Multivariate Data Analysis and Machine Learning in High Energy Physics (III)

Helge Voss (MPI–K, Heidelberg)

Graduierten-Kolleg , Freiburg, 11.5-15.5, 2009

Outline

- Summary of last lecture
- 1-dimensional Likelihood:
  - data preprocessing → decorrelation
- Recap of Fisher's Discriminant
- Neural Networks

What we've learned yesterday:

- Neyman Pearson: Actually he states the "obvious" if you think about it:
  - if you want to know to which probability an event with measured variables "x" is either signal or background, use the ratio between the "true probability" between observing an event with "x" from either signal or background.
    - Who is surprised that this is the "best you can do" ?

- Unfortunately we basically never have access to the "true probability density":
  - hence we try to either estimate the $PDF_S(x)$ and $PDF_B(x)$ using Kernel Density Estimator/Mulitdimensional likelihood (effectively averages of the PDF over regions of the variable space, derived from training events) → curse of dimensionality!!
  - or neglect correlations and use 1-dimensional PDF's → no problems with dimensionality
  - or try a different approach of directly determining hyperplanes in the feature space that separate signal and background events.

- There is no magic, you still need to "tune" parameters (e.g size of the region you want to average over in PDF estimate) in order to get good results.

Naïve Bayesian Classifier
"often called: (projective or 1D) Likelihood"

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

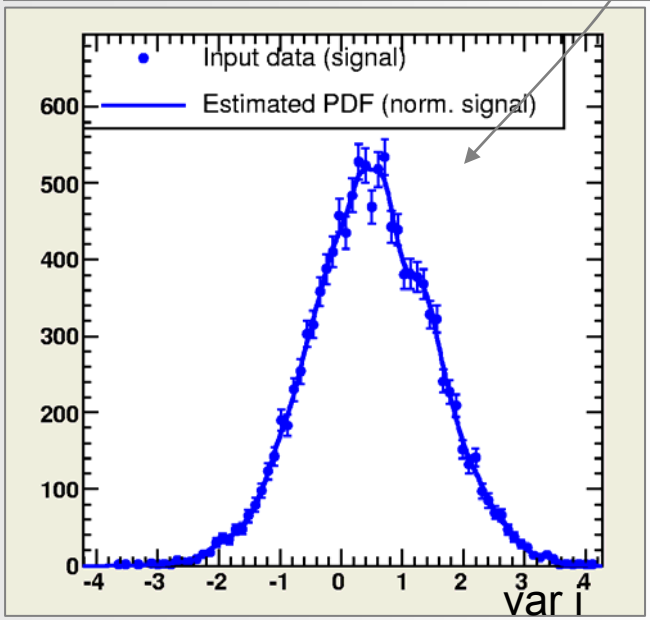
PDFs

discriminating variables

Likelihood ratio for event *event*

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

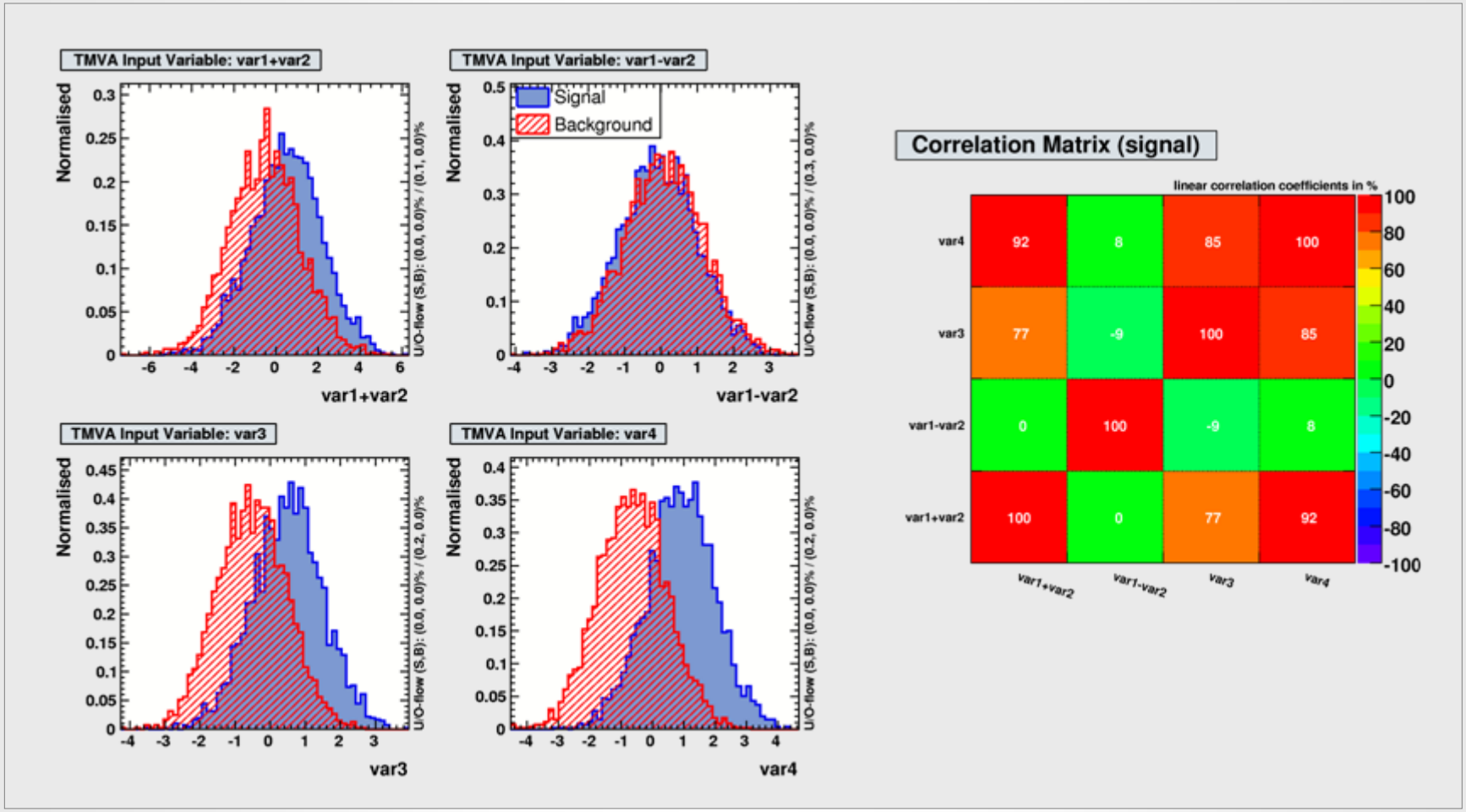
Classes: signal, background types



- 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?

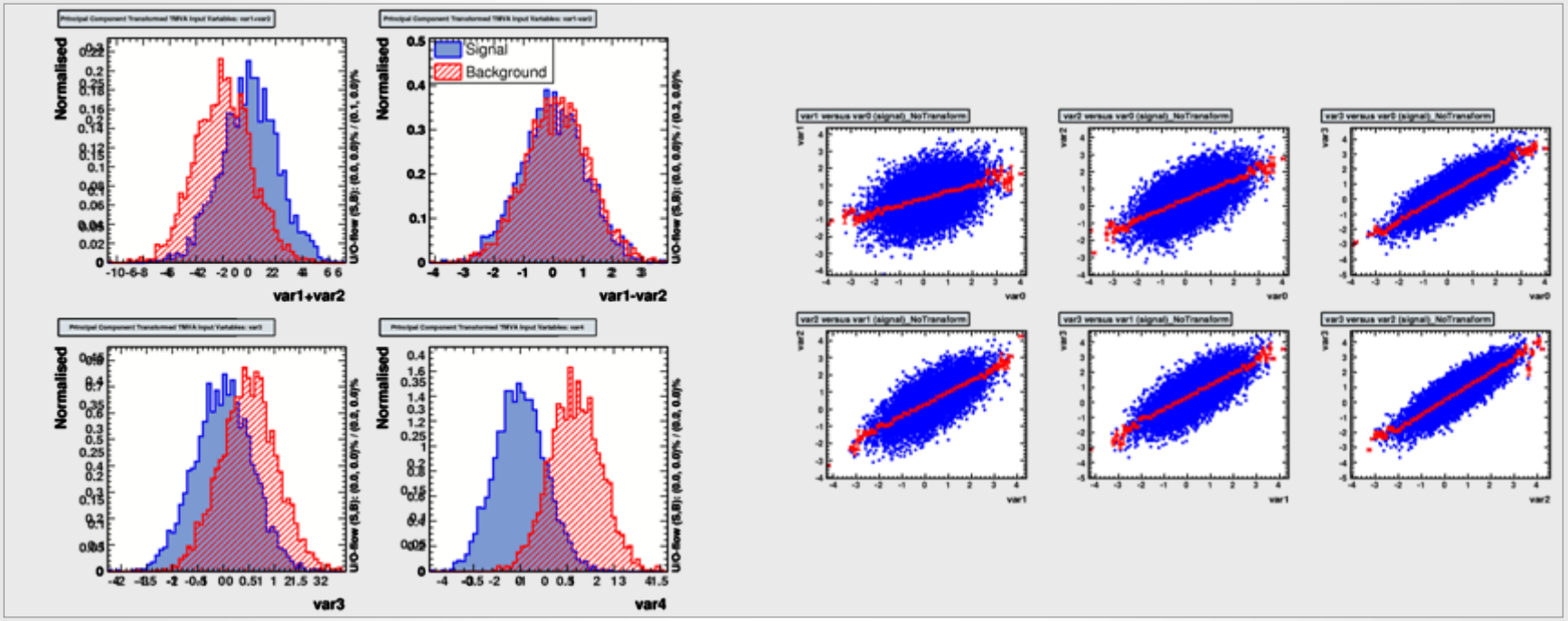
- Typically correlations are present: $C_{ij}=\mathrm{cov}[\,x_i\,,\,x_j\,]=E[\,x_i\,x_j\,]-E[\,x_i\,]E[\,x_j\,]\neq 0 \quad (i\neq j)$



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

Decorrelation

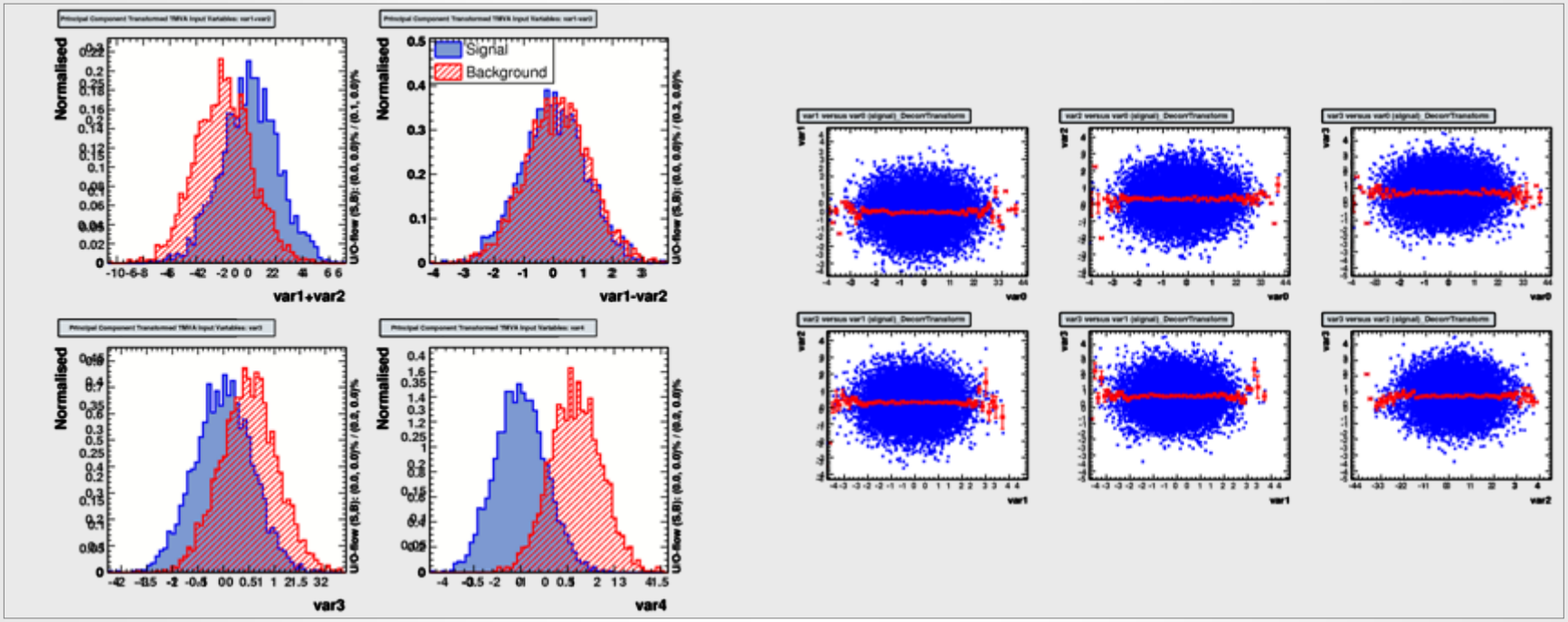
- Determine *square-root* C' of correlation matrix C, *i.e.*, $C = C'C'$
  - compute C' by diagonalising C: $D = S^T C S \;\Rightarrow\; C' = S\sqrt{D}S^T$
  - transformation from original (x) in de-correlated variable space (x') by: $x' = C'^{-1}x$



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

Decorrelation: Principal Component Analysis

- PCA is typically used to:
  - reduce 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.

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

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

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

$$\Downarrow$$

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

→ for other estimators, one need to decide on one of the two…

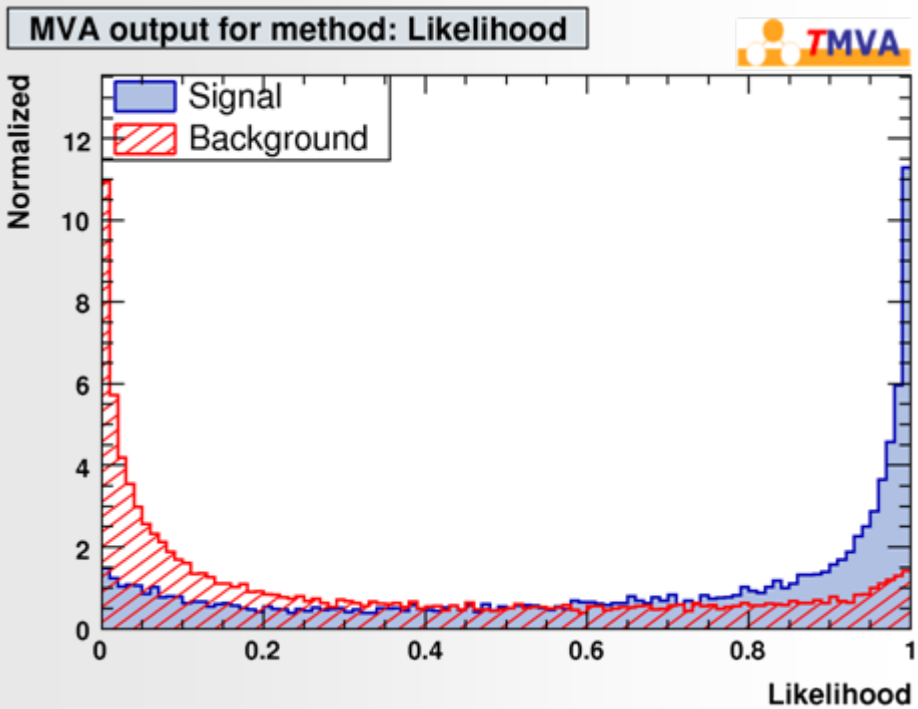
Decorrelation at Work

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

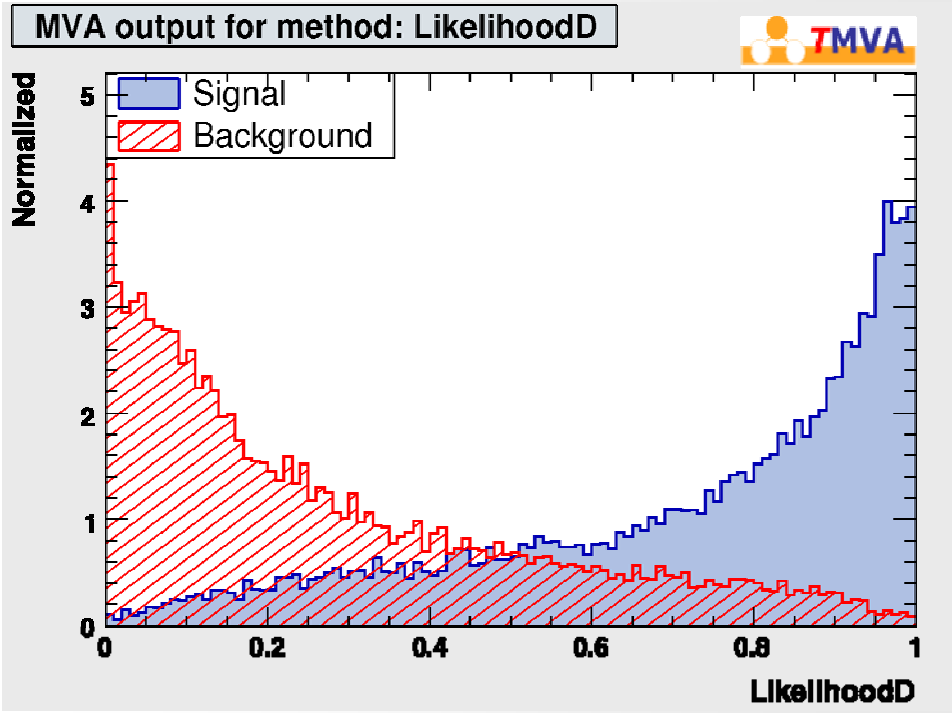
→1-D Likelihood on decorrelated sample give best possible performance

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

correlated variables:



after decorrelation



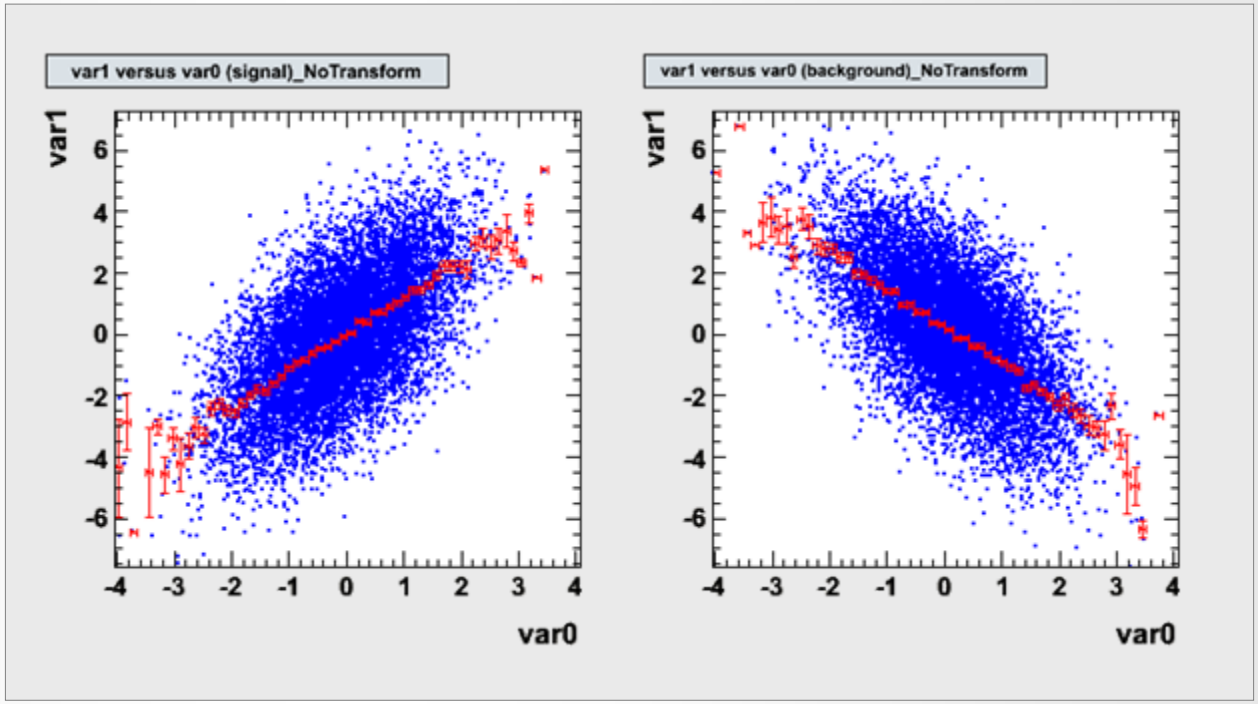
(note the different scale on the y-axis… sorry)

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

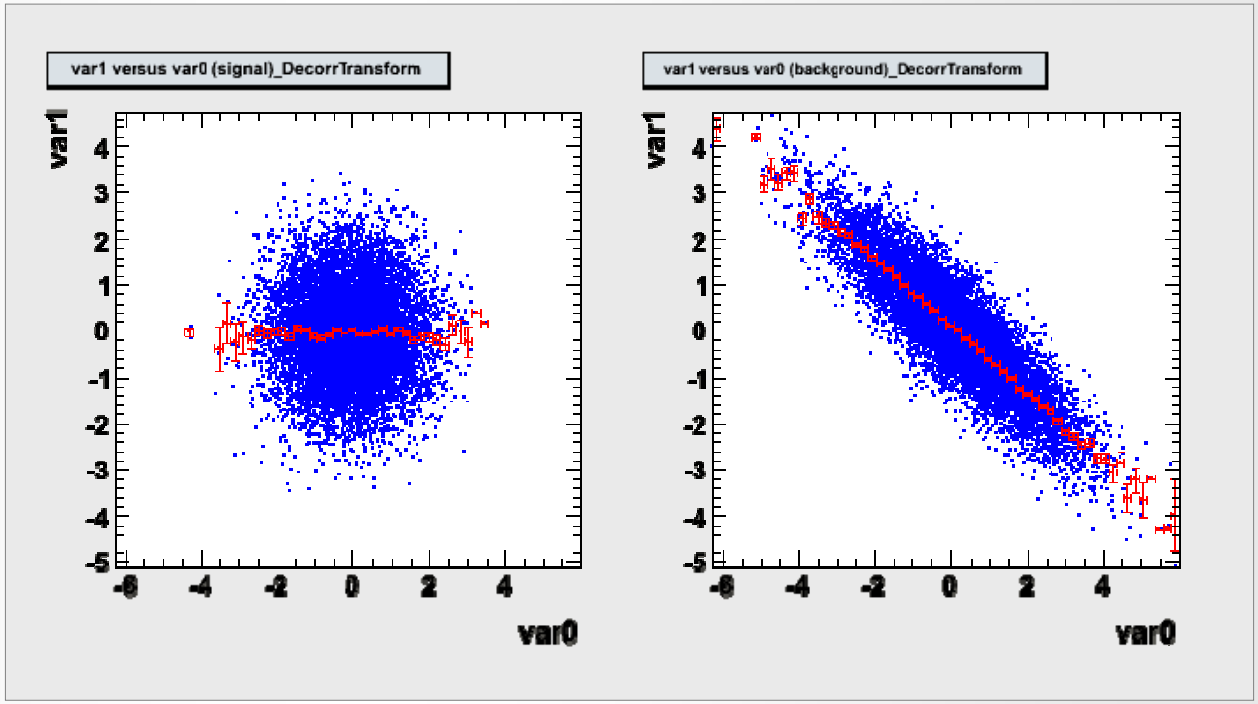
Limitations of the Decorrelation

- 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?

Original correlations

SQRT decorrelation



Signal

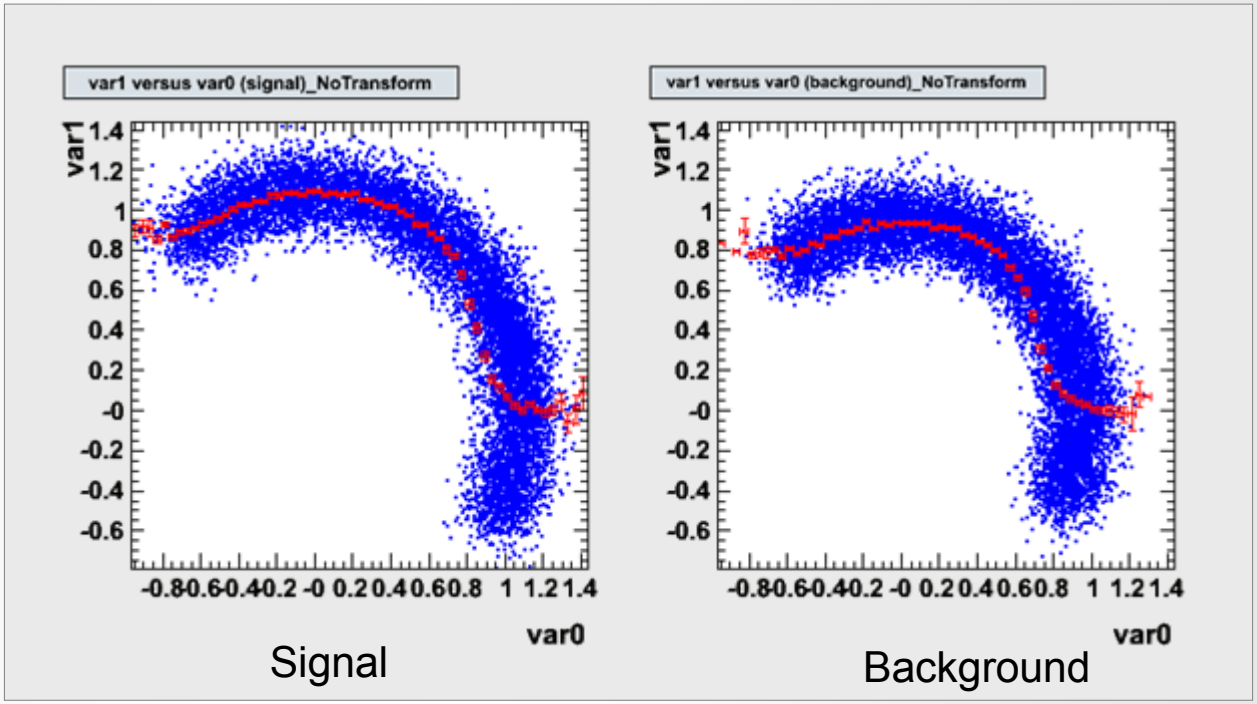
Background

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

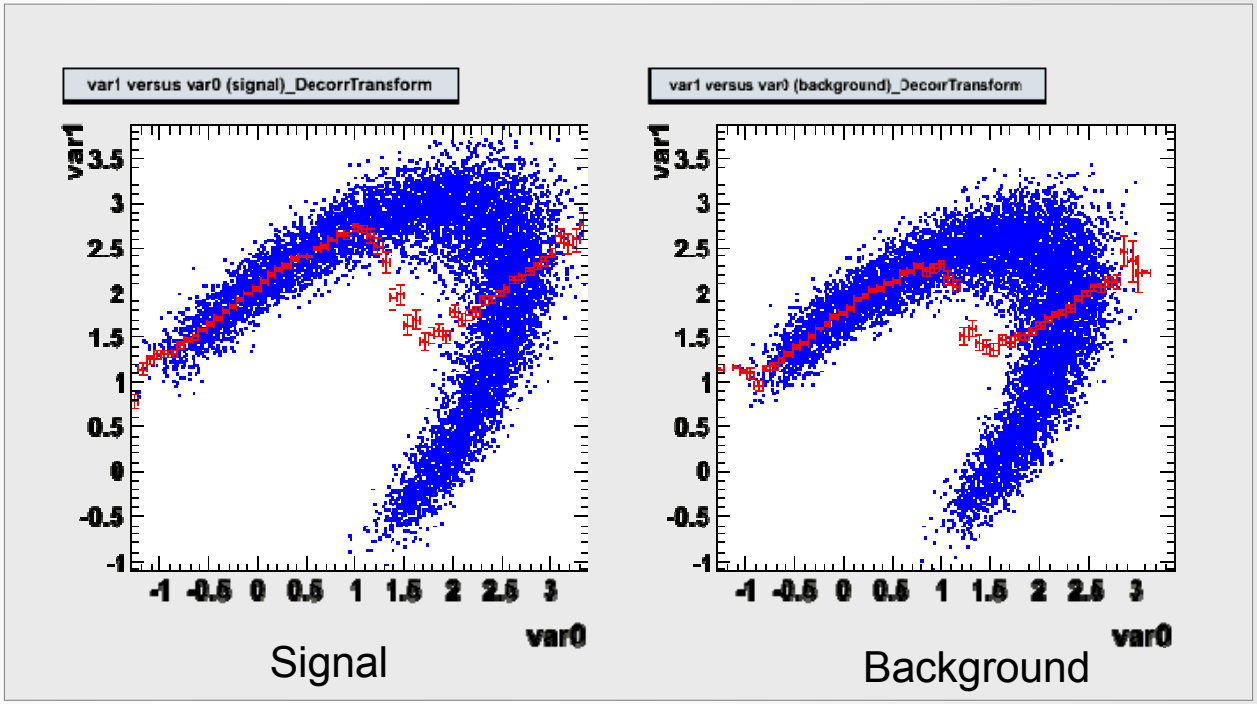
Limitations of the Decorrelation

- 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 ?

Original correlations

SQRT decorrelation



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

“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)

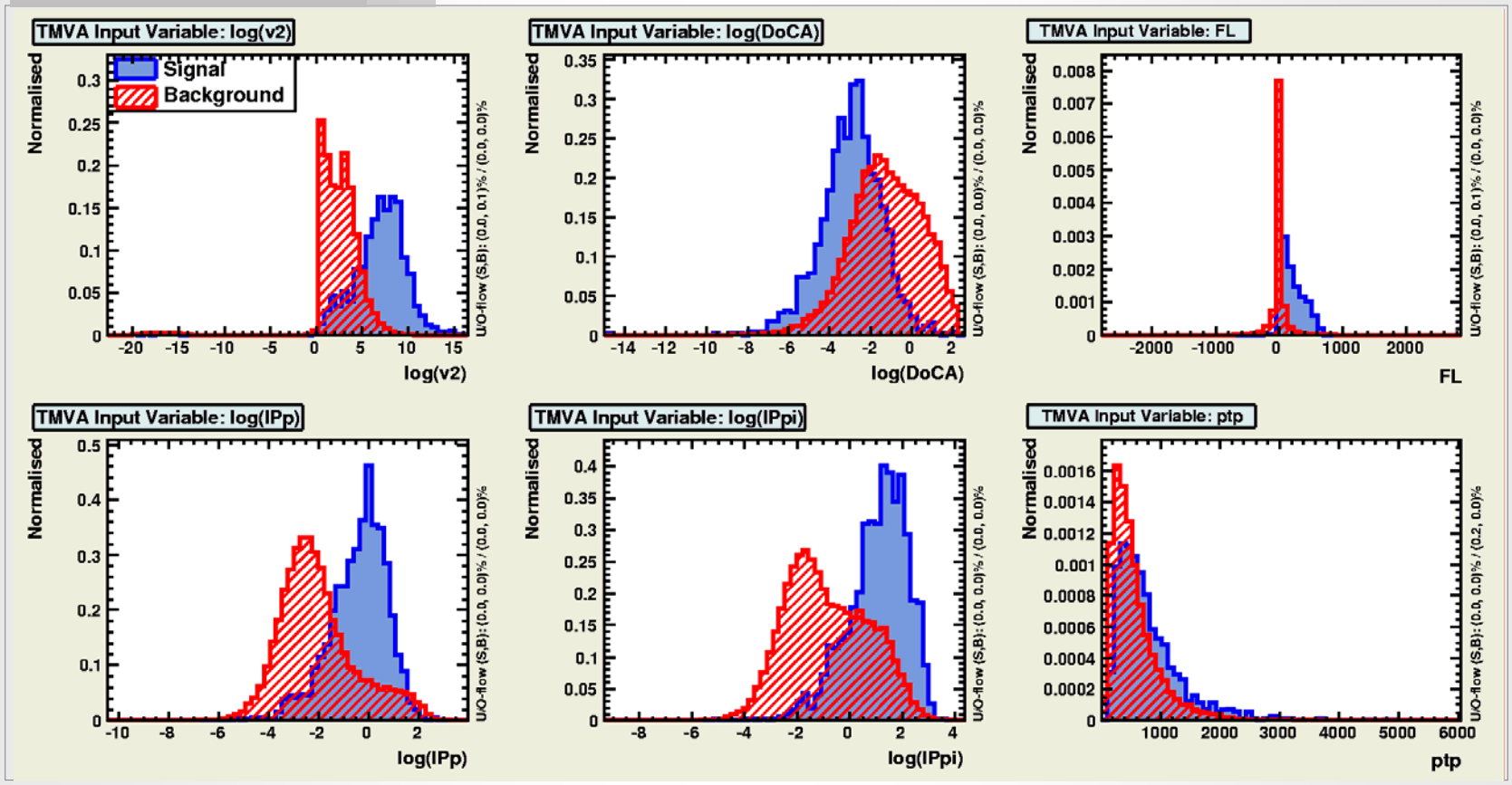
  - Second: make Gaussian via inverse error function: $\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$

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

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

“Gaussian-isation”

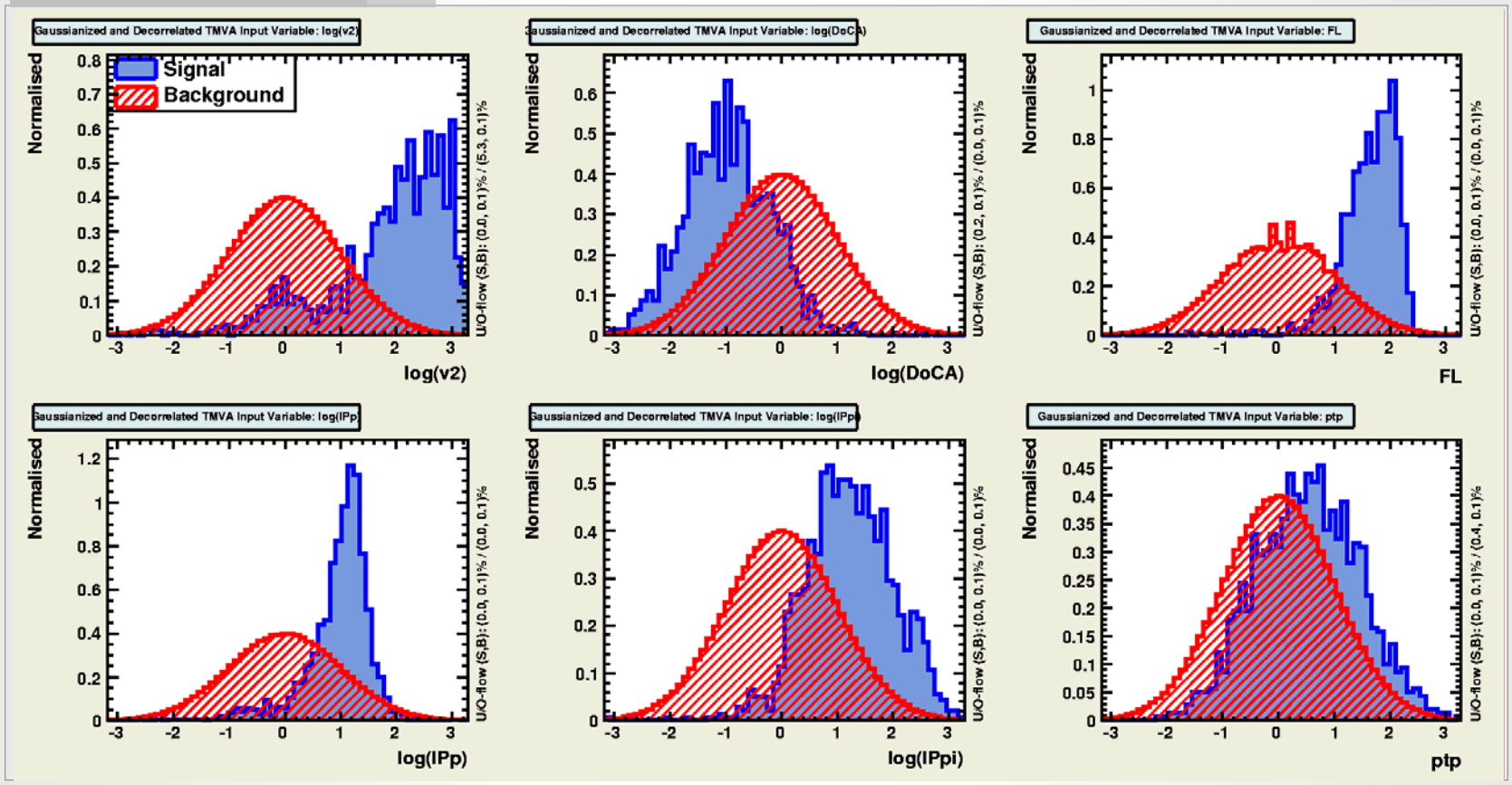
Background - Gaussianised



We cannot simultaneously G[illegible]oth signal and background ?!

“Gaussian-isation”

Background - Gaussianised



We cannot simultaneously G[illegible]oth signal and background ?!

Linear Discriminant

If non parametric methods like ‘k-Nearest-Neighbour” or “Kernel Density Estimators” suffer from

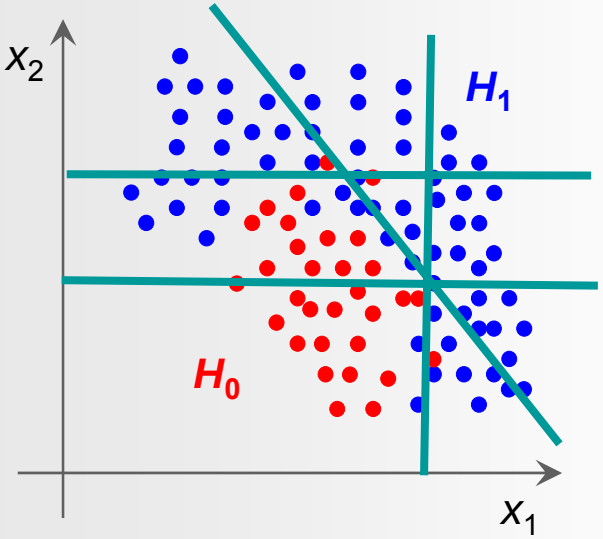
- lack of training data → “curse of dimensionality”
- slow response time → need to evaluate the whole training data for every test event

→ 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

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

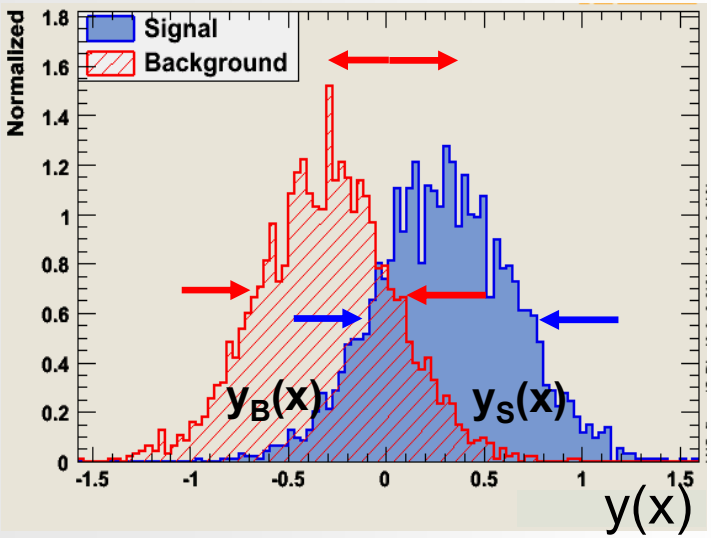


How do we determine the “weights” w that do “best”??

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:

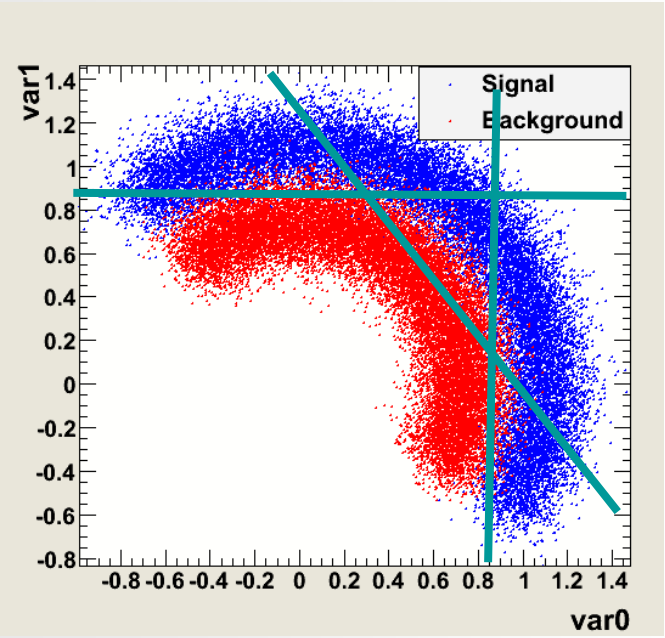
- the Linear discriminant obviousl doesn't do a very good job here:

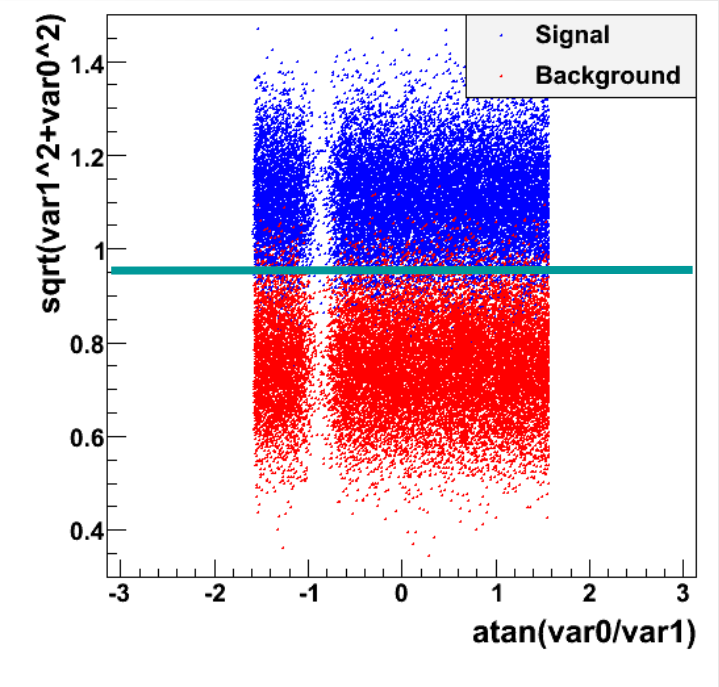
- Of course, these can easily de-correlated:

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

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

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





Linear Discriminant with Quadratic input:

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

while:
- var0
- var1

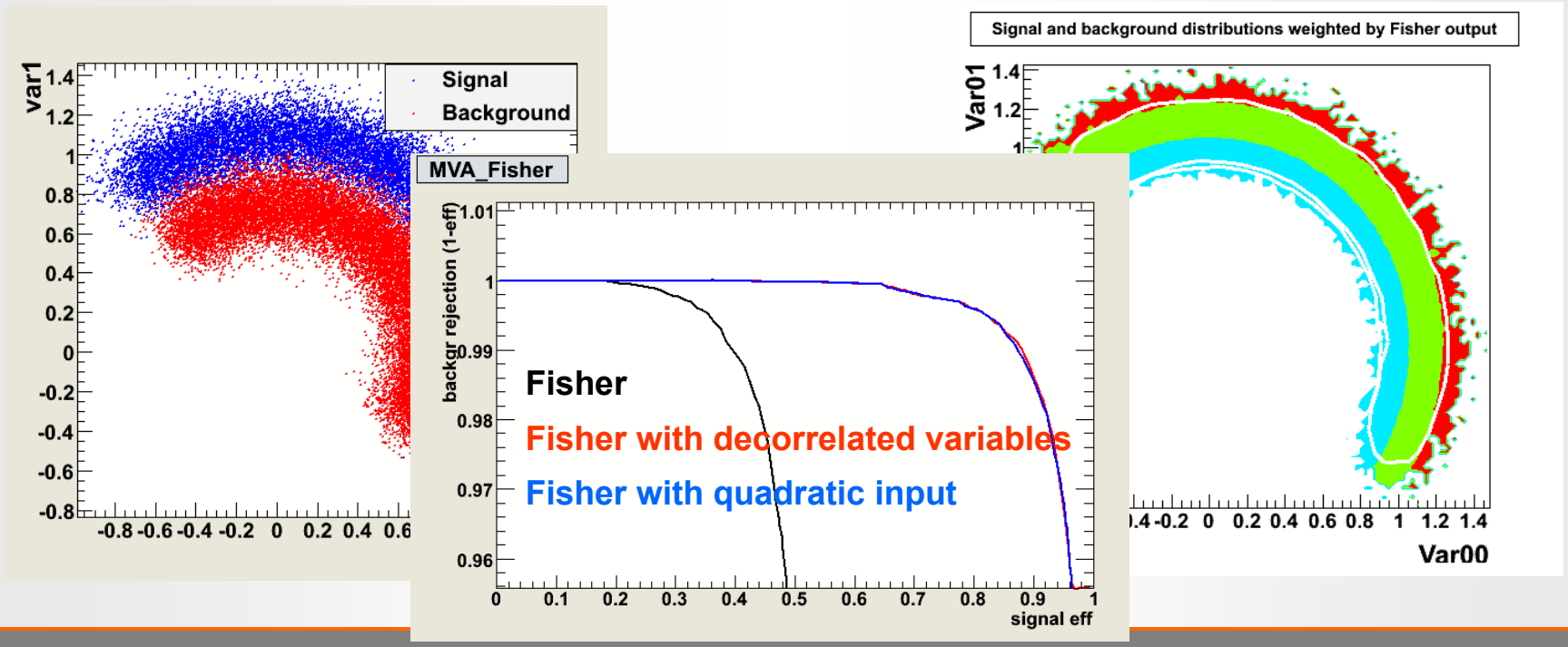
→ linear decision boundaries in var0,var1

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

→ quadratic decision boundaries in var0,var1

Performance of Fisher Discriminant:

with quadratic input:



Training Classifiers and Loss-Function

- All classifiers (KNN,Likelihood,Fisher) could be calculated and didn't require parameter fitting
- Other classifiers provide a set of "basis" functions (or model) that need to optimally adjusted to find the appropriate separating hyperplane(surface)
- Let $x \in R^n$ 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: $y(x):R^n \rightarrow R$:
- 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 the "arbitrary" non-linear decision boundaries, y(x) needs to be constructed in "any" non-linear fashion

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

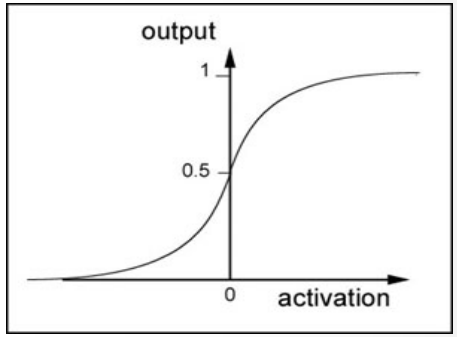
- Think of $h_i(x)$ as a set of "basis" functions
- If h(x) is sufficiently general (i.e. non linear), a linear combination of "enough" basis function should allow to describe any possible discriminating function y(x)

K.Weierstrass Theorem: proves just that previous statement.

Imagine you chose $h_i(x)$ to be such that:

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

$$A(x)=\frac{1}{1-e^{x}}\text{: the sigmoid function}$$

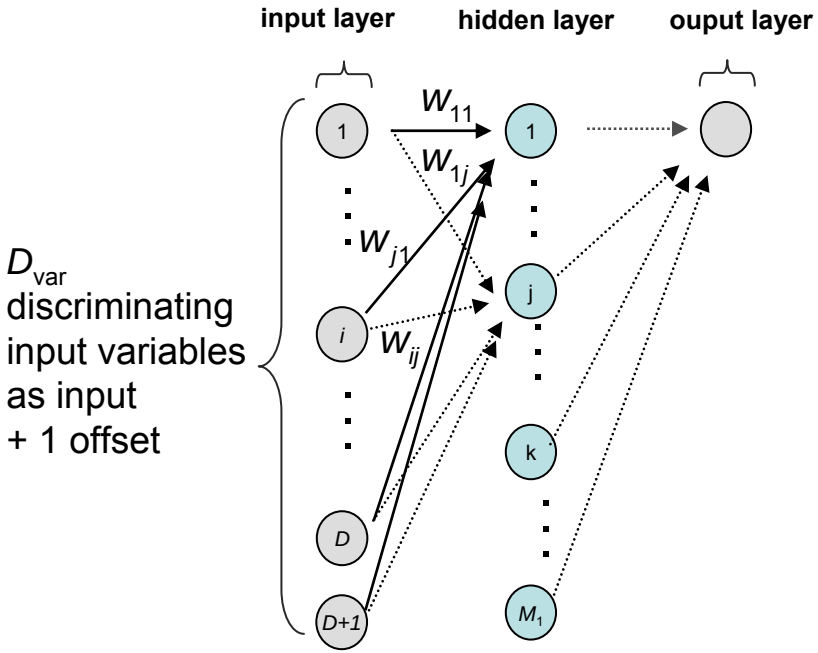


y(x) =
a linear combination of
non linear functions of
linear combination of
the input data

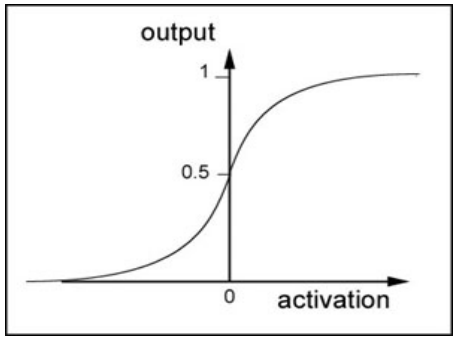
Now we "only" need to find the appropriate "weights" w

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:

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

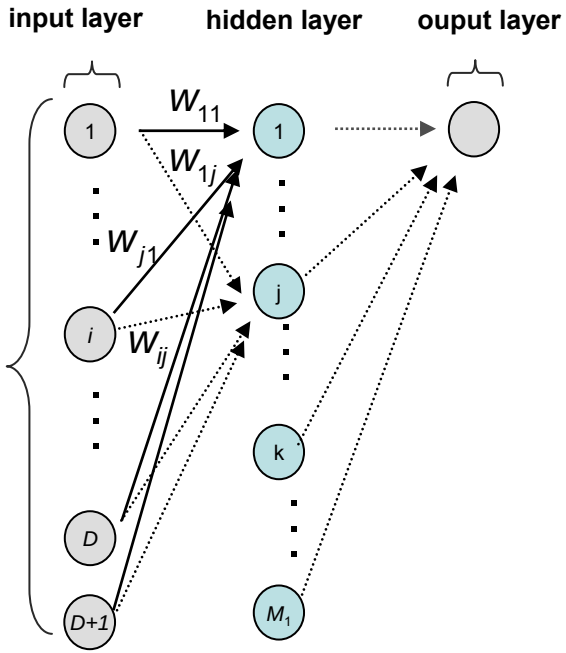
or tanh

- 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

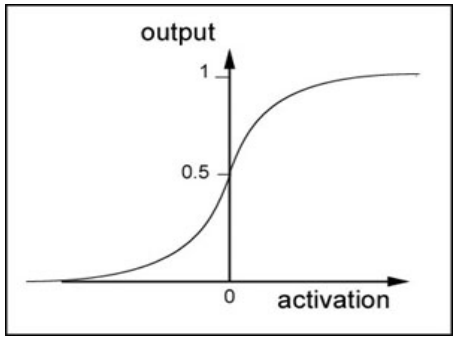
Neural Networks: Multilayer Perceptron MLP

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

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

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

i.e. use usual "sum of squares"

how do we adjust ?

- minimize Loss function:

$$L(w) = \sum_{i}^{\text{events}} (\underbrace{y(x_i)}_{\substack{\textbf{predicted}\\ \textbf{event type}}} - \underbrace{y(C)}_{\substack{\textbf{true}\\ \textbf{event type}}})^2 \quad \text{where} \quad y(C) = \begin{cases} 1 \text{ for } C = \text{signal} \\ 0 \text{ for } C = \text{backgr.} \end{cases}$$

- 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
  - → <u>back propagation</u> (learn from experience, gradually adjust your perception to match reality
  - → <u>online learning</u> (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

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

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

- for weights connected to output nodes

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

- for weights connected to output nodes

  … a bit more complicated formula

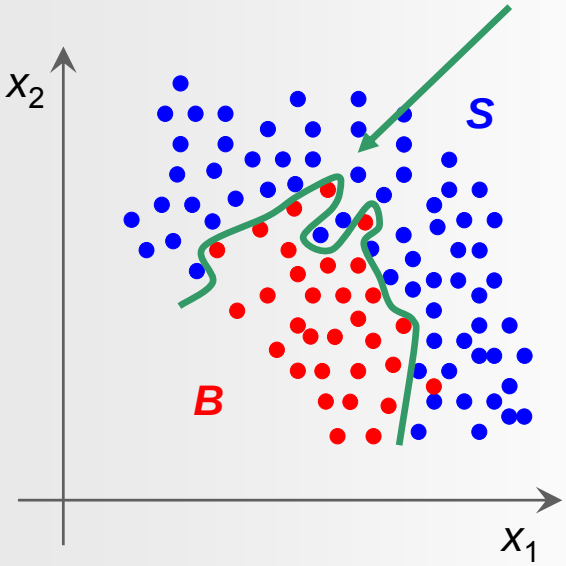
- note: all these gradients are easily calculated from the training event

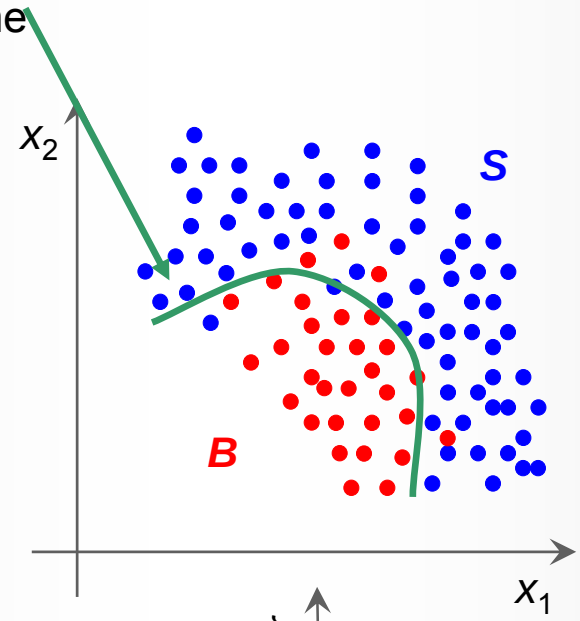
- training is repeated n-times over the whole training data sample. how often ??

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!

Overtraining

- training is repeated n-times over the whole training data sample. how often ??
- it seems intuitive that this boundary will give better results in another statistically independent data set than that one





→ stop training before you learn statistical fluctuations in the data

→ verify on independent "test" sample

→ possible overtraining is concern for every "tunable parameter" α of classifiers: Smoothing parameter, n-nodes…

Cross Validation

- many (all) classifiers have tuning parameters "α" that need to be controlled against overtraining
  - number of training cycles, number of nodes (neural net)
  - smoothing parameter h (kernel density estimator)
  - ….
- the more free parameter a classifier has to adjust internally → more prone to overtraining
- more training data → better training results
- division of data set into "training" and "test" sample reduces the "training" data

Cross Validation: divide the data sample into say 5 sub-sets

Train	Train	Train	Train	Test

- train 5 classifiers: $y_i(x,\alpha)$: i=1,..5,
- classifier $y_i(x,\alpha)$ is trained without the i-th sub sample
- calculate the test error: $$CV(\alpha) = \frac{1}{N_{events}} \sum_{k}^{events} L(y_i(x_k,\alpha)) \quad L: \text{loss function}$$
- choose tuning parameter α for which $CV(\alpha)$ is minimum and train the final classifier using all data

What is the best Network Architecture?

- Theoretically a single hidden layer is enough for any problem, provided one allows for sufficient number of nodes. (K.Weierstrass theorem)

- "Relatively little is known concerning advantages and disadvantages of using a single hidden layer with many nodes over many hidden layers with fewer nodes. The mathematics and approximation theory of the MLP model with more than one hdden layer is not very well understood ……"

  …."nonetheless there seems to be reason to conjecture that the two hidden layer model may be significantly more promising than the single hidden layer model"

  A.Pinkus, "Approximation theory of the MLP model with neural networks", Acta Numerica (1999),pp.143-195

  (Glen Cowan)

- Typically in high-energy physics, non-linearities are reasonably simple,
  - → in many cases 1 layer with a larger number of nodes should be enough
  - → but it might well be worth trying more layers (and less nodes in each layer)