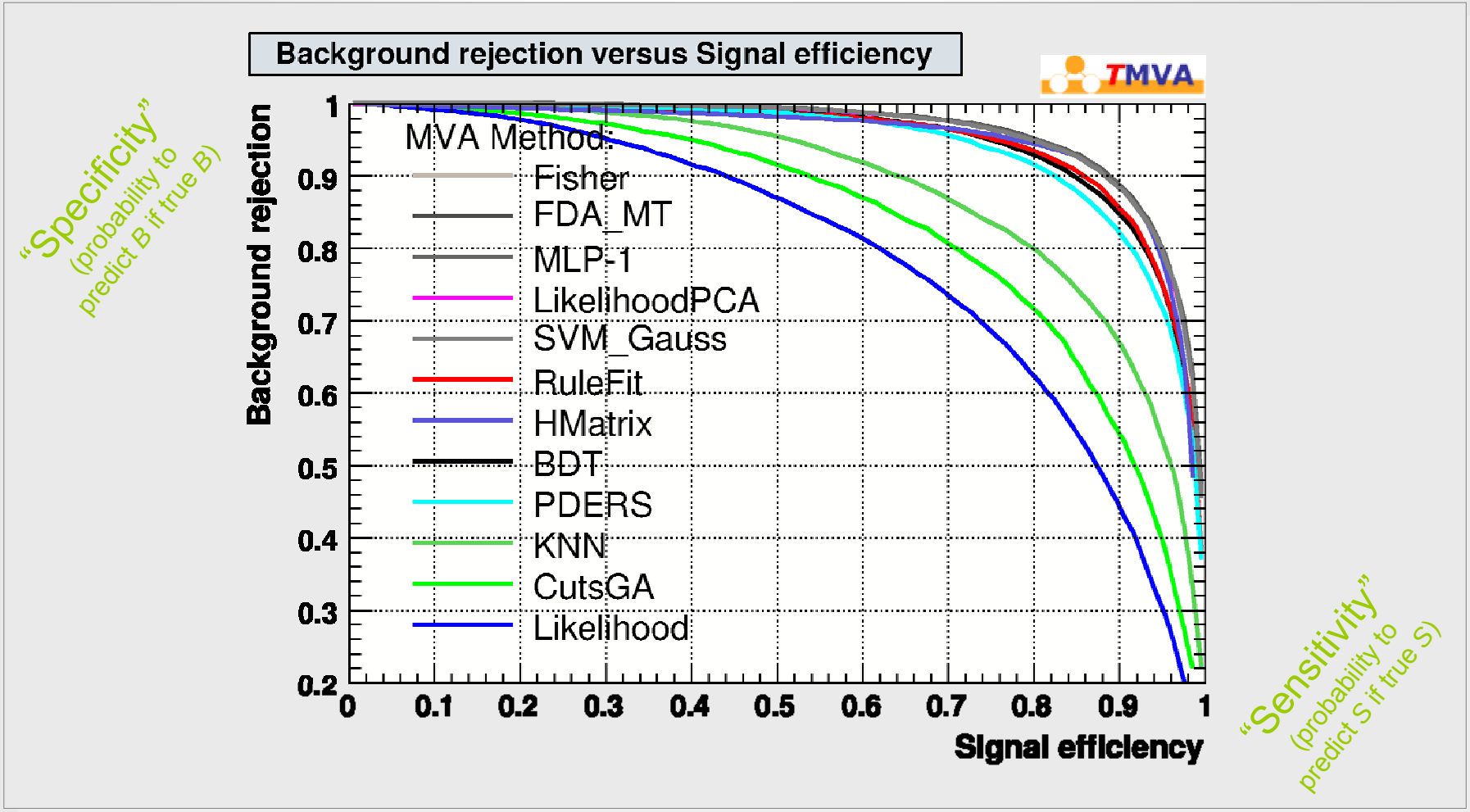
Evaluation results ranked by best signal efficiency and purity (area)

MVA Methods:	Signal efficiency at bkg eff. (error): @B=0.01	@B=0.10	@B=0.30	Area	Separation:	Significance:
Fisher	0.268(03)	0.653(03)	0.873(02)	0.882	0.444	1.189
MLP	0.266(03)	0.656(03)	0.873(02)	0.882	0.444	1.260
LikelihoodD	0.259(03)	0.649(03)	0.871(02)	0.880	0.441	1.251
PDERS	0.223(03)	0.628(03)	0.861(02)	0.870	0.417	1.192
RuleFit	0.196(03)	0.607(03)	0.845(02)	0.859	0.390	1.092
HMatrix	0.058(01)	0.622(03)	0.868(02)	0.855	0.410	1.093
BDT	0.154(02)	0.594(04)	0.838(03)	0.852	0.380	1.099
CutsGA	0.109(02)	1.000(00)	0.717(03)	0.784	0.000	0.000
Likelihood	0.086(02)	0.387(03)	0.677(03)	0.757	0.199	0.682

Better classifier

Receiver Operating Characteristics (ROC) Curve

- Smooth background rejection versus signal efficiency curve:
  (from cut on classifier output)



Summary of Classifiers and their Properties

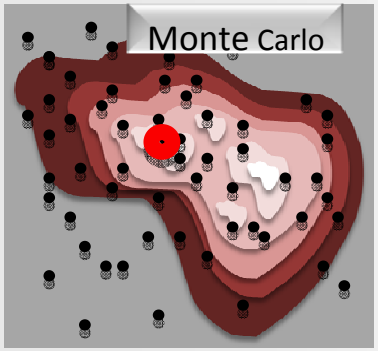
Criteria		Classifiers								
		Cuts	Likeli-hood	PDERS / k-NN	H-Matrix	Fisher	MLP	BDT	RuleFit	SVM
Perfor-mance	no / linear correlations	😐	☺	☺	😐	☺	☺	😐	☺	☺
	nonlinear correlations	😐	☹	☺	☹	☹	☺	☺	😐	☺
Speed	Training	☹	☺	☺	☺	☺	😐	☹	😐	☹
	Response	☺	☺	☹/😐	☺	☺	☺	😐	😐	😐
Robust-ness	Overtraining	☺	😐	😐	☺	☺	☹	☹	😐	😐
	Weak input variables	☺	☺	☹	☺	☺	😐	😐	😐	😐
Curse of dimensionality		☹	☺	☹	☺	☺	😐	☺	😐	😐
Transparency		☺	☺	😐	☺	☺	☹	☹	☹	☹

The properties of the Function discriminant (FDA) depend on the chosen function A

Summary

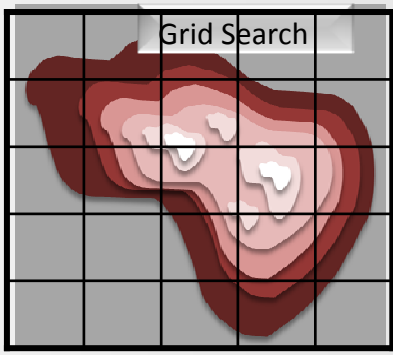
- Seen the general idea of MVA's for Classification and Regression:
- The most important classifiers implemented in TMVA:
  - reconstructing the PDF and use Likelihood Ratio:
    - Nearest Neighbour (Multidimensional Likelihood)
    - Naïve-Bayesian classifier (1dim (projective) Likelihood)
  - fitting directly the decision boundary:
    - Linear discriminant (Fisher)
    - Neuronal Network
    - Support Vector Machine
    - Boosted Decision Tress
- Seen some actual decision boundaries for simple 2D/3D problems
- Genearl analysis advise (for MVAs)
- Systematic errors
  - be as careful as with "cuts" and check against data
- Introduction to TMVA

Minimisation Techniques



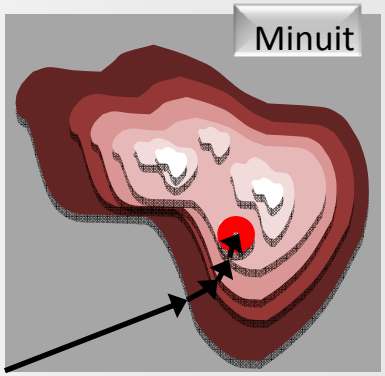
Brute force methods: Random Monte Carlo or Grid search

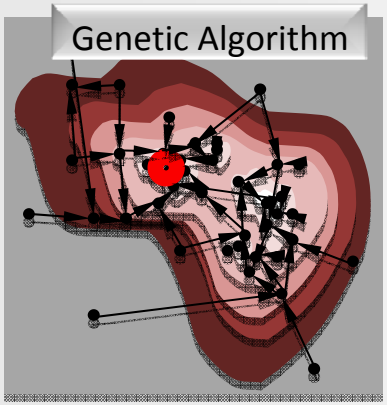
- Sample entire solution space, and chose solution providing minimum estimator
- Good global minimum finder, but poor accuracy



Default solution in HEP: Minuit

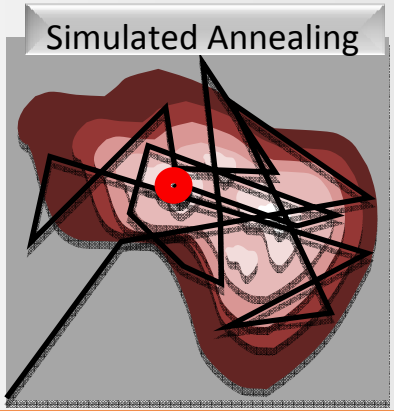
- Gradient-driven search, using variable metric, can use quadratic Newton-type solution
- Poor global minimum finder, gets quickly stuck in presence of local minima





Biology-inspired

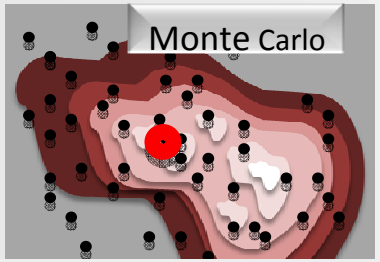
- "Genetic" representation of points in the parameter space
- Uses mutation and "crossover"
- Finds approximately global minima



Like heating up metal and slowly cooling it down ("annealing")

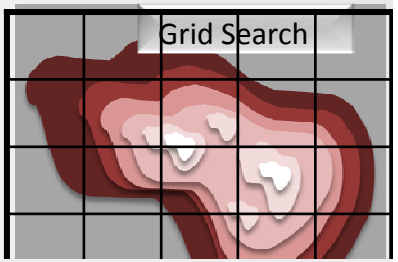
Atoms in metal move towards the state of lowest energy while for sudden cooling atoms tend to freeze in intermediate higher energy states → slow "cooling" of system to avoid "freezing" in local solution

Minimisation Techniques



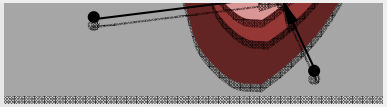
Brute force methods: Random Monte Carlo or Grid search

- Sample entire solution space, and chose solution providing minimum estimator
- Good global minimum finder, but poor accuracy



one can also chain minimisers

For example, one can use MC sampling to detect the vicinity of a global minimum, and then use Minuit to accurately converge to it.



Like heating up metal and slowly cooling it down ("annealing")

Atoms in metal move towards the state of lowest energy while for sudden cooling atoms tend to freeze in intermediate higher energy states → slow "cooling" of system to avoid "freezing" in local solution