Multivariate Discriminators

INFN School of Statistics 2015
Ischia (Napoli, Italy)

Helge Voss

MAX-PLANCK-INSTITUT FÜR KERNPHYSIK IN HEIDELBERG

Overview

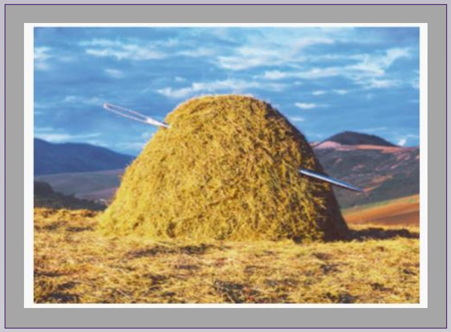
Max-Planck-Institut für Kernphysik

- Multivariate classification/regression algorithms (MVA)
  - what they are
  - how they work
- Overview over some classifiers
  - Multidimensional Likelihood (kNN : k-Nearest Neighbour)
  - Projective Likelihood (naïve Bayes)
  - Linear Classifier
  - Non linear Classifiers
    - Neural Networks
    - Boosted Decision Trees
    - Support Vector Machines
- General comments about:
  - Overtraining
  - Systematic errors

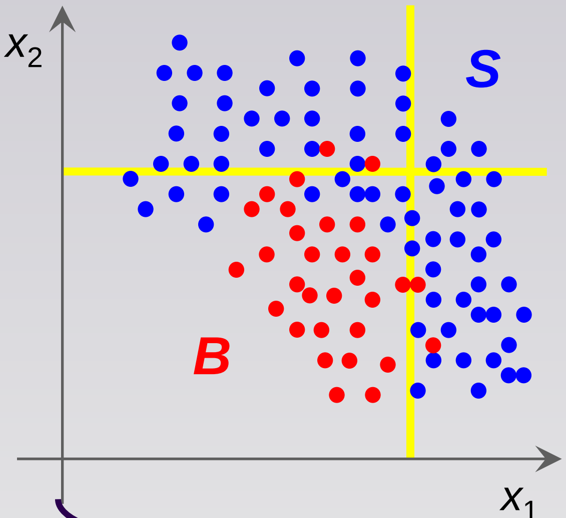
Event Classification

- Discriminate *Signal* from *Background*
  - we have discriminating observed variables x_1, x_2, …

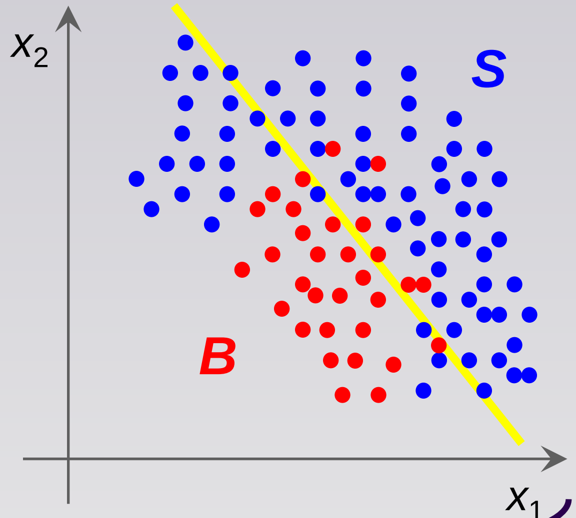
  → decision boundary to select events of type *S* ?



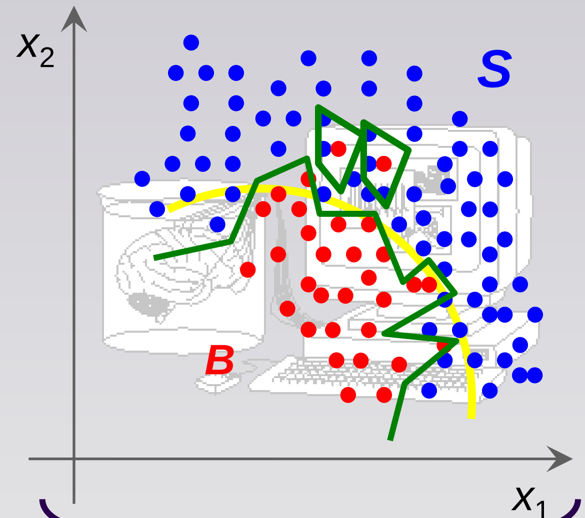
Rectangular cuts?



A linear boundary?



A nonlinear one?



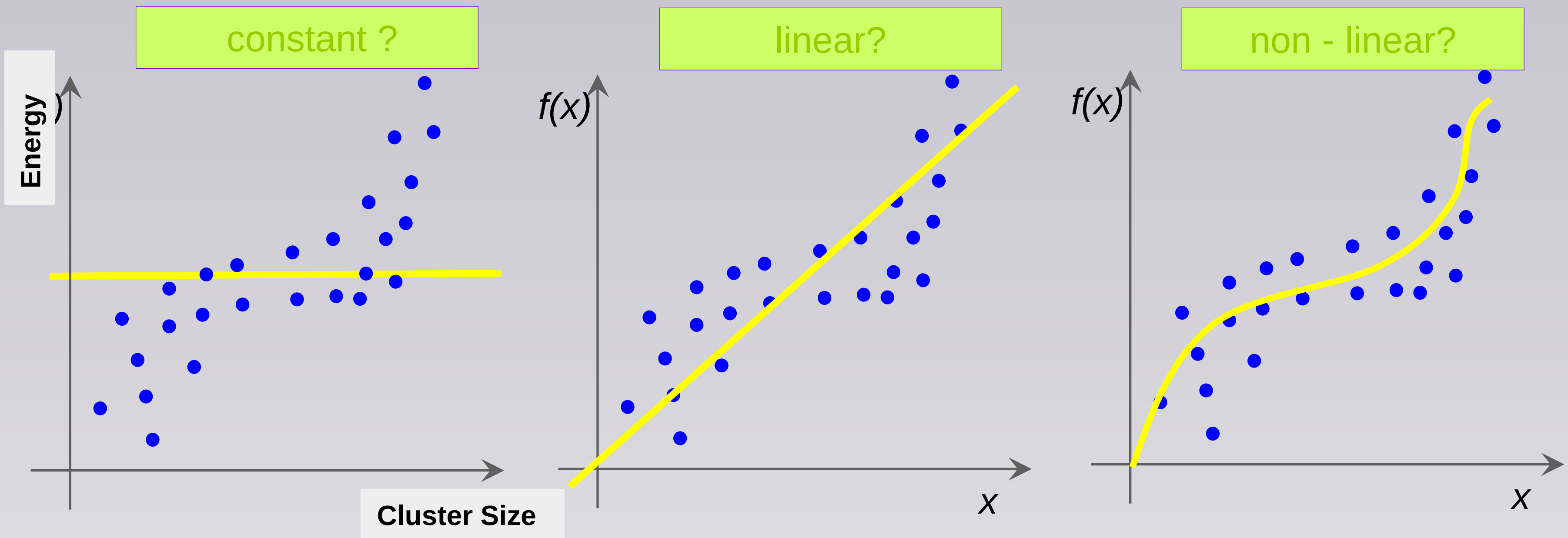
Low variance (stable), high bias methods

High variance, small bias methods

- Which model/class ? Pro and cons ?
- Once decided on a class of boundaries, how to find the "optimal" one ?

Function Estimation: Regression

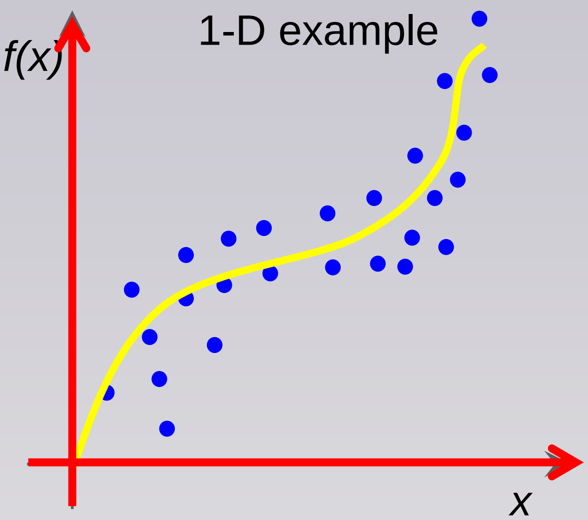
- estimate “functional behaviour” from a set of ‘known measurements” ?
- e.g. : photon energy as function “D”-variables ECAL shower parameters + …



- known analytic model (i.e. nth -order polynomial) □ Maximum Likelihood Fit)
- no model ?
  - → “draw any kind of curve” and parameterize it?
- seems trivial ? → human brain has very good pattern recognition capabilities!
- what if you have **many** input variables?

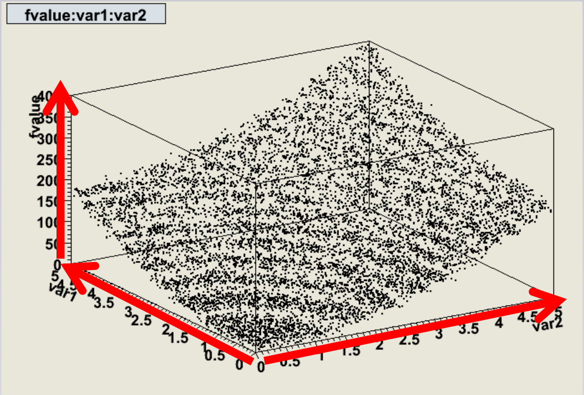
Regression → model functional behaviour

- Estimate the 'Functional Value'
- From measured parameters

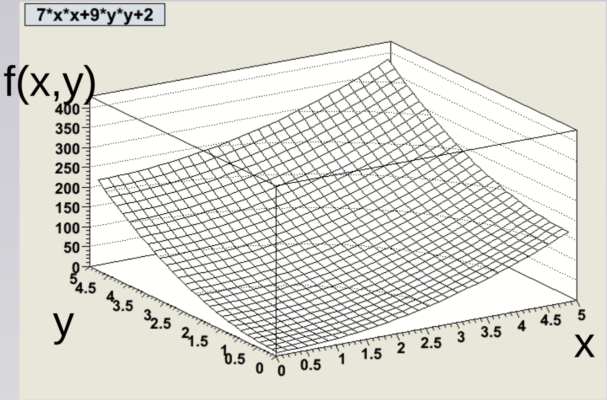


2-D example

events generated according:



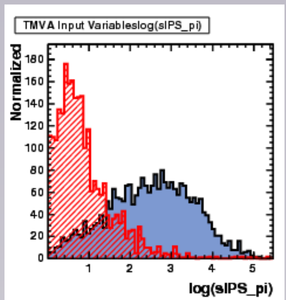
underlying distribution

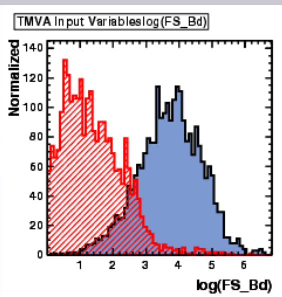


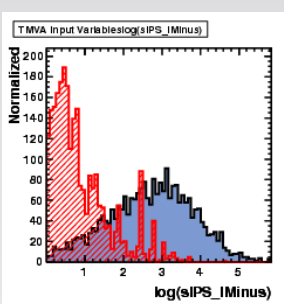
- better known: (linear) regression → fit a known analytic function
  - *e.g.* the above 2-D example → reasonable function would be: f(x) = ax^2+by^2+c
- don't have a reasonable "model" ? → need something more general:
  - *e.g.* piecewise defined splines, kernel estimators, decision trees to approximate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

→ NOT in order to "fit a parameter"
→ provide prediction of function value f(x) for new measurements x (where f(x) is not known)

Event Classification







$\mathbf{P}^D$

“feature space”

$\longrightarrow$ $\mathbf{P}$

- 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 label

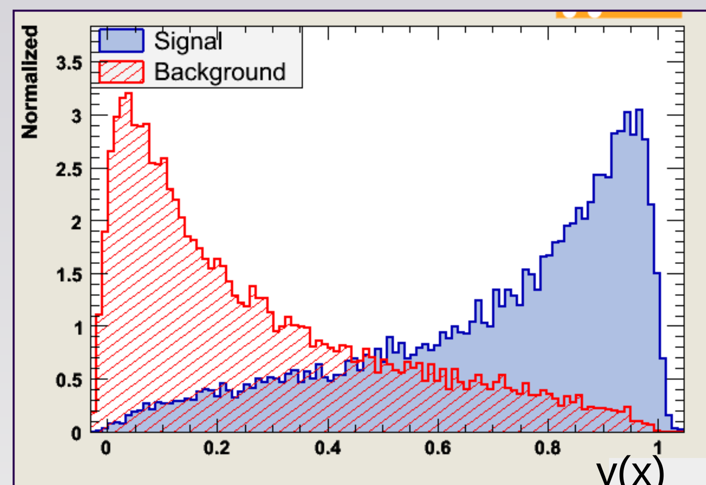
  $y(x)$: $R^D \rightarrow R$:

  most general form

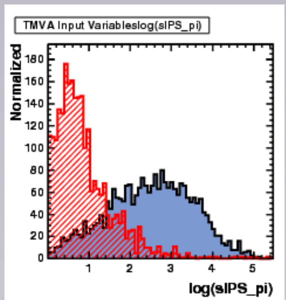
  $y = y(\mathbf{x})$; $\mathbf{x} \in \mathbf{P}^D$

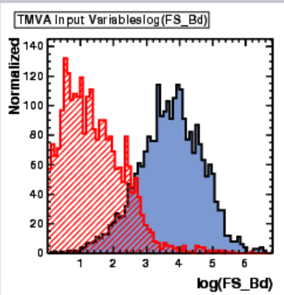
  $\mathbf{x}=\{x_1,\ldots,x_D\}$: input variables

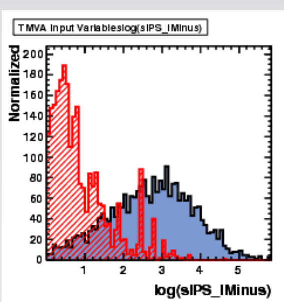
- plotting (historamming) the resulting y(x) values:



Event Classification







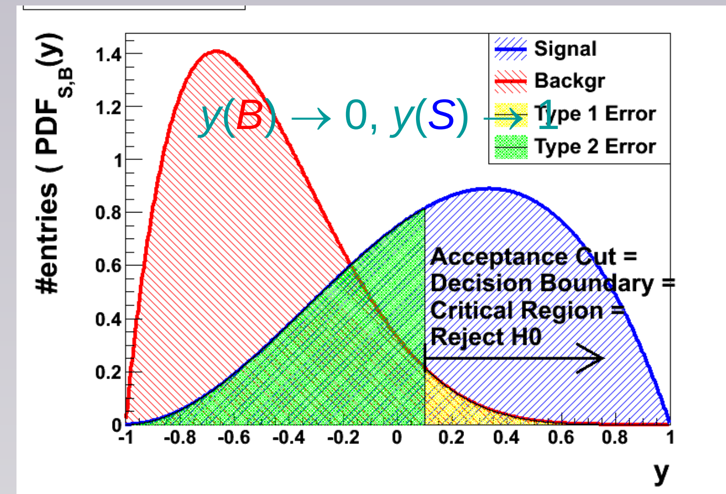
P^D

"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 labels

$y(x)$: $R^n \rightarrow R$:

$\longrightarrow P$



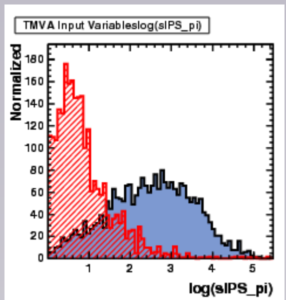
- $y(x)$: "test statistic" in D-dimensional space of input variables
- distributions of $y(x)$: $PDF_S(y)$ and $PDF_B(y)$
- used to set the selection cut!

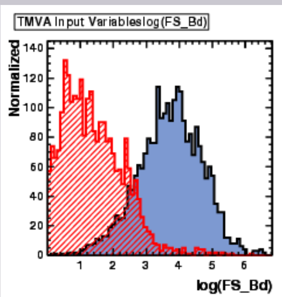
→ efficiency and purity

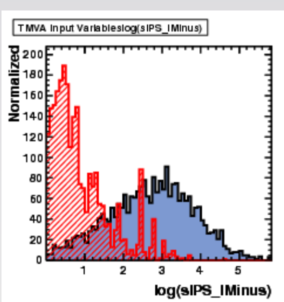
$y(x)$:
- > cut: signal
- = cut: decision boundary
- < cut: background

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

Classification ↔ Regression







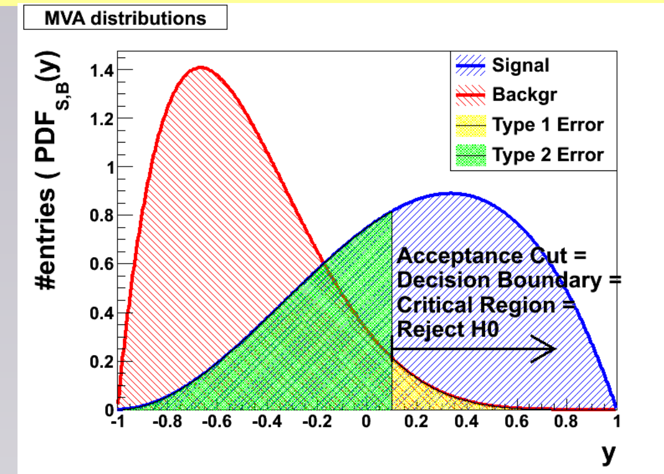
P^D

"feature space"

<u>Classification:</u>

- 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.

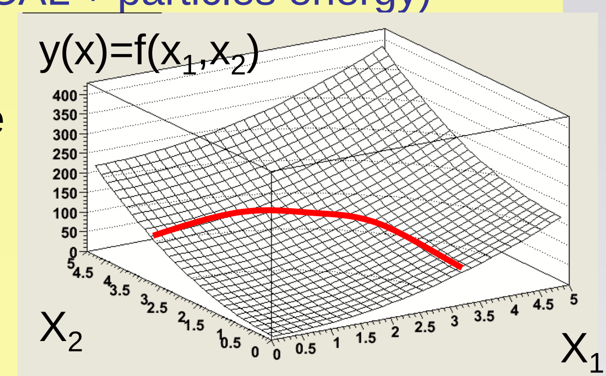


$y(x)$: $R^D \rightarrow R$:

P

<u>Regression:</u>

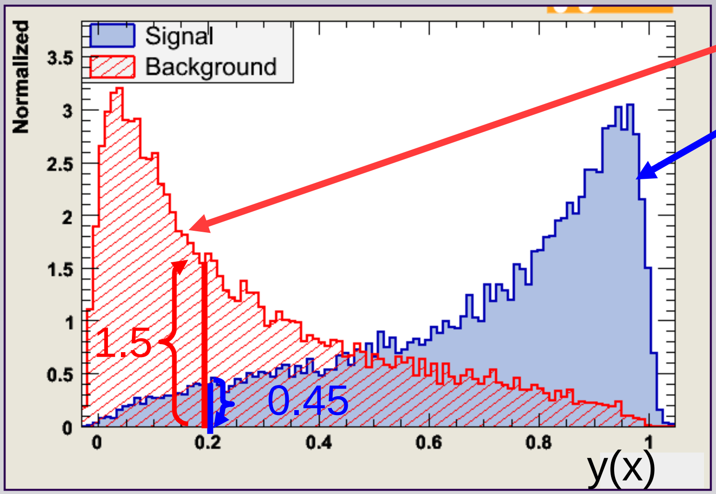
- Each event has "D" measured variables + one function value (*e.g.* cluster shape variables in the ECAL + particles energy)
- $y(x)$: $R^D \rightarrow R$ "regression function"
- $y(x)$=const → hyperplanes where the target function is constant



Now, $y(x)$ needs to be build such that it best approximates the target, not such that it best separates signal from bkg by demanding $y(x)$ > const → signal and $y(x)$ < const → background

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 $y(x) = 0.2$

$\rightarrow$ $PDF_B(y(x)) = 1.5$ 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

Event Classification

P(Class=C|**x**) (or simply P(C|**x**)) : probability that the event class is of C, given the measured observables **x**=$\{x_1, \ldots, x_D\}$ → y(**x**)

Probability density distribution according to the measurements **x** and the given mapping function

Prior probability to observe an event of "class C" *i.e.* the relative abundance of "signal" versus "background" → $\mathrm{P(C)} = f_C = \frac{n_C}{n_{tot}}$

$$P(\text{Class} = C \mid y) = \frac{P(y \mid C) \cdot P(C)}{P(y)}$$

Posterior probability

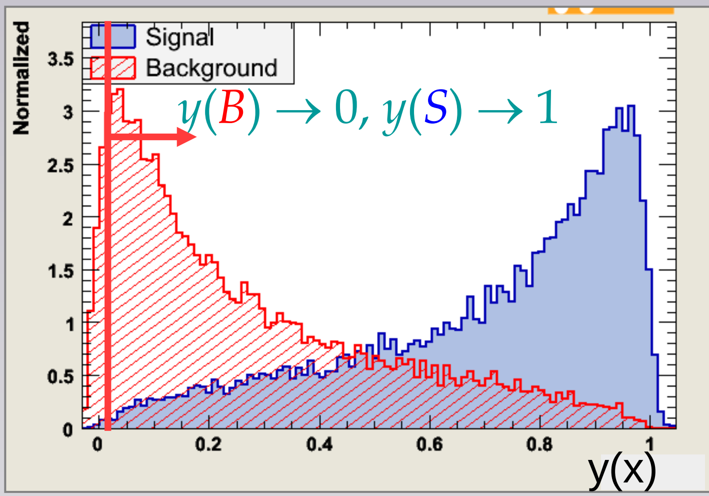
Overall probability density to observe the actual measurement y(x). *i.e.* $P(y) = \sum_{\text{Classes}} P(y \mid \text{Class}) P(\text{Class})$

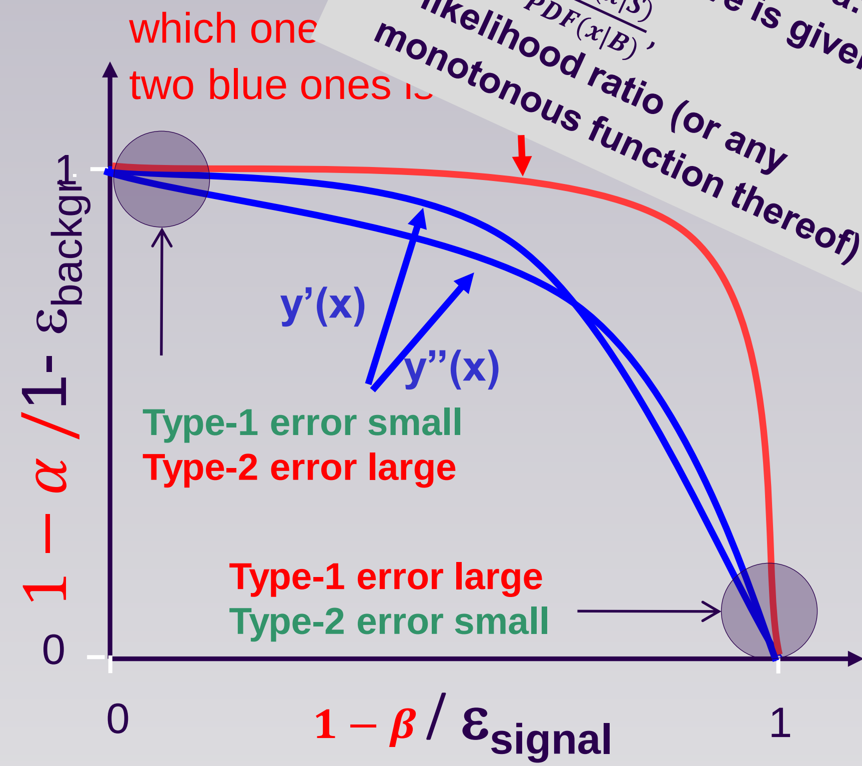
- **It's a nice "exercise" to show that this application of Bayes' Theorem gives exactly the formula on the previous slide !**

Receiver Operation Chara tersic (ROC) curve

"Neyman-Pearson Lemma: "limit" in ROC curve is given by
$y(x) = \frac{PDF(x|S)}{PDF(x|B)}$,
the likelihood ratio (or any monotonous function thereof)

Signal(H_1) /Background(H_0) discrimination:





- **Type 1 error: reject H_0 (i.e. the 'is bkg' hypothesis) although it would haven been true**
  - **→ background contamination**
  - Significance α: background sel. efficiency **1– α: background rejection**
- **Type 2 error: accept H_0 although false**
  - **→ loss of efficiency**
  - Power: **1– β signal selection efficiency**

MVA and Machine Learning

- Finding y(x) : $R^n \rightarrow R$
  - given a certain type of model class y(x)
  - **"fits" (learns) from events with known type the parameters in y(x)**

    such that y:
    - CLASSIFICATION: separates well Signal from Background in training data
    - REGRESSION: fits well the target function for training events
  - **use for yet unknown events → predictions**

→ supervised machine learning

Event Classification → finding the mapping function y(x)

- Neyman-Persons: $\boldsymbol{y}(\boldsymbol{x}) = \frac{\boldsymbol{PDF}(\boldsymbol{x}|\boldsymbol{S})}{\boldsymbol{PDF}(\boldsymbol{x}|\boldsymbol{B})}$
- p(x|S) and p(x|B) 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"

- Use these "training" events to:

  - estimate p(x|S) and p(x|B): (e.g. the differential cross section folded with the detector influences) and use the likelihood ratio

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

    →(generative algorithms)

  OR

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

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

    →(discriminative algorithms)

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

Recap:

Multivariate Algorithms → combine all 'discriminating' measured variables into ONE single "MVA-variable" y(x): $R^D \rightarrow R$

→ contains 'all' information from the "D"-measurements
- → allows to place ONE final cut
  - → corresponding to an (complicated) decision boundary in D-dimensions
- → may also be used to "weight" events rather than to 'cut' them away

y(x) is found by

→ estimating the pdfs and using the likelihood ratio

OR

→ Via training:
- → fitting the free parameters "w" (weights) in some model y(x; w) to 'known data'

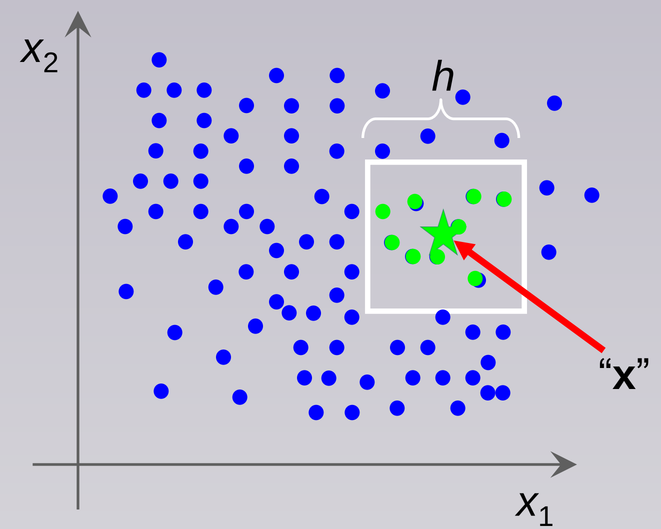
Overview

- Multivariate classification/regression algorithms (MVA)
  - what they are
  - how they work
- Overview over some classifiers
  - Multidimensional Likelihood (kNN : k-Nearest Neighbour)
  - Projective Likelihood (naïve Bayes)
  - Linear Classifier
  - Non linear Classifiers
    - Neural Networks
    - Boosted Decision Trees
    - Support Vector Machines
- General comments about:
  - Overtraining
  - Systematic errors

K- Nearest Neighbour

- estimate probability density P(x) in D-dimensional space:
- 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:

“events” distributed according to P(x)



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

k(u): is called a Kernel function:

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

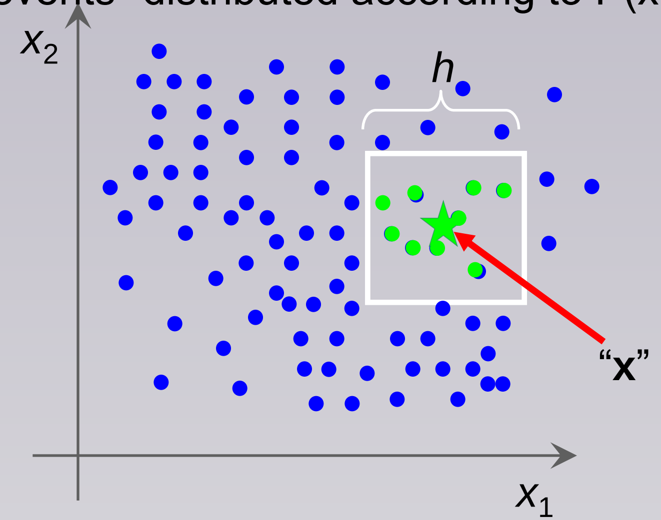
- Classification: Determine PDF$_S$(x) and PDF$_B$(x)
  → likelihood ratio as classifier!

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

→ Kernel Density estimator of the probability density

Nearest Neighbour and Kernel Density Estimator

- estimate probability density P(x) in D-dimensional space:
- 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:



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

k(u): is called a Kernel function:

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

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

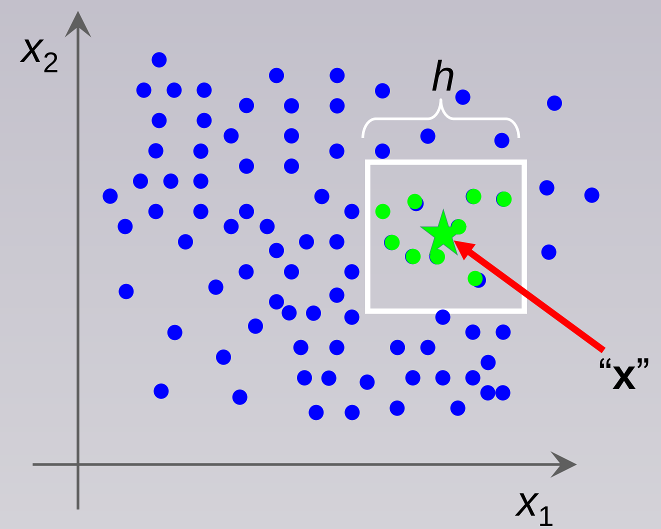
$$\frac{1}{N}\sum_i^N k(\vec{x}^i - \vec{x}) f(\vec{x}^i) = \int_V \hat{f}(\vec{x}) P(\vec{x}) d\vec{x}$$

i.e.: the average function value

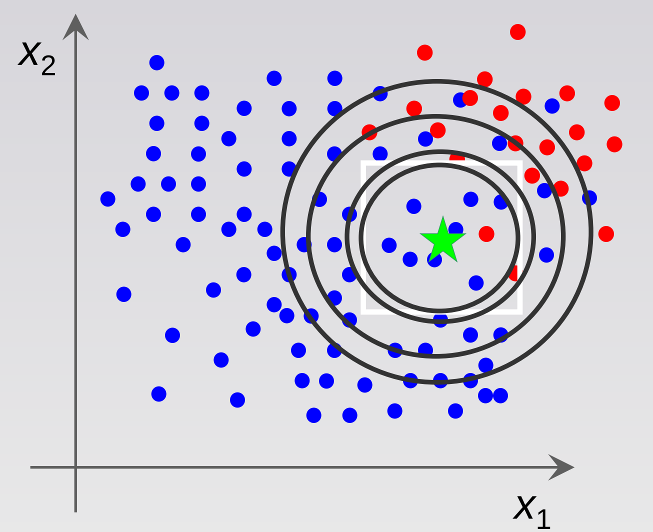
Nearest Neighbour and Kernel Density Estimator

- estimate probability density P(x) in D-dimensional space:
- 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:

"events" distributed according to P(x)



- 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

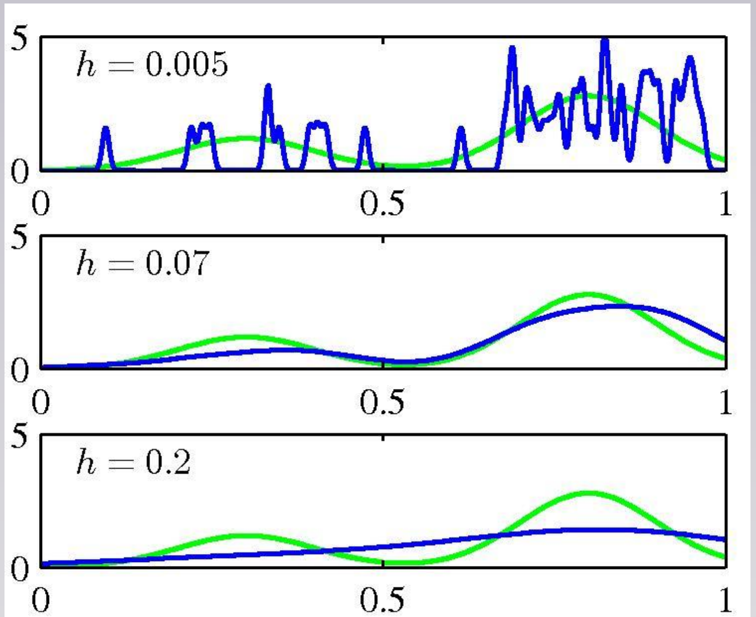
$$y(x) = \frac{n_S}{K}$$

- Kernel Density Estimator: replace "window" by "smooth" kernel function → weight events by distance (e.g. via Gaussian)

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)
- which metric for the Kernel (window)?
  - normalise all variables to same range
  - include correlations ?
    - Mahalanobis Metric: x*x → $xV^{-1}x$



(Christopher M.Bishop)

- a drawback of Kernel density estimators:

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

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

"Curse of Dimensionality"

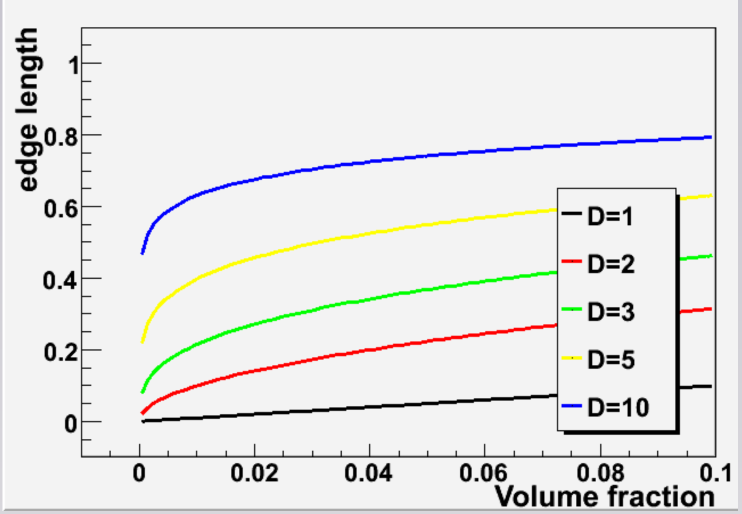
Max-Planck-Institut für Kernphysik

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:

- higher dimensional cases K-events often are not in a small "vicinity" of the space point anymore:



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

for a cube of a particular fraction of the volume:

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

- 10 dimensions: capture 1% of the phase space

→ 63% of range in each variable necessary → that's not "local" anymore..☹

→ develop all the alternative classification/regression techniques

Naïve Bayesian Classifier (projective Likelihood Classifier)

Multivariate Likelihood (k-Nearest Neighbour)

→ estimate the full D-dimensional joint probability density

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

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

PDFs

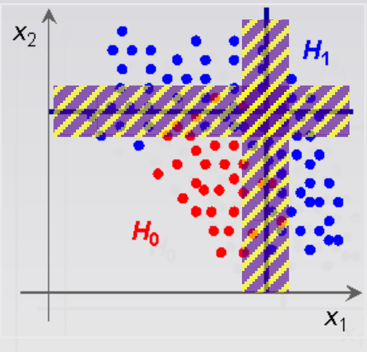
discriminating variables

Likelihood ratio for event *event*

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

Classes: signal, background types

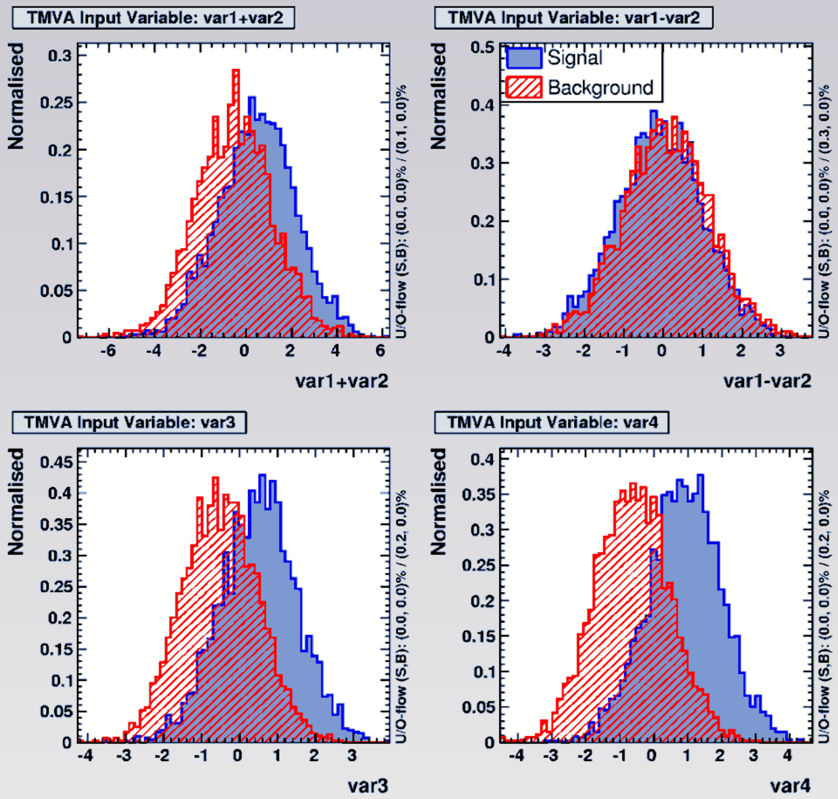
- One of the first and still very popular MVA-algorithm in HEP
  - No hard cuts on individual variables,
  - allow for some "fuzzyness": one very signal like variable may counterweigh another less signal like variable
- optimal method if correlations == 0 (Neyman Pearson Lemma)
  - try to "eliminate" correlations → e.g. linear de-correlation

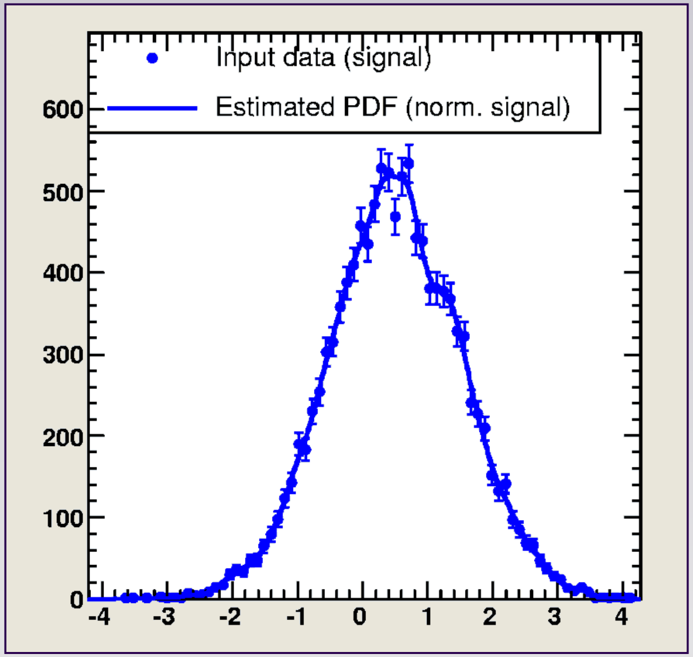


PDE introduces fuzzy logic

Naïve Bayesian Classifier (projective Likelihood Classifier)

- Where to get the PDF's ?
  - Simple histograms
  - Smoothing (e.g. spline or kernel function)





Overview

- Multivariate classification/regression algorithms (MVA)
  - what they are
  - how they work
- Overview over some classifiers
  - Multidimensional Likelihood (kNN : k-Nearest Neighbour)
  - Projective Likelihood (naïve Bayes)
  - **Linear Classifier**
  - **Non linear Classifiers**
    - **Neural Networks**
    - Boosted Decision Trees
    - Support Vector Machines
- General comments about:
  - Overtraining
  - Systematic errors

Classifier Training and Loss-Function

- Discriminative algorithms:
  - No PDF estimation

  - But fit a “decision boundary” directly: i.e.

    → provide a set of “basis” functions h_i (“a model”):
    → $y(x) = \sum w_i h_i(x)$

    - adjust parameters w_i

      → optimally separating hyperplane (surface) → “training”

Linear Discriminant

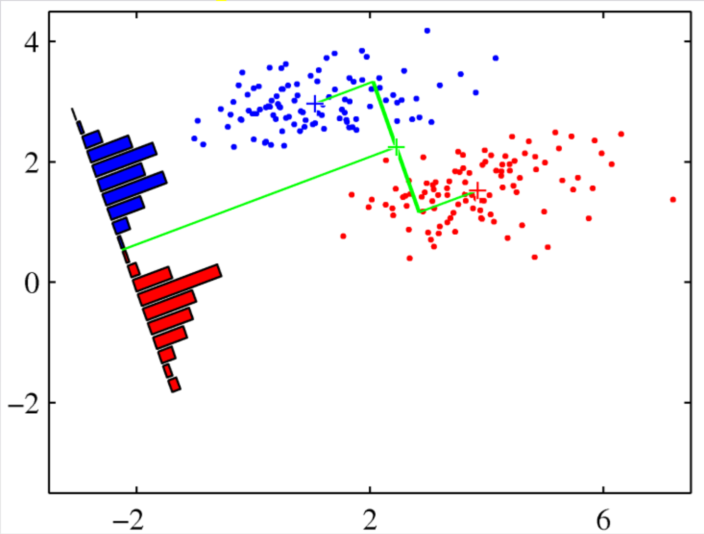
General:

$$y(x = \{x_1, \dots, x_D\}) = \sum_{i=0}^{M} w_i h_i(x)$$

Linear Discriminant:

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

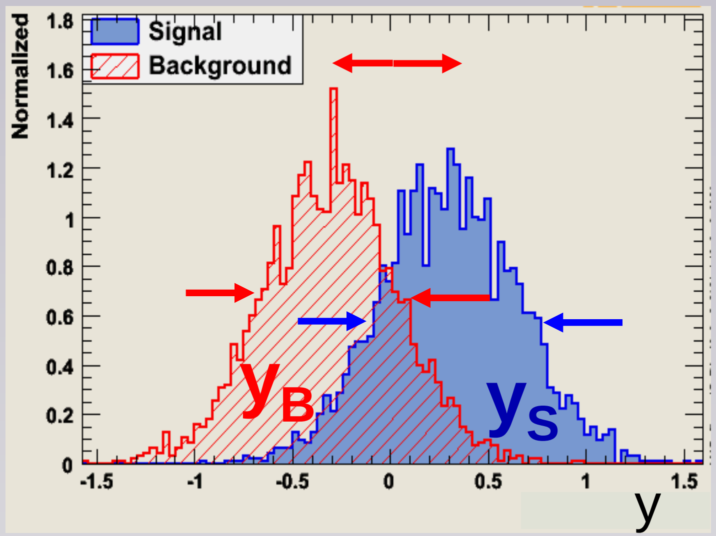
i.e. any linear function of the input variables: → linear decision boundaries



PDF of the test statistic y(x)
→ determine the "weights" w that separate "best" PDF_S from PDF_B

Fisher's Linear Discriminant

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



determine the "weights" w that do "best"

- Maximise "separation" between the S and B

→ minimise overlap of the distributions of y_S and y_B

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

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

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

note: these quantities can be calculated from the training data

Classifier Training and Loss-Function

More general: rather than: maximize $J(\vec{w})$ constructed "by hand"

→ minimize a the expectation value of a "Loss function" $L(y^{train}, y(x))$

which penalizes prediction errors for training events

regression: y_i^{train} = the functional value of training event i which happens to have the measured observables x_i

classification: y_i^{train} =1 for signal, =0 (-1) background

What to choose for $L(y^{train}, y(x))$?

- Regression:

→ $E[L] = E\left[\left(y^{train} - y(x)\right)^2\right]$ squared error loss (regression)

- Classification:

→ $E[L] = E[y_i^{train} \log(y(x_i)) + (1 - y_i^{train}) \log(1 - y(x_i))]$ binomial loss

Classifier Training and Loss-Function

- Regression: y_i^{train} : Gaussian distributed around a mean value
  - Remember: Maximum Likelihood estimatior (Tuesday by Glen Cowan)

  → Maximise: log probability of the observed training data:

$$\log L = \log \prod_i^{events} P\left(y_i^{train} | y(x_i)\right) = \sum_i^{events} \log(P(y_i^{train} | y(x_i)) = \sum_i^{events} \left(y_i^{train} - y(x_i)\right)^2$$

→ $E[L] = E\left[\left(y^{train} - y(x\ )\right)^2\right]$ squared error loss (regression)

- Classification: now: y_i^{train} (i.e. is it 'signal' or 'background') is Bernoulli distributed

$$\log L = \sum_i^{events} \log(P(y_i^{train} | y(x_i)) = \sum_i \log(P(S|x_i)^{y_i^{train}} P(B|x_i)^{1-y_i^{train}})$$

If we now say y(x) should simply parametrize P(S|x); P(B|x)=1-P(B|x) →

→ $E[L] = E[y_i^{train} \log\left(y(x_i)\right) + \left(1 - y_i^{train}\right) \log(1 - y(x_i))]$ binomial loss

Logistic Regression*

*Actually, although called 'regression' it is a 'classification' algorithm!

Fisher Discriminant:

→ equivalent to Linear Discriminant with 'squared loss function'

→ Ups: didn't we just show that "classification" would naturally use 'binomial loss function" ?

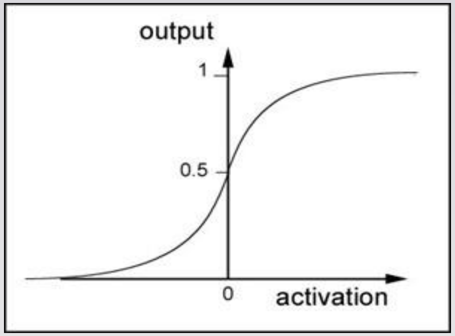
→ O.k. let's build a linear classifier that maximizes 'binomial loss':

- → For y(x) to parametrize P(S|x), we clearly cannot 'use a linear function for 'y(x)'
- → But we can 'squeeze' any linear function $w_0 + \sum w_j x^j$ = Wx into the proper interval $0 \leq y(x) \leq 1$ using the 'logistic function' (i.e. sigmoid function)

Logistic Regression

→ $y(x) = P(S|x) = sigmoid(Wx) = \frac{1}{1+e^{-Wx}}$

→ $Log(Odds) = Log\left(\frac{P(S|x)}{P(B|x)}\right) = Wx$ is linear!



Note: Now y(x) has a 'probability' interpretation. y(x) of the Fisher discriminant was 'just' a discriminator.

Neural Networks

for “arbitrary” non-linear decision boundaries → y(x) non-linear function

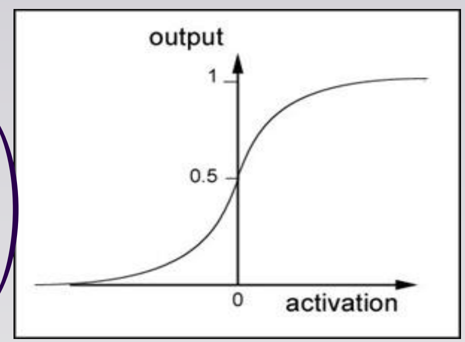
$$y(\vec{x}) = sigmoid\left(\sum_k^M w_k h_k(\vec{x})\right)$$

- Think of $h_k(x)$ 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)

there are also mathematical proves for this statement.

Imagine you chose do the following:

$$y(x) = A\left(\sum_k^M w_k \overbrace{A\left(w_{k0} + \sum_{j=1}^D w_{kj} x_j\right)}^{h_i(\mathbf{x})}\right)$$



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

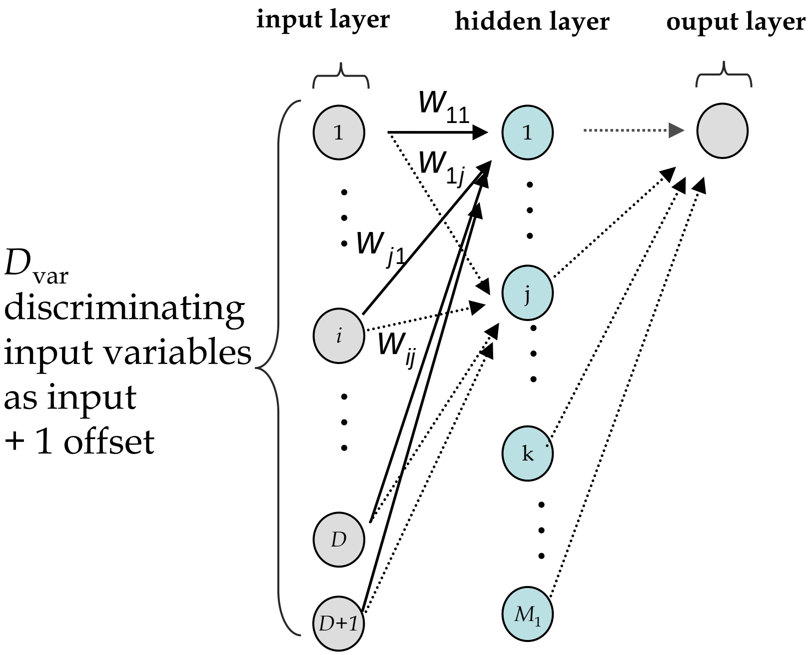
the sigmoid function

A non linear (sigmoid) function of
a linear combination of
non linear function(s) of
linear combination(s) of
the input data

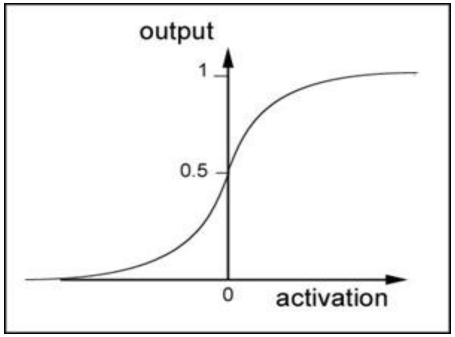
Ready is the Neural Network
Now we “only” need to find the appropriate “weights” w

Neural Networks: Multilayer Perceptron MLP

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



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 …

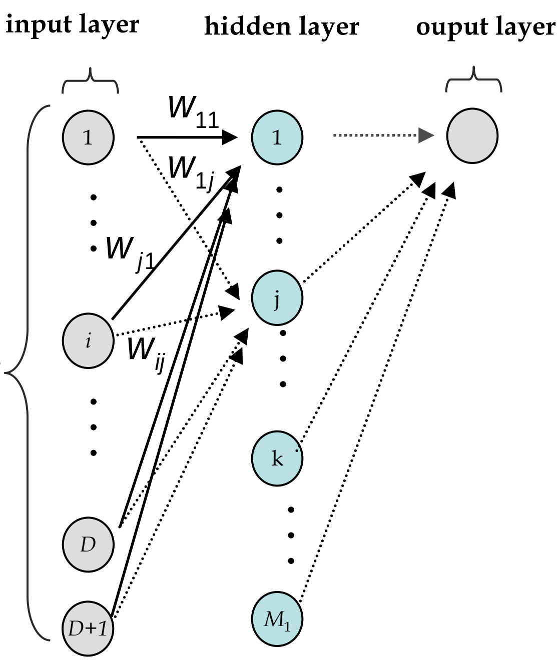
- Nodes in hidden layer represent the "activation functions" whose arguments are linear combinations of input variables → non-linear response to the input
- The output is a linear combination of the output of the activation functions at the internal nodes
- Input to the layers from preceding nodes only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

Neural Networks: Multilayer Perceptron MLP

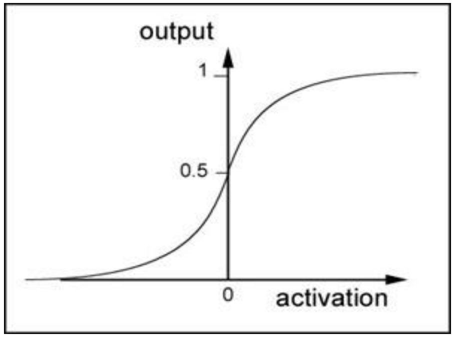
Max-Planck-Institut für Kernphysik

try to “interpret” the formula as a Neural Network:

D_{var} discriminating input variables as input + 1 offset



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

Now we just need to fix the parameters by ? → Minimizing Loss function:

$$L(w) = \sum_{i}^{events} \Big(\underbrace{y_i^{train}}_{\text{true}} - \underbrace{y(x_i)}_{\text{predicted}}\Big)^2$$

i.e. use usual "sum of squares"

classification: Binomial loss

$$L(w) = \sum_{i}^{events} \left(y_i^{train} \log\big(y(x_i)\big) + \left(1 - y_i^{train}\right)\log\big(1 - y(x_i)\big)\right)$$

where $y^{train} = \begin{cases} 1, & signal \\ 0, & backgr \end{cases}$

- y(x): very "wiggly" function → many local minima.

→ one global overall fit not efficient/reliable

Back-propagation

<u>back propagation</u> (nice recursive formulation of the gradient $\frac{\partial L}{\partial w_{ij}}$ using 'chain rule')

→(Stochastic) gradient decent: update weights 'along the gradient' at each training step

→$w_{ij} \rightarrow w_{ij} - \eta \frac{\partial L}{\partial w_{ij}}$; $\eta =$ learning rate

- online learning: update event by event
- (mini) batch learning: update after seeing the whole (parts of the) sample

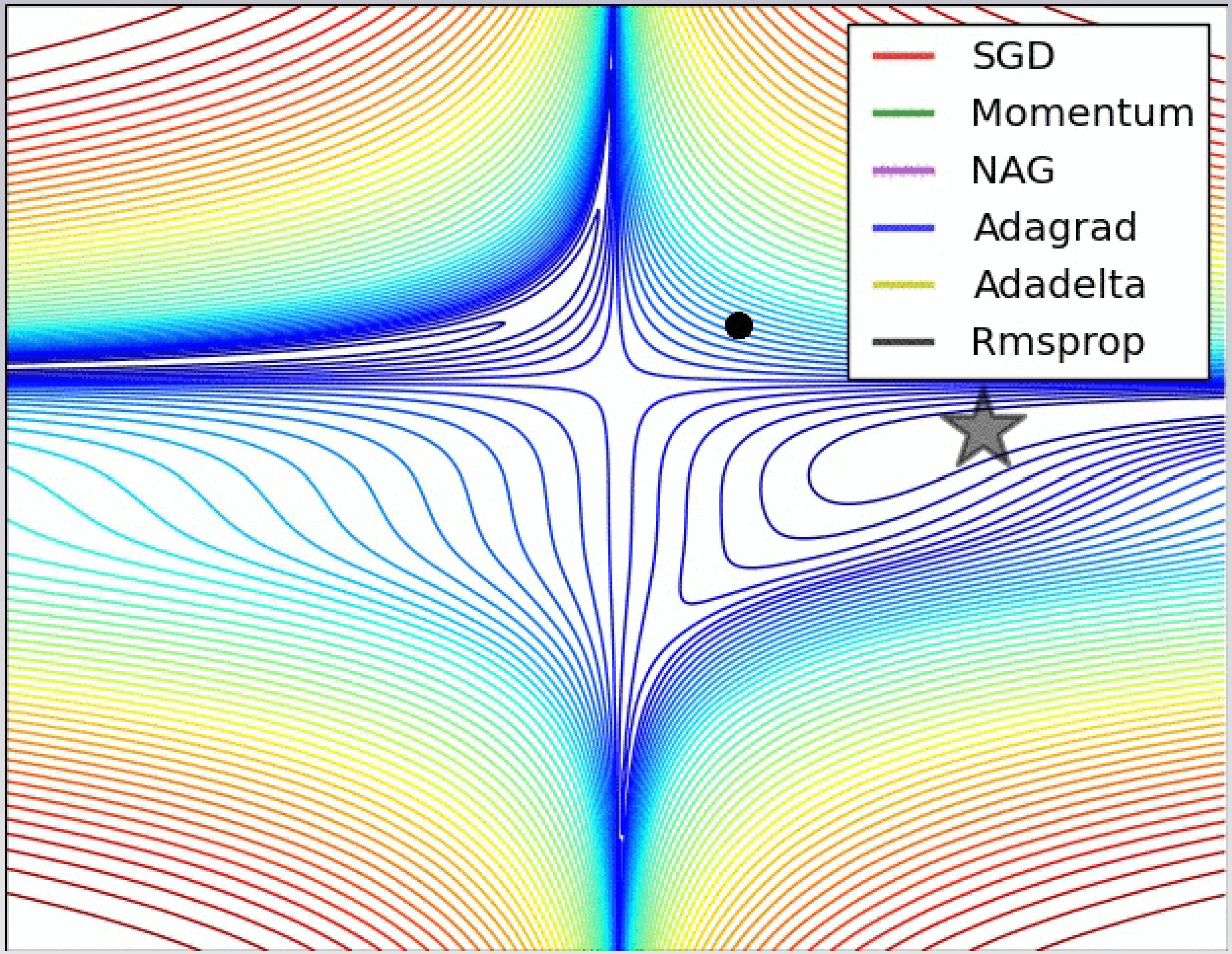
Simple "gradient" is typically not the most effective function minimizer:

→ Use function curvature ("hessian" matrix) à la Newton method

→ "Momentum" - accelerate the learning when gradient direction stays 'constant' e.g.:

→ $v \rightarrow \mu v - \eta \nabla L$; $w_{ij} \rightarrow w_{ij} + v$ (classical momentum)

Gradient Descent



What is "Deep Learning"

Neural networks with 'many hidden layers'

- Learn a hierarchy of features: i.e. successive layers learn: 4-vectors → invariant masses → decays)
- Used to be 'impossible to train' → vanishing gradient problem

- Enormous progress in recent years
  - Layerwise pre-training using 'auto-encoders' or 'restricted-Boltzman machines'
  - 'intelligent' random weight initialisation
  - Stochastic gradient decent with 'momentum'
  - 'new' activation functions:

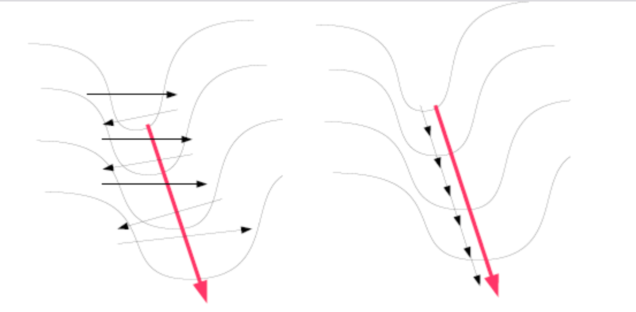


Figure 1. Optimization in a long narrow valley

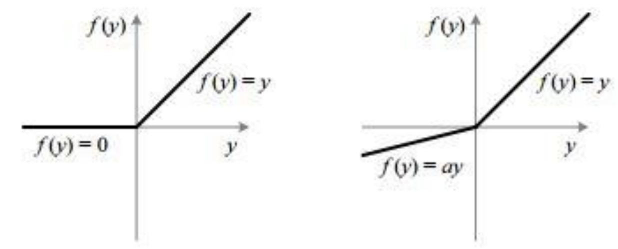


Figure 1. ReLU vs. PReLU. For PReLU, the coefficient of the negative part is not constant and is adaptively learned.

Summary

- Multivariate Algorithms are a powerful alternative to "classical cuts" that:
  - Do not use hard selection criteria (cuts) on each individual observables
  - Look at all observables "together"
    → eg. combining them into 1 variable

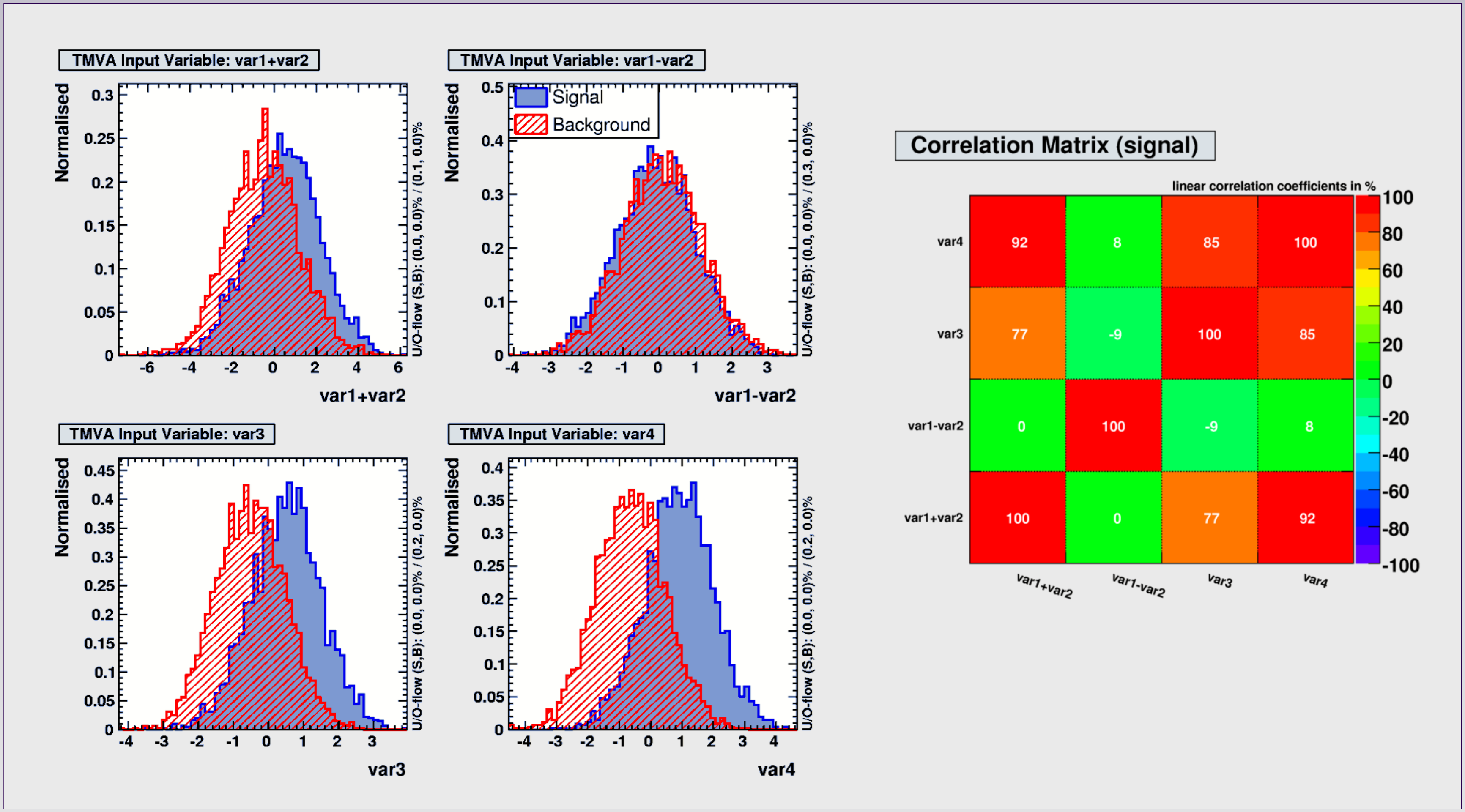
→ Mulitdimensional Likelihood → PDF in D-dimensions
→ Projective Likelihood (Naïve Bayesian) → PDF in D times 1 dimension
  → Be careful about correlations

→ Linear classifiers : y(x) = 'linear combination of observables "x" '
  → decision boundary (y(x) = const) is a linear hyperplane

→ Non-linear classifier: Neural networks → any kind of hyperplane

What if there are correlations?

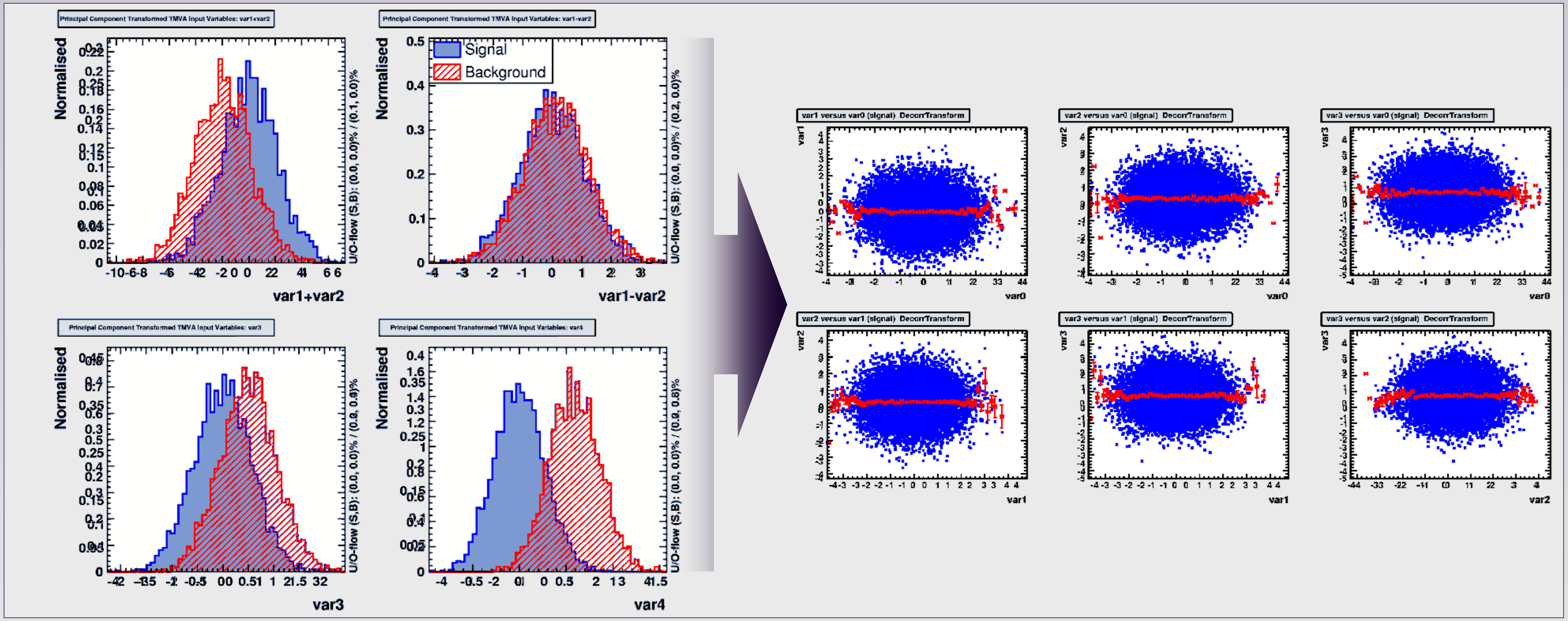
- Typically correlations are present: $C_{ij}=\mathrm{cov}[\, x_i\, ,\, x_j\,]=E[\, x_i\, x_j\,]-E[\, x_i\,]E[\, x_j\,]\neq 0\ \ (i\neq j)$



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

De-Correlation

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



Attention: eliminates only linear correlations!!

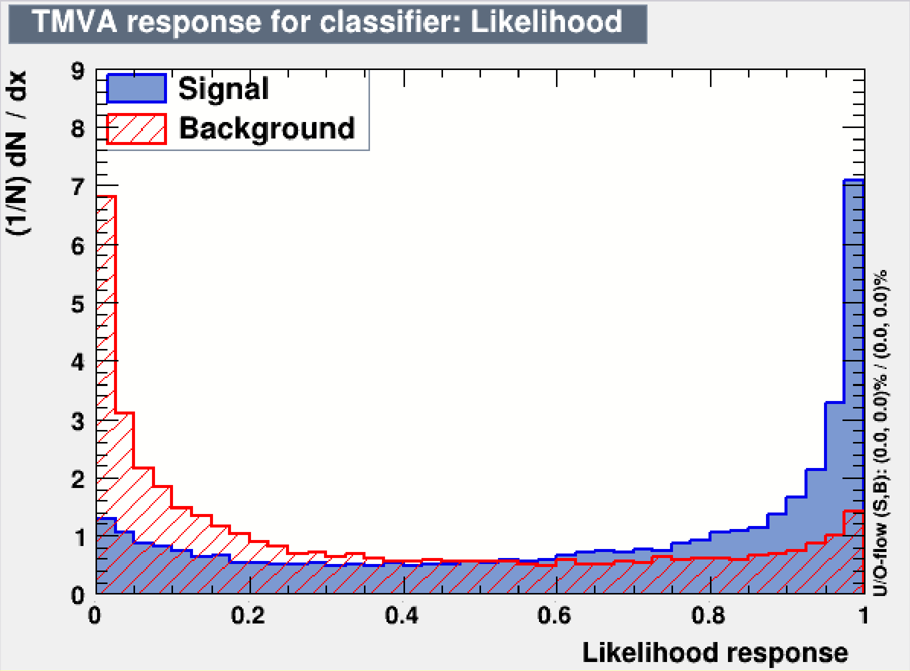
Decorrelation at Work

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

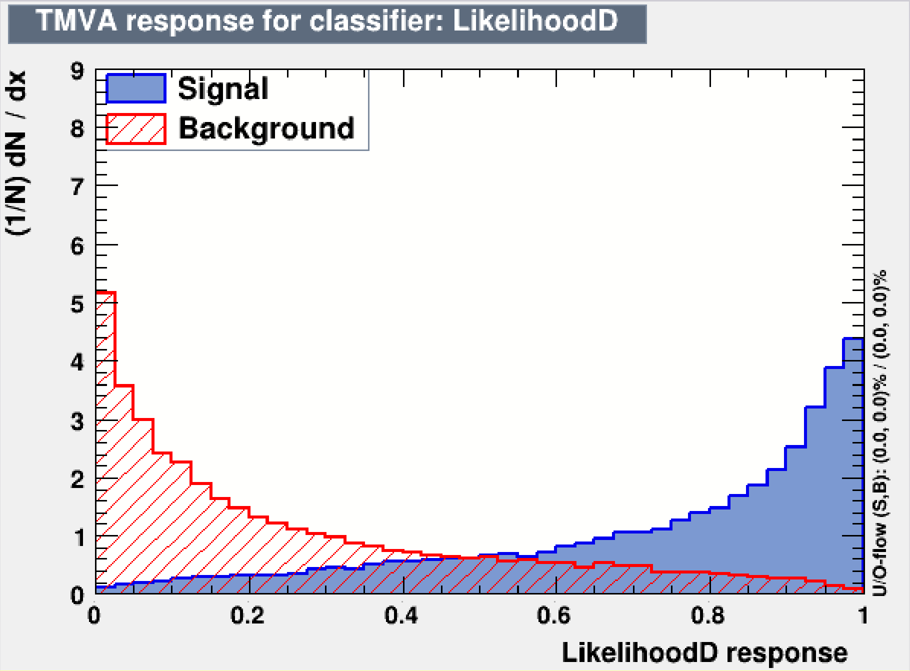
→ 1-D Likelihood on decorrelated sample give best possible performance

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

correlated variables:



after decorrelation



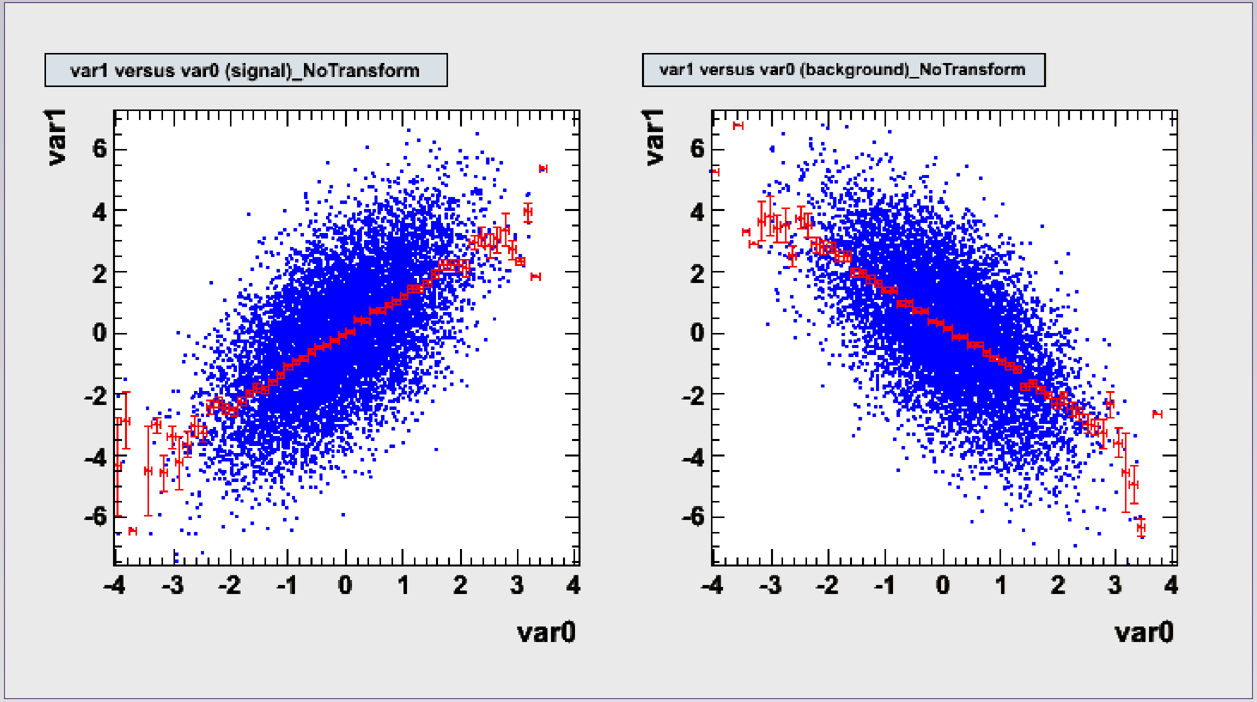
Watch out! Things might look very different for non-linear correlations!

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 cases where correlations between signal and background differ?

Original correlations



Signal

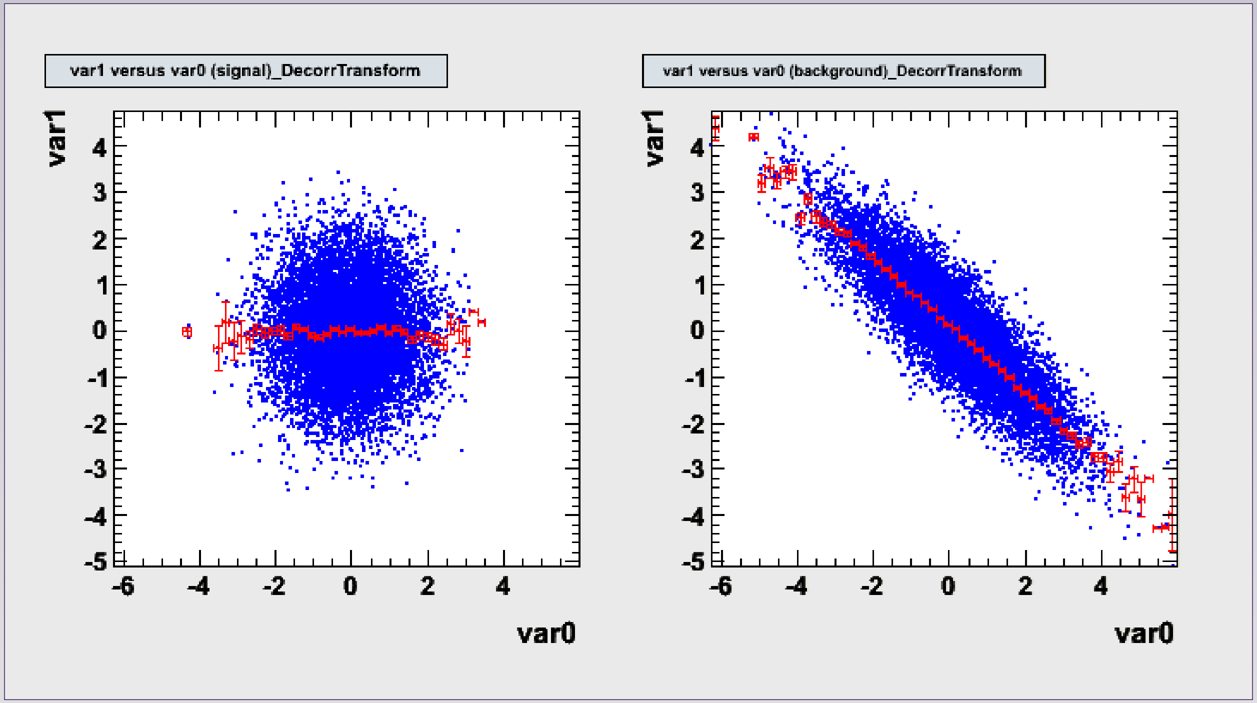
Background

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

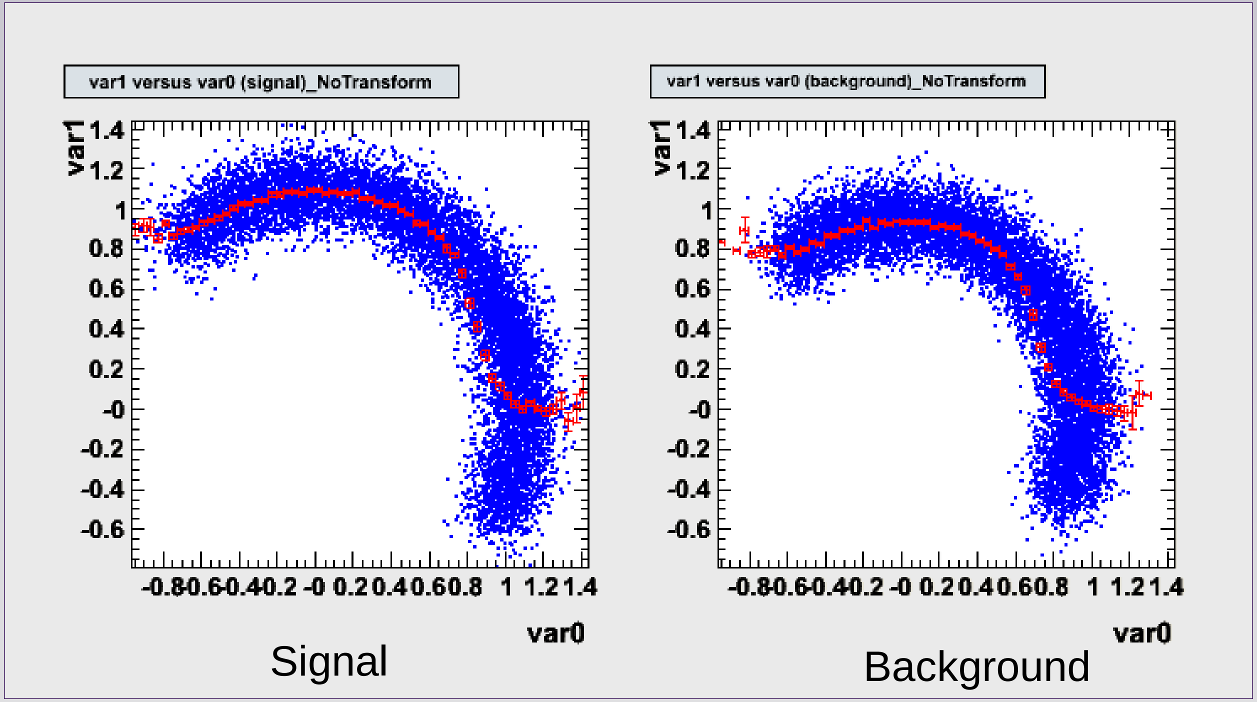
Background

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



Signal

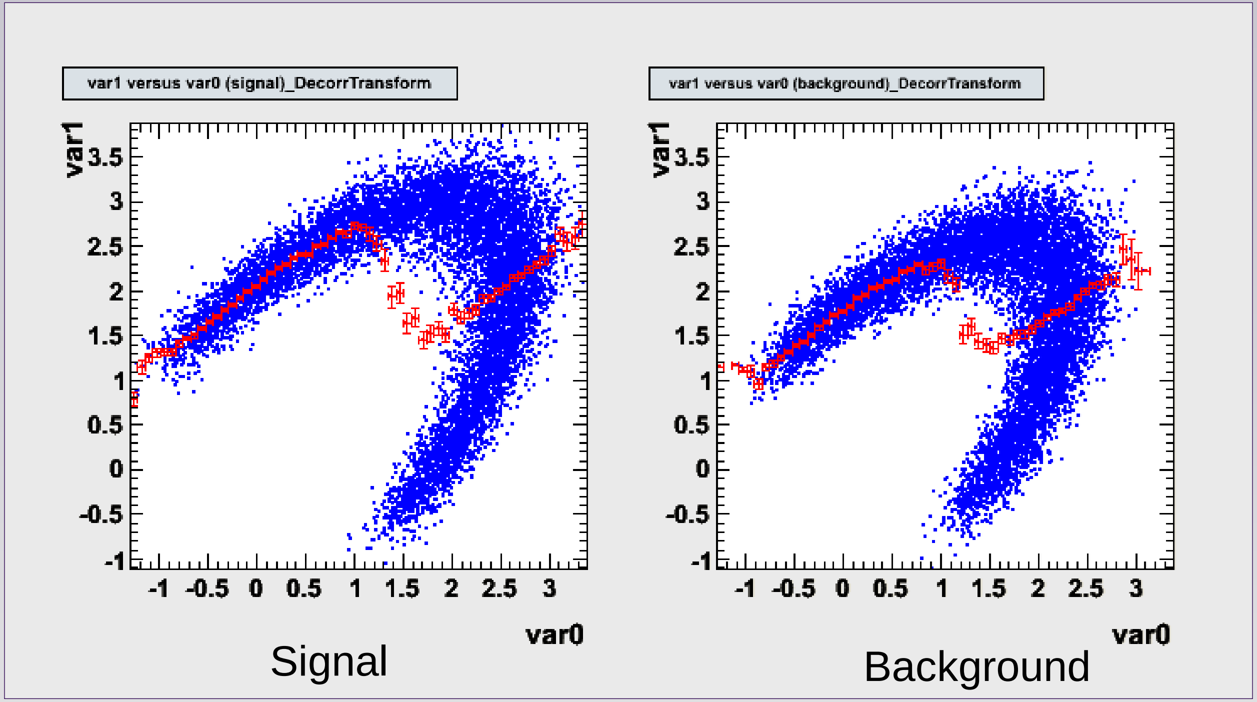
Background

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 ?

SQRT decorrelation



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

→ Perhaps “decorrelate” only a subspace!

How to Apply the Pre-Processing Transformation?

- Correlation (decorrelation): different for signal and background variables
- ☹ we don't know beforehand if it is signal or background.
  → What do we do?

→ for <u>likelihood ratio</u>, decorrelate signal and background independently

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

$\Downarrow$

signal transformation

background transformation

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

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

De-Correlation: Principal Component Analysis

- **PCA** (unsupervised learning algorithm)
  - **reduce dimensionality of a problem**
  - **find most dominant features in a distribution**

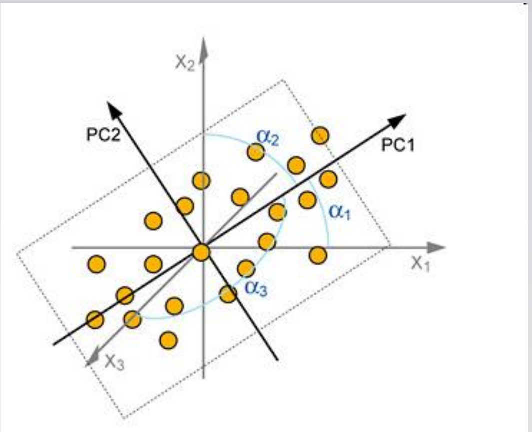
- **Eigenvectors of covariance matrix → "axis" in transformed variable space**
  - **large eigenvalue → large variance along the axis (principal component)**
    - **→ sort eigenvectors according to their eigenvalues**
    - **→ transform dataset accordingly**
    - **→ diagonalised covariance matrix with first "variable" → variable with largest variance**

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

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

"Gaussian-isation"

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

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

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)

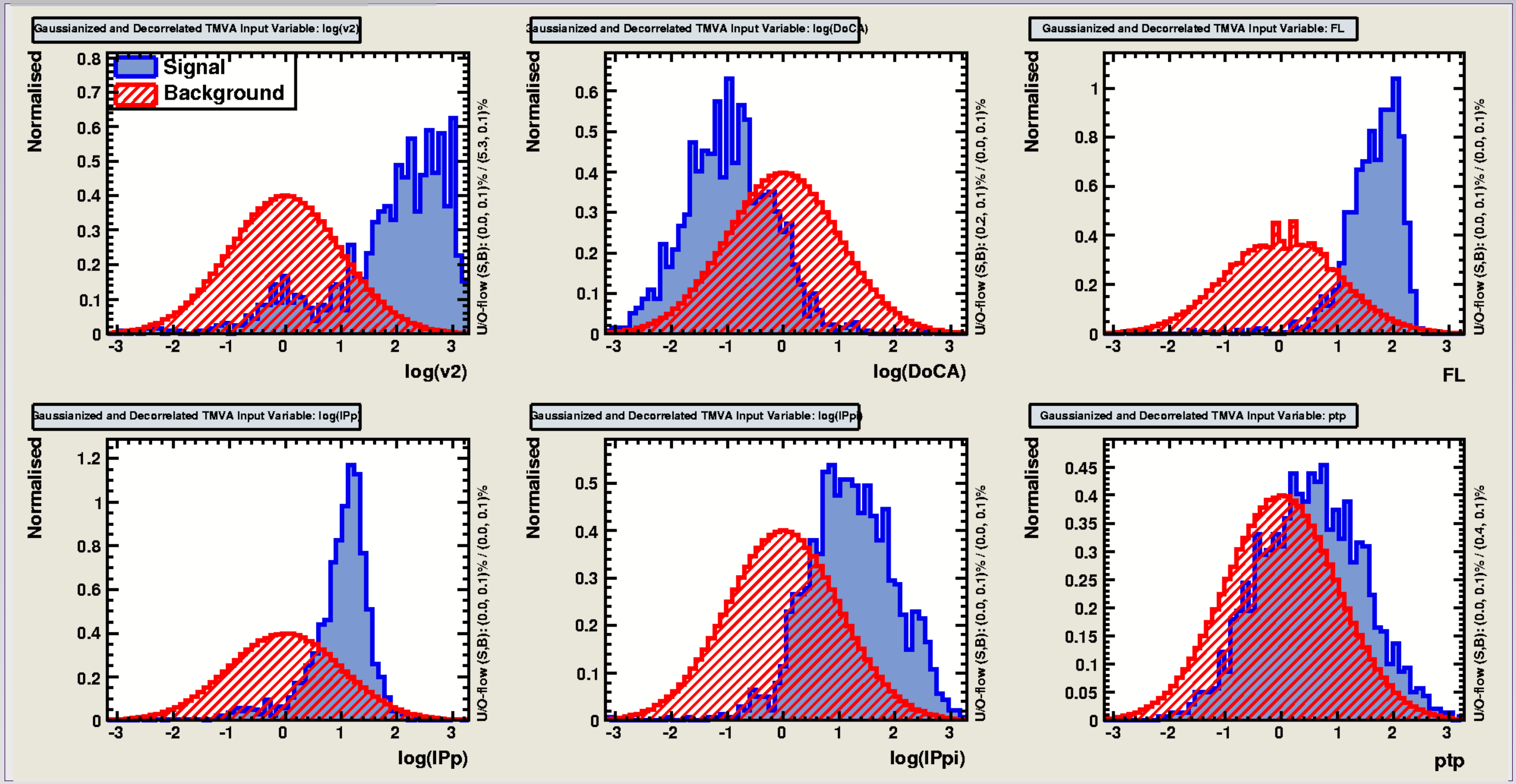
  - Second: make Gaussian via inverse error function: $\text{erf}\left(x\right) = \frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}dt$

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

  - Third: decorrelate (and "iterate" this procedure)

"Gaussian-isation"

Background - Gaussianised



We cannot simultaneously "Gaussianise" both signal and background !

Linear Discriminant and non linear correlations

assume the following non-linear correlated data:

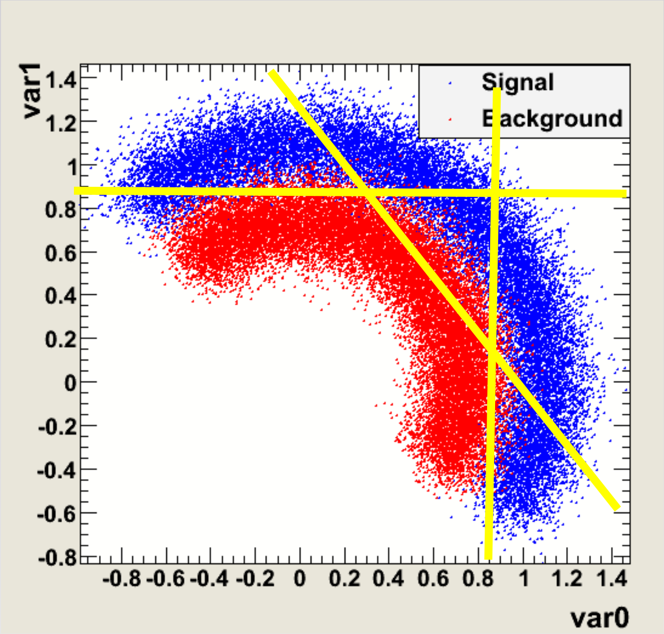
- **the Linear discriminant obviously doesn't do a very good job here:**

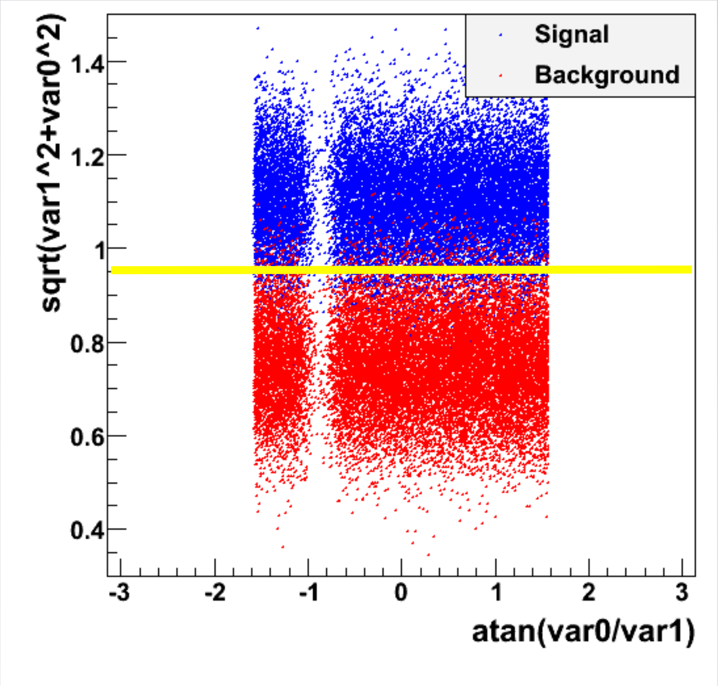
- **Of course, these can easily be de-correlated:**

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

$$var0' = \sqrt{var0^2 + var1^2}$$

$$var1' = a\tan\left(\frac{var0}{var1}\right)$$





Linear Discriminant with Quadratic input:

- **A simple 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: