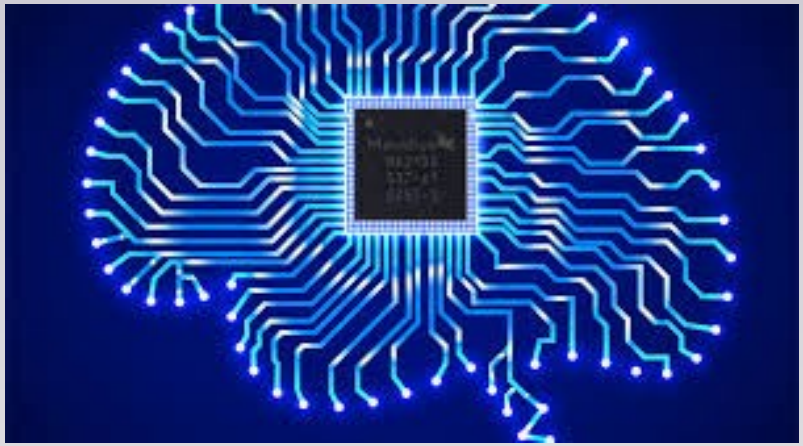
Multivariate Analysis (Machine Learning) … in HEP

IN2P3 School of Statistics 2016



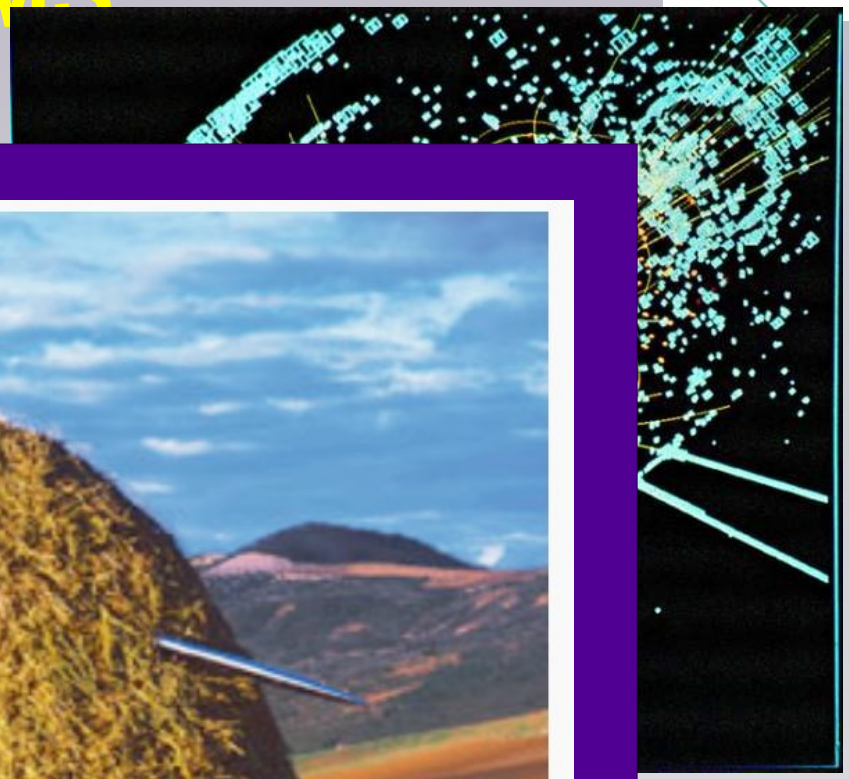
Helge Voss

MAX-PLANCK-INSTITUT FÜR KERNPHYSIK IN HEIDELBERG

HEP Experiments: Simulated Higgs Event in CMS

Max-Planck-Institut für Kernphysik

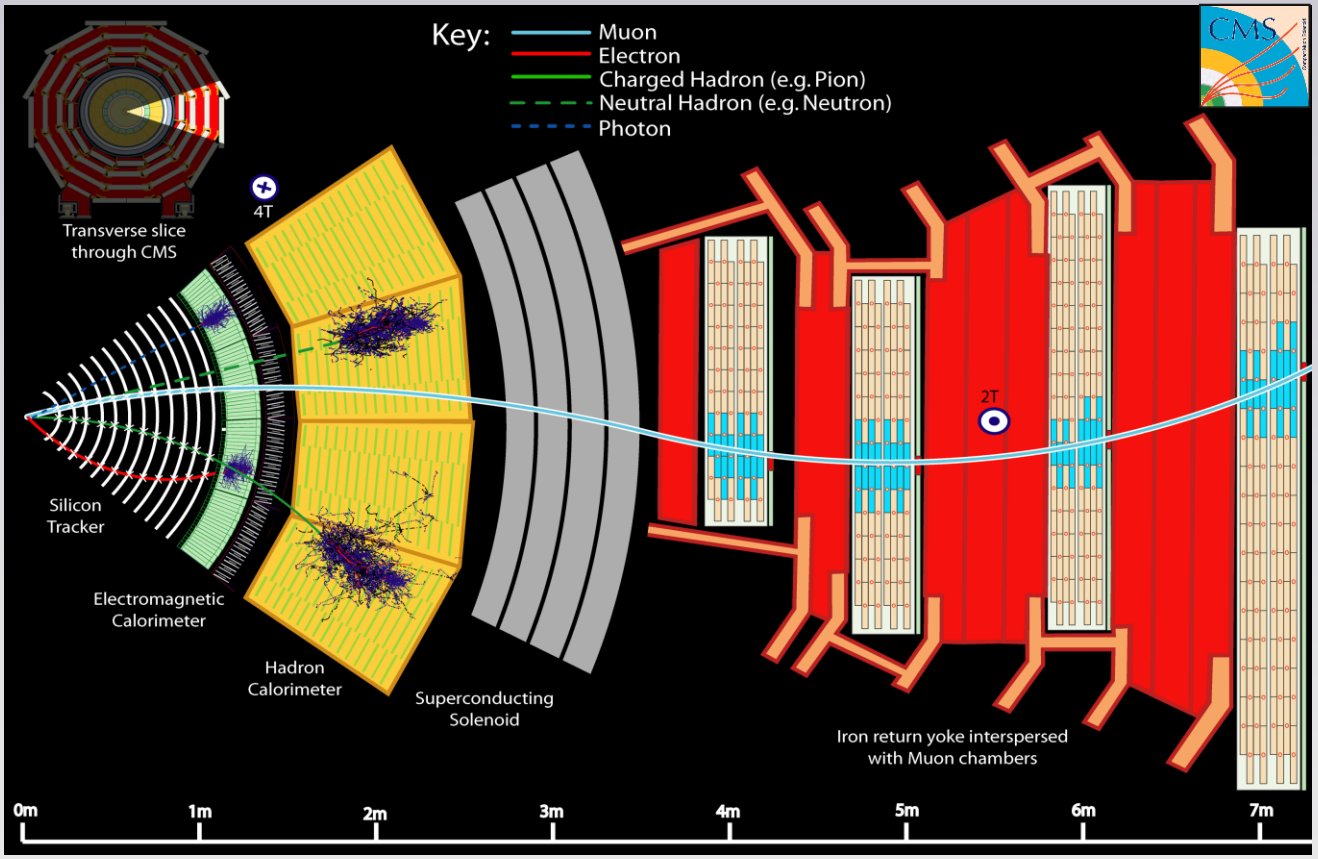
- That's how a "typical" higgs event looks like: (underly



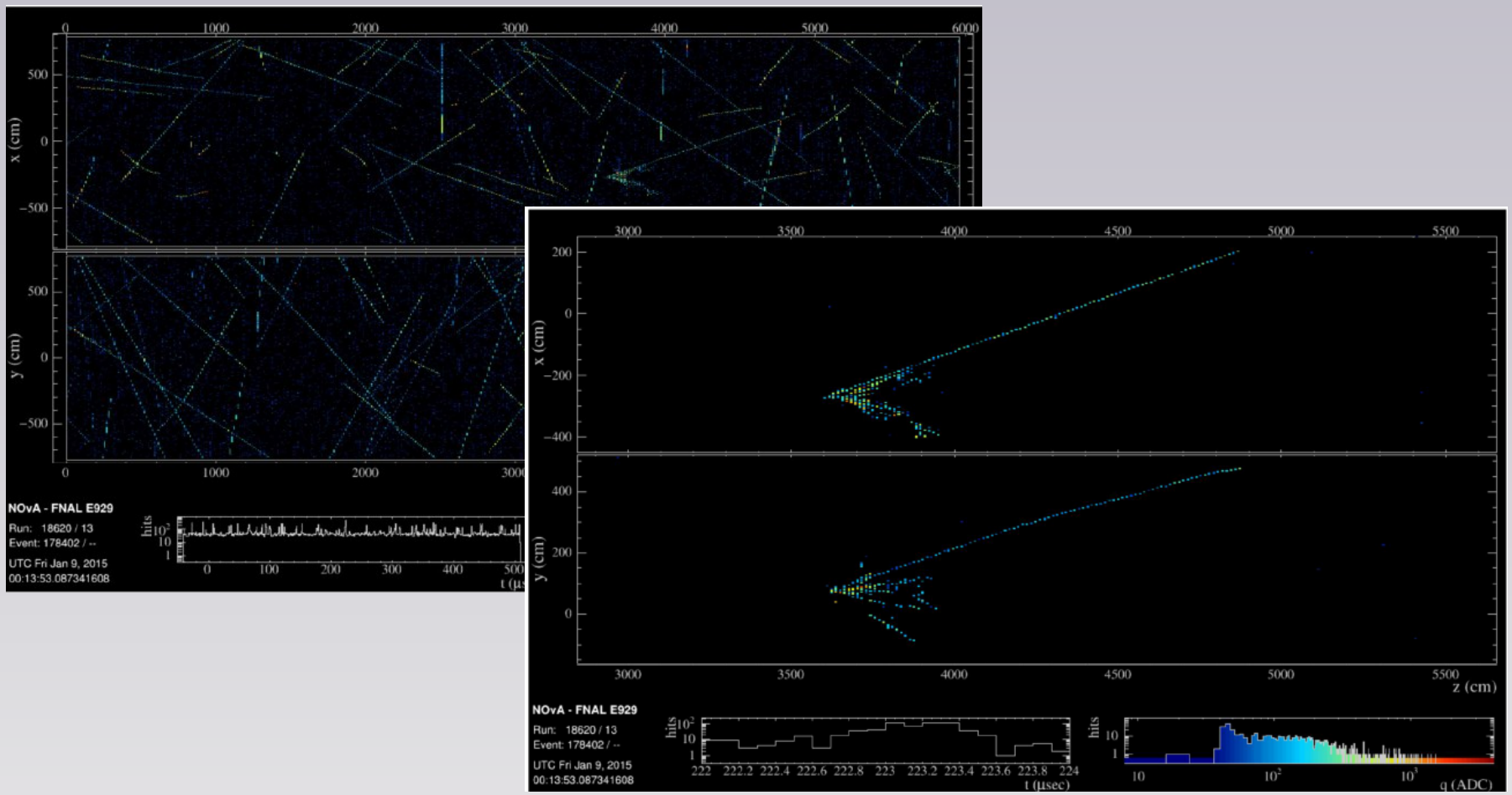
n a tiny

HEP Experiments: Event Signatures in the Detector

- the needle in the hay-stack is already "one piece" … but:
  - (Higgs-) particles need to be reconstructed from decay products
  - decay products need to be reconstructed from detector signatures
  - etc..

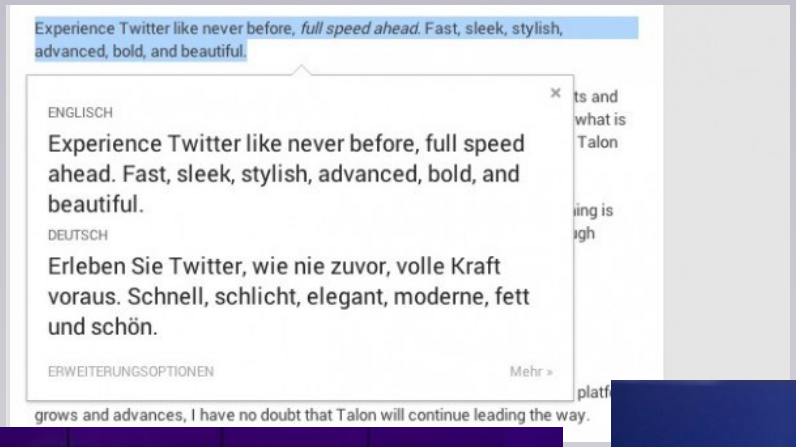


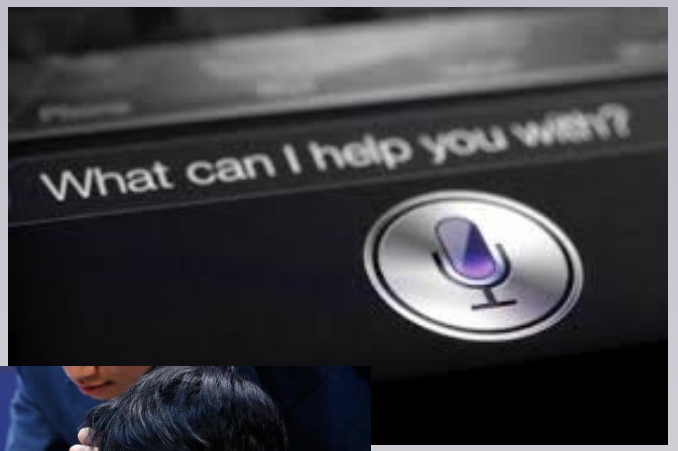
NOνA long baseline oscillation exp. $(\nu_\mu)/\nu_e$ (dis-)/appearance

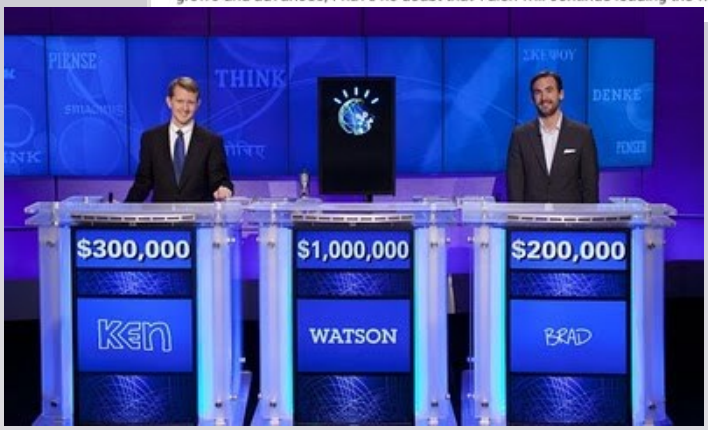


O(100k) background, O(100) ν_μ , O(10) ν_e per year

Machine Learning ‘elsewhere’



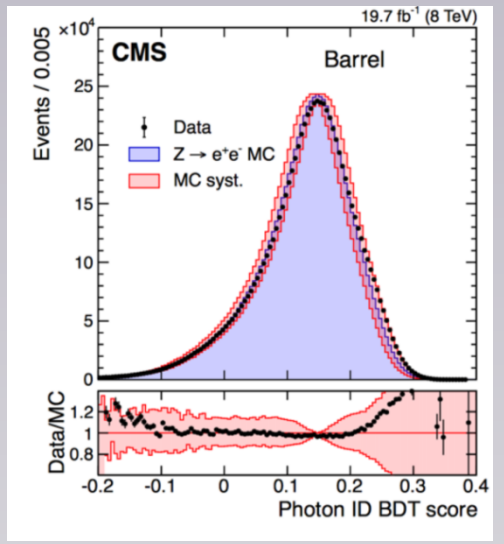




…. is ‘everywhere’

Outline

Max-Planck-Institut für Kernphysik



- **What is: Machine Learning (ML) & Multivariate Analysis/Technique (MVA)**
  - **Basics (classification, regression)**
  - **ROC-curve**
  - **generative vs predictive models**
- **MVA/ML algorithms**
  - **Naïve Basian, KNN,**
  - **Linear discriminators, SVM**
  - **model fitting – gradient decent and loss function**
  - **General comments about MVAs**
  - **BDT (Yann Coadou) this afternoon)**
  - **Neural Networks (tomorrow)**

What is Machine Learning

- *"[Machine Learning is the] field of study that gives computers the ability to learn without being explicitly programmed."* Arthur Samuel (1959)

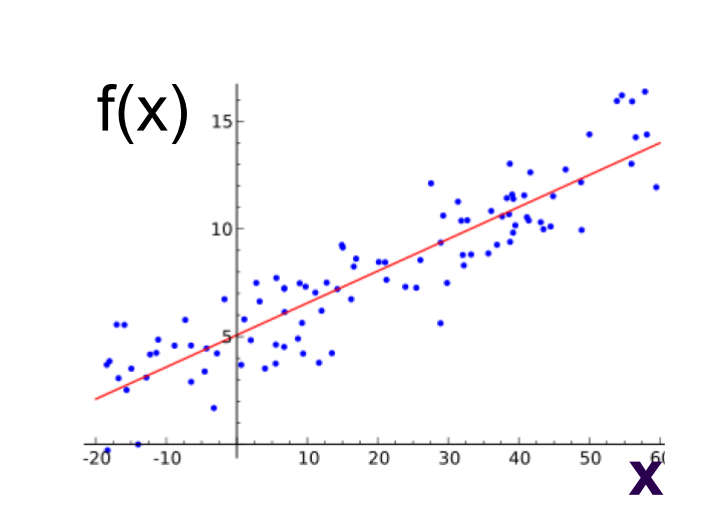
- *"A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E."* Tom Mitchell, Carnegie Mellon University (1997)

I suggest: forget about 'fancy definitions':

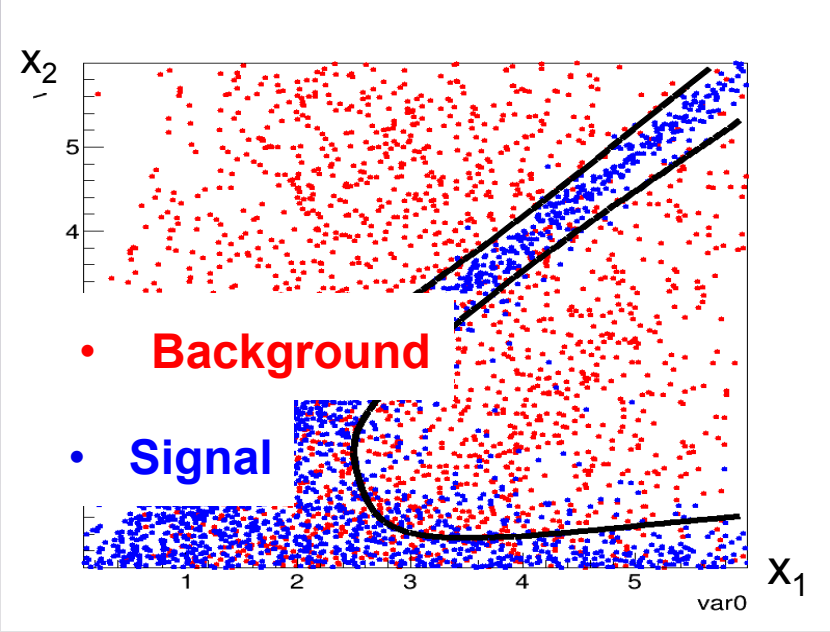
'understanding/modeling your data' …
and if you cannot do it in multi-dimensions on "analytic first principles" let the computer help ☺

What are Multivariate Techniques

→ Many things … starting from “linear regression” …



to multivariate event classification



→ or w/o prior ‘analytic’ model

→ typically “multivariate”

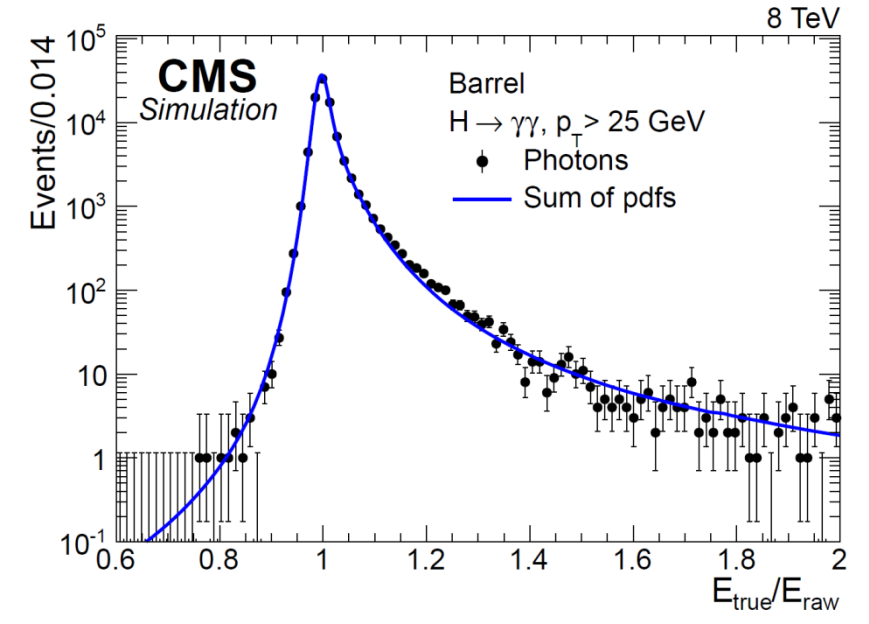
- Parameters depend on the ‘joint distribution’ $f(x_1, x_2)$

- ‘learning from experience’ → known data points

Machine Learning - Multivariate Techniques

- fitted (non-)analytic function may approximate:
  - target value → 'regression'

    (e.g. calorimeter calibration/correction function)



MC sample: γ +jets

- Raw energy in crystals, η, Φ
- Cluster shape variables
- Local cluster position variables

  (energy leakage)
- Pile-up estimators

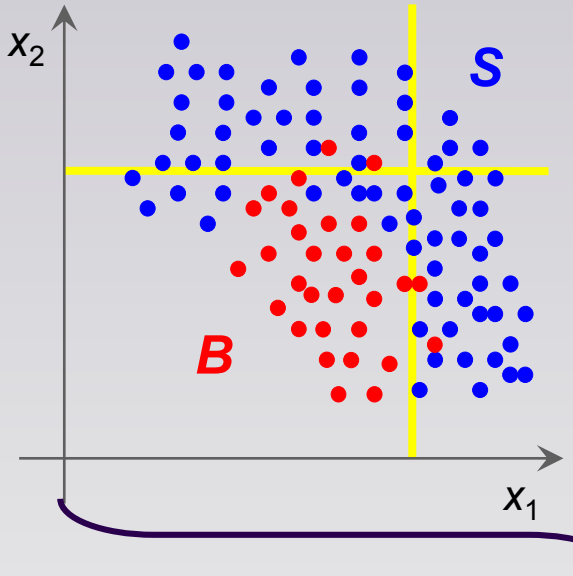
→ predict energy correction (i.e. parameters in crystal-ball: pdf for energy measurement)

Event Classification

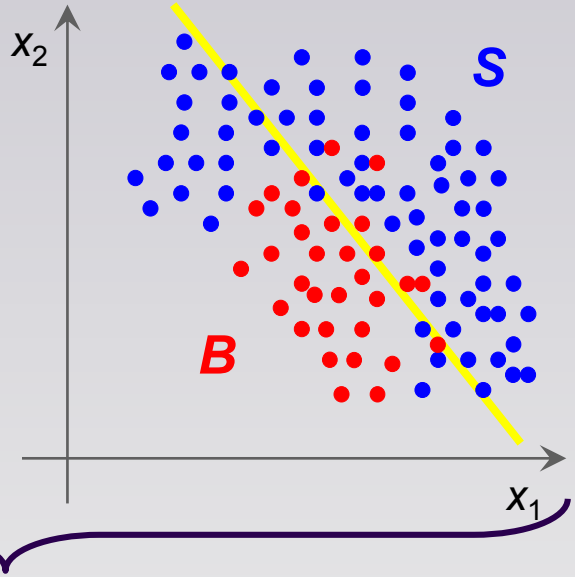
- ***Signal*** **and** ***Background***
  - **discriminating observed variables x_1, x_2, …**

  → decision boundary ?

Rectangular cuts?

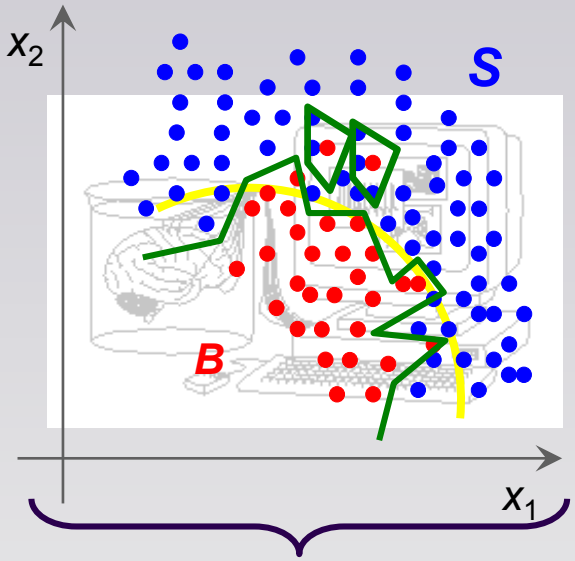


A linear boundary?



Low variance (stable), high bias methods

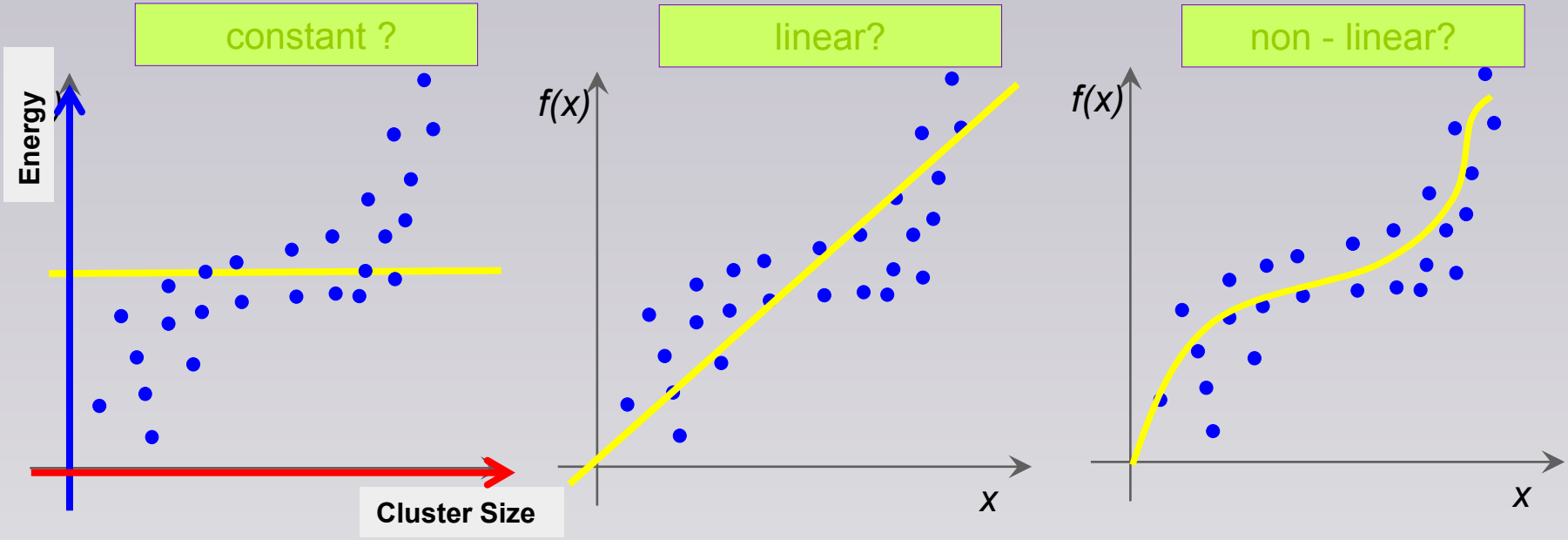
A nonlinear one?



High variance, small bias methods

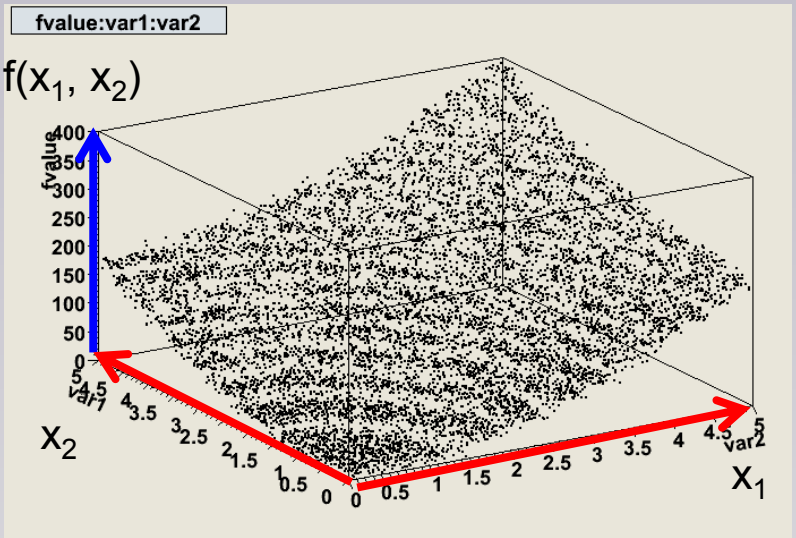
Regression

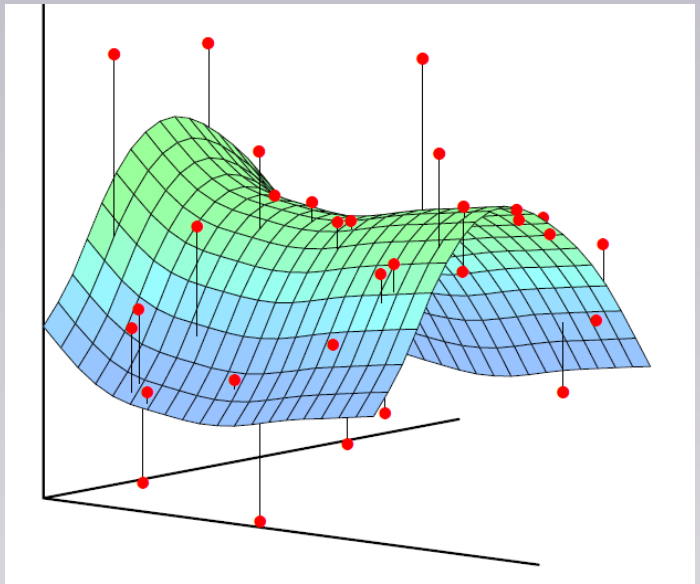
- 'known measurements" → model "functional behaviour"
- e.g. : photon energy as function "D"-variables: ECAL shower parameters + …



- 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





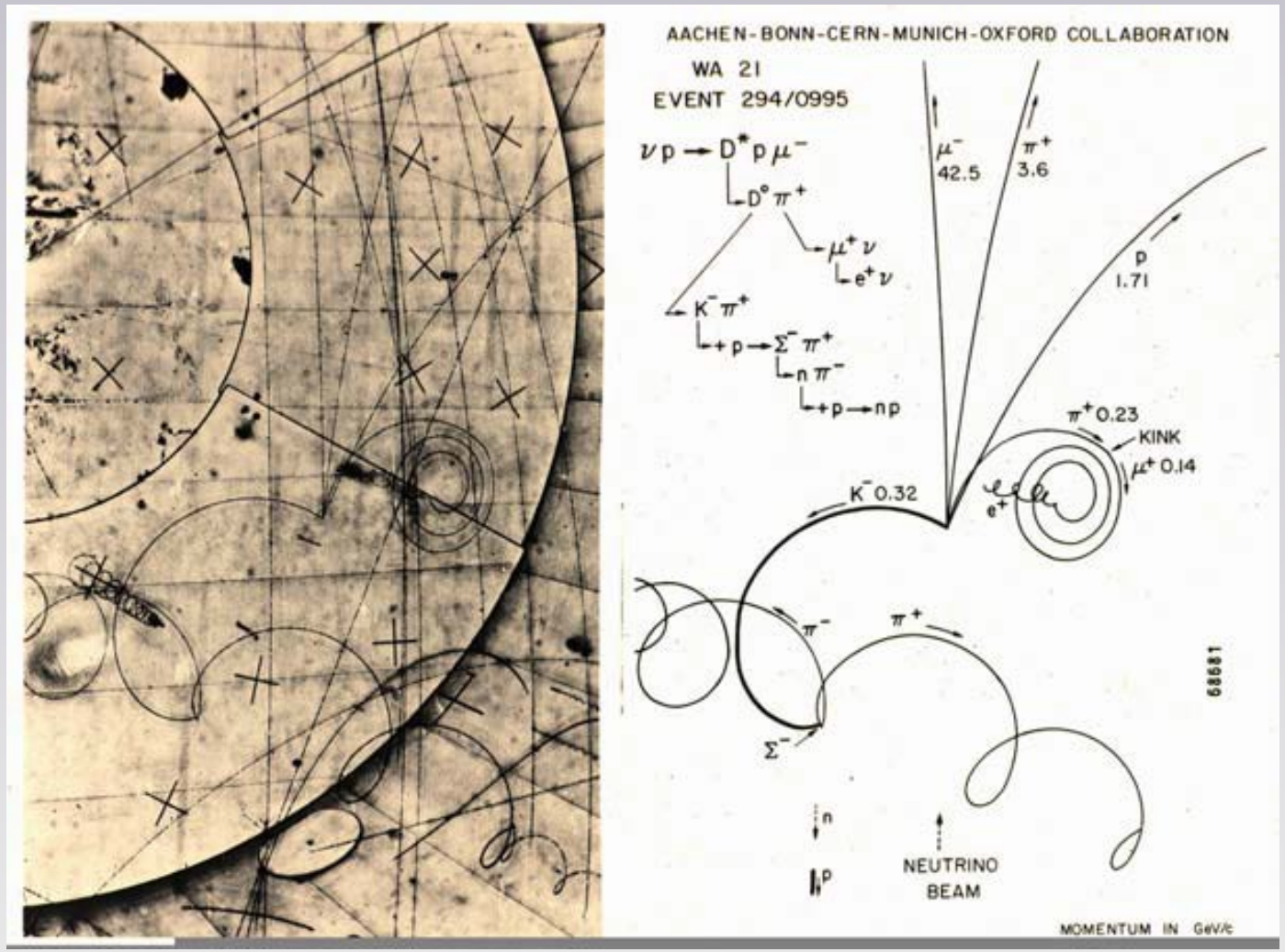
- "standard" regression → fit a known analytic function
  - *e.g.* $f(\mathbf{x}) = ax_1^2+bx_2^2+c$
- BUT most times: don't have a reasonable "model" ? → need something more general:
  - *e.g.* piecewise defined splines, kernel estimators, decision trees to approximate f(x)

Note: we are not interested in the 'fitted parameter(s)', it is not: "Newton deriving F=m·a"
→ just provide prediction of function values f(x) for new measurements x

HEP: Everying startet Multivariate

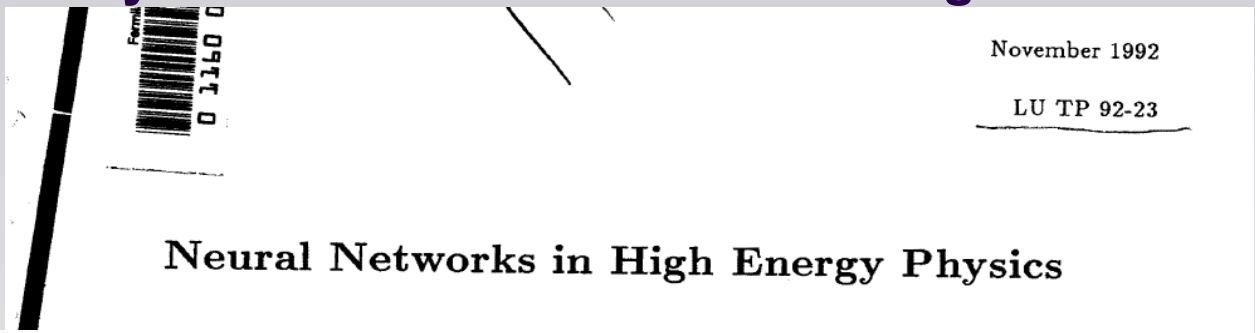
Max-Planck-Institut für Kernphysik

- intelligent “Multivariate Pattern Recognition” used to identify particles



Machine Learning in HEP

- Later: 'MVAs got out of fashion' → replaced by
  - if (..) then … ; → 'cuts on individual variables'
    - Fear of "black box fears" or because it is easier to program?
  - Some 'Fisher discriminants', Naïve Bayesian (Likelihood) even NNs…. have always been around before becoming mainstream again ☺



November 1992

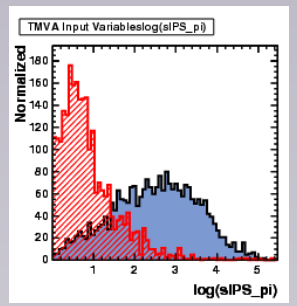
LU TP 92-23

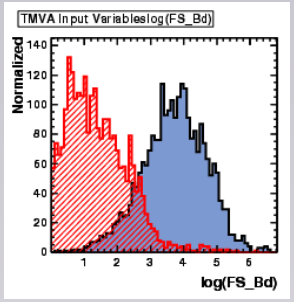
Neural Networks in High Energy Physics

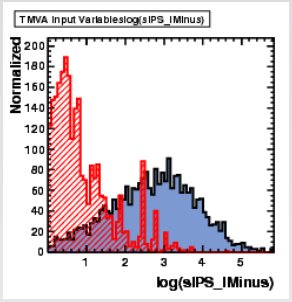
High Energy Physics

The progress of exploiting ANN in high enregy physics has been somewhat slow. Partly this conservatism is due to the a misconception that ANN approaches contain an element of "black box magic" as compared to conventional approaches. I hope I have convinced the reader that this is not the case. Statistical interpretation of the answers makes the ANN approach as well-defined to use as the discriminant ones.

Event Classification





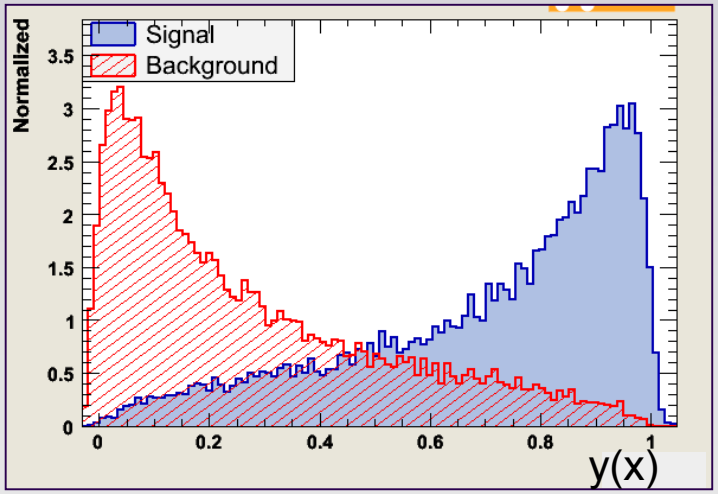


$\mathbf{P}^D$ "feature space"

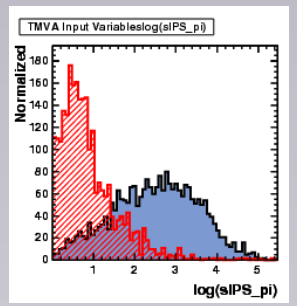
- Each event, if Signal or Background, has "D" measured variables.
- Test statistic:
  y(x): $R^D \to R$:
  most general form
  $y = y(\mathbf{x})$; $\mathbf{x} \in \mathbf{P}^D$
  $\mathbf{x}=\{x_1,\ldots,x_D\}$: input variables
- Find a mapping from D-dimensional input-observable ("feature" space) to one dimensional output → class label

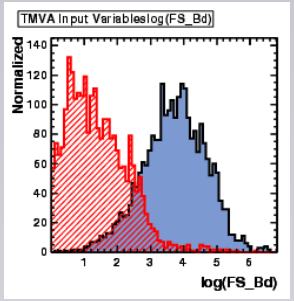
$\mathbf{P}^D \longrightarrow \mathbf{P}$

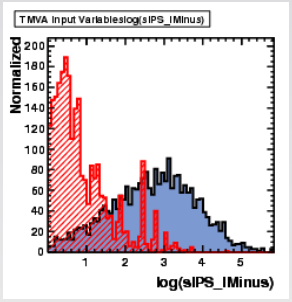
- plotting (histogramming) the resulting y(x) values:



Event Classification





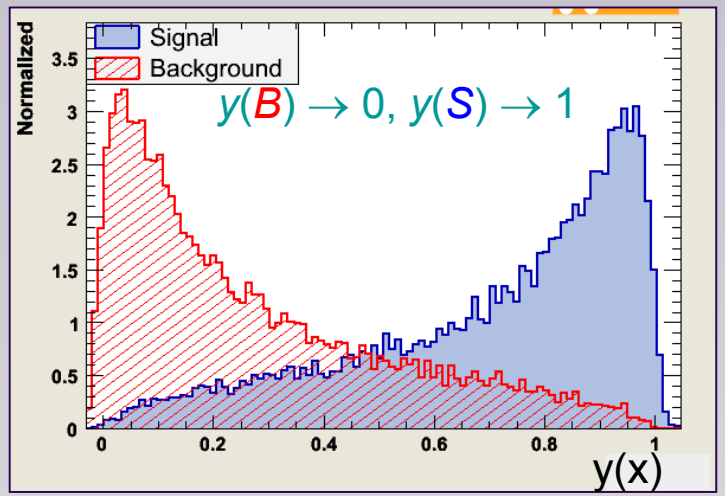


P^D

"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

Test statistic: y(x): $R^D \rightarrow R$: → P



- 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

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

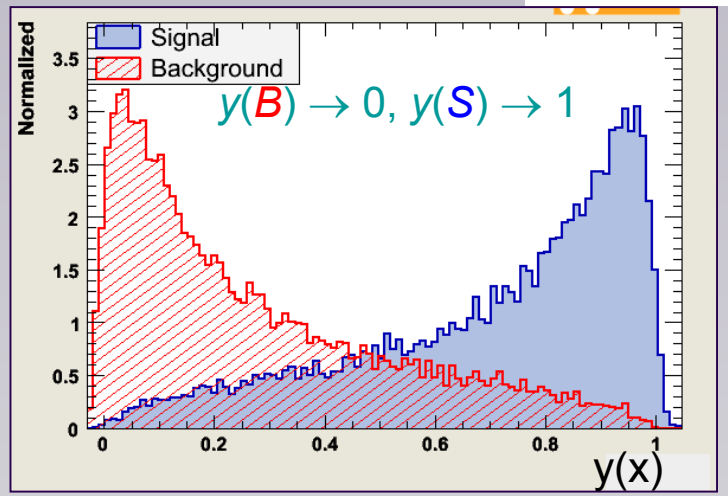
Classification ↔ Regression

Max-Planck-Institut für Kernphysik

Classification:

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

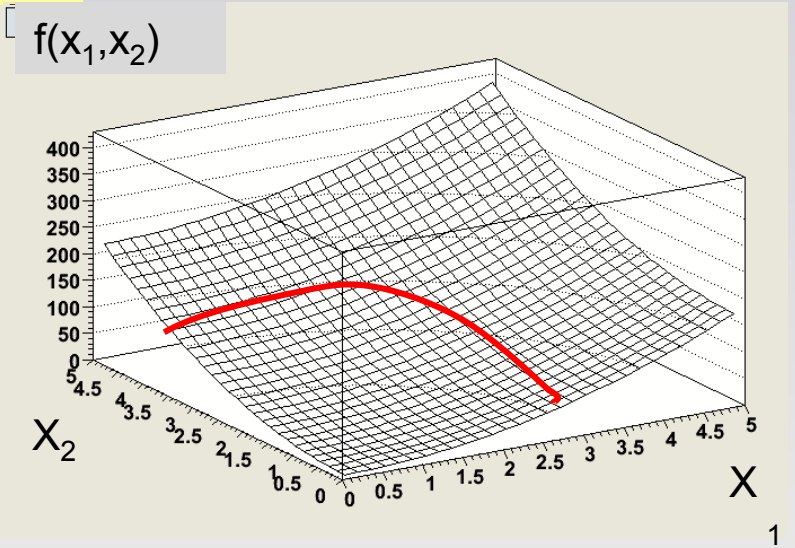
y(x): $R^D \rightarrow R$:



Regression:

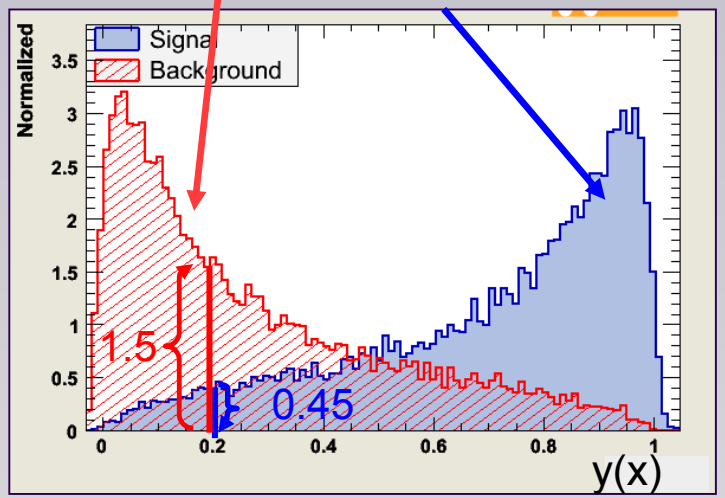
- "D" measured variables + one function value (*e.g.* cluster shape variables in the ECAL + particles energy)
- y(x): $R^D \rightarrow R$ "regression function"
- y(x)=const → hyperplanes where the target function is constant

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



Event Classification

$PDF_B(y)$. $PDF_S(y)$:



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

→ Probability densities for y given background or signal

e.g.: for an event with y(x) = 0.2

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

f_S ,f_B : fraction of **S** and **B** in the sample:

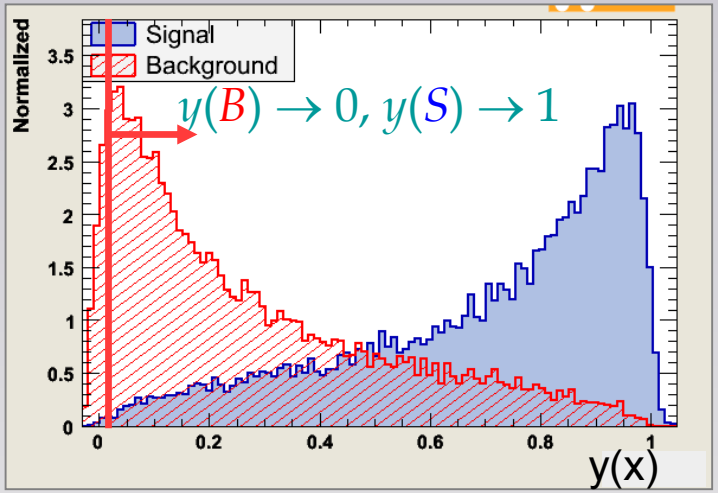
$$\frac{f_S PDF_S(y)}{f_S PDF_S(y) + f_B PDF_B(y)} = P(C = S \mid y)$$

is the probability of an event with measured **x**={x_1,….,x_D} that gives y(x) to be of type signal

Receiver Operation Characteristic (ROC) curve

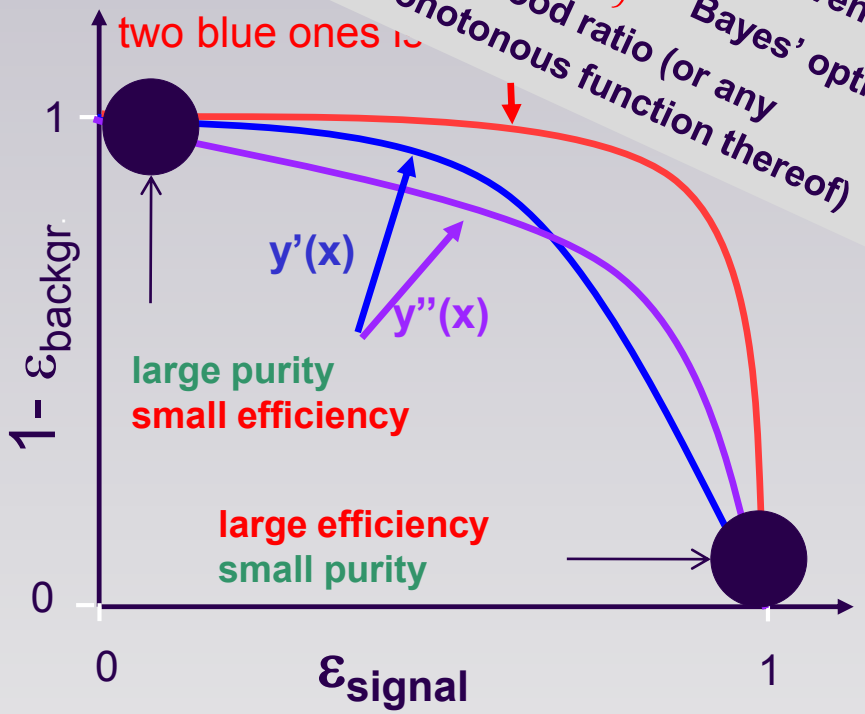
Max-Planck-Institut für Kernphysik

Signal(H_1) /Background(H_0) discrimination:



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

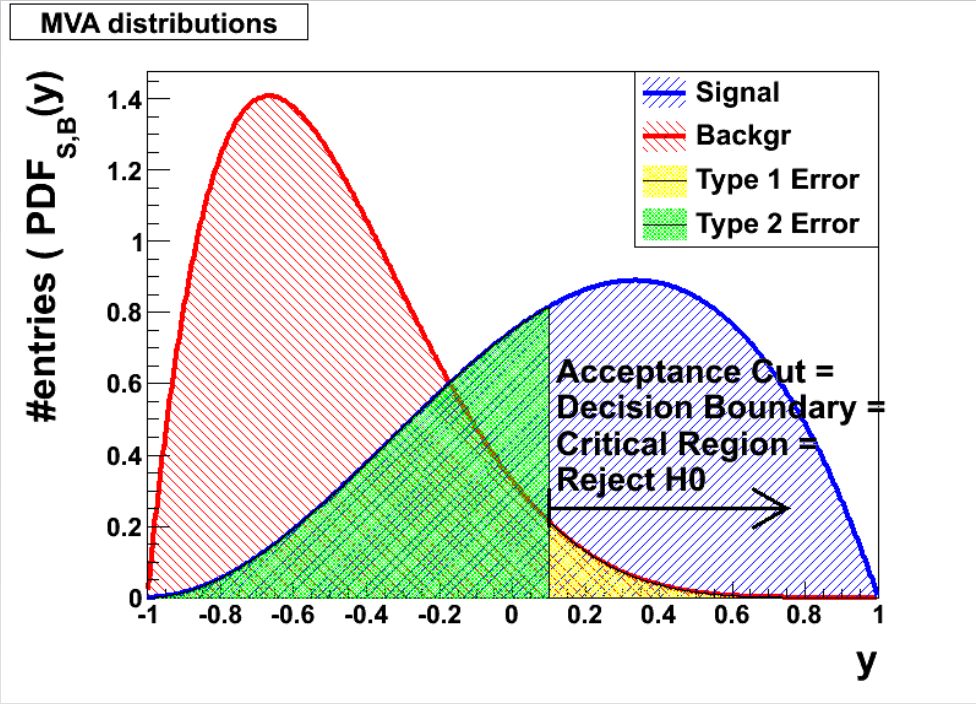
which one two blue ones is



Receiver Operation Characteristic (ROC) curve

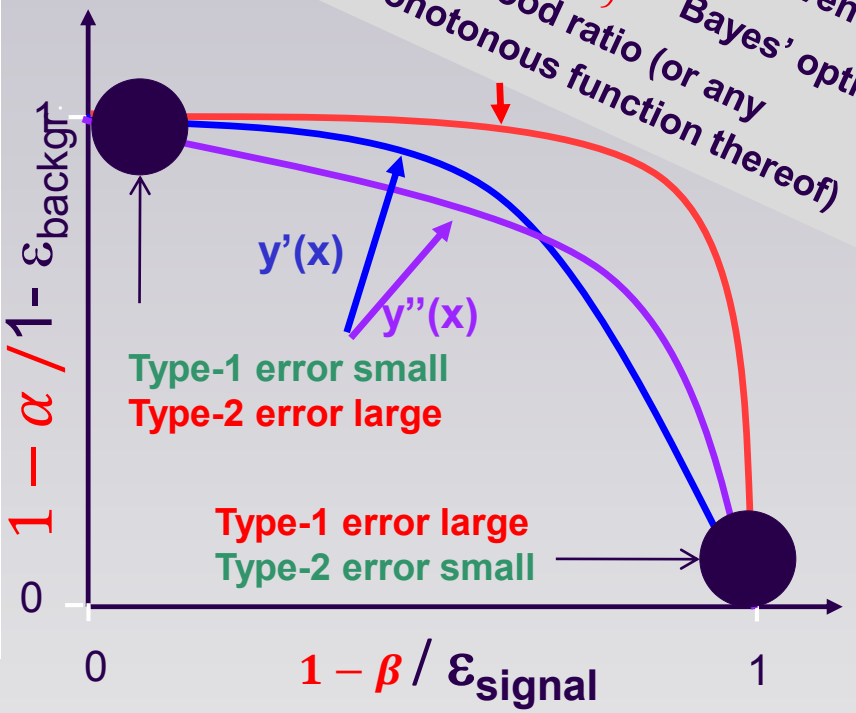
Max-Planck-Institut für Kernphysik

Signal(H_1) /Background(H_0)



"Neyman-Pearson Lemma: "limit" in ROC curve is given by the likelihood ratio (or any monotonous function thereof)

$y(x) = \frac{PDF(x|S)}{PDF(x|B)}$, Bayes' optimal



- Type 1 error: reject H_0 (i.e. the 'is bkg' hypothesis) although it would haven been true
  - → background contamination
- Type 2 error: accept H_0 although false
  - → loss of efficiency

Event Classification → finding the mapping function y(x)

- $y(x) = \frac{PDF(x|S)}{PDF(x|B)}$ → best possible classifier

→but p(x|S), p(x|B) are typically unknown

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

- use already classified "events" (e.g. MonteCarlo) 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

  - approximate the "likelihood ratio" (or a monotonic transformation thereof).
    - find a y(x) whose hyperplanes* in the "feature space":

      (y(x) = const) optimally separate 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

Machine Learning Categories

Max-Planck-Institut für Kernphysik

supervised: - training "events" with known type (i.e. Signal or Backgr, target value)

un-supervised: - no prior notion of "Signal" or "Background"

- cluster analysis: if different "groups" are found → class labels
- principal component analysis:

find basis in observable space with biggest hierarchical differences in the variance

→ infer something about underlying substructure

reinforcement-learning:

- learn from "success" or "failure" of some "action policy"

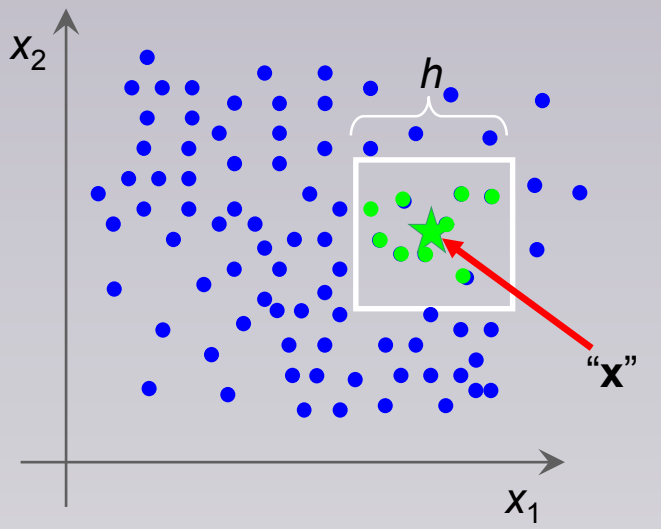
(i.e. a robot achieves his goal or does not / falls or does not fall/ wins or looses the game)

This lecture: supervised learning

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

"events" distributed according to P(x)



→ K-events:

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

k(u): is called a Kernel function:

→K(x)/N: estimate of average P(x) in the volume V

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

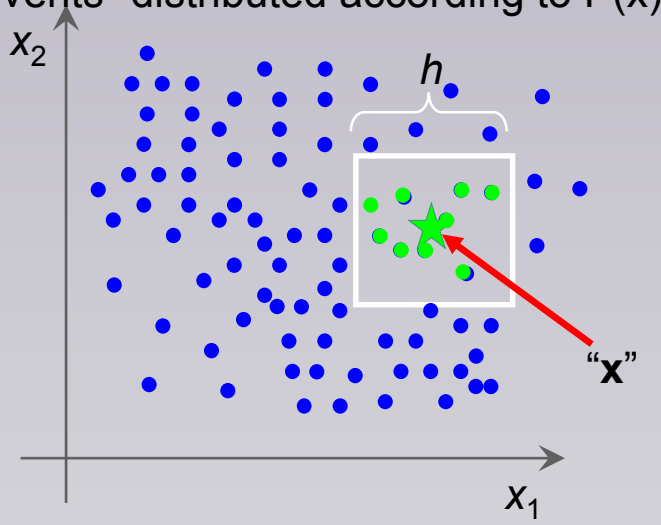
$$P(\mathbf{x}) = \frac{1}{N} \sum_{n=1}^{N} \frac{1}{h^D} k\left(\frac{\mathbf{x} - \mathbf{x}_n}{h}\right)$$

→ Kernel Density estimator of the probability density

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

"events" distributed according to P(x)



→ 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(x)/N: estimate of average P(x) in the volume V

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

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

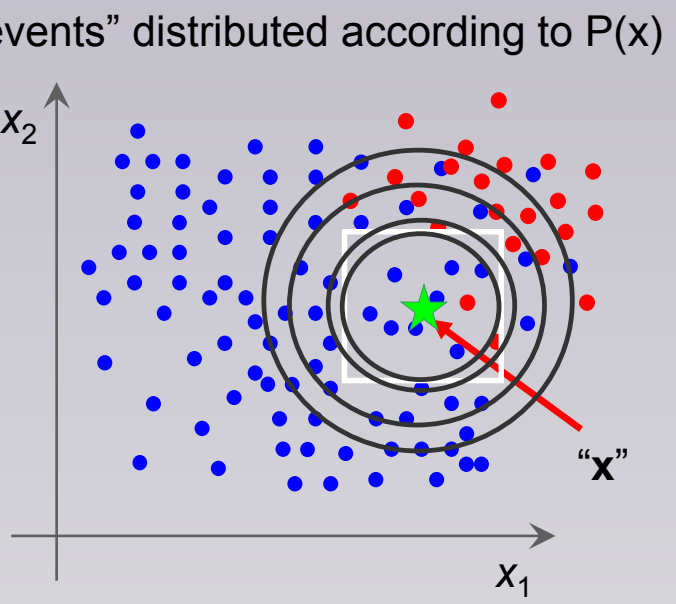
i.e.: the average function value

K-Nearest Neighbour

→kNN : k-Nearest Neighbours

relative number events of the various classes amongst the k-nearest neighbours

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



keep K fixed → variable window size

→automatically 'adapt' resolution to the available data

→ may replace "window" by "smooth" kernel function (i.e. weight events by distance via Gaussian)

Kernel Density Estimator

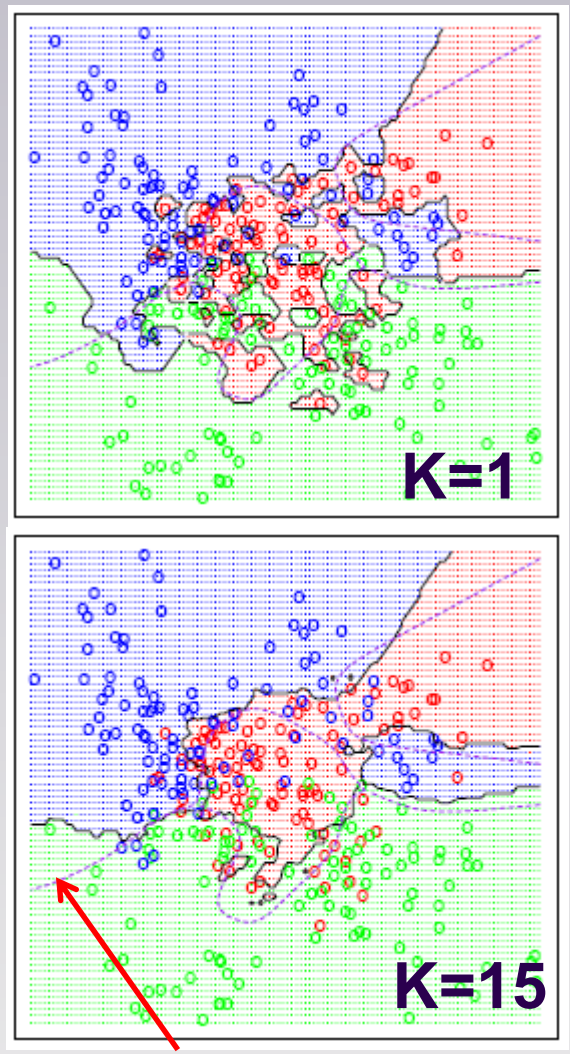
$$P(\mathbf{x}) = \frac{1}{N}\sum_{n=1}^{N} K_h(\mathbf{x} - \mathbf{x}_n)$$: a general probability density estimator using kernel K

- K or h: "size" of the Kernel → "smoothing"
  - too small: overtraining/overfitting
  - too large: not sensitive to features in P(x)

- Kernel types: window/Gaussian …
- which metric for the Kernel ?
  - normalise all variables to same range
  - include correlations ?
    - Mahalanobis Metric: $x^*x \rightarrow xV^{-1}x$

- a drawback of Kernel density estimators:

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



(Elements of statistical learning)

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

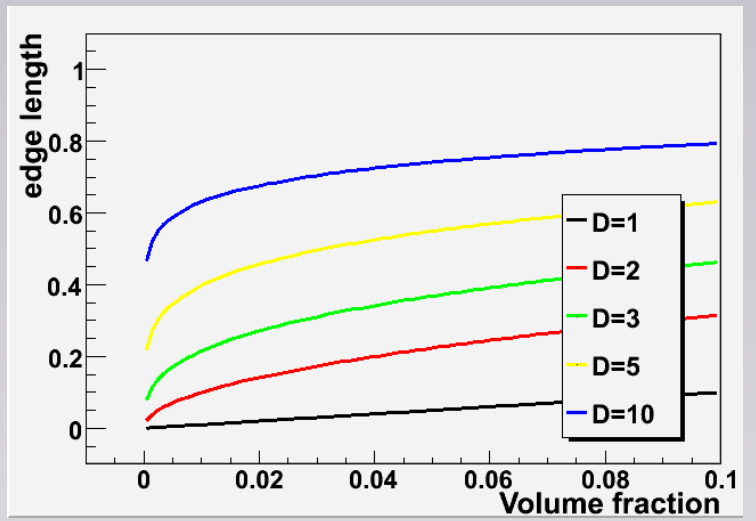
"Curse of Dimensionality"

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)

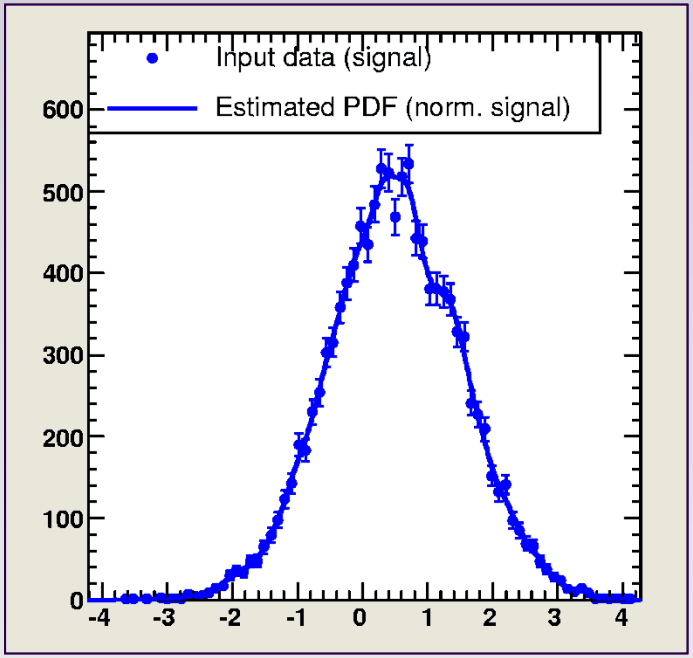
Multivariate Likelihood (k-Nearest Neighbour)
→ estimate the full D-dimensional joint probability density

Naïve Bayesian
→ ignore correlations

$$P(\mathbf{x}) \cong \prod_{i=0}^{D} P_i(\mathbf{x})$$

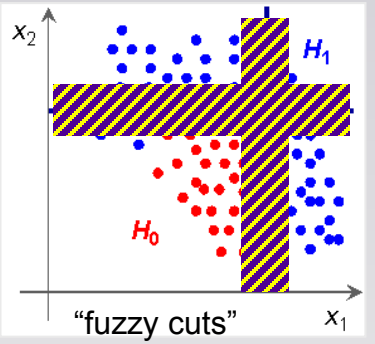
product of marginal PDFs
(1-dim "histograms")

pdf: histogram + smoothing



- No hard cuts on individual variables → "fuzzy",

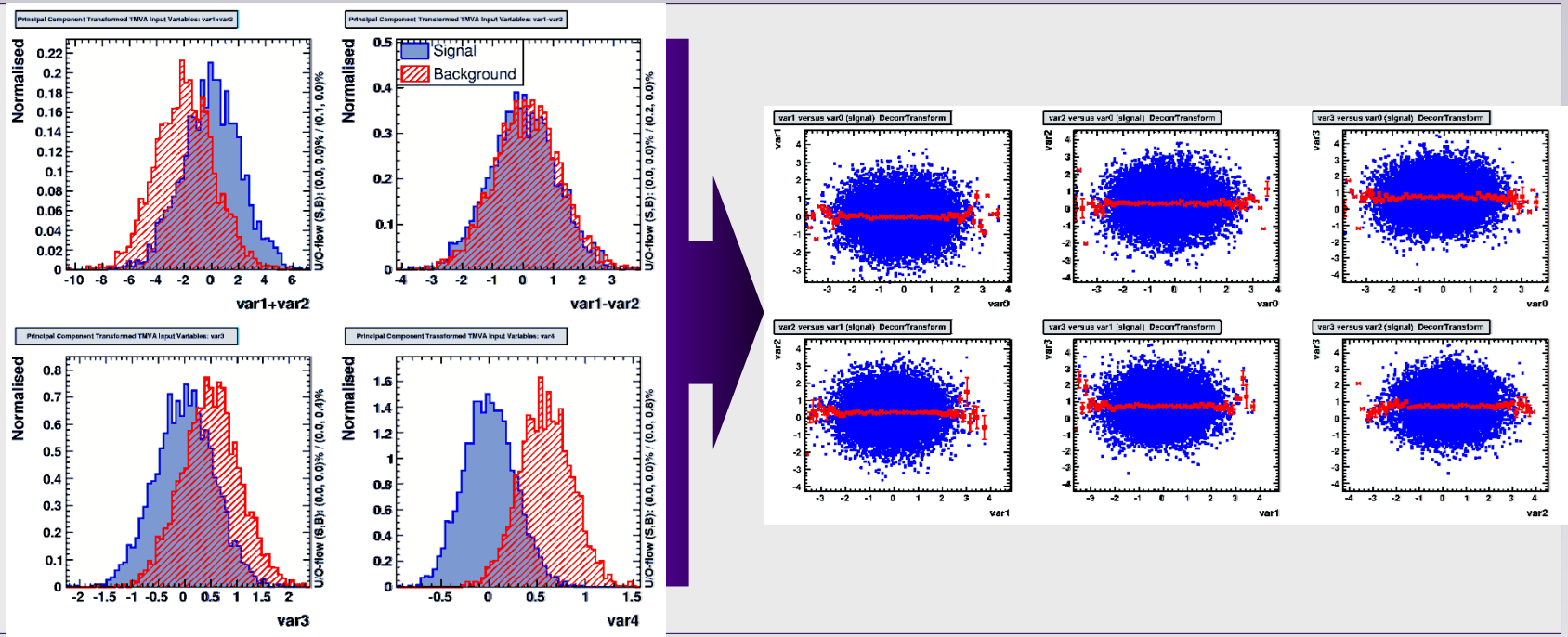
(a very signal like variable may counterweigh another, less signal like variable)



- optimal method if correlations == 0
  - try to "eliminate" correlations

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: $D = S^T C S \Rightarrow C' = S\sqrt{D}S^T$
    - transformation from original (x) in de-correlated variable space (x') by: $x' = C'^{-1}x$

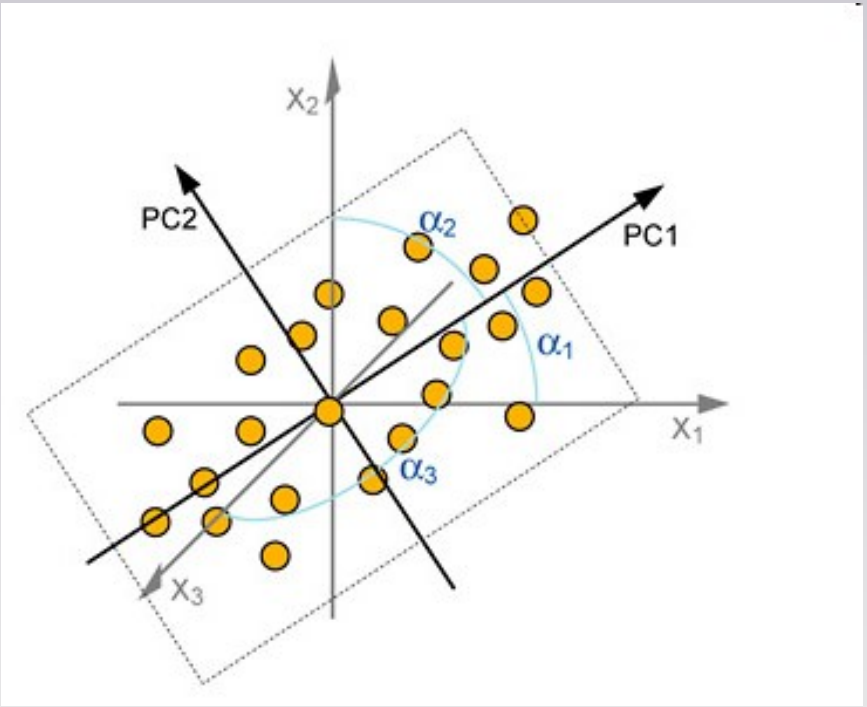


Attention: eliminates only linear correlations!!

De-Correlation via PCA (Principal Component Analysis)

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

- **Eigenvectors of covariance matrix → "axes" in transformed variable space**
  - large eigenvalue → large variance along the axis (principal component)



→ PCA eliminates correlations!

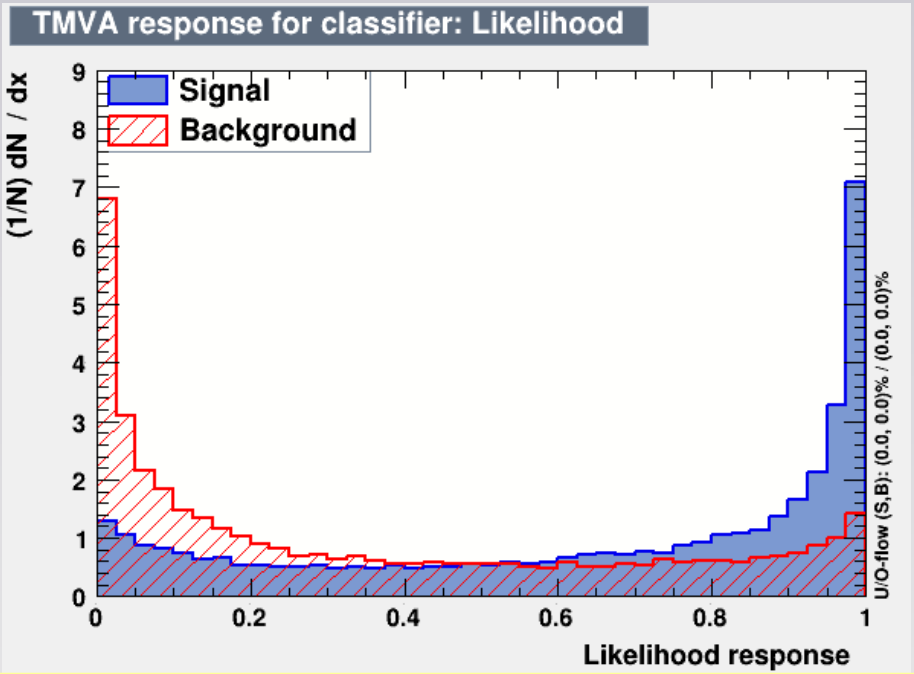
Decorrelation at Work

- Example: linear correlated Gaussians → de-correlation works to 100%

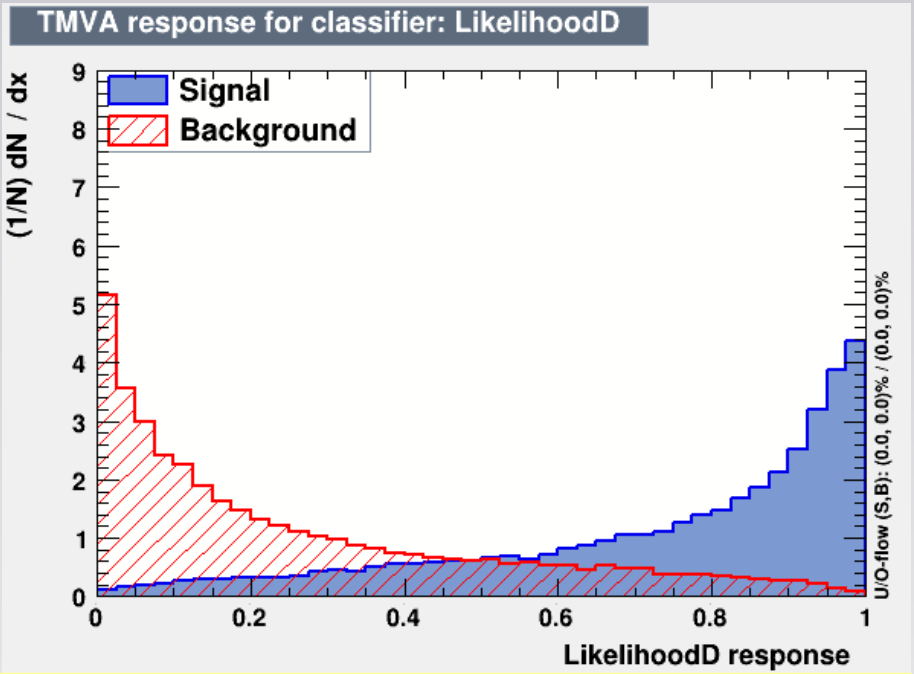
→1-D Likelihood on de-correlated sample give best possible performance

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

correlated variables:



after decorrelation



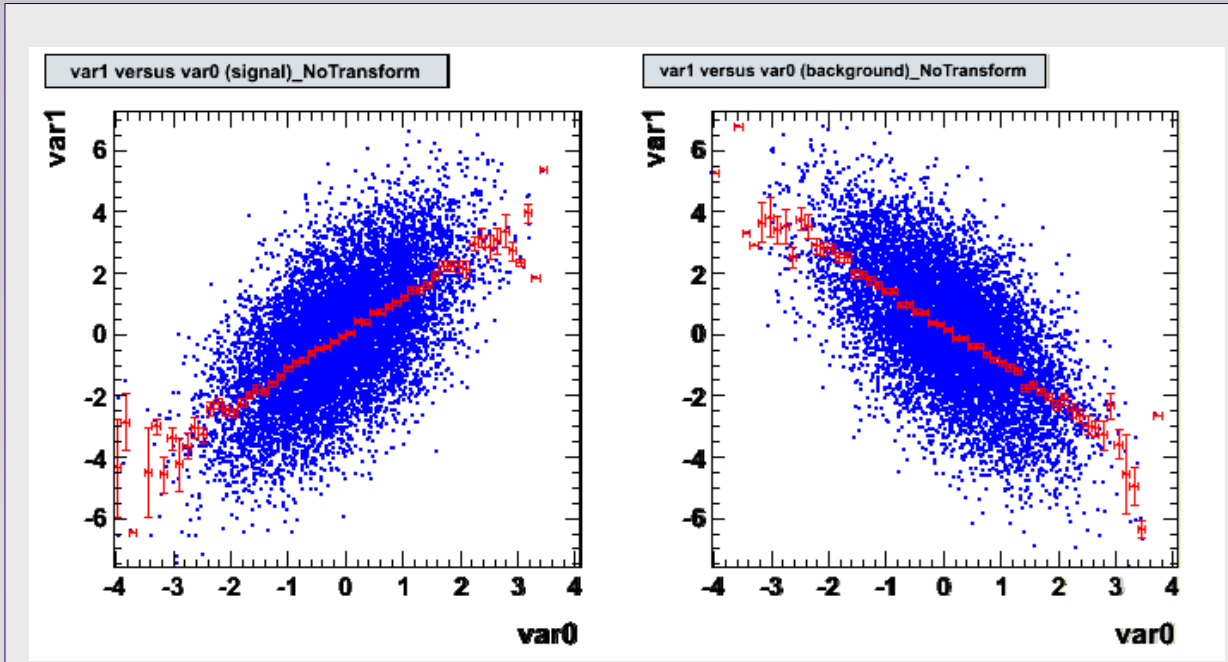
Watch out! Things might look very different for non-linear correlations!

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

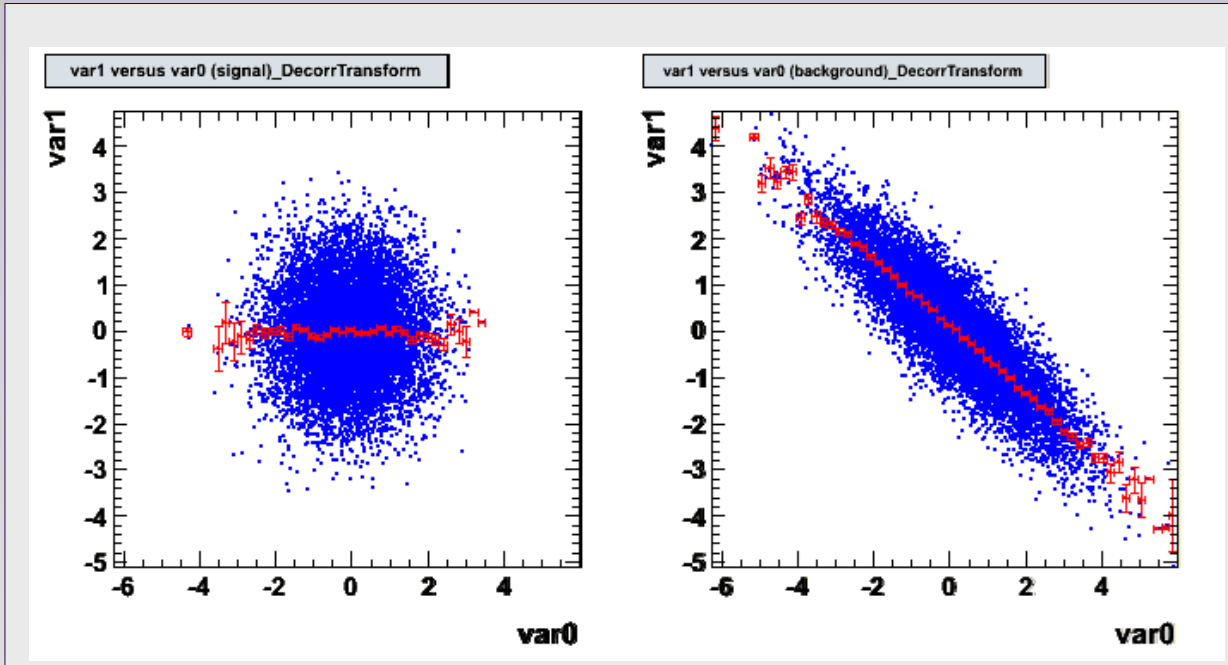
Background

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

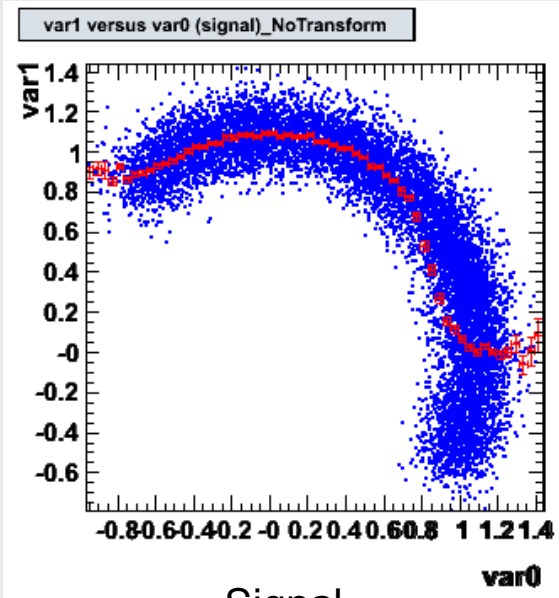
Background

Limitations of the Decorrelation

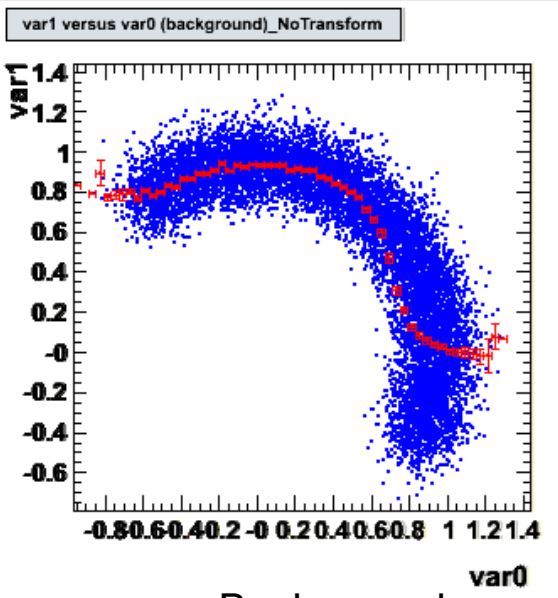
- in cases with non-Gaussian distributions and/or nonlinear correlations, the decorrelation needs to be treated with care

- How does linear decorrelation affect strongly nonlinear cases ?

Original correlations



Signal



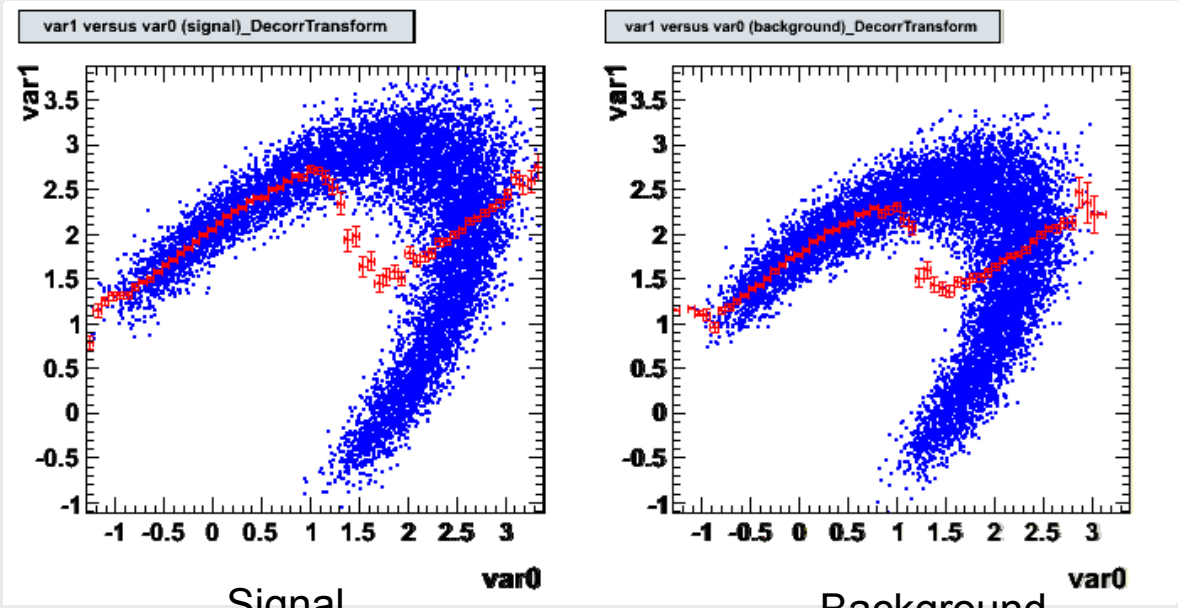
Background

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 using decorrelation “blindly”!!

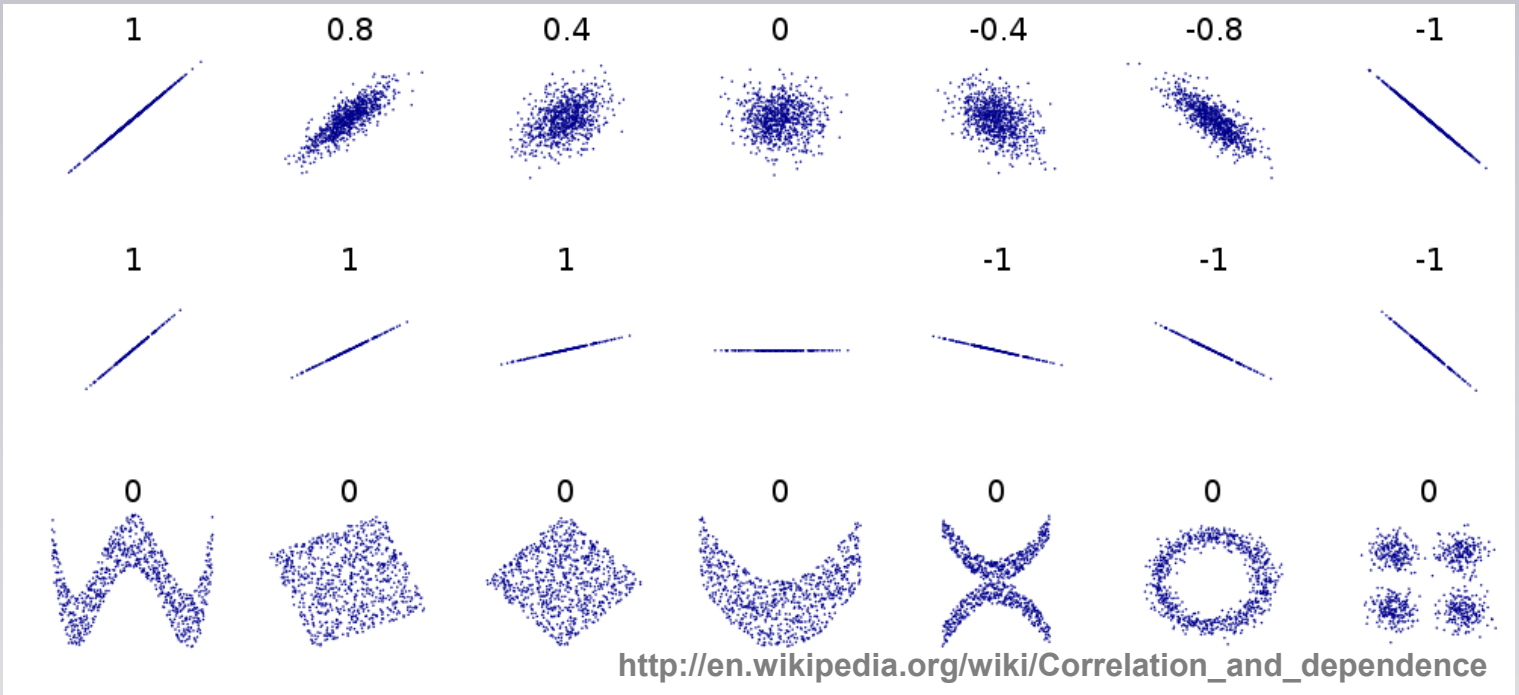
→Perhaps “de-correlate” only a subspace!

Correlation Coefficients

Max-Planck-Institut für Kernphysik

'correlations' , 'linear-correlations', 'interaction/dependence'

→ phsicist's slang often different from statistitans' !



http://en.wikipedia.org/wiki/Correlation_and_dependence

- **to capture "non-linear correlations" → mutual information**

- $I(x,y) = \int\int p_{xy}(x,y) log\left(\frac{p_{xy}(x,y)}{p_x(x)p_y(y)}\right) dxdy$

- $I(x,y) = 0$ **only if** x, y **are really statistically independent !**

Discriminative Classifiers

- **KNN and Naïve Bayesian** (Multi-dimensional and Projective Likelihood)
  - generative methods - estimate the pdf
- **discriminative methods**
  - impose model-specific restrictions (i.e. linear decision boundaries)
  - fit directly the decision boundaries

"Neyman-Pearson Lemma:
"limit" in ROC curve is given by

$$y(x) = \frac{PDF(x|S)}{PDF(x|B)},$$

Bayes' optimal the likelihood ratio (or any monotonous function thereof)

→ in the limit, a 'perfect' discriminative classifier y(x) parametrizes the likeihood ratio (or a monotonic function thereof)

→ use as 'event weights'

arXiv:1506.02169 for a 'more theoretical' analysis

Linear Discriminant

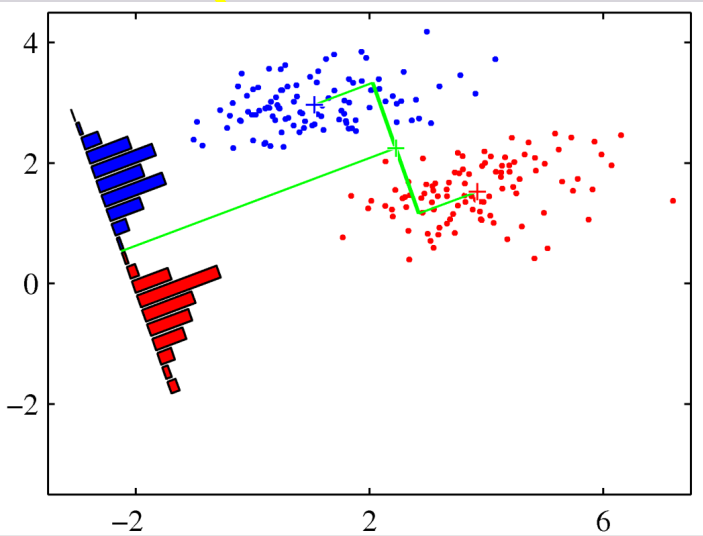
General:

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

Linear Discriminant:

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

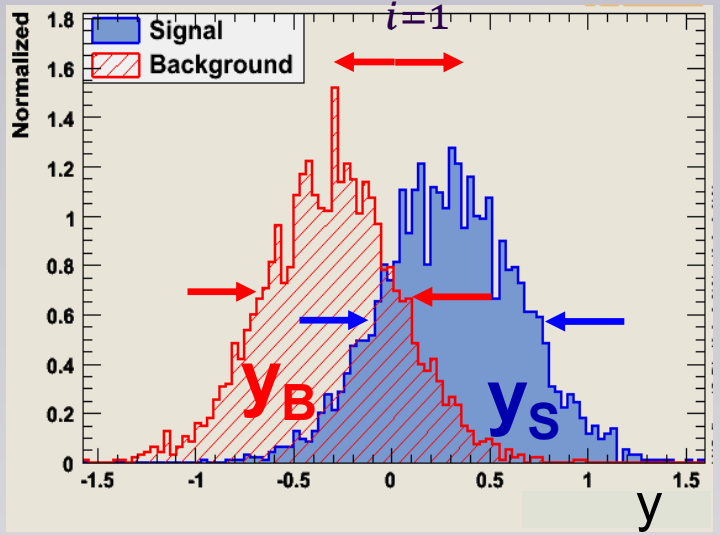
i.e. any linear function of the input variables: → linear decision boundaries



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

Fisher's Linear Discriminant

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



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

Linear Discriminant and non linear correlations

assume the following non-linear correlated data:

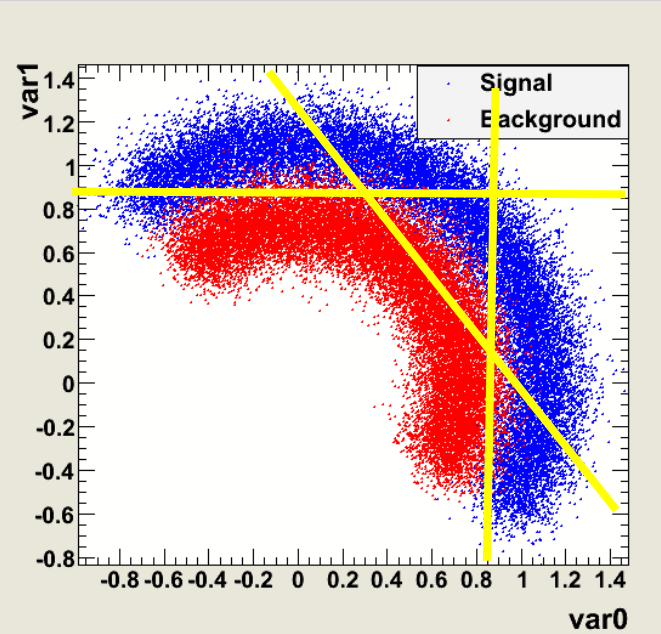
- **the Linear discriminant obviously doesn't do a very good job here:**

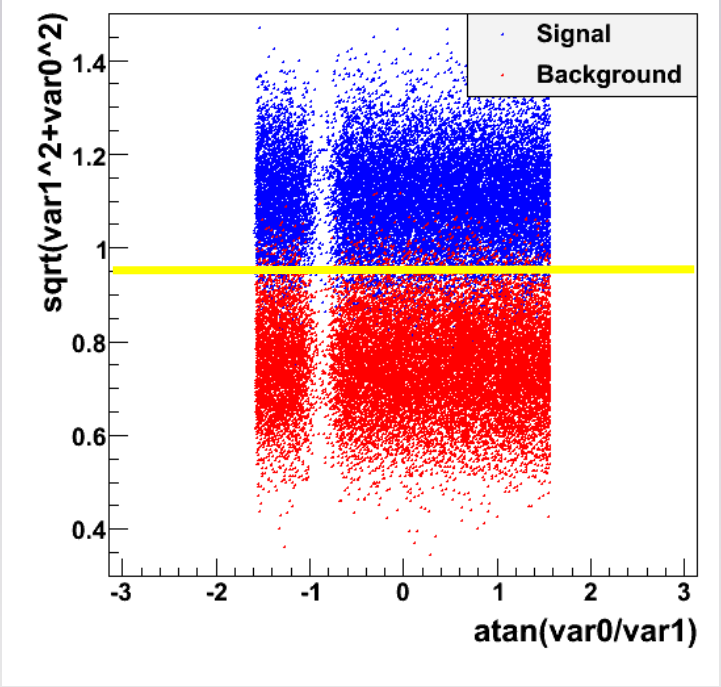
- **Of course, these can easily be de-correlated:**

  →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:**

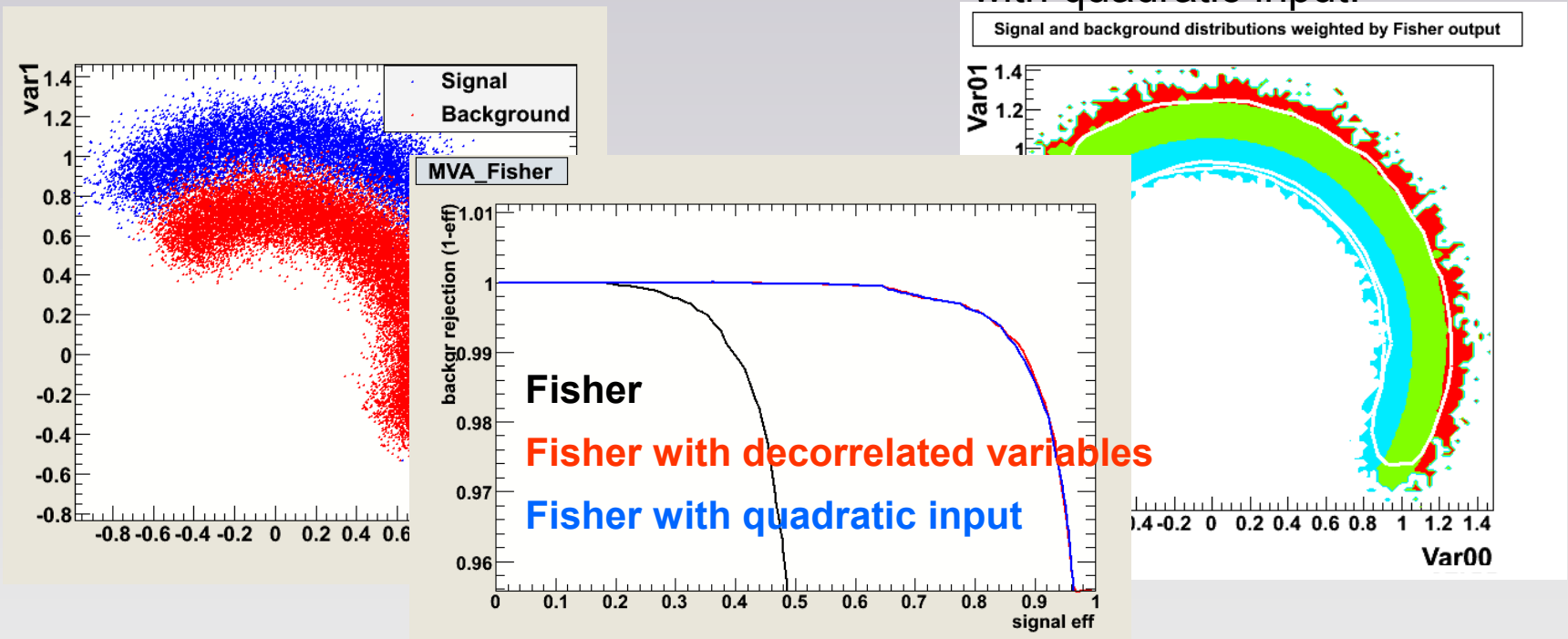
while:
- var0
- var1

→ linear decision boundaries in var0,var1

'feature engineering'
- var0 * var0
- var1 * var1
- var0 * var1

→ quadratic decision boundaries in var0,var1

Performance of Fisher Discriminant with quadratic input:



Classifier Training and Loss-Function

What about a more 'general approach' than 'constructing $J(\vec{w})$' ?

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

$L(y^{train}, y(x))$: penalizing prediction errors for training events

- Regression:

→ $E[L] = E\left[\frac{1}{2}(y^{train} - y(x))^2\right]$ squared error loss

- Classification:

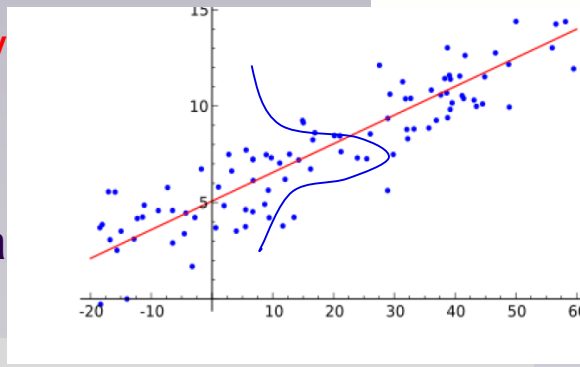
→ $E[L] = E[y_i^{train} \log(y(x_i)) + (1 - y_i^{train}) \log(1 - y(x_i))]$ binomial loss

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

Classifier Training and Loss-Function

- Regression: y_i^{train} : Gaussian distributed around a mean v
  - Remember: Maximum Likelihood estimatior
  - Maximise: log probability of the observed training data



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

$$\rightarrow E[L] = E\left[\frac{1}{2}\left(y^{train} - y(x)\right)^2\right] \quad \text{squared error loss (regression)}$$

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

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

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

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

Logistic Regression*

*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' ?

→ build a linear classifier that maximizes 'binomial loss':

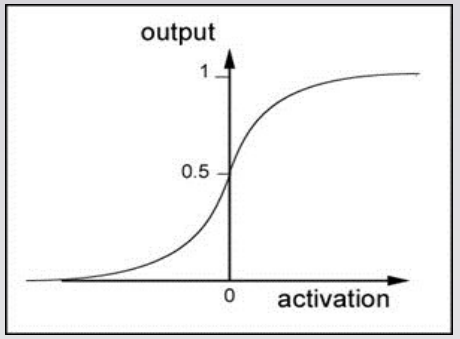
→ y(x) to parameterize P(S|x), we clearly cannot 'use a linear function for 'y(x)'

→ '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!

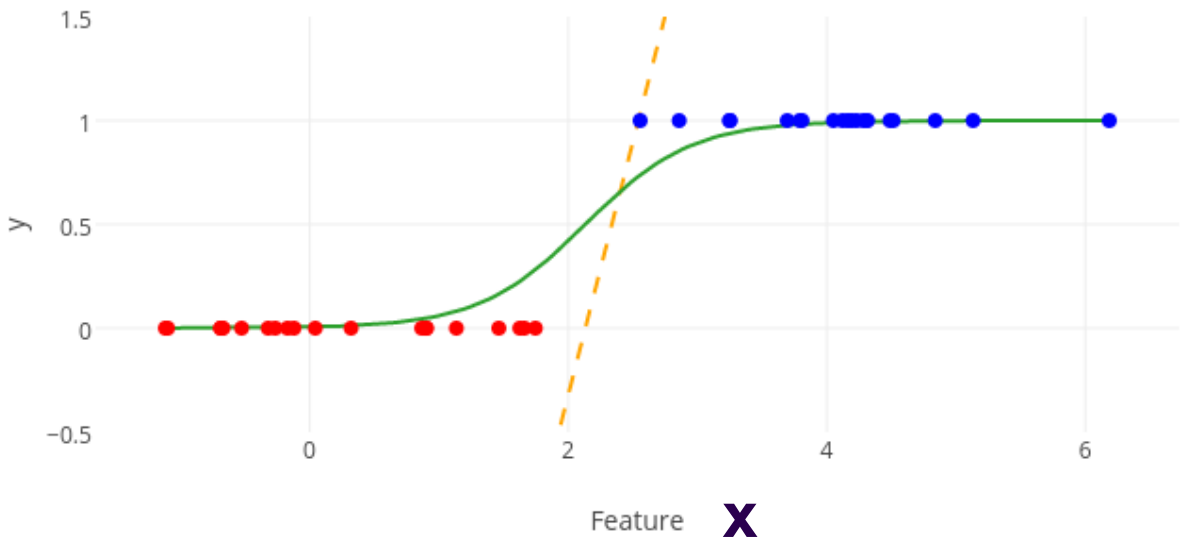


Note: Now y(x) has a 'probability' interpretation. y(x) of the Fisher discriminant was 'just' a discriminator.

Logistic Regression

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

1D example:



y(x) =sigm(wx)

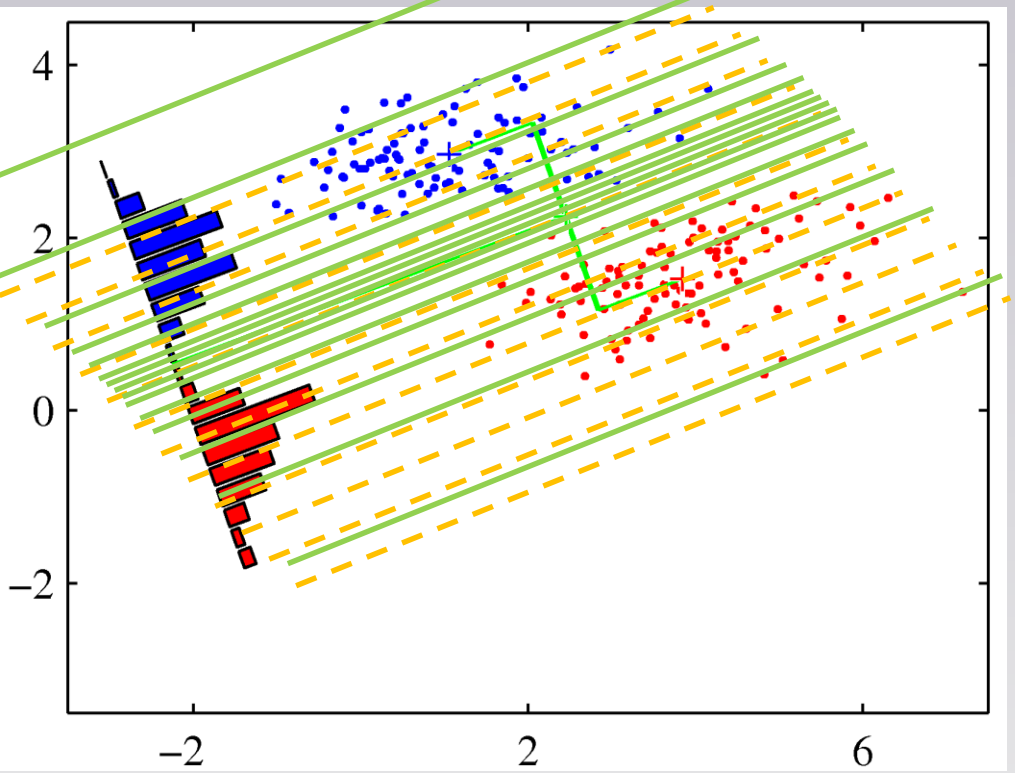
y(x) =wx

Note: decision boundaries are still ‘linear’, just the ‘contour lines’ (y(x)=const) are non-linear, parametrizing the probability of the event being y=0 or y=1 as ‘distance’ from the boundary….

Logistic Regression

Difference between ‘linear classifier’ and ‘logistic regression’

→ distribution of decision boundaries



a ‘monotonous’ transformation of y(x)

→ does not change ‘relative overlap’ for pdfs of y_S and y_B

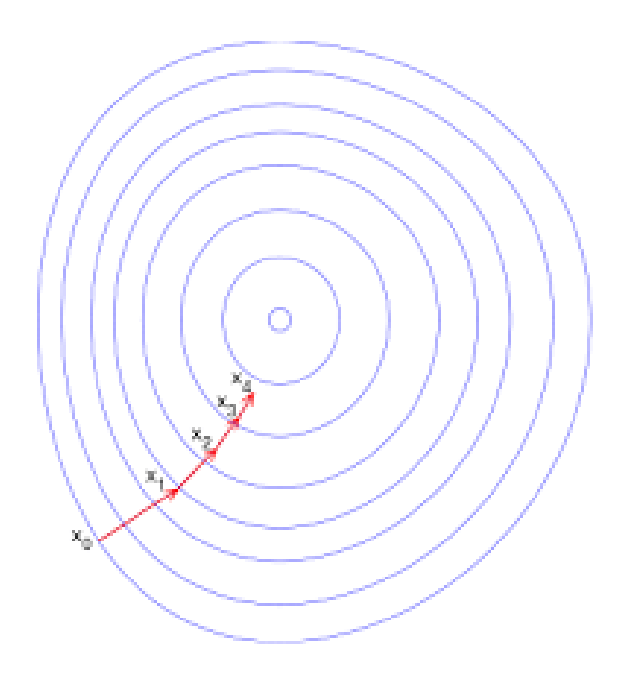
→ Does not change performance

(Stochastic) Gradient Decent SDG

minimize the "loss function" → "W" ?

e.g. $E[L(\boldsymbol{W})] = E[y_i^{train} \log(y(x_i)) + (1 - y_i^{train}) \log(1 - y(x_i))]$

with $y(x) = \frac{1}{1+e^{-Wx}}$;



learning rate

$\boldsymbol{W} \rightarrow \boldsymbol{W} - \boldsymbol{\eta} \frac{\partial \boldsymbol{E}(\boldsymbol{L})}{\partial \boldsymbol{w}}$: gradient decent

and if you don't want to evaluate the expectation value every time for the whole sample:

$\boldsymbol{W} \rightarrow \boldsymbol{W} - \boldsymbol{\eta} \frac{\partial \boldsymbol{L}}{\partial \boldsymbol{w}}$: stochastic gradient decent

mostly: something in between → mini-batches

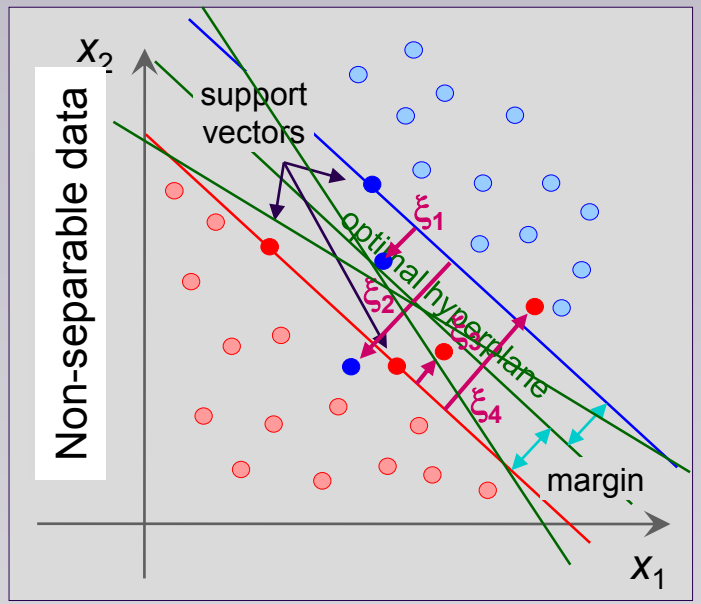
Support Vector Machine

- 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
- suitable variable transformations into a higher dimensional space may allow separation with linear decision boundaries non linear problems
- →Support Vector Machine

Support Vector Machines

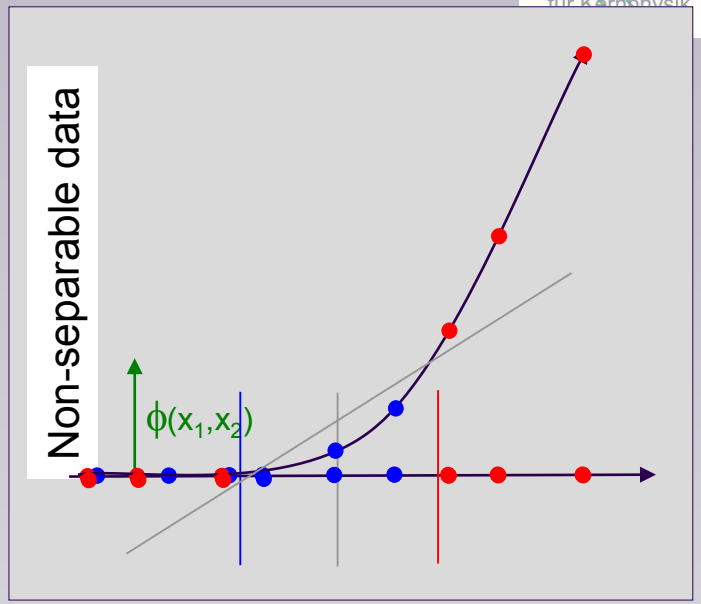
- hyperplane that separates **S** from **B**
  - Linear decision boundary
  - Best separation: maximum distance (margin) between closest events (*support*) to hyperplane
  - If data non-separable add *misclassification cost* parameter $C \cdot \Sigma_i \xi_i$ to minimisation function
  - **Solution of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

  → **quadratic minimisation problem**



Support Vector Machines

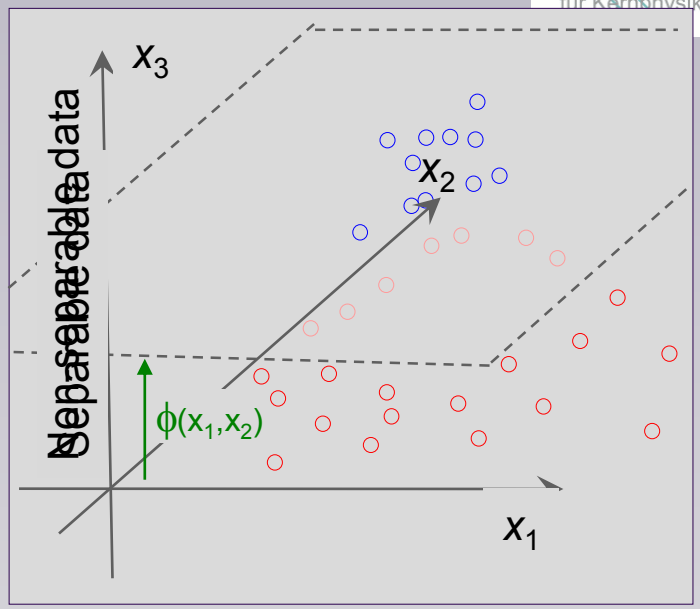
- hyperplane that separates **S** from **B**
  - Linear decision boundary
  - Best separation: maximum distance (margin) between closest events (*support*) to hyperplane
  - If data non-separable add *misclassification cost* parameter $C \cdot \Sigma_i \xi_i$ to minimisation function
  - **largest margin - inner product of support vectors (distances) → 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
  - 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
  - **largest margin - inner product of support vectors (distances) → quadratic minimisation problem**



- Non-linear cases:
  - non linear variable transformation → linear separation in transformed feature space
  - no explicit transformation specified → Only its "scalar product" $x \cdot x \rightarrow \Phi(x) \cdot \Phi(x)$ needed.
    - 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

- How does this “Kernel” business work?
- Kernel function == scalar product in “some transformed” variable space

→ standard: $\vec{x} \cdot \vec{y} = \sum x_i \; y_i = |x||y| * cos(\theta)$

→ large if : $\vec{x} \cdot \vec{y}$ are in the same “direction”

→ zero if : $\vec{x} \cdot \vec{y}$ are orthogonal (i.e. point along different axes / dimension)

→ e.g. Gauss kernel: $\Phi(\vec{x}) \cdot \Phi(\vec{y}) = K(\vec{x}, \vec{y}) = exp(-\frac{(\vec{x}-\vec{y})^2}{2\sigma^2})$

→ zero if points: $\vec{x}\; and\; \vec{y}$ “far apart” in original data space

→ large only in “vicinity” of each other

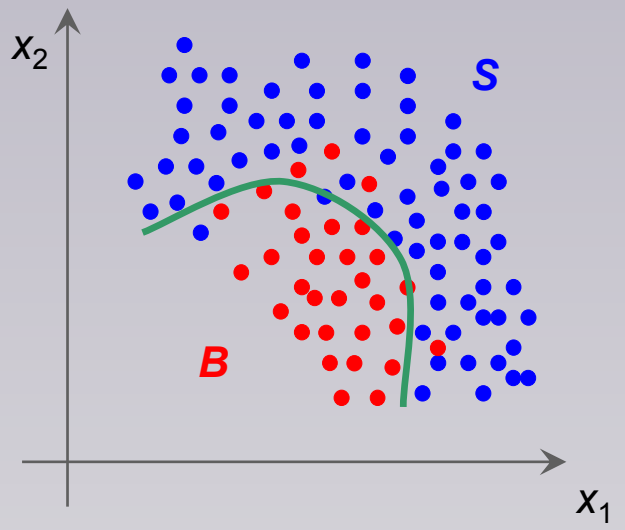
→ $\sigma <$ distance between training data points:

→ each data point is “lifted” into its “own” dimension

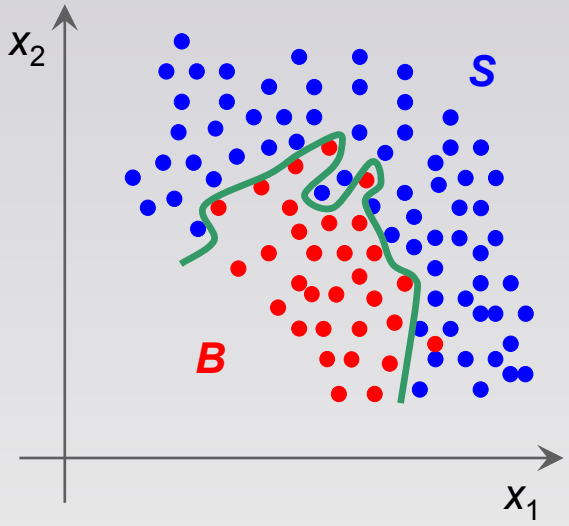
→ full separation of “any” event configuration with decision boundary along coordinate axis

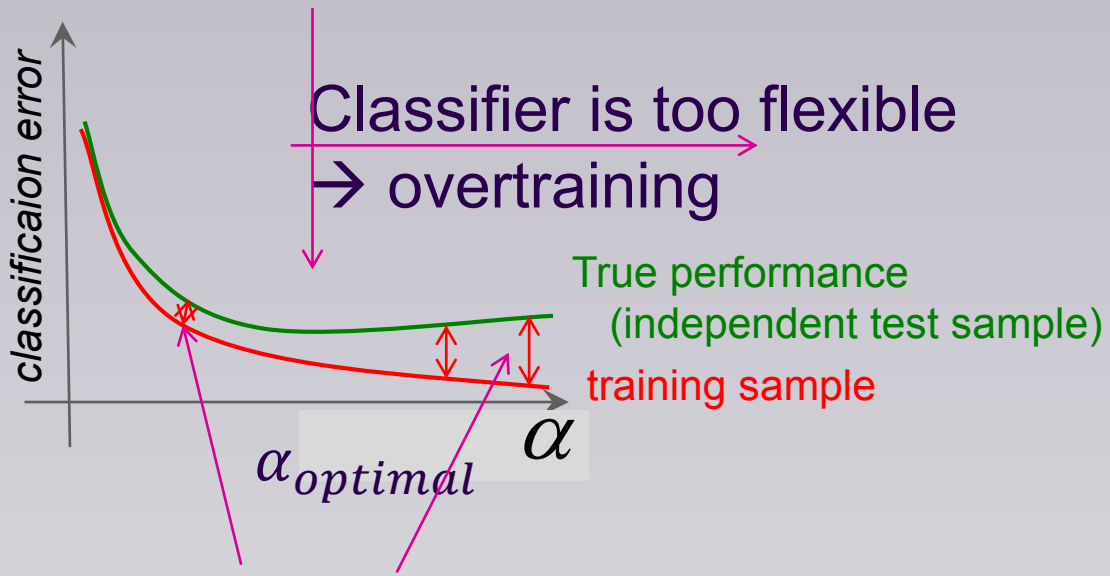
→ well, that would of course be: overtraining

Overtraining



Or ?





Bias if 'performance' is estimated from the training sample

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

→ **verify on independent "test" sample**

Regularisation

Minimize loss function: e.g. **via** $\boldsymbol{W} \rightarrow \boldsymbol{W} - \boldsymbol{\eta}\frac{\boldsymbol{\partial L}}{\boldsymbol{\partial w}}$: SDG

Include prior distribution on 'weights'/'parameters' *w*:

$$L = \log\Big(\prod_{i}^{events} P\big(y_i^{train}|y(x_i)\big) * p(w)\Big)$$

$$= \sum_{i}^{events} \log(P(y_i^{train}|y(x_i)) + \log\big(p(w)\big)$$

often (e.g if y = polynomial or y = neural network)

w "small" → model is less 'flexible'

→ reasonable prior $p(w)$ would be: Gaussian with mean zero

→ $L = L + \frac{1}{2}\alpha\sum w^2$ α: factor of 'how much you want to penalize"

Digression

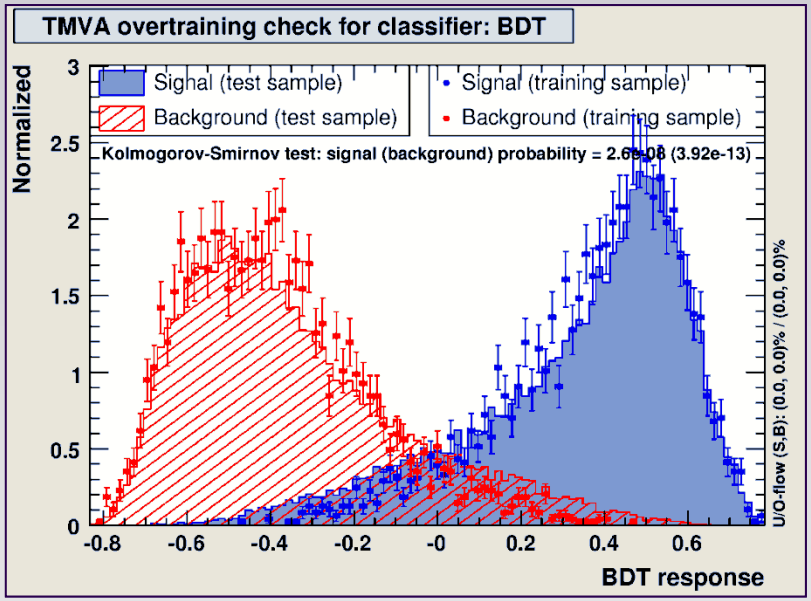
Max-Planck-Institut für Kernphysik

Kolmogorov Smirnov Test:

Tests if two sample distributions are compatible with coming from the same parent distribution

→ Statistical test: if they are indeed random samples of the same parent distribution, then the KS-test gives an uniformly distributed value between 0 and 1 !!

→ Note: that means an average value of 0.5 !!



Please: don't misunderstand the title of this plot as:

$$KS = \begin{cases} \sim 1: & ok \\ else: & trouble \end{cases}$$

Was meant as quick sanity check ONLY!!

Cross Validation

- parameters "α" → control performance
  - #training cycles, #nodes, #layers, regularisation parameter (neural net)
  - smoothing parameter h (kernel density estimator)
  - ….
- more training data → better training results
- division of data set into "training" and "test" and "validation" sample? ☹

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,
- i-th classifier is trained without the i-th sub sample → used as 'test/validation'
- calculate the test error: $CV(\alpha) = \frac{1}{N_{events}} \sum_{k}^{events} L(y_i(x_k,\alpha))$ L : loss function
- use α for which CV(α) is minimum → train the final classifier using all data

General Advice for (MVA) Analyses

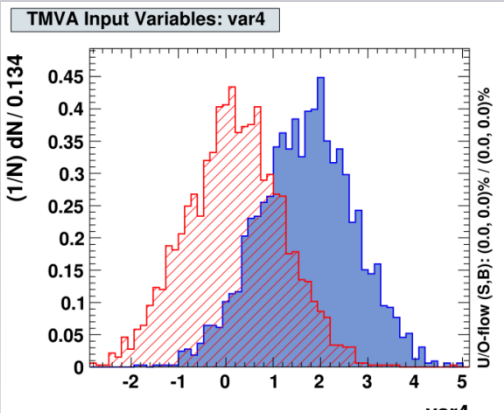
- no magic in MVA- or ML-Methods:
  - no “artificial intelligence” ☹ … just “fitting decision boundaries” in a given model
- most important: finding good observables
  - good separation power between S and B
  - little correlations amongst each other → have ‘new information’
  - no correlation with the parameters you try to measure in your signal sample!
- combination of variables → feature engineering !
  - eliminate correlations: *you are MUCH more intelligent than the algorithm*
- scale features to similar numeric range
- apply pure pre-selection cuts yourself.
- avoid “sharp features” → numerical problems, binning loss
  - often simple variable transformations (i.e. log(variable)) do the trick
- treat regions with different features “independent”
  - Introduces unnecessary correlations, ‘kinks’ in decision boundaries

MVA Categories

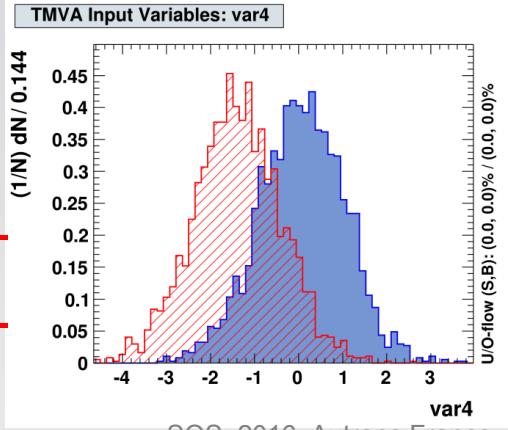
- **one classifier per 'region'**
- 'regions' in the detector (data) with different features treated independent
  - **improves performance**
  - **avoids additional** correlations where otherwise the variables would be uncorrelated!

Example: var4 depends on some variable "eta"

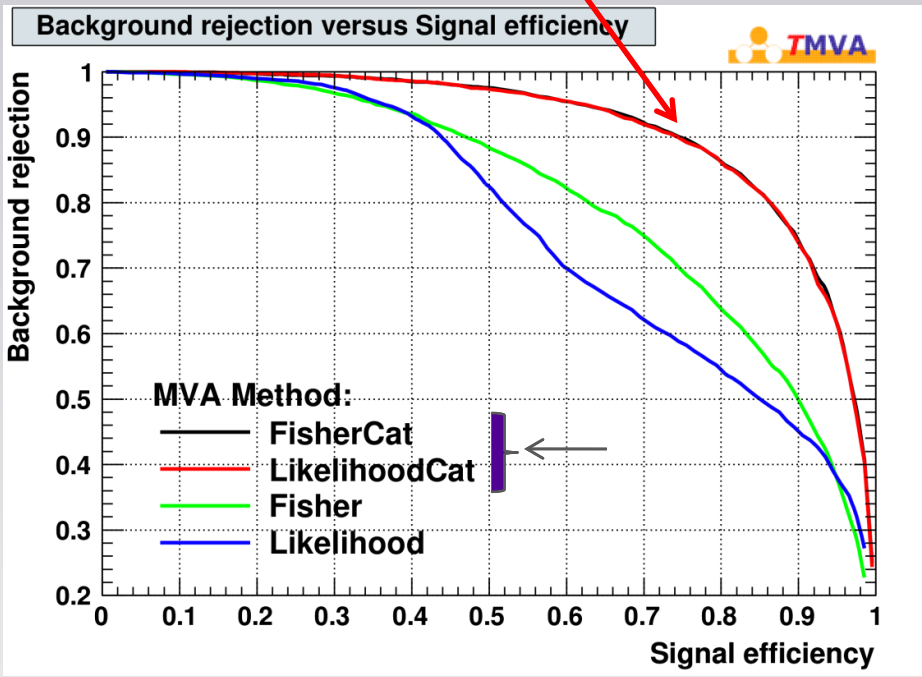
|eta| > 1.3



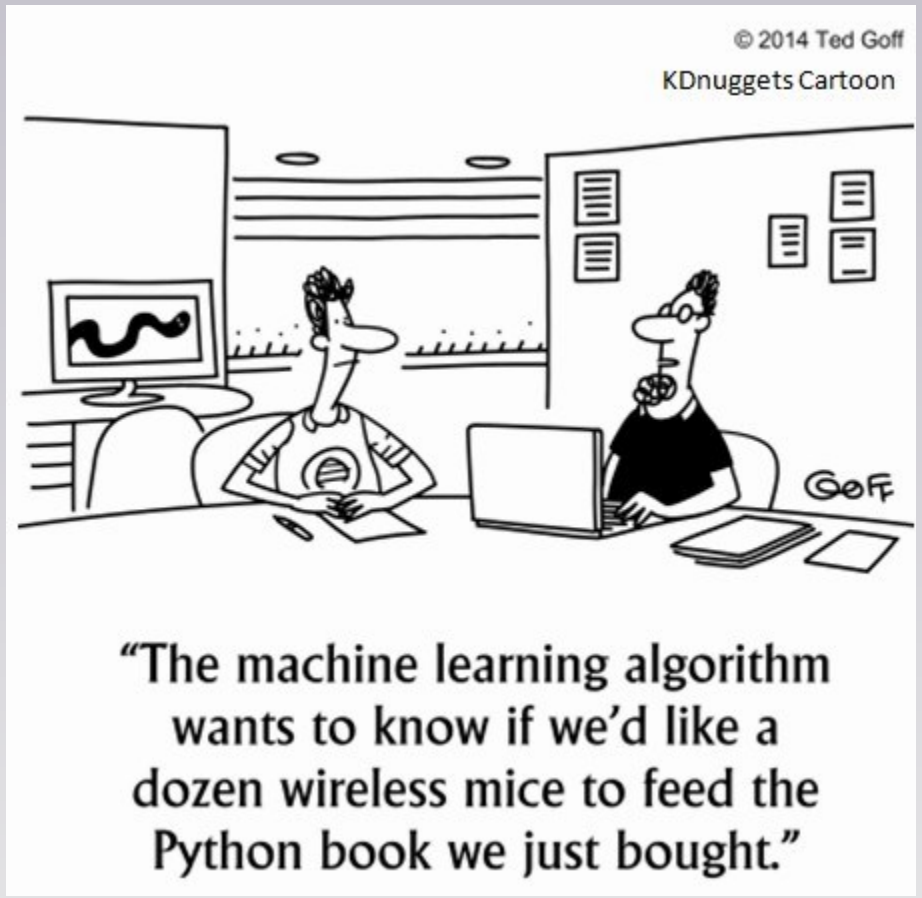
|eta| < 1.3



Recover optimal performance after splitting into categories



About Systematic Errors



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

MVA and Systematic Uncertainties

- Multivariate Classifiers THEMSELVES don't have systematic uncertainties
  → even if trained on a "phantasy Monte Carlo sample"
  - there are only "bad" and "good" performing classifiers !
    - OVERTRAINING is NOT a systematic uncertainty !!
    - difference between two classifiers resulting from two different training runs DO NOT CAUSE SYSTEMATIC ERRORS
  - same as with "well" and "badly" tuned classical cuts
  - MVA classifiers: → only select regions in observable space

- Efficiency estimate (Monte Carlo) → statistical/systematic uncertainty
  - involves "estimating" (uncertainties in) distribution of $PDF_{y_{S(B)}}$
    - statistical "fluctuations" → re-sampling (Bootstrap)
    - "smear/shift/change" input distributions and determine $PDF_{y_{S(B)}}$

  → estimate systematic error/uncertainty on efficiencies

- Only involves "test" sample..
  - systematic uncertainties have nothing to do with the training !!

Summary

- **MVA or ML algorithms**
  - → **parametrize likelihood** ratio (or a monotonic function thereof)
    - → **decision boundaries or 'event weights'**
  - → **Parametrize the 'target function'**
    - → **'regression'**

→ **Generative or discriminative algorithms**

- → **Multidimensional/projective Likelihood (rec. pdf)**
- → **(Linear) discriminators etc. → minimize a loss function**

→ **Take care in training, validation and testing**

- → **Don't want over/'under'-training but the best classifier!**

Backup

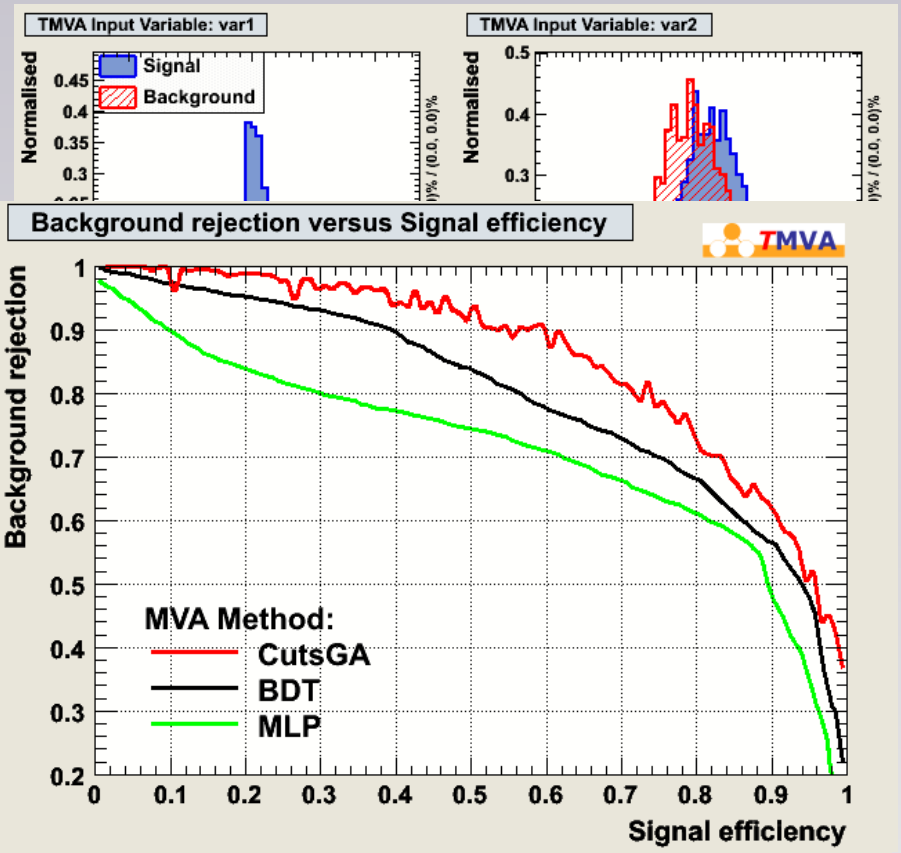
MVA and Systematic Uncertainties

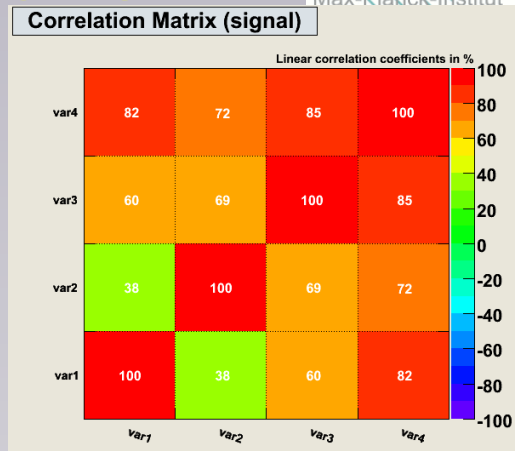
Max-Planck-Institut für Kernphysik

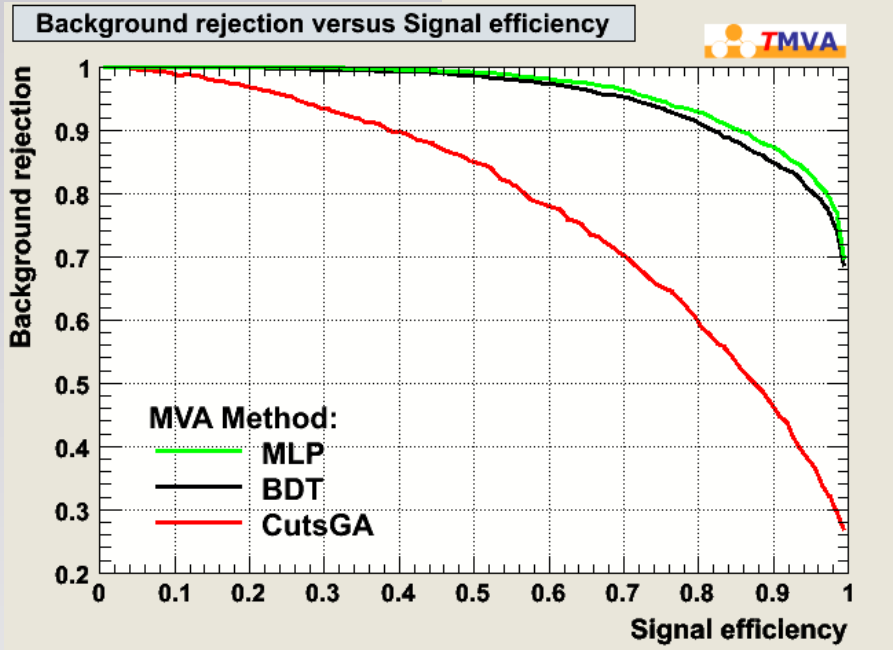
- Don't be afraid of correlations!
  - typically "kinematically generated" → easily modeled correctly
  - "classical cuts" are also affected by "wrongly modeled correlations"
  - MVA method let's you spot mis-modeled correlations!
    - "projections" of input variables
    - + the combined MVA test statistic "y(x)" !

Systematic "Error" in Correlations

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





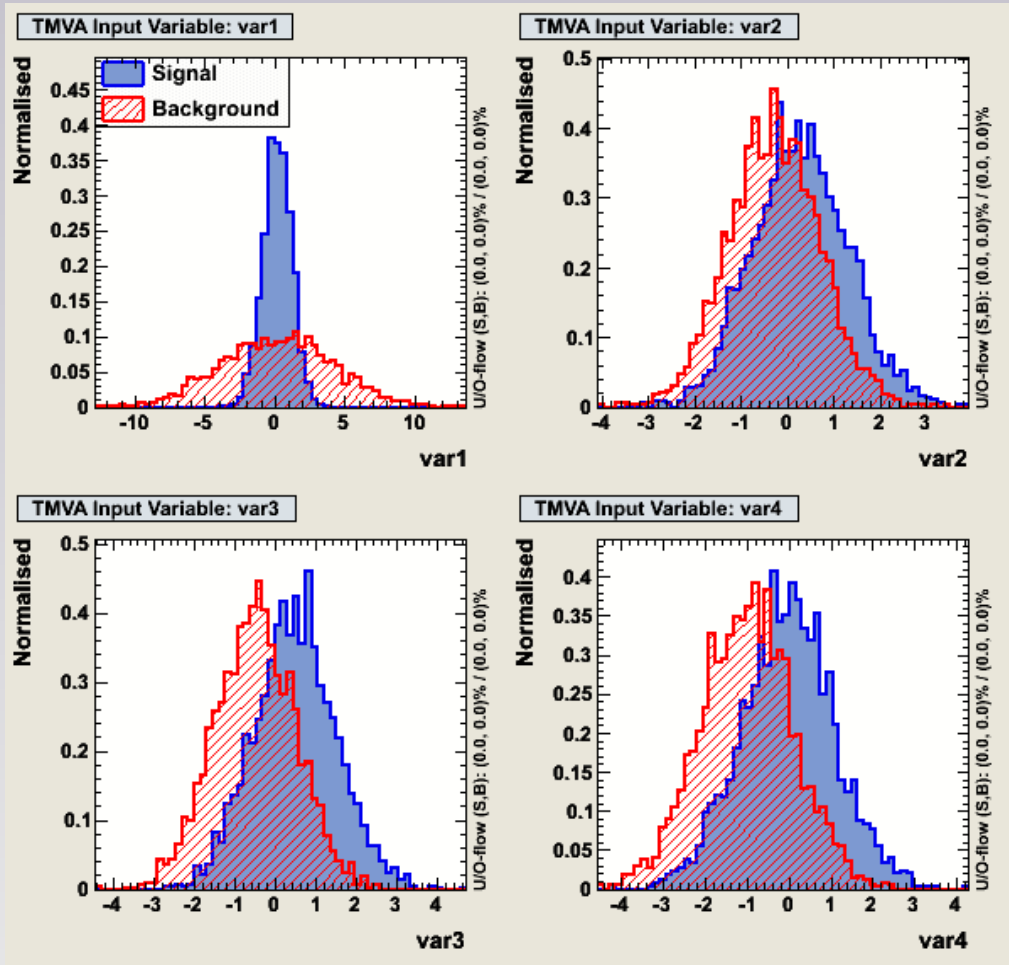


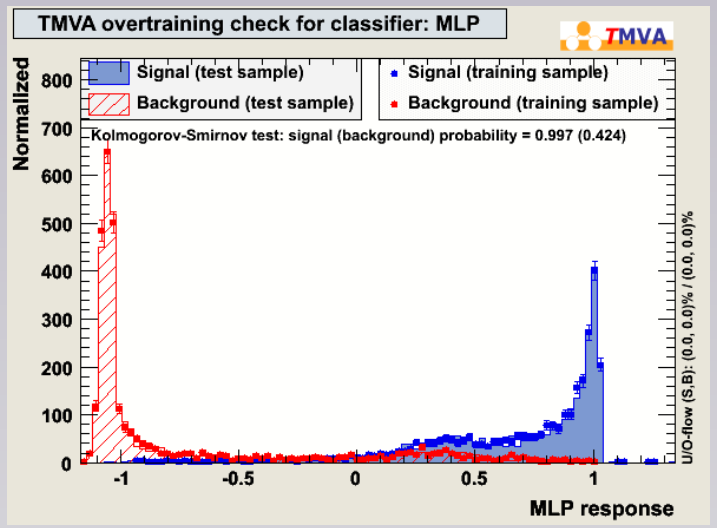
- Assume in "real data" there are NO correlations → SEE what happens!!

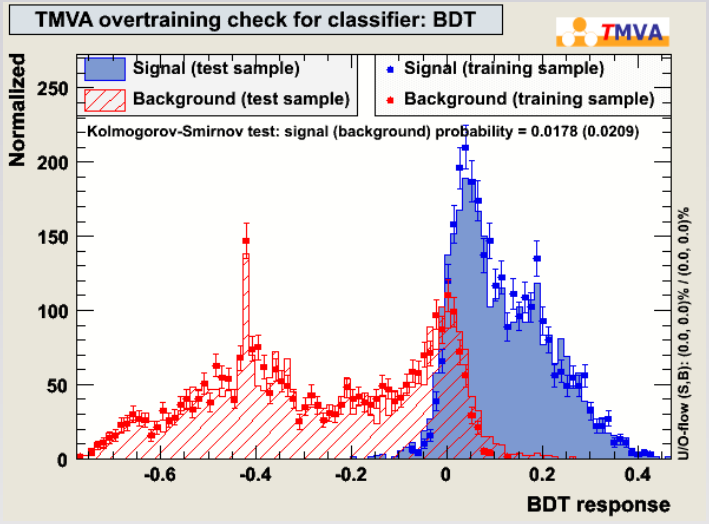
Systematic “Error” in Correlations

Max-Planck-Institut für Kernphysik

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





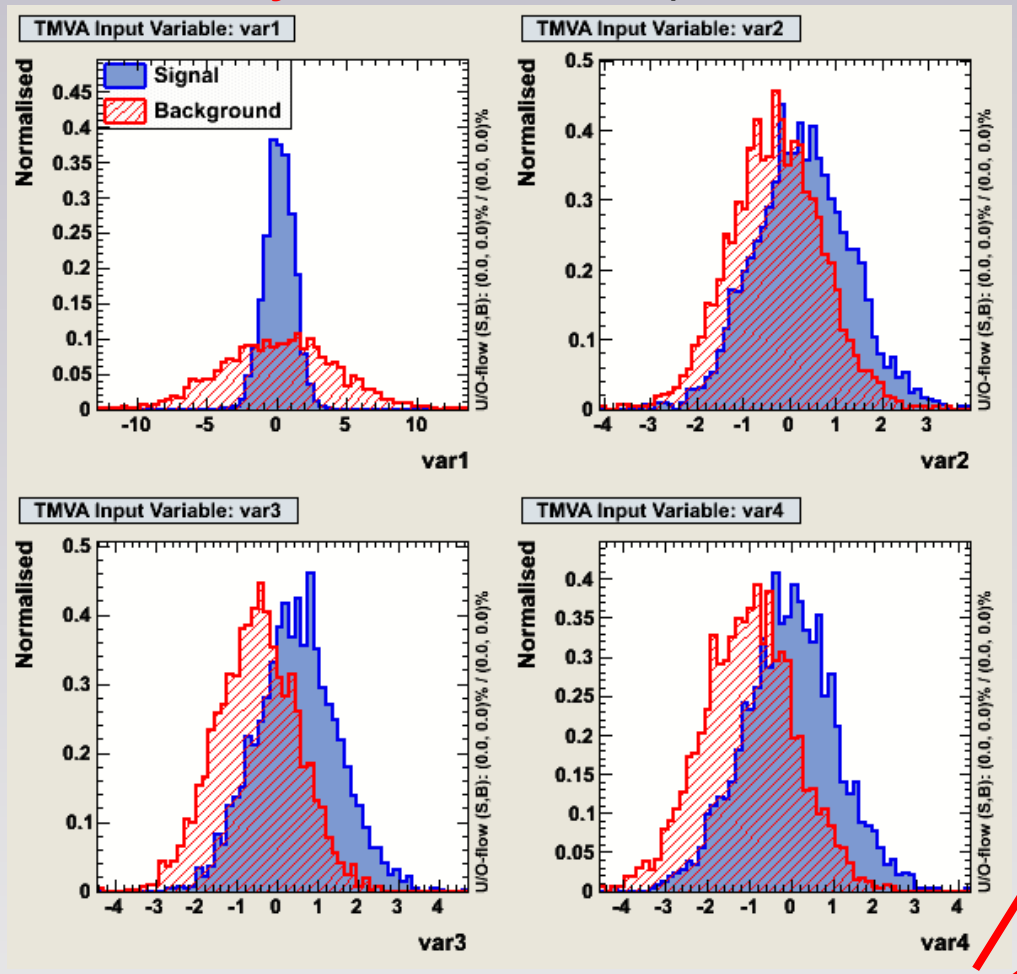


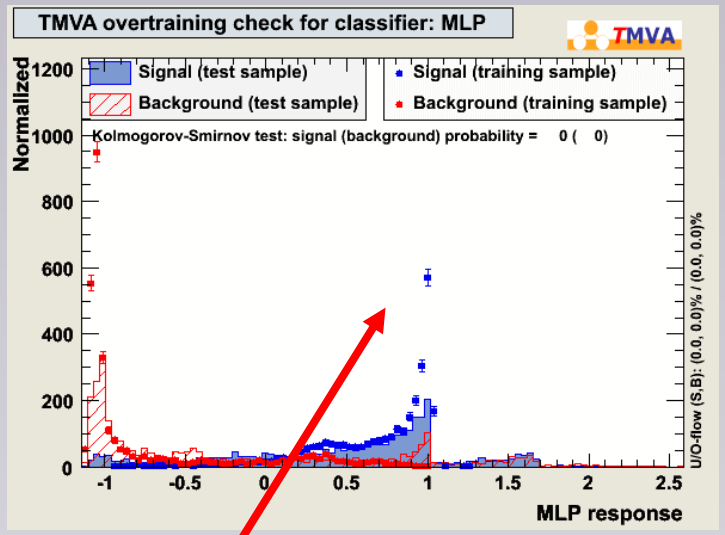
Systematic “Error” in Correlations

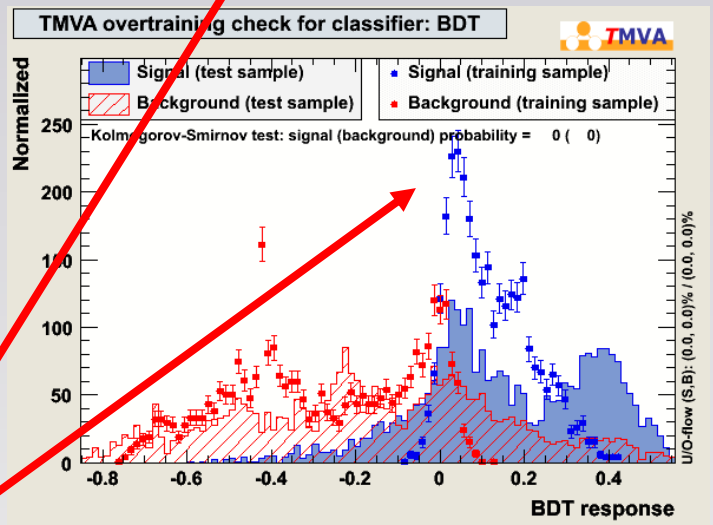
Max-Planck-Institut für Kernphysik

- **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….)

Robustness against systematic Uncertainties

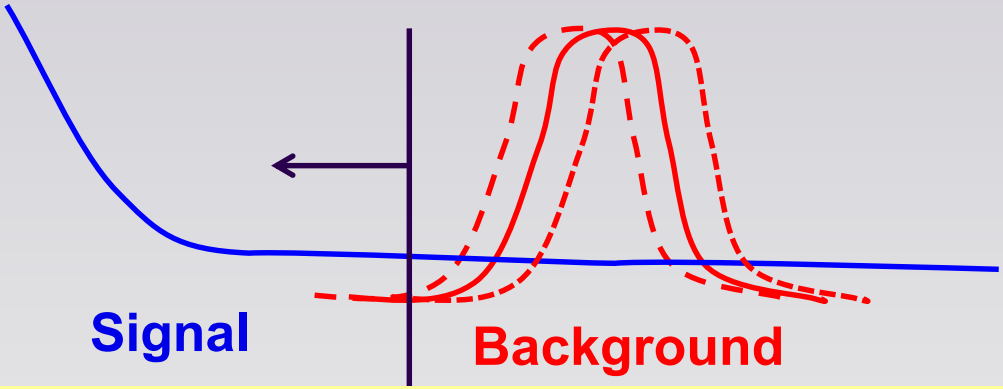
- minimize “systematic” uncertainties (robustness)

→ “classical cuts” : do not cut near steep edges, or in regions of large sys. uncertainty

→ hard to translate to MVAs:

  → artificially degrade discriminative power (shifting/smearing) of systematically “uncertain” observables IN THE TRAINING

    → remove/smooth the ‘edges’ → MVA does not try to exploit them



- **Note: if I KNEW about the error, I’d correct for it. I’m talking about ‘unknown’ systematics**

How does this look in 2D?

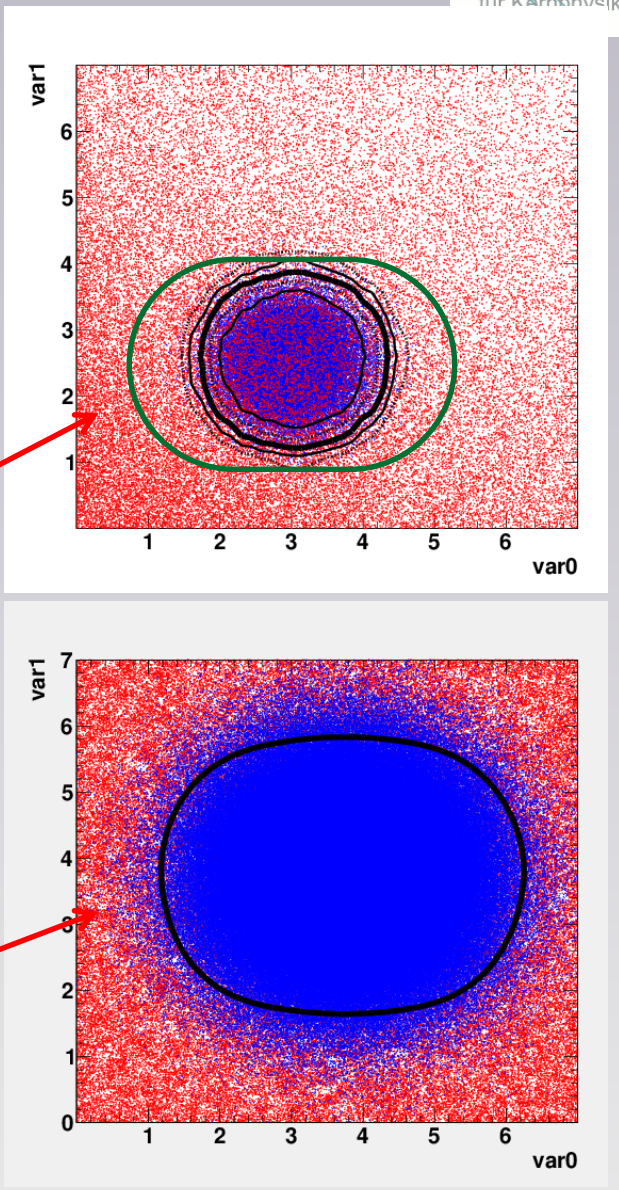
MVA-decision boundaries

- **Looser MVA-cut → wider boundaries in BOTH variables**

- **You actually want a boundary like THIS**
  - **Tight boundaries in var1**
  - **Loose boundaries in var0**

→ train MVA algorithm with 'problematic variables' transformed to make them less discriminant:

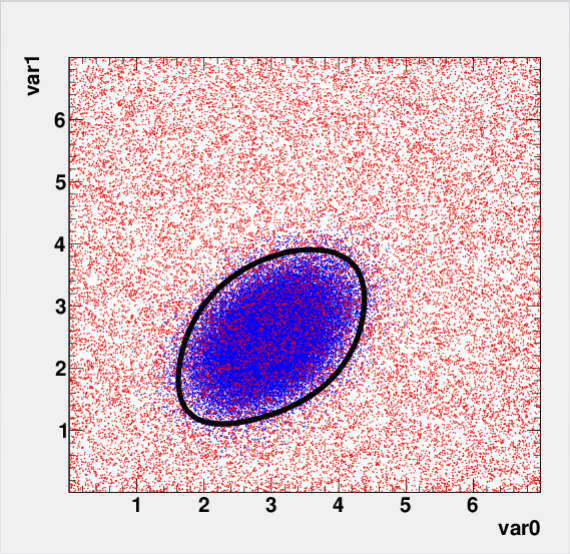
- **YES it works !**



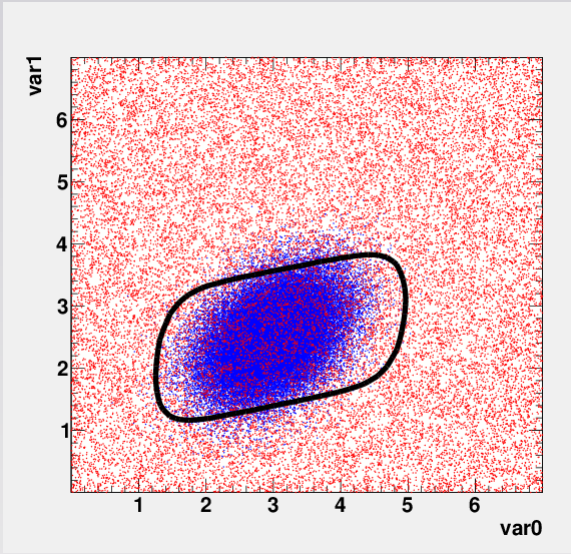
Another example..

- Hmm… also here, I'd still say it does exactly what I want it to do
  - The difficulty is to 'evaluate' or 'estimate' the advantage (reduction in systematic ←→ loss in performance)

bad decsision boundary

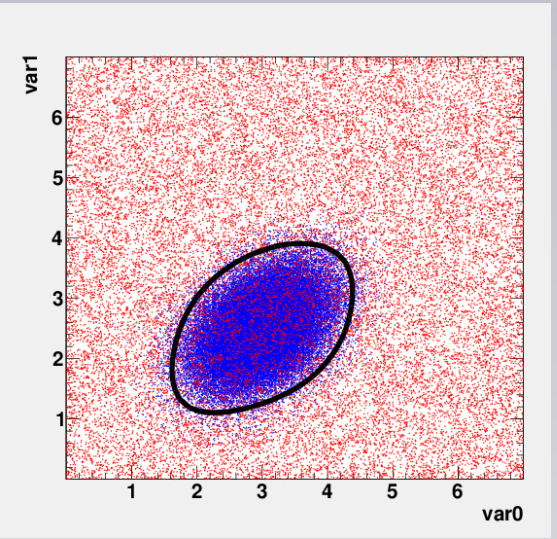


better

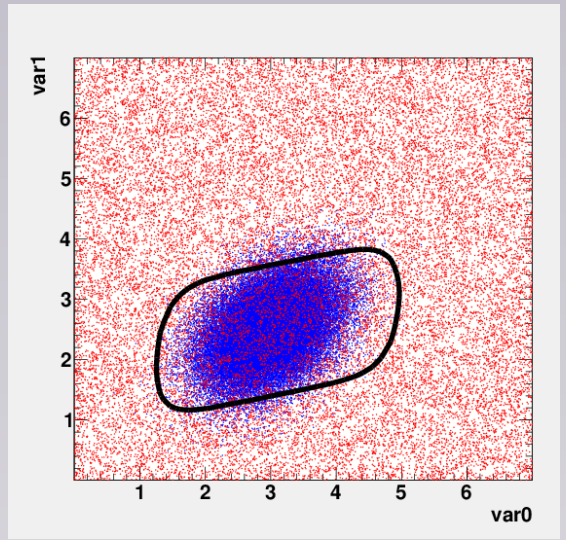


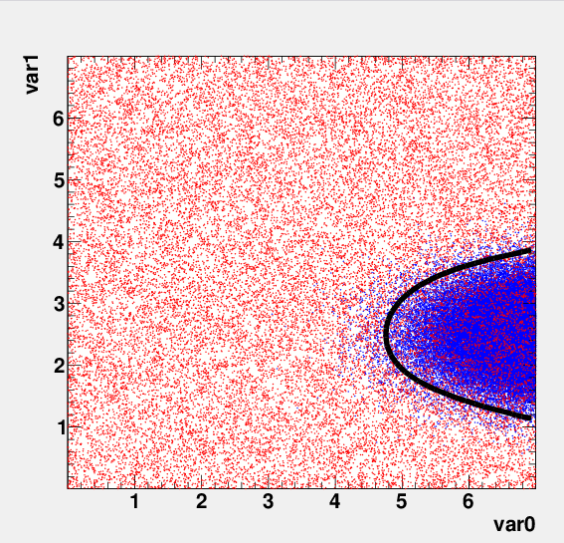
other examples..

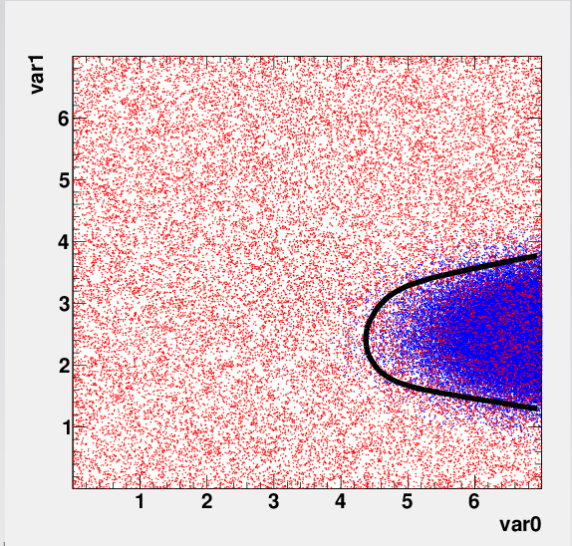
bad decsision boundary



better







Seems to work but:

difficult to 'evaluate' or 'estimate' the advantage (reduction in systematic ←→ loss in performance)

How does this look in 2D?

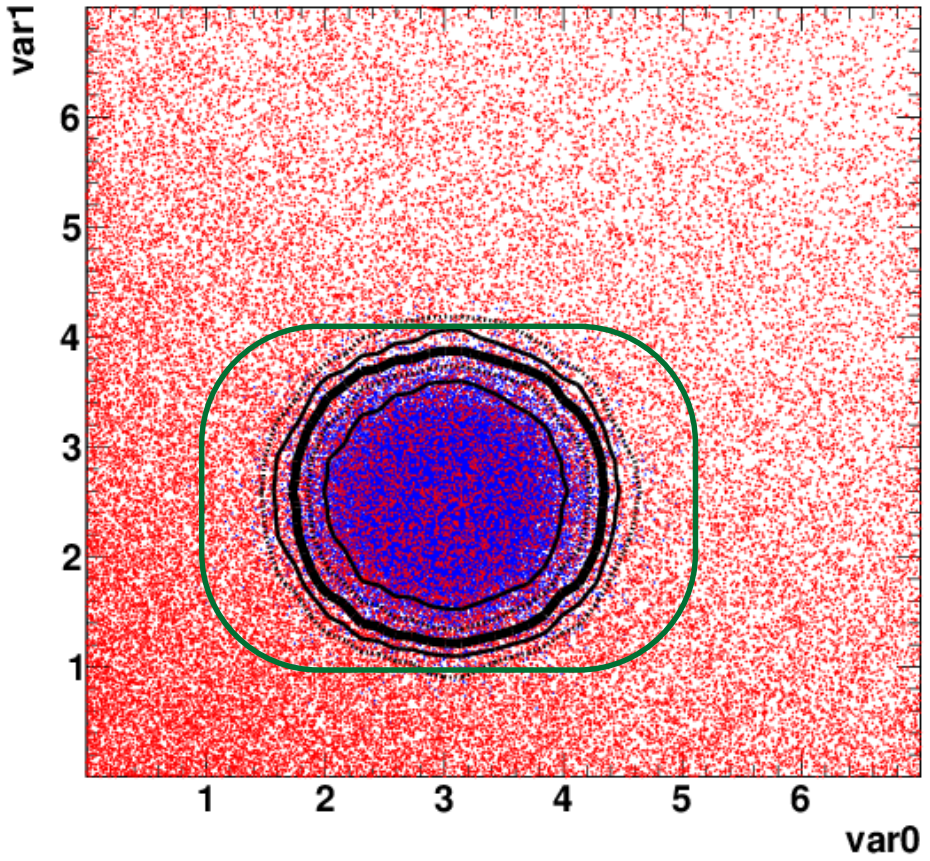
Max-Planck-Institut für Kernphysik

MVA-decision boundaries

- **Looser MVA-cut → wider boundaries in BOTH variables**

What if you are sure about the peak's position in var1, but less sure about var0 ?

- **You actually want a boundary like THIS**
  - **Tight boundaries in var1**
  - **Loose boundaries in var0**

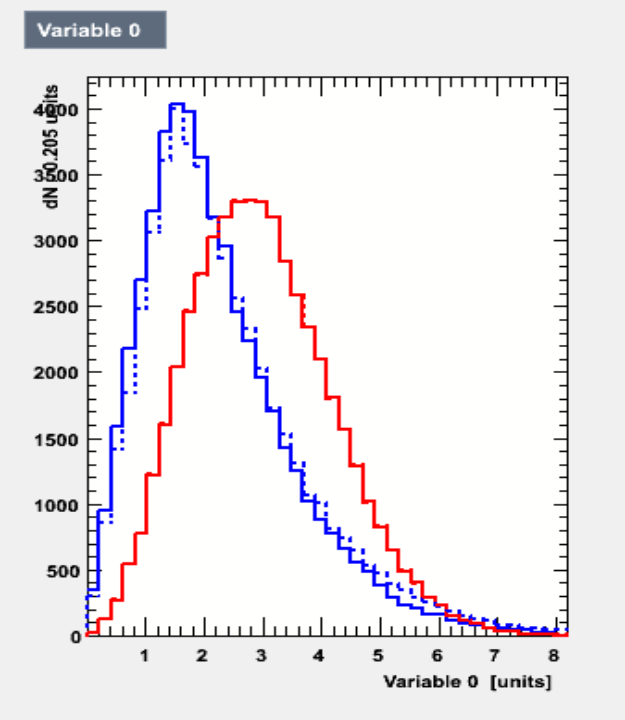


Reduce information content

- **Looking for a general tool to 'force' any MVA algorithm, not to rely too much on exact feature:**
  - Similar: early stopping techniques in Neural networks to avoid overtraining

→ reduce difference between "signal" and "background"

→ or reduce information content in each, "signal" and "background"



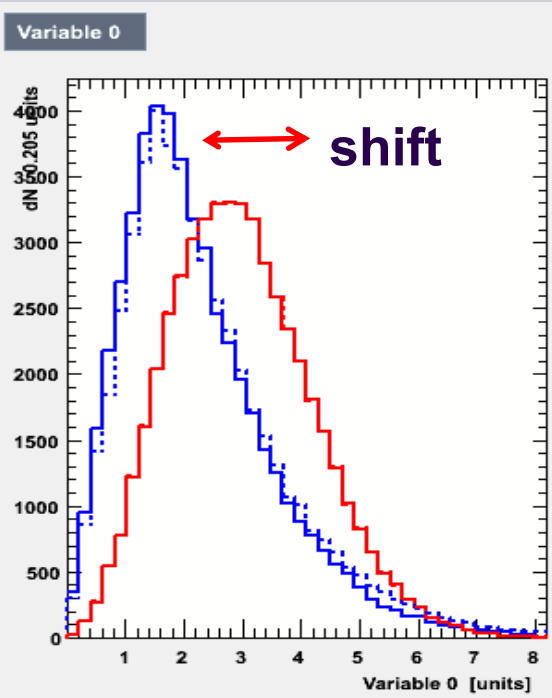
- **Here: one would for example "shift" such that signal and backgr. are less separated**
- **However, that's not "universal"**

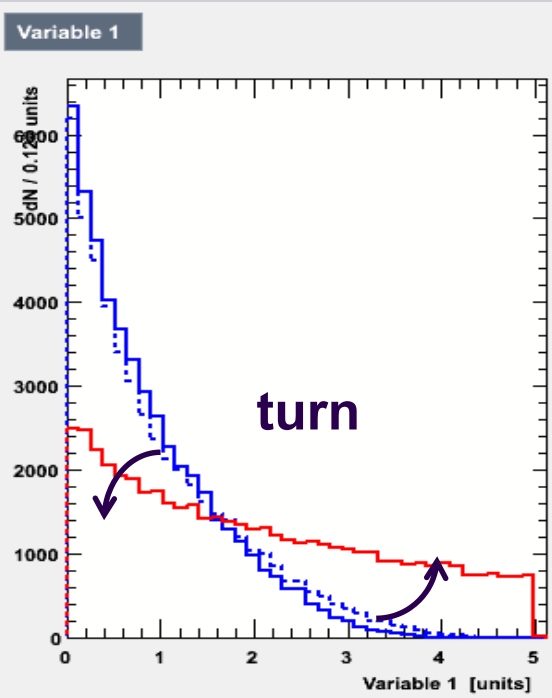
Reduce information content

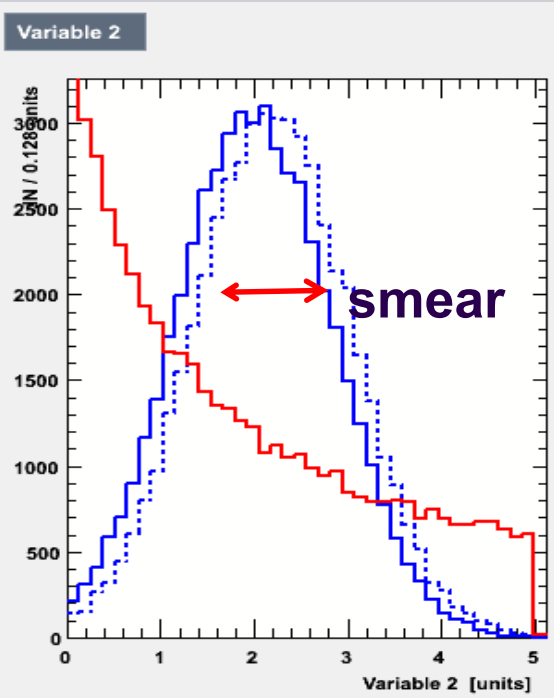
- **Looking for a general tool to 'force' any MVA algorithm, not to rely too much on exact feature:**
  - Similar: early stopping techniques in Neural networks to avoid overtraining

→ reduce difference between "signal" and "background"

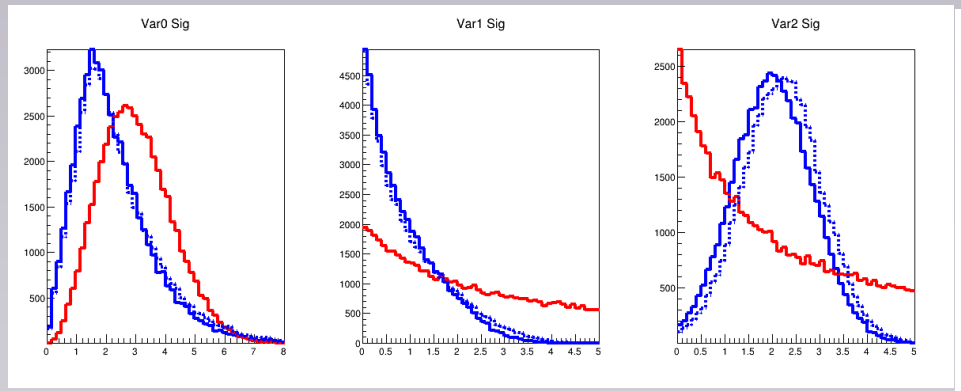
→ or reduce information content in each, "signal" and "background"

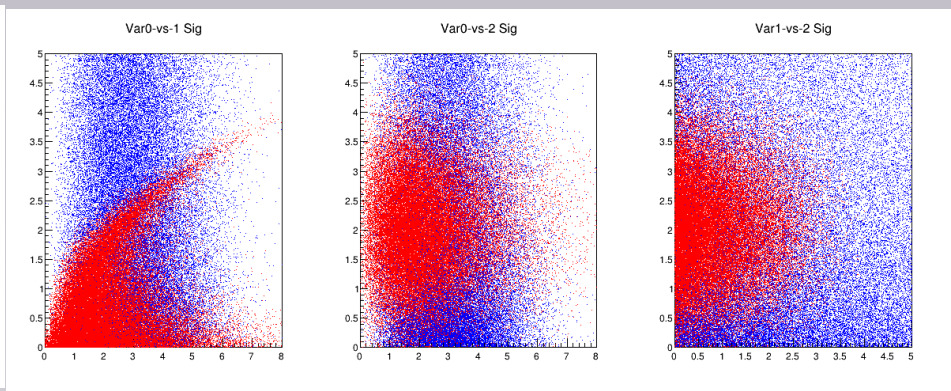


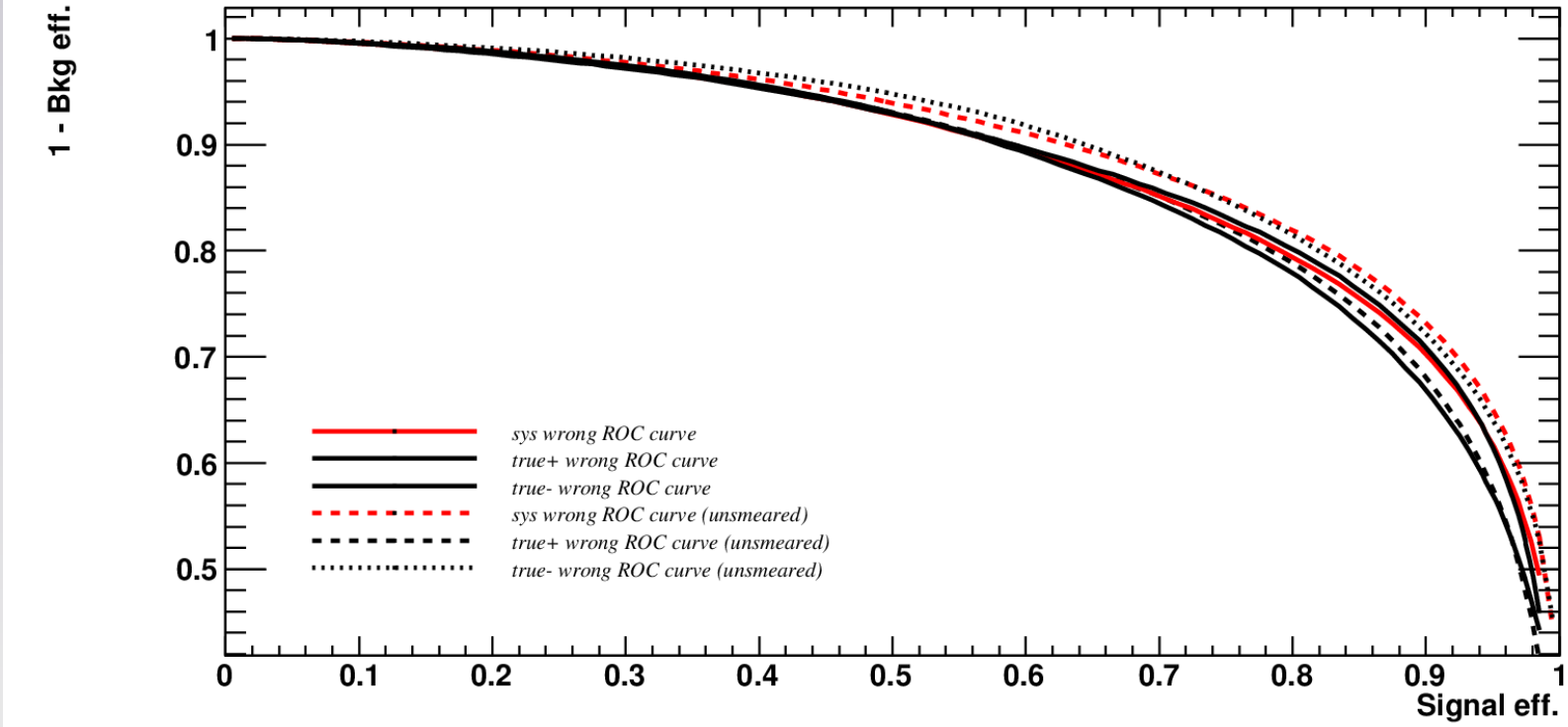




Back to my "complicated" 3D example

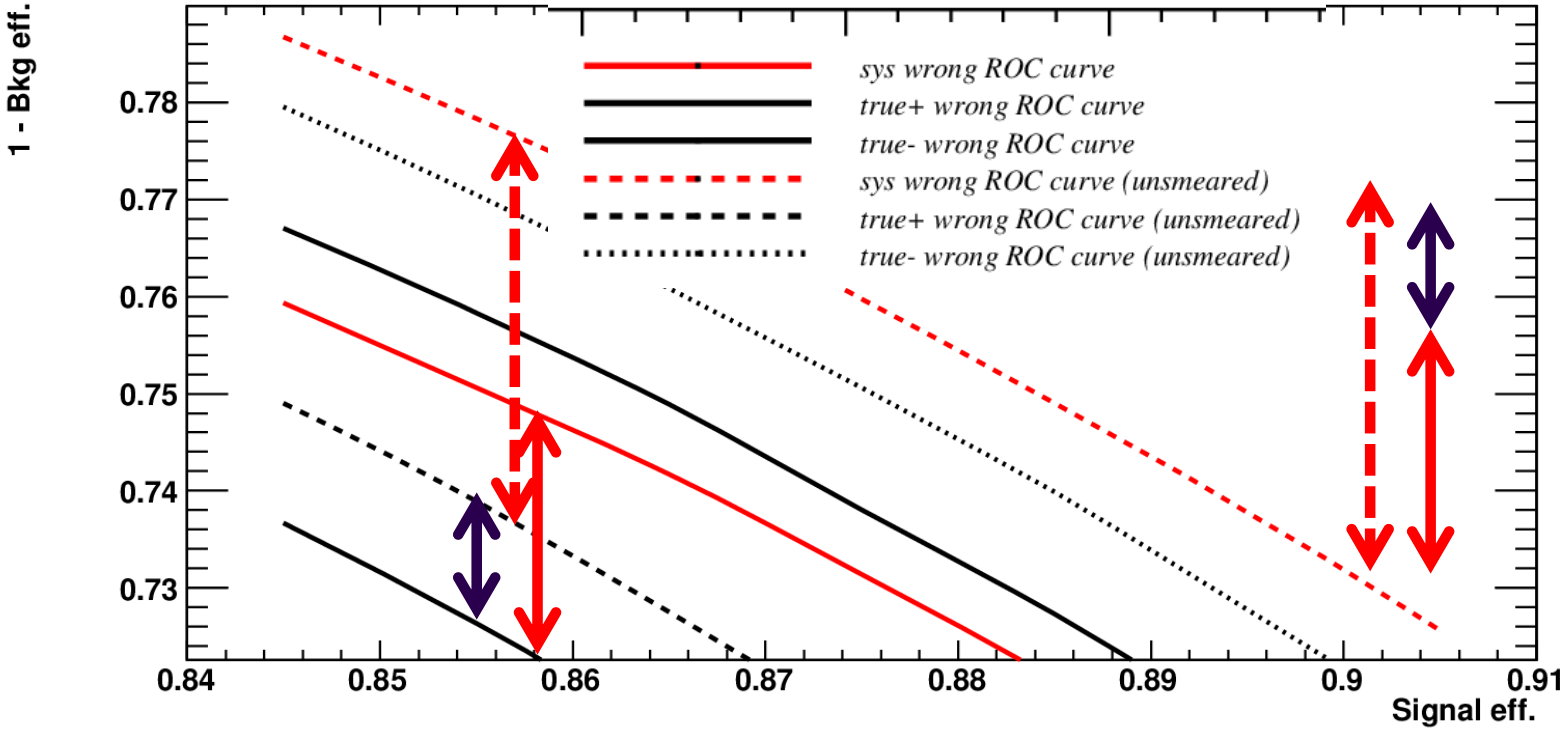






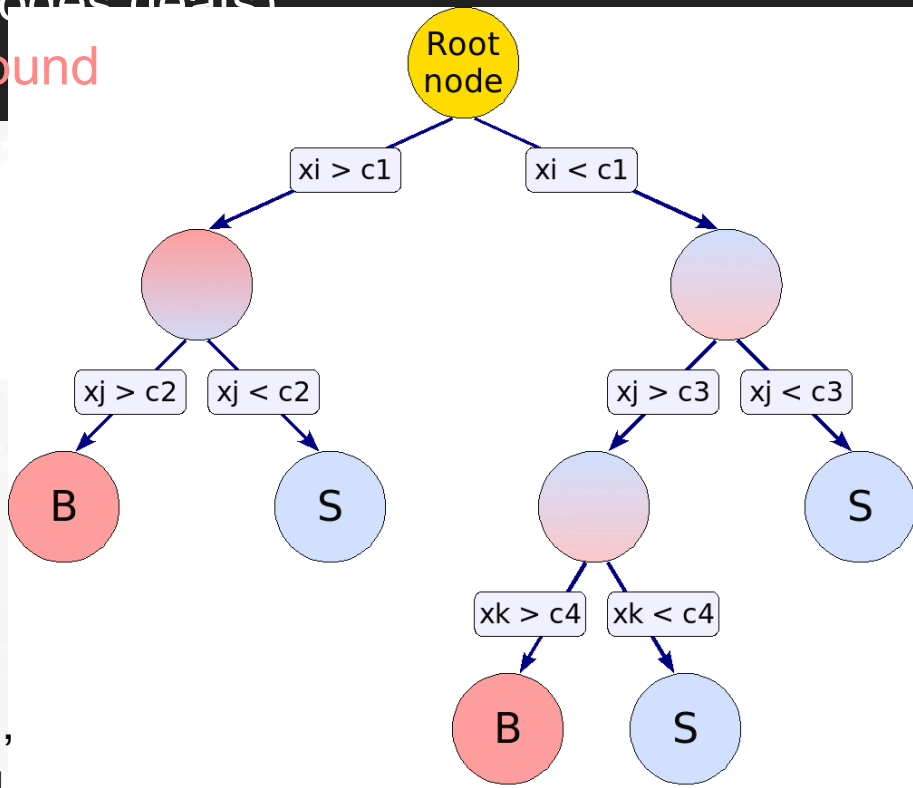
ROC Cuve - Zoom

- compare: difference between red (what you think you have) and black (what your algorithm applied to nature might actually provide)
- do this for solid (smeared) and dashed (unsmeared) classifiers



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

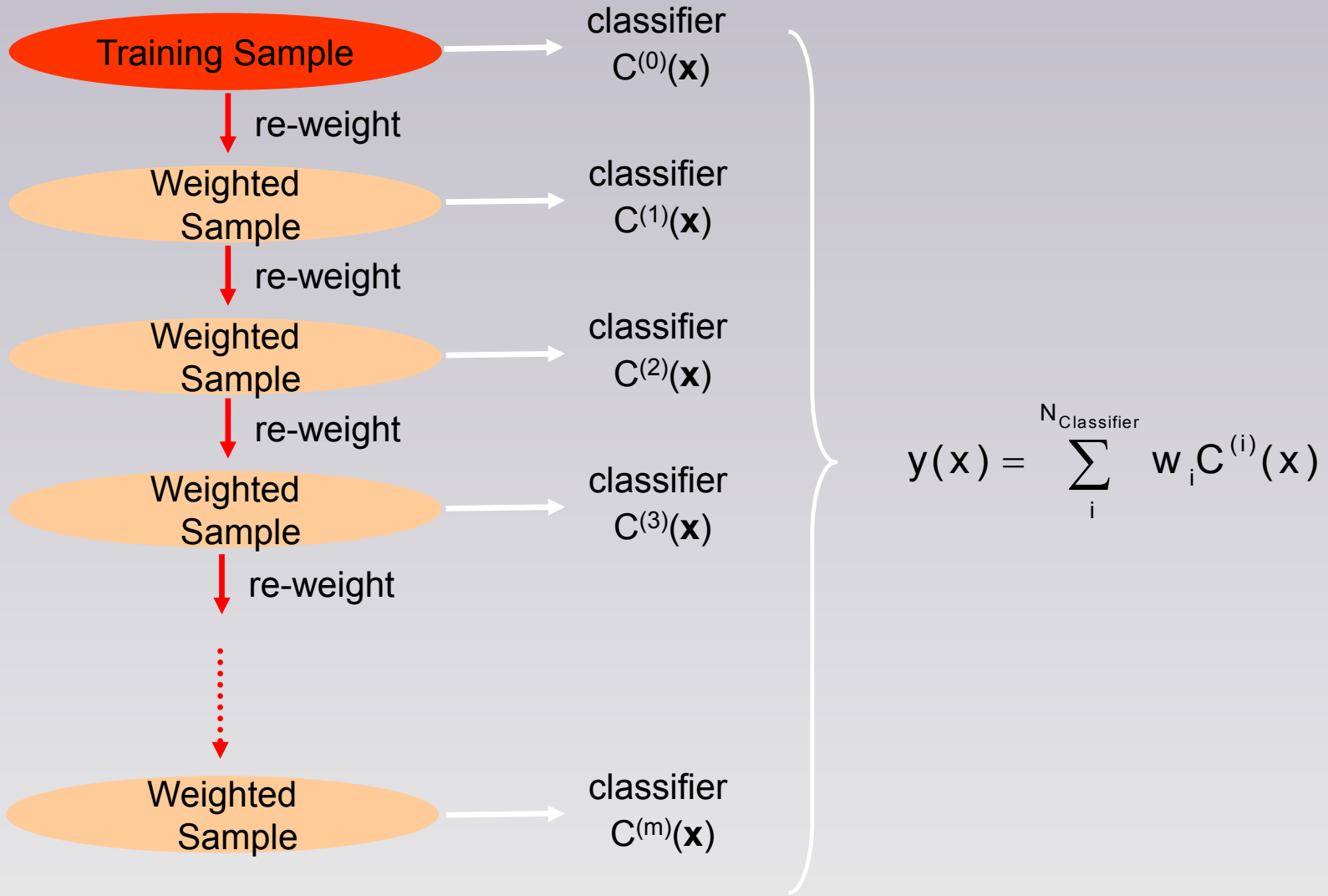


- Each branch → one standard "cut" sequence
  - easy to interpret, visualised
- Disadvatage → very sensitive to statistical fluctuations in training data

- 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
  - increases performance

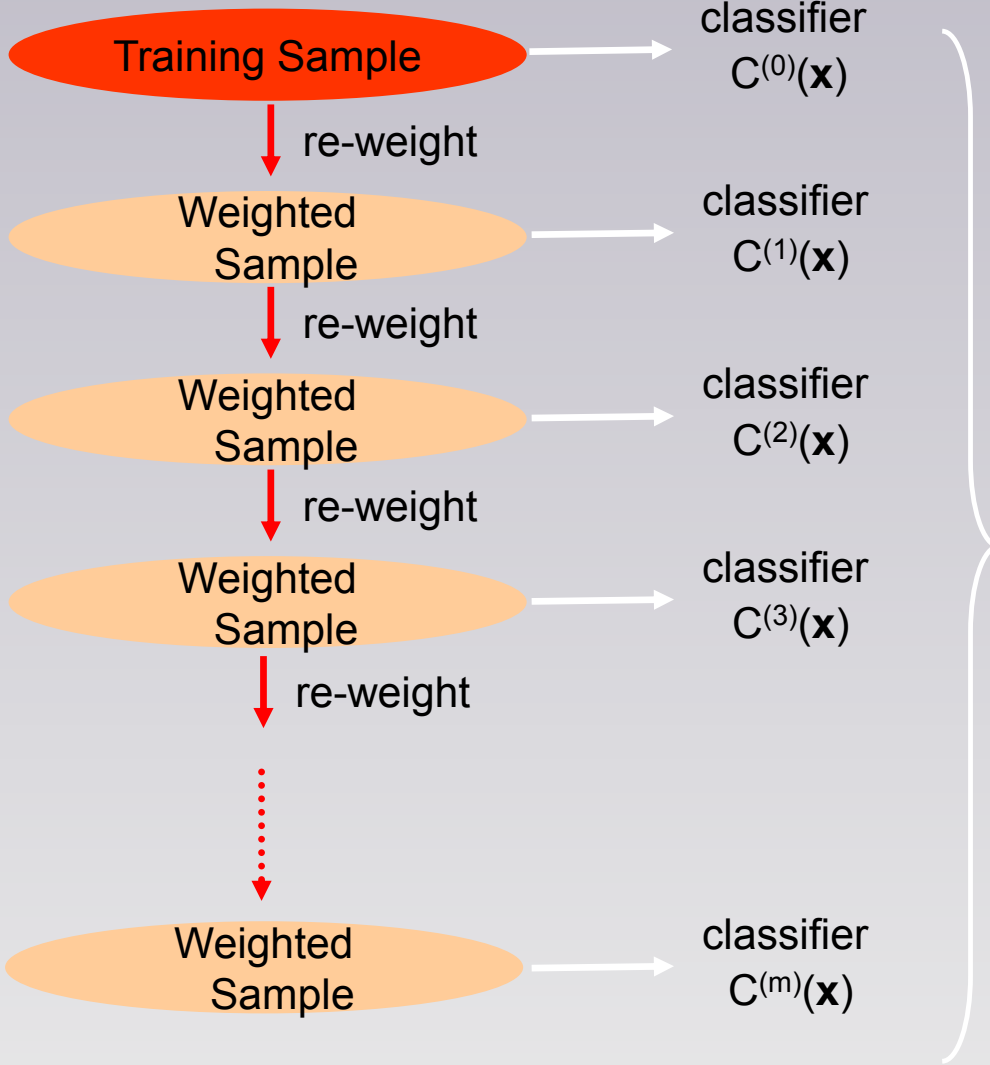
→ became popular in HEP since MiniBooNE, B.Roe et.a., NIM 543(2005)

Boosting



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

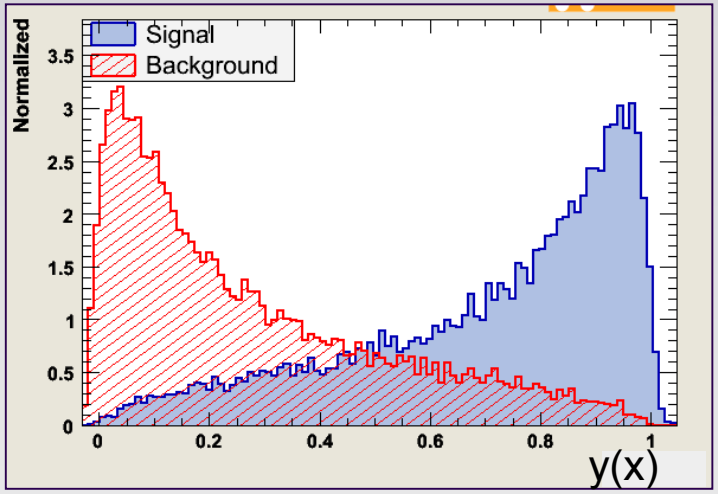
Adaptive Boosting (AdaBoost)



- AdaBoost re-weights events misclassified by previous classifier:

$$\frac{1-f_{err}}{f_{err}}\;;\; f_{err}=\frac{misclassified}{all\ events}$$

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



Boosted Decision Trees

- Are very popular in HEP
  - Robust and easy to train,
  - get good results
- But: when we adopted BDTs,
  - In 2006 ANNs just started their big breakthrough in the ML community with remarkable advances in DEEP Learning !

→ Let's move on to Neural Networks

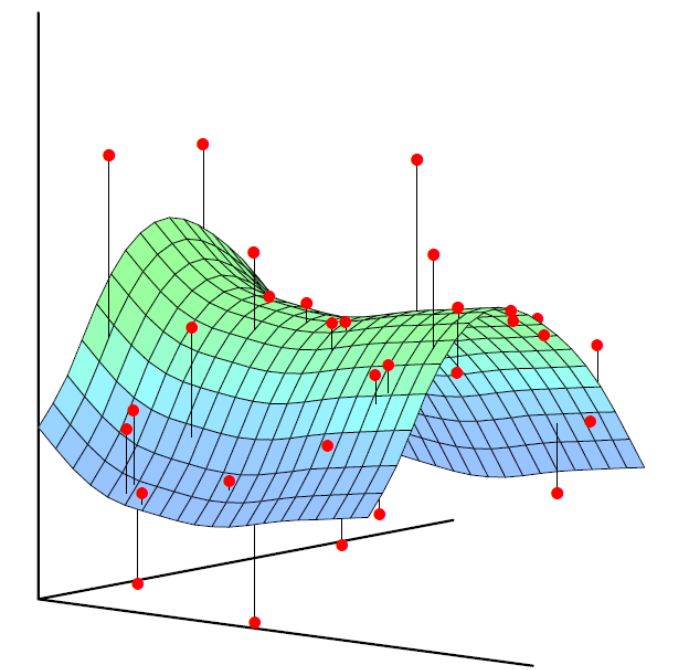
Machine Learning - Multivariate Techniques

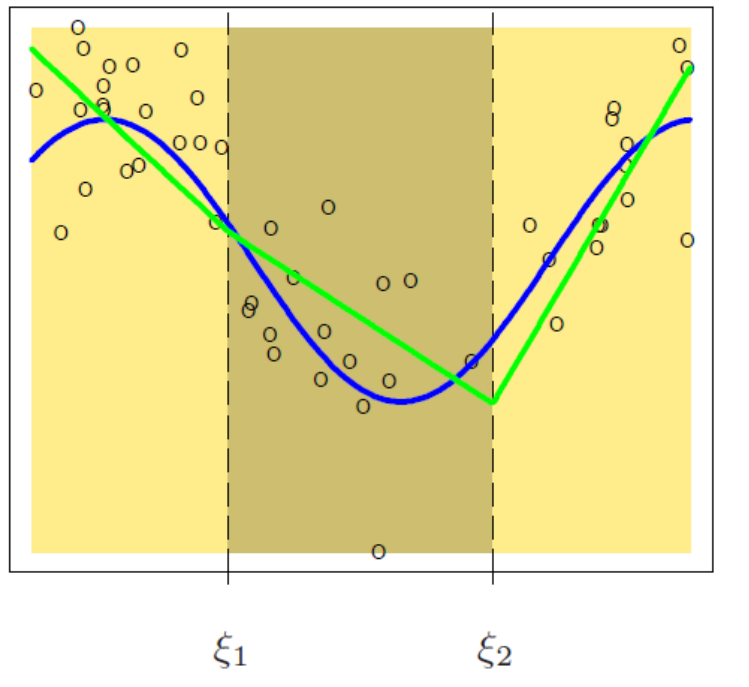
Max-Planck-Institut für Kernphysik

- if we do not know that 'straight line' or 'polynomial' is a good model (particularly in higher dimension) ?

→ general, simple, piecewise models

→ fit non-analytic → computer → machine learning





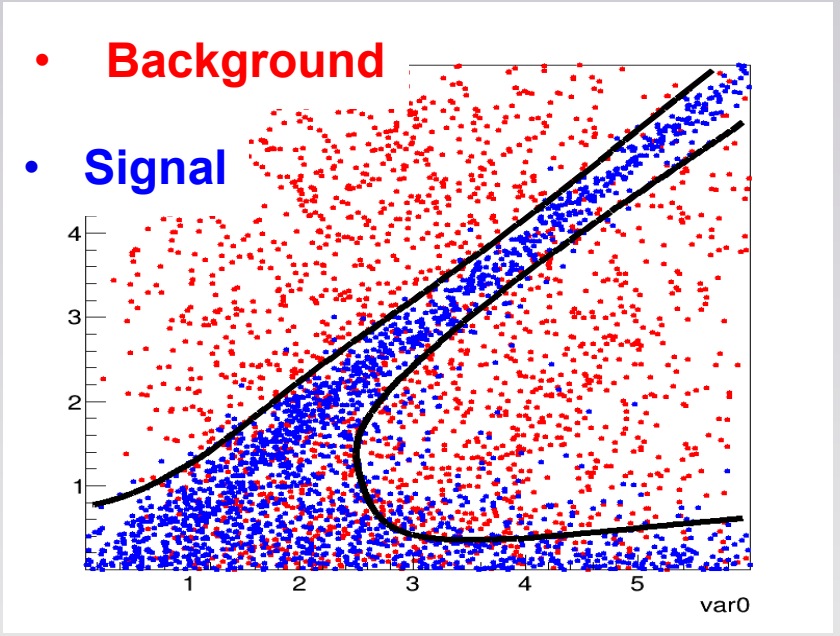
Machine Learning - Multivariate Techniques

- fitted non-analytic function may approximate:
  - Likelihood ratio:

$$y(x) = \frac{PDF(x|S)}{PDF(x|B)} \quad ; \; x = \begin{pmatrix} var0 \\ var1 \\ . \\ . \\ . \end{pmatrix}$$

$$y(x) = const$$

→ decision boundary



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" → $P(C) = f_C = \frac{n_C}{n_{tot}}$

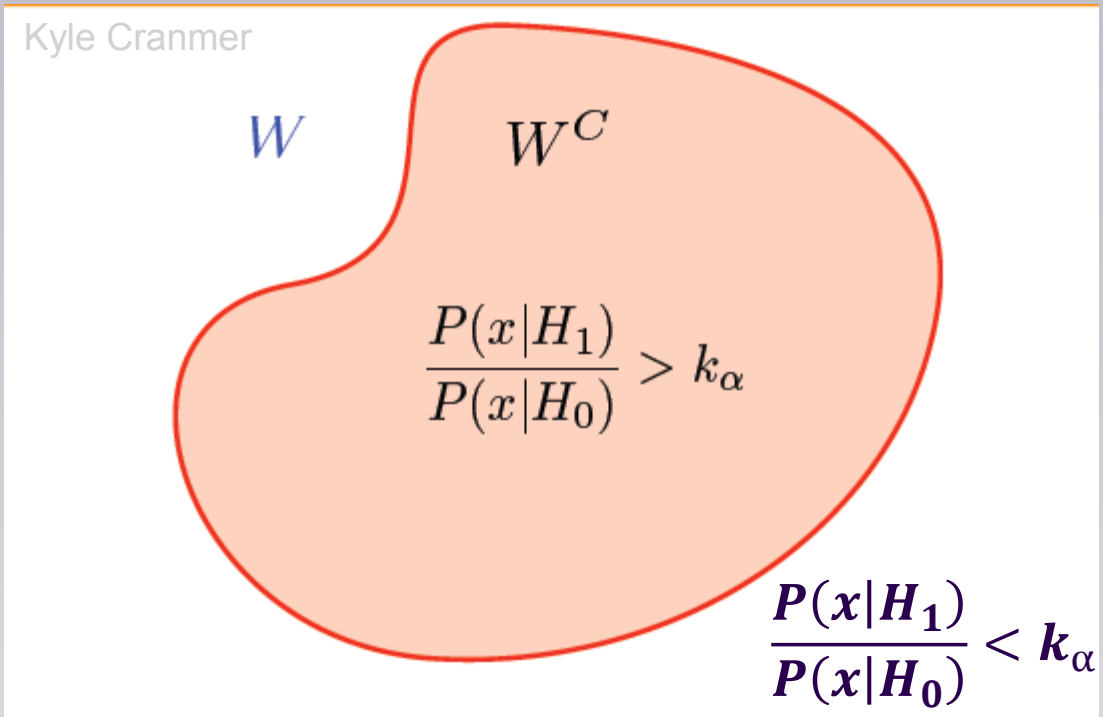
$$P(Class = C \mid y) = \frac{P(y \mid C)}{P(y)}$$

Posterior probability

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

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

Neyman Pearson Lemma



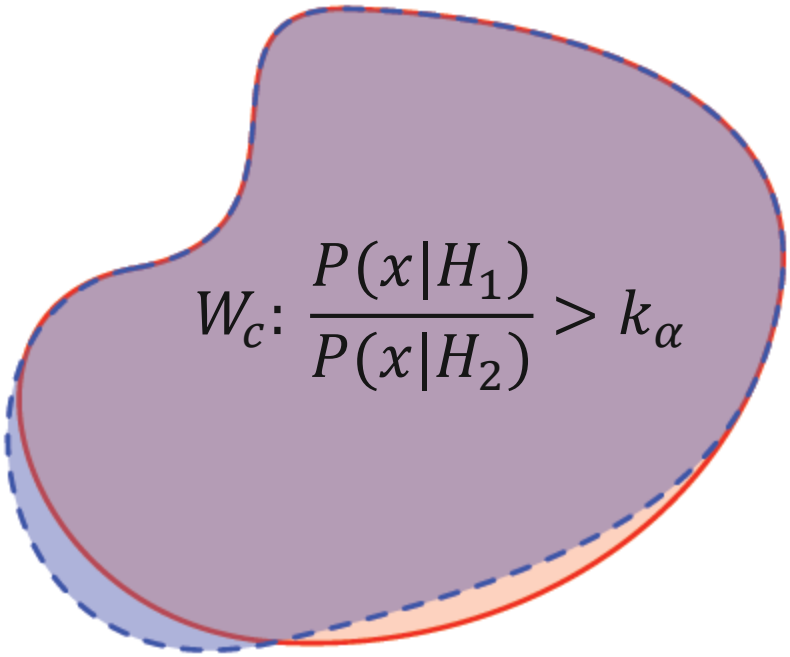
graphical proof of Neyman Pearson's Lemma:
(graphics/idea taken from Kyle Cranmer)

- **the critical region W^C given by the likelihood ratio $\frac{P(x|H_1)}{P(x|H_0)}$**

→ **for each given size α (risk of e.g. actually making a false discovery)**
= **the statistical test with the largest power $1-\beta$ (chances of actually discovering something given it's there)**

Neyman Pearson Lemma

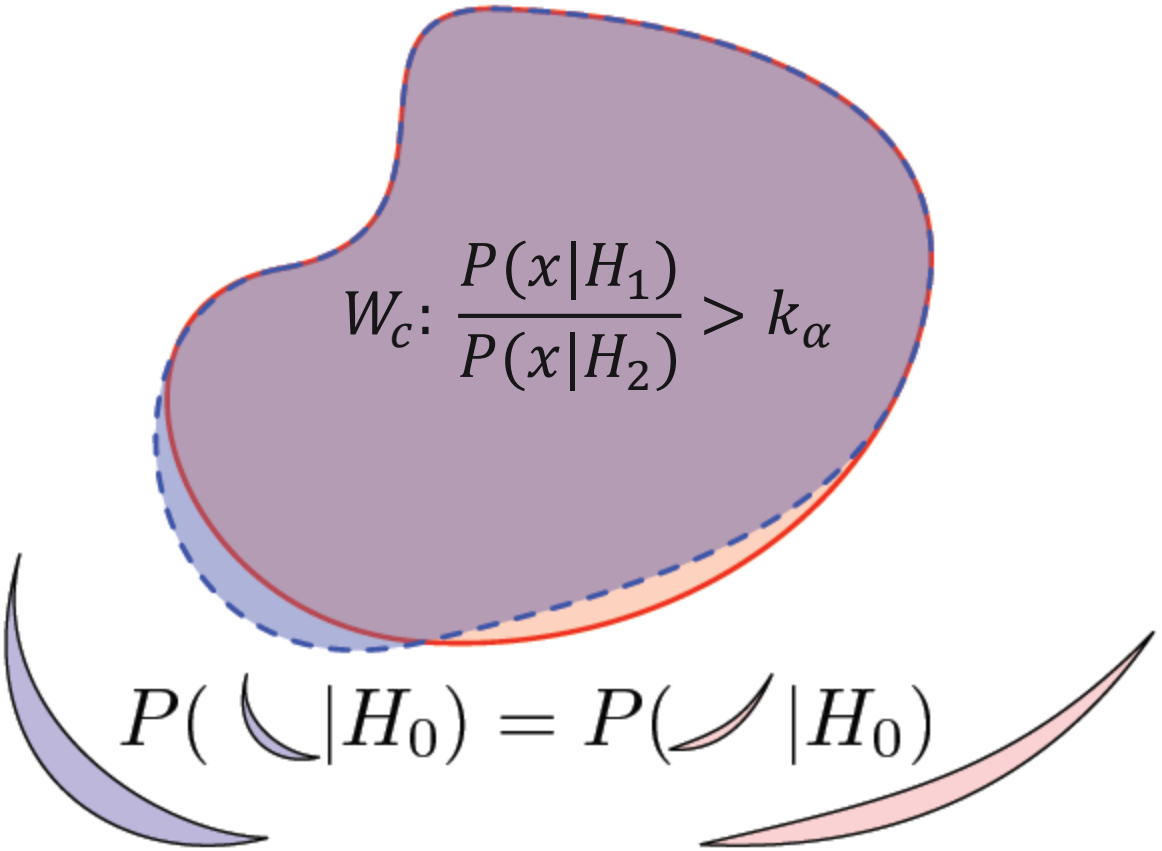
Kyle Cranmer



assume we want to modify/find another “critical” region with same size (α) i.e. same probability under H_0

Neyman Pearson Lemma

Kyle Cranmer

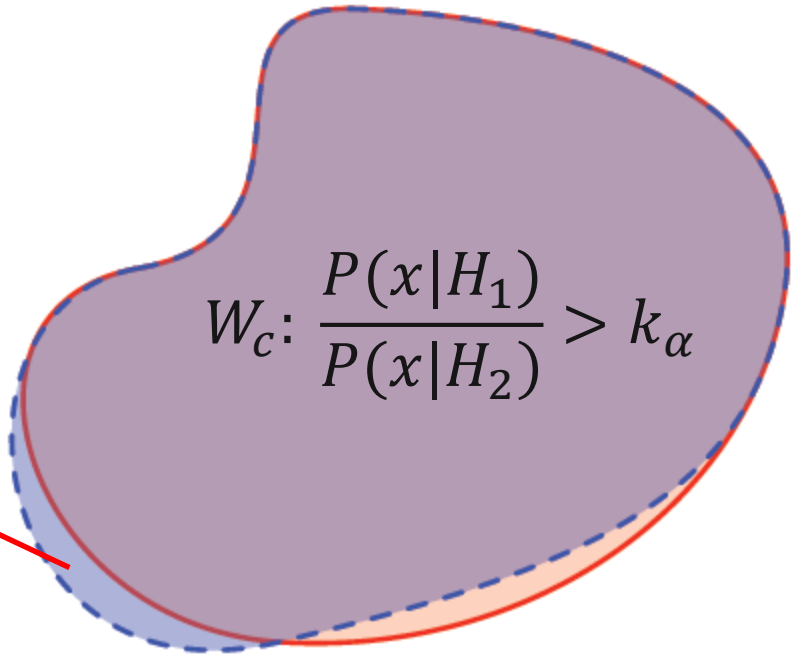


… as size (α) is fixed

$$\alpha = \int_C \boldsymbol{P(x|H_0)dx}$$

Neyman Pearson Lemma

Kyle Cranmer



outside "critical region" given by LL-ratio

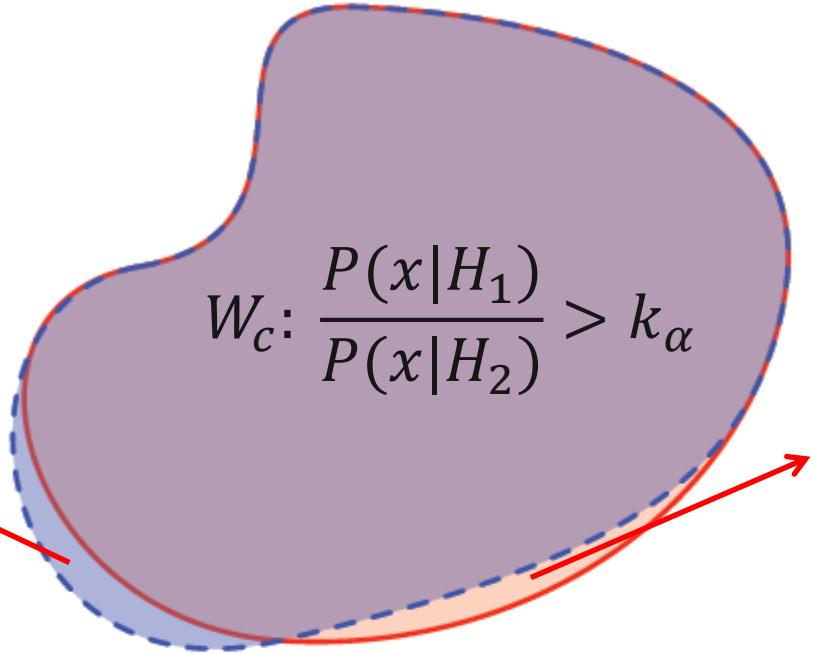
$$P(\text{ }|H_0) = P(\text{ }|H_0)$$

$$\frac{P(x|H_1)}{P(x|H_0)} < k_\alpha$$

$$P(\text{ }|H_1) < P(\text{ }|H_0)k_\alpha$$

Neyman Pearson Lemma

Kyle Cranmer



$$W_c: \frac{P(x|H_1)}{P(x|H_2)} > k_\alpha$$

outside "critical region" given by LL-ratio

inside "critical region" given by LL-ratio

$$P(\text{⌒}|H_0) = P(\text{⌒}|H_0)$$

$$\frac{P(x|H_1)}{P(x|H_0)} < k_\alpha$$

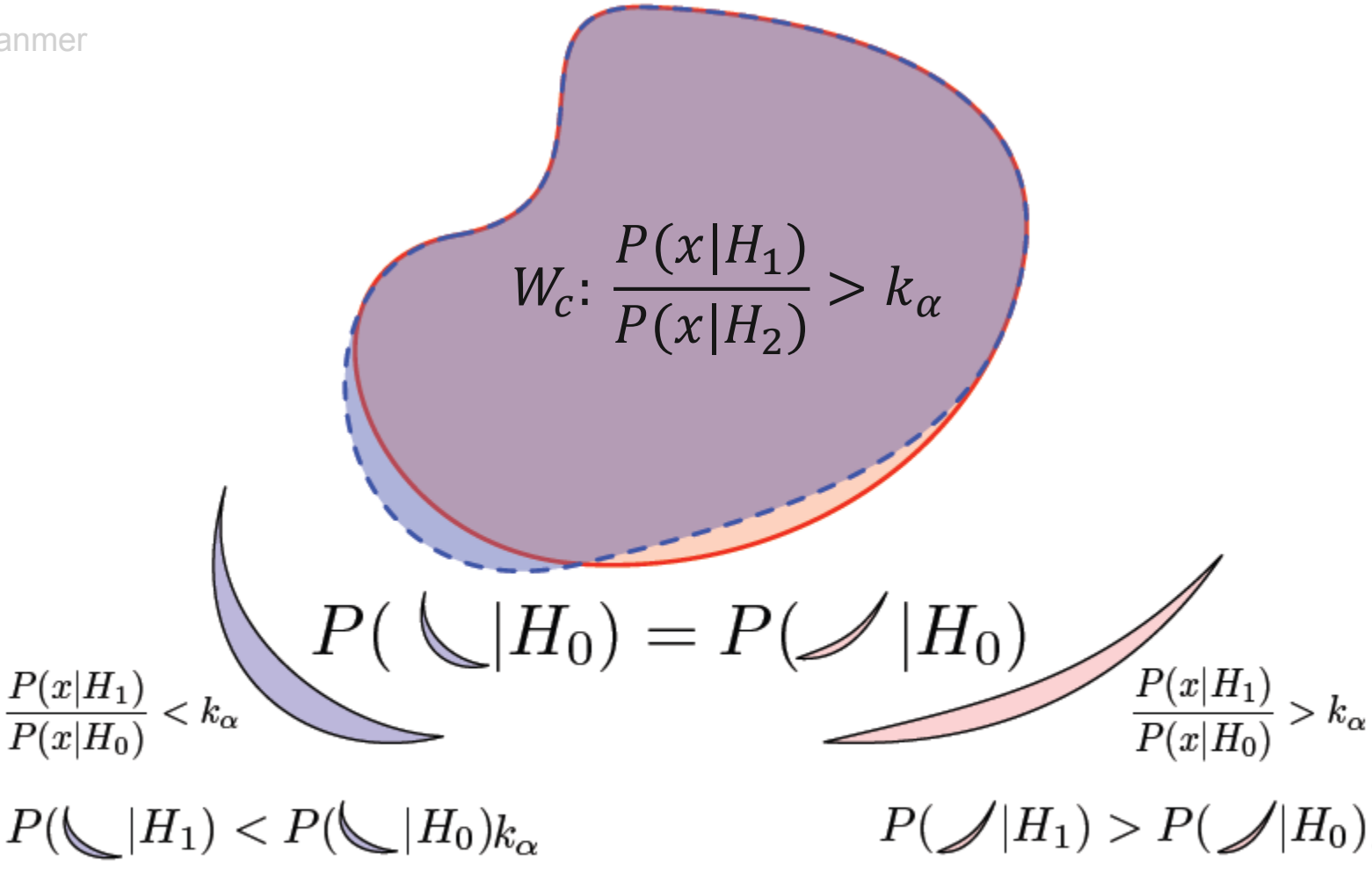
$$\frac{P(x|H_1)}{P(x|H_0)} > k_\alpha$$

$$P(\text{⌒}|H_1) < P(\text{⌒}|H_0) k_\alpha$$

$$P(\text{⌒}|H_1) > P(\text{⌒}|H_0) k_\alpha$$

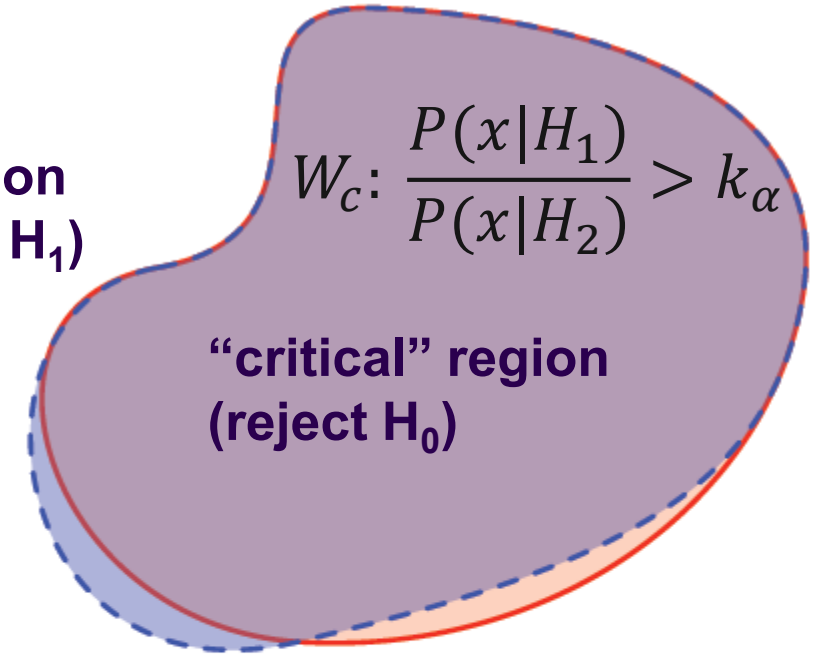
Neyman Pearson Lemma

Kyle Cranmer



Neyman Pearson Lemma

Kyle Cranmer



$$P(\text{left}|H_0) = P(\text{right}|H_0)$$

$$\frac{P(x|H_1)}{P(x|H_0)} < k_\alpha \qquad \frac{P(x|H_1)}{P(x|H_0)} > k_\alpha$$

$$P(\text{left}|H_1) < P(\text{left}|H_0)k_\alpha \qquad P(\text{right}|H_1) > P(\text{right}|H_0)k_\alpha$$

The NEW "acceptance" region has less power! (i.e. probability under H_1) q.e.d