Machine Learning and Multivariate Techniques in HEP data Analyses

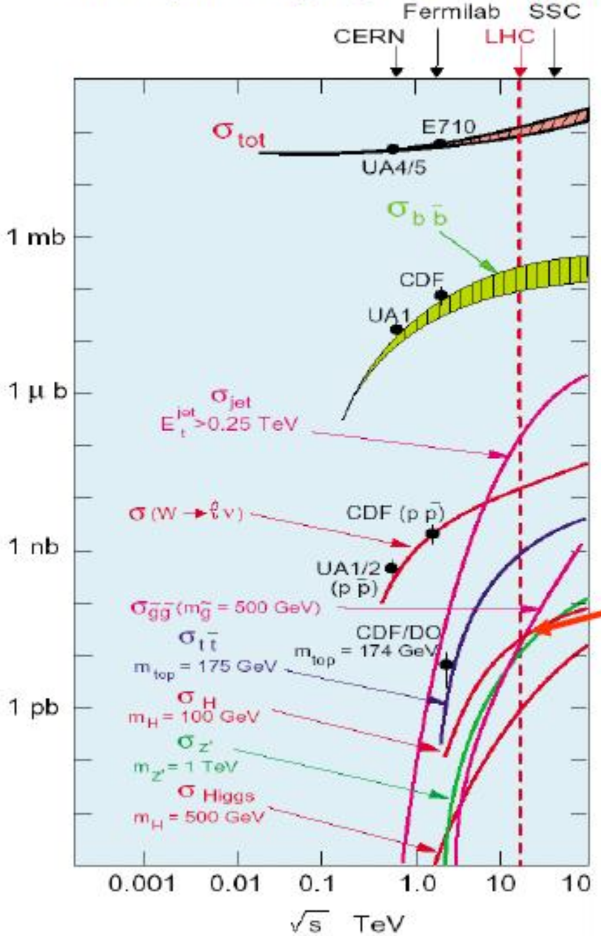
- **What is: Machine Learning (ML) & Multivariate Analysis/Technique (MVA)**
  - **Basics (classification, regression)**
  - **ROC-curve**
  - **generative vs predictive models**
- **MVA/ML algorithms**
  - **Naïve Basian, KNN,**
  - **Linear discriminators, SVM**
  - **model fitting – gradient decent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

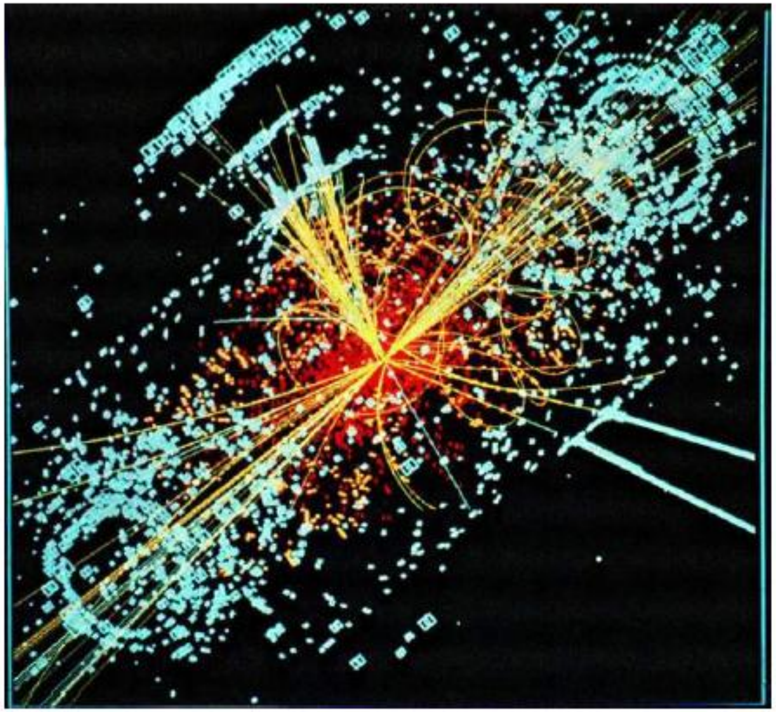
Extracted from slides by:
G. Cowan's lectures at RH London Univ., H. Voss at SOS 2016, K. Reygers lectures at Heilderbeg Univ.

Prof. dr hab. Elżbieta Richter-Wąs

HEP Experiments: Simulated Higgs event in CMS

- That's how a "typical" Higgs event looks like: (underlying ~23 'minimum bias' events)

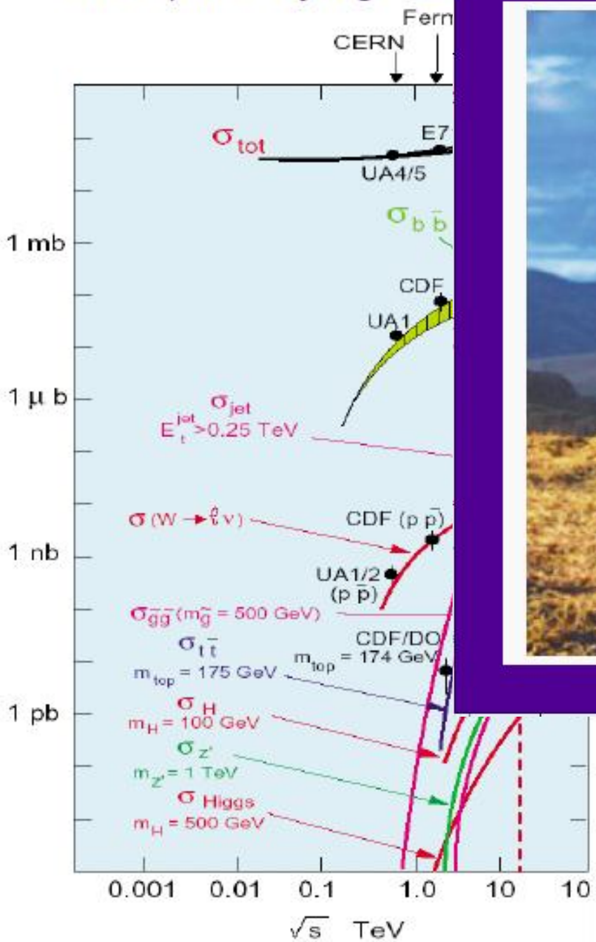


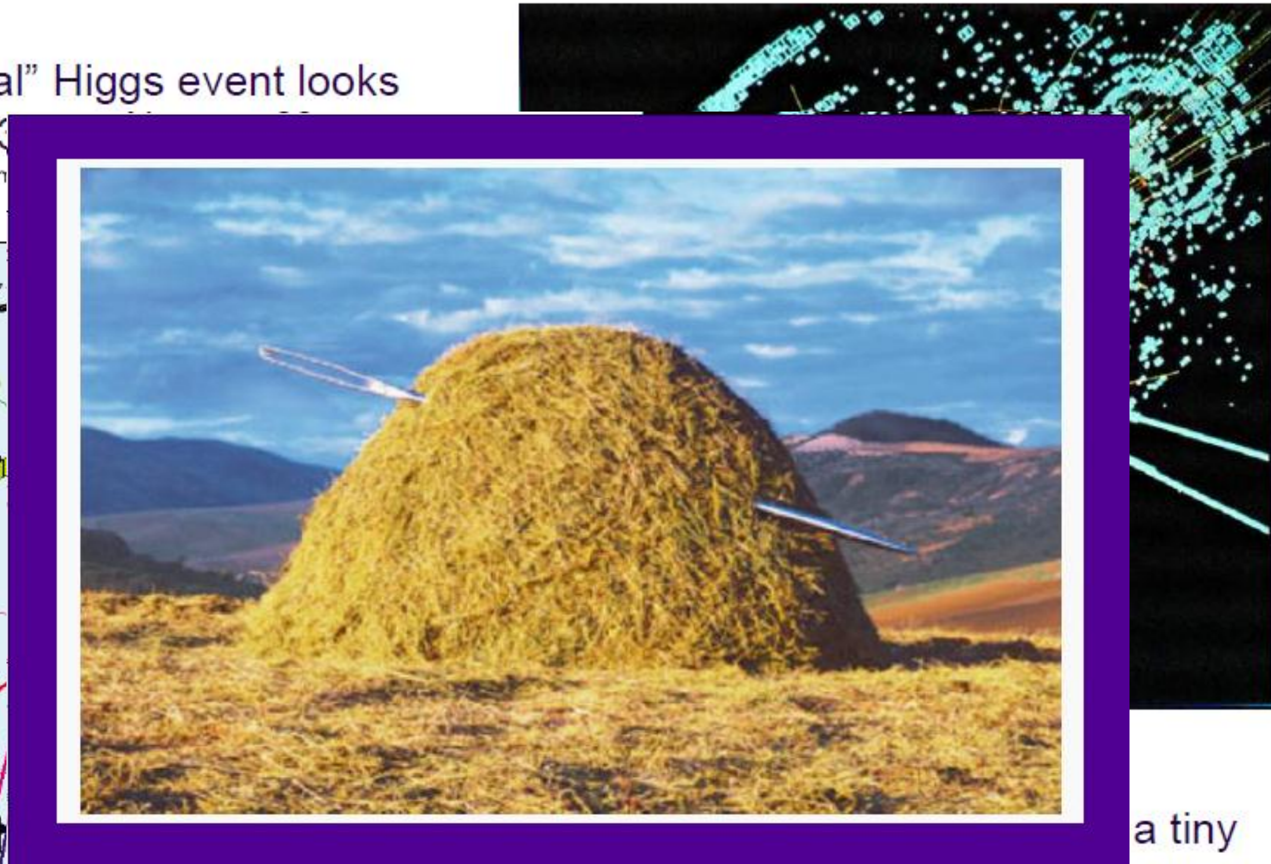


- And not only this: These event happen only in a tiny fraction of the collisions $O(10^{-11})$

HEP Experiments: Simulated Higgs event in CMS

- That's how a "typical" Higgs event looks like: (underlying ~23

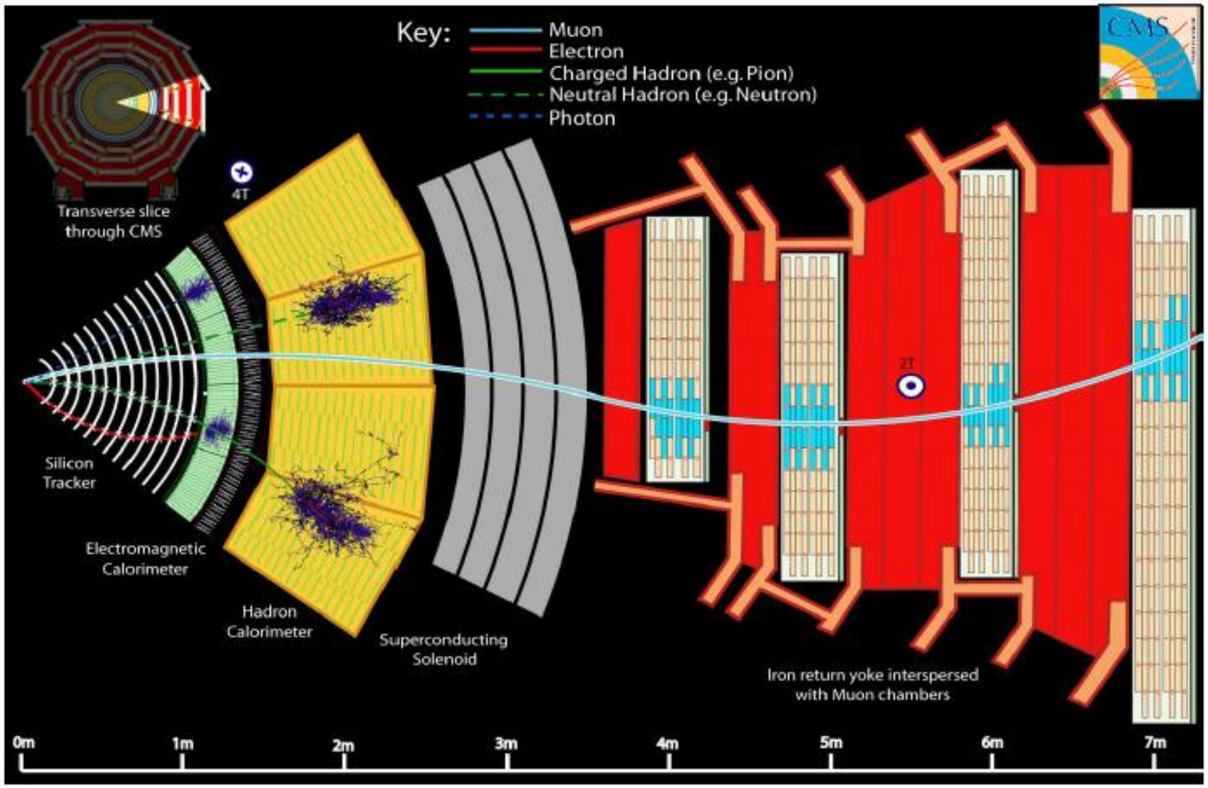




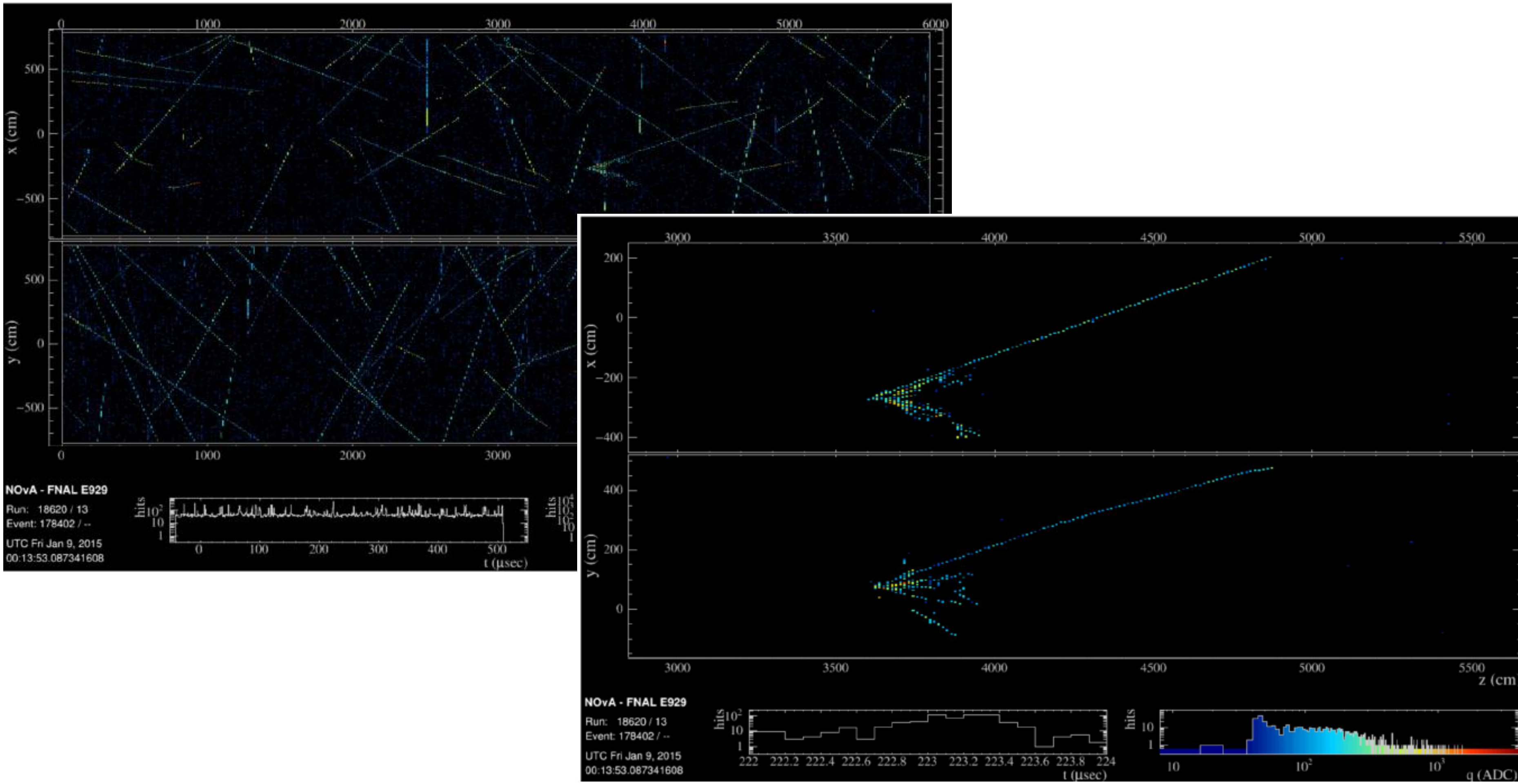
a tiny fraction of the collisions $O(10^{-11})$

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. (ν_μ)/ν_e (dis-)/appearance



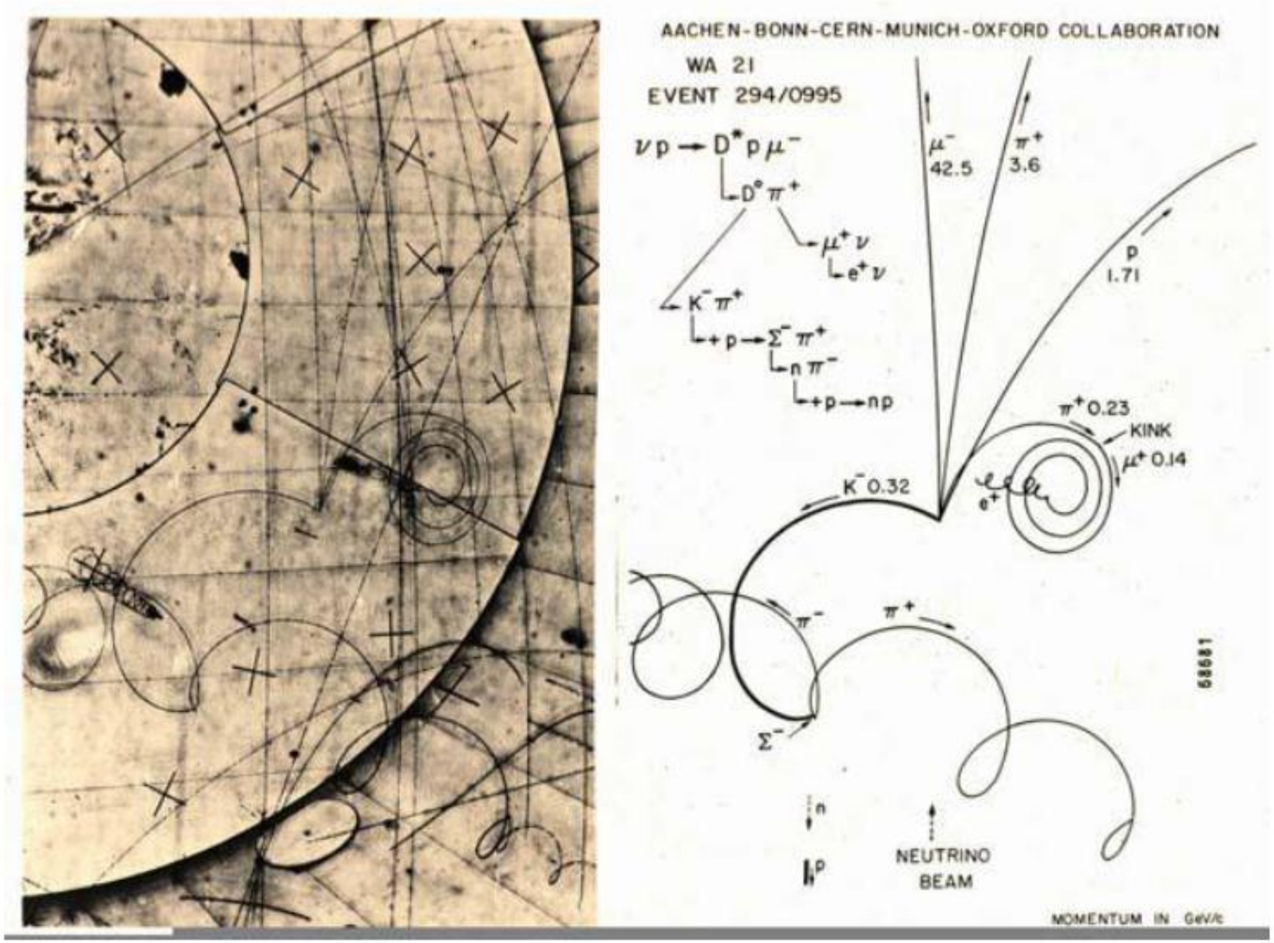
O(100k) background, O(100) ν_μ , O(10) ν_e per year

Machine Learning ‚elsewhere'

…. is 'everywhere'

HEP: Everything started Multivariate

- intelligent "Multivariate Pattern Recognition" used to identify particles



Outline

- 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

What is Machine Learning

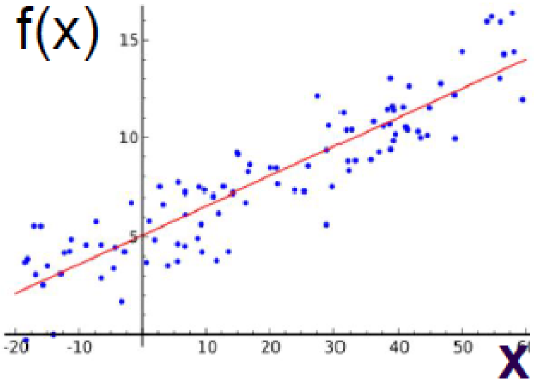
- *"[Machine Learning is the] field of study that gives computers the ability to learn without being explicitly programmed."* Arthur Samuel (1959)

- *"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)

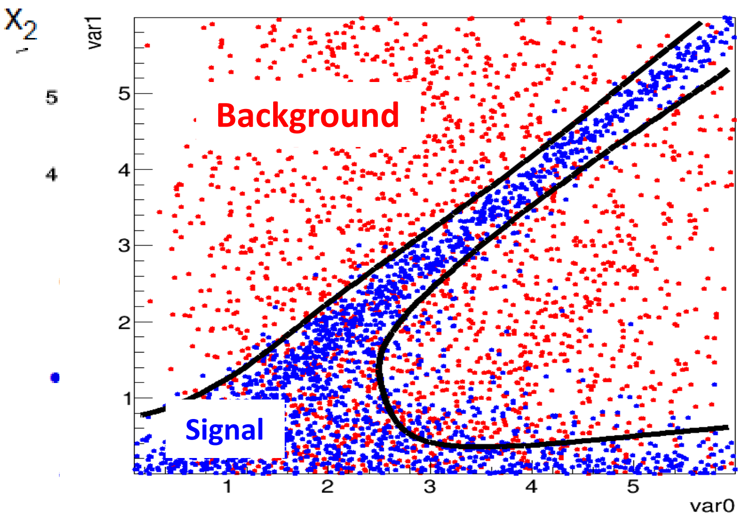
'understanding/modeling your data' …
and if you cannot do it in multi-dimensions on "analytic first principles" let the computer help ☺

What are Multivariate Techniques

→ Many things … starting from "linear regression" …



to multivariate event classification



→ or w/o prior 'analytic' model

→ typically "multivariate"

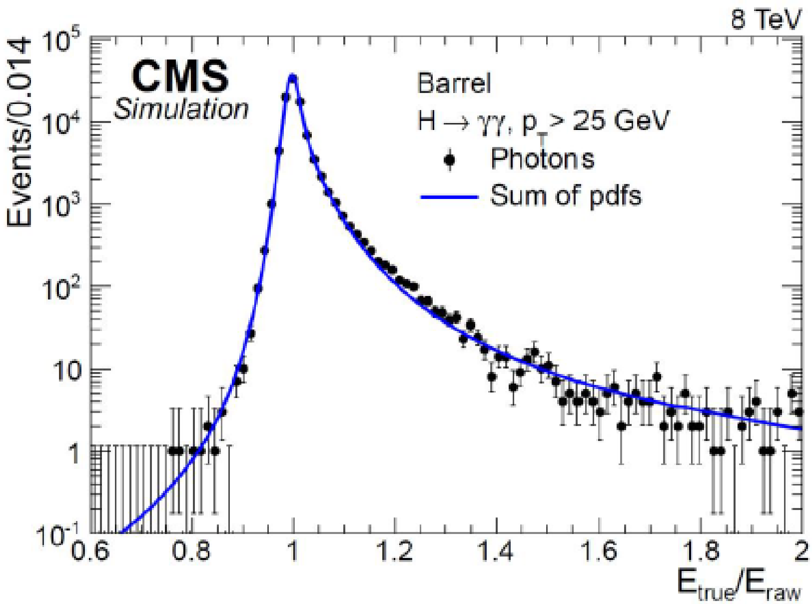
- Parameters depend on the 'joint distribution' $f(x_1, x_2)$

- '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)



<u>MC sample: γ +jets</u>

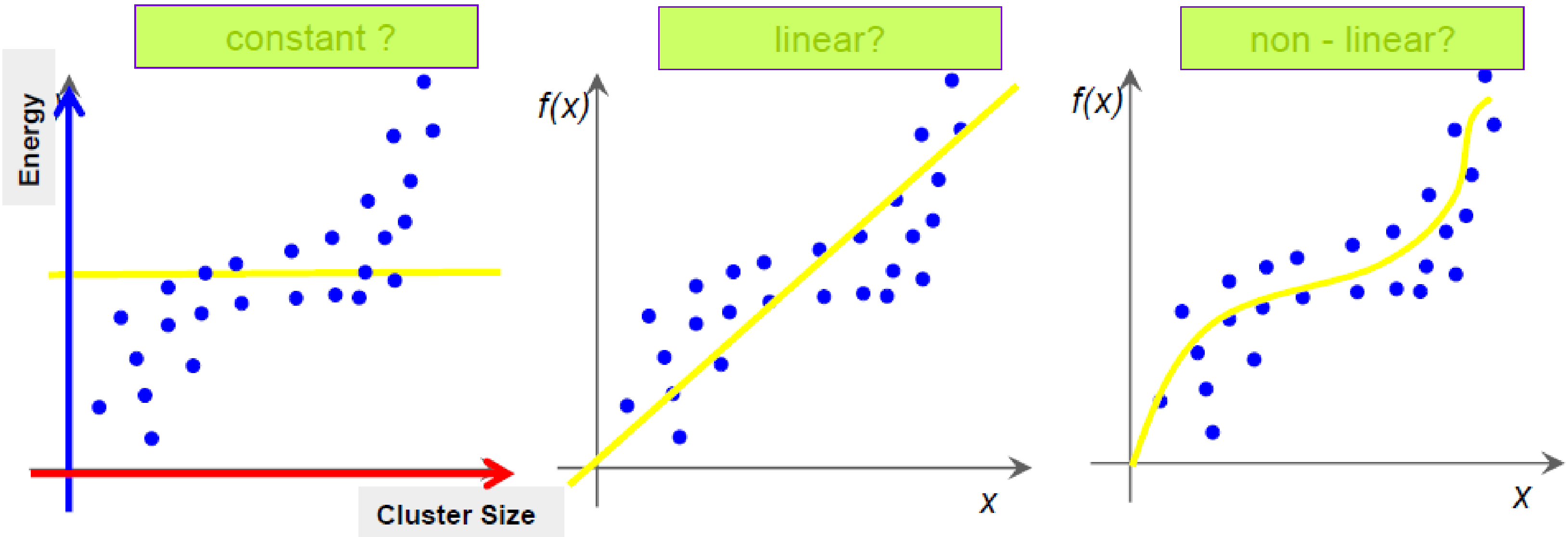
- Raw energy in crystals, η, Φ
- Cluster shape variables
- Local cluster position variables

  (energy leakage)
- Pile-up estimators

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

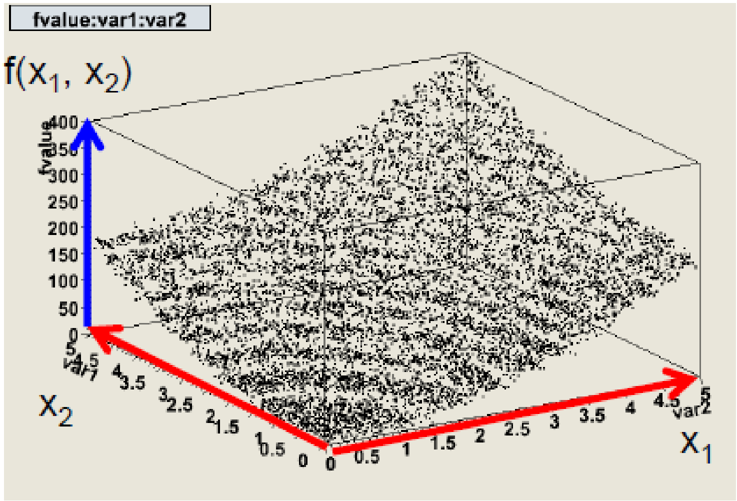
Regression

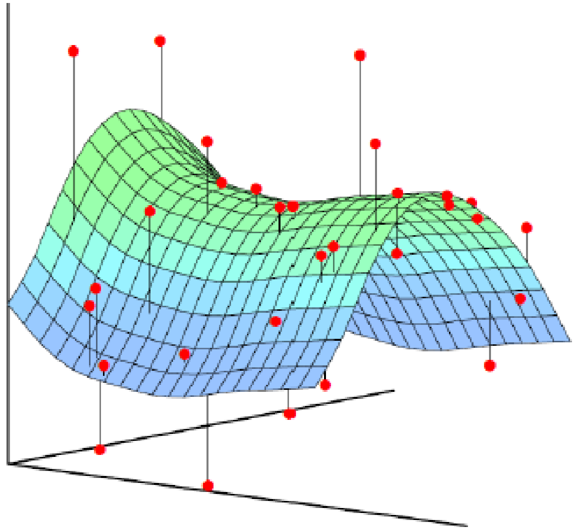
- 'known measurements" → model "functional behaviour"
- e.g. : photon energy as function "D"-variables: ECAL shower parameters + …



- 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

Multi-Variate Classification

Consider events which can be either signal or background events.

Each event is characterized by *n* observables:

$$\vec{x} = (x_1, \ldots, x_n) \qquad \text{"feature vector"}$$

Goal: classify events as signal or background in an optimal way.

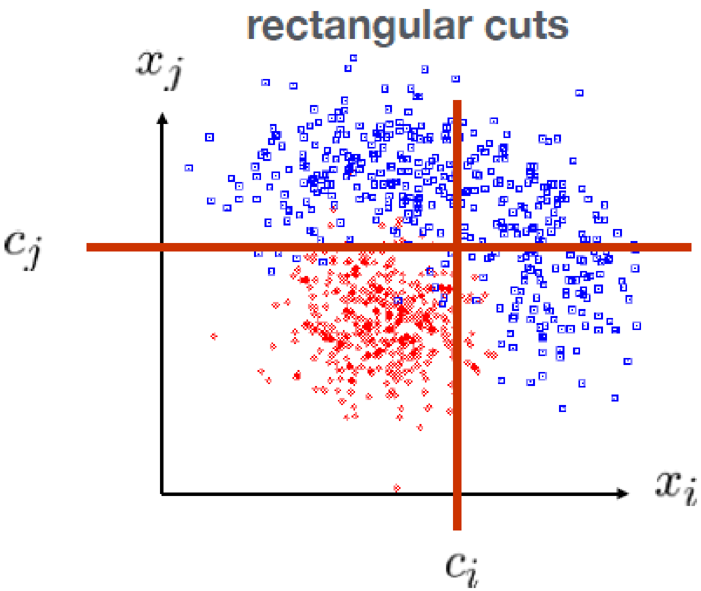
This is usually done by mapping the feature vector to a single variable, i.e., to scalar test statistic:

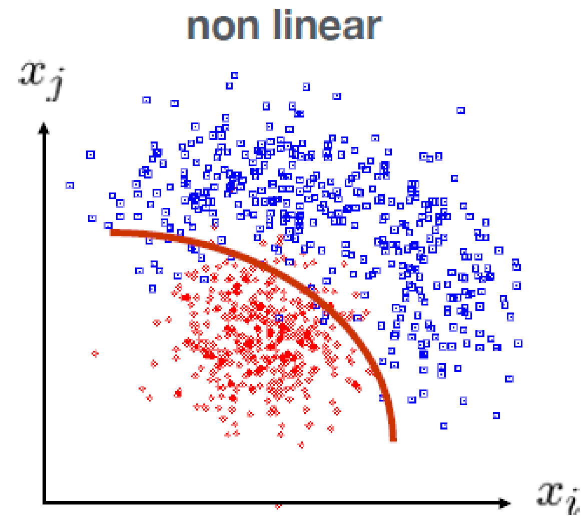
$$\mathbb{R}^n \to \mathbb{R} : \quad y(\vec{x})$$

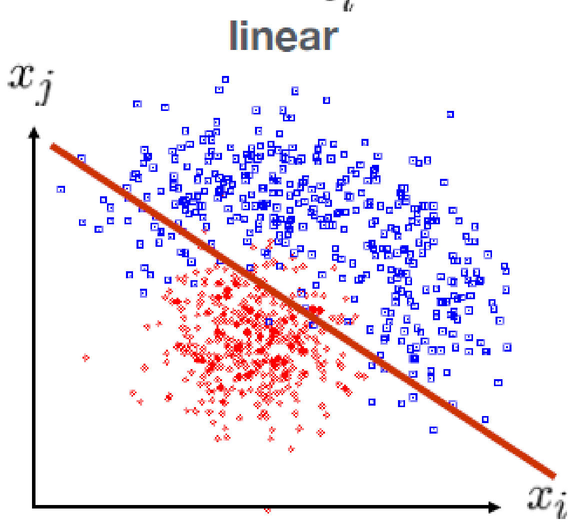
A cut $y > c$ to classify events as signal corresponds to selecting a potentially complicated hyper-surface in feature space. In general superior to classical "rectangular" cuts on the x_i.

Problem closely related to *machine learning* (*pattern recognition*, *data mining*, …)

Classification: Different Approaches



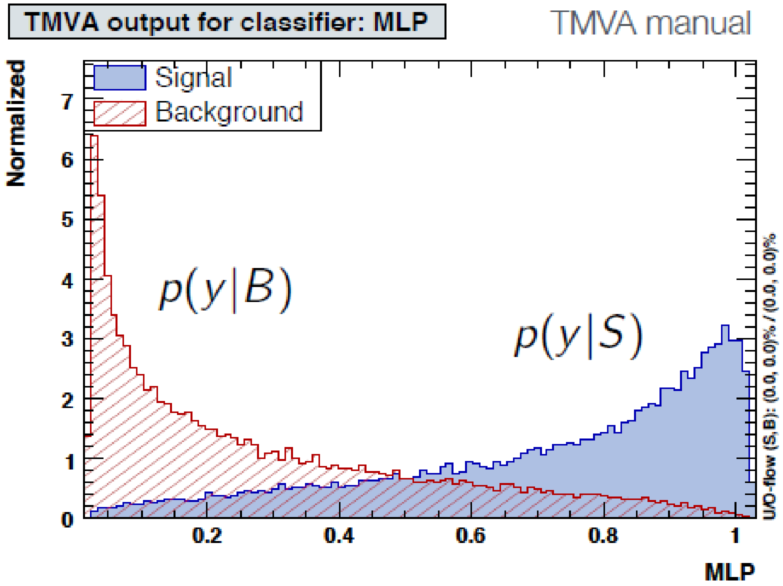




k-Nearest-Neighbor,
Boosted Decision Trees,
Multi-Layer Perceptrons,
Support Vector Machines
…

Signal Probability Instead of Hard Decisions

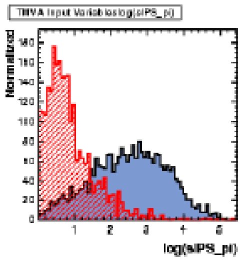
Example: test statistic y for signal and background from a Multi-Layer Perceptron (MLP):

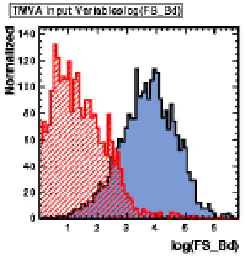


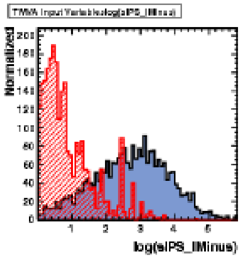
Instead of a hard yes/no decision one can also define the probability of an event to be a signal event:

$$P_s(y) \equiv P(S|y) = \frac{p(y|S) \cdot f_s}{p(y|S) \cdot f_s + p(y|B) \cdot (1 - f_s)}, \qquad f_s = \frac{n_s}{n_s + n_b}$$

Event Classification







$\mathbb{P}^D$

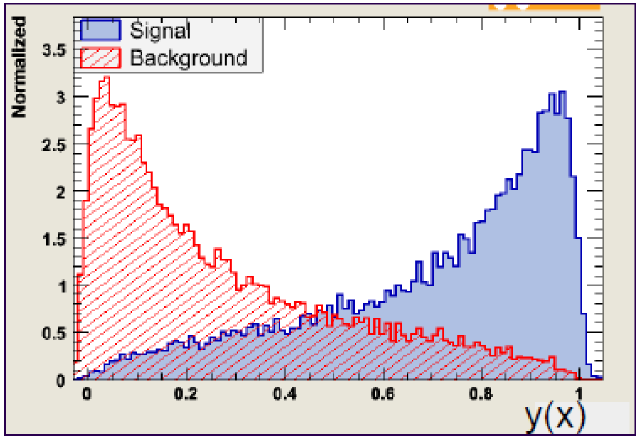
"feature space"

- 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 label

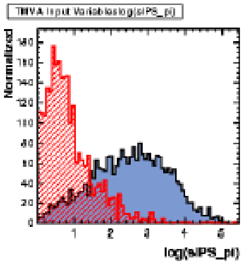
Test statistic:
$y(x): R^D \rightarrow R$: → $\mathbb{P}$

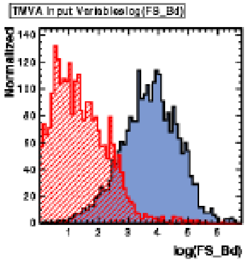
most general form
$y = y(\mathbf{x}); \ \mathbf{x} \in \mathbb{P}^D$
$\mathbf{x}=\{x_1,\ldots,x_D\}$: input variables

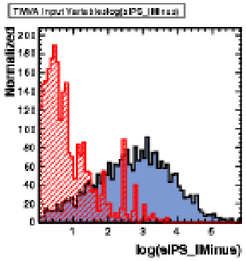
- plotting (histogramming) the resulting y(x) values:



Event Classification







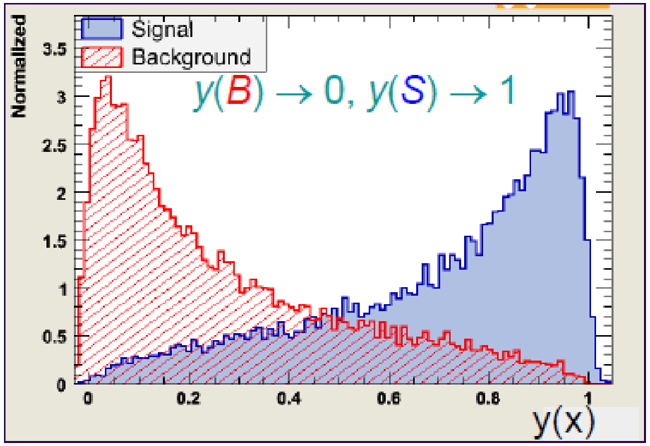
P^D

"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

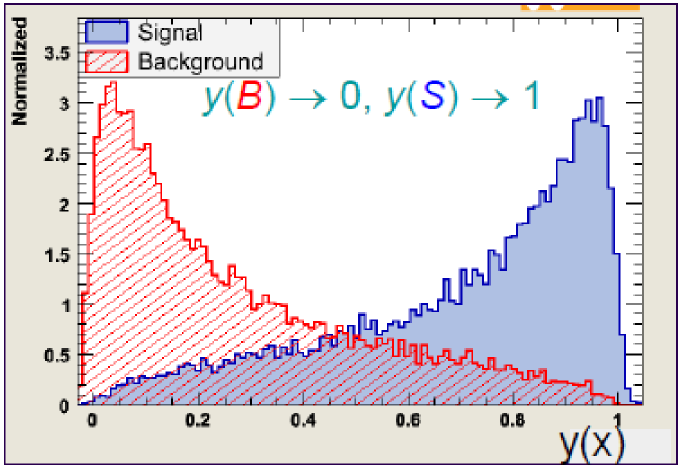
- overlap of $PDF_S(y)$ and $PDF_B(y)$ → separation power , purity
- y(x)=const: surface defining the decision boundary.

Classification <-> Regression

Classification:

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

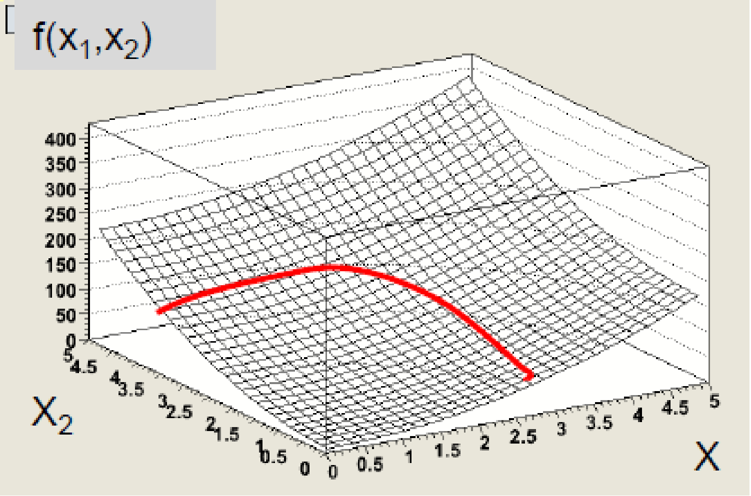
y(x): $R^D \to R$:



Regression:

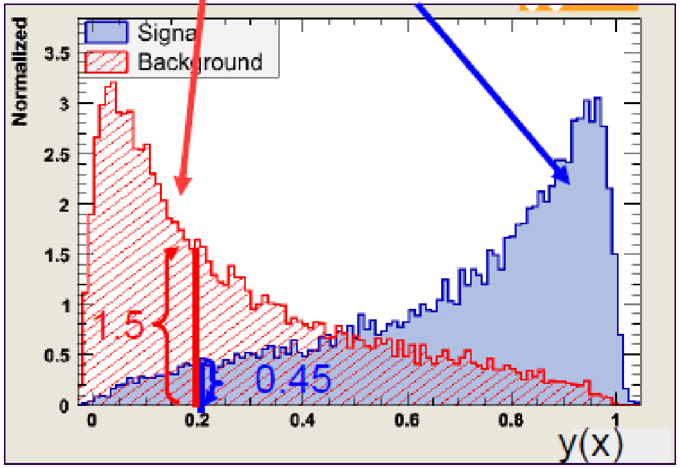
- "D" measured variables + one function value (*e.g.* cluster shape variables in the ECAL + particles energy)
- y(x): $R^D \to R$ "regression function"
- y(x)=const → hyperplanes where the target function is constant

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 $PDF_S(y(x)) = 0.45$

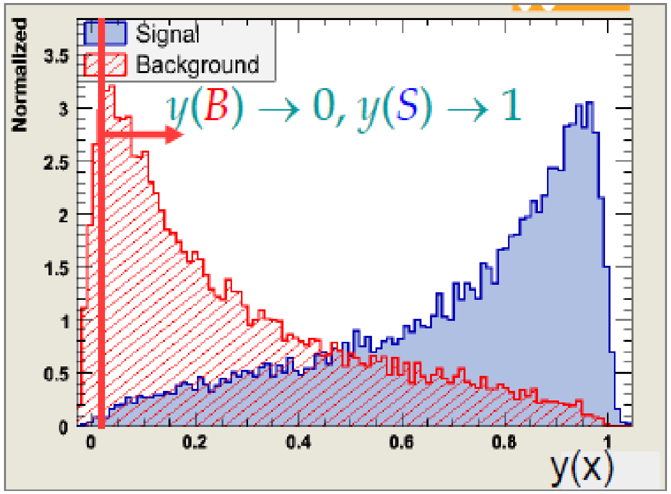
f_S, f_B : fraction of **S** and **B** in the sample:

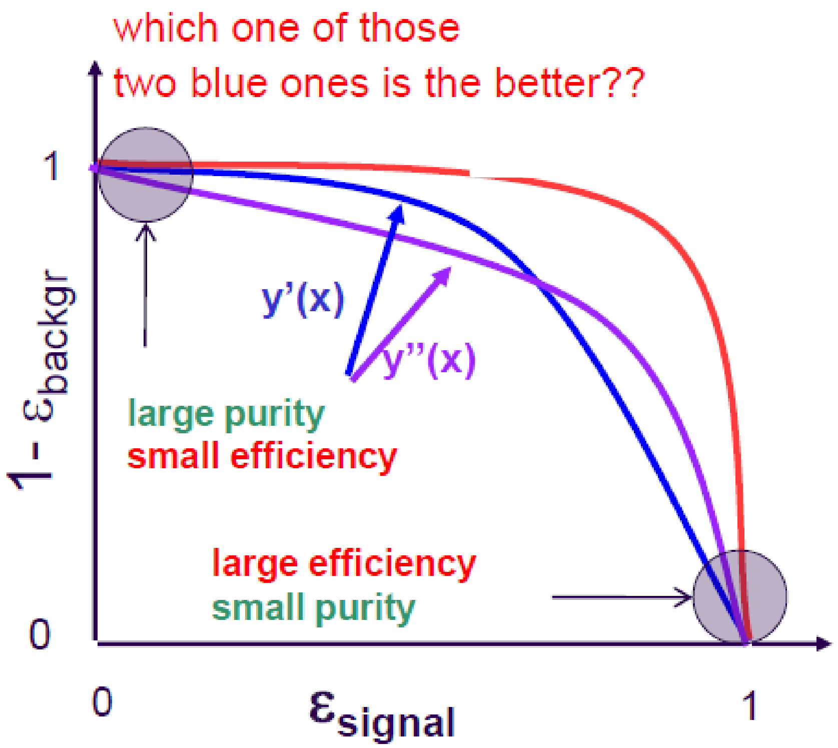
$$\frac{f_S PDF_S(y)}{f_S PDF_S(y) + f_B PDF_B(y)} = P(C = S \mid y)$$

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

Receiver Operation Characteristic (ROC) curve

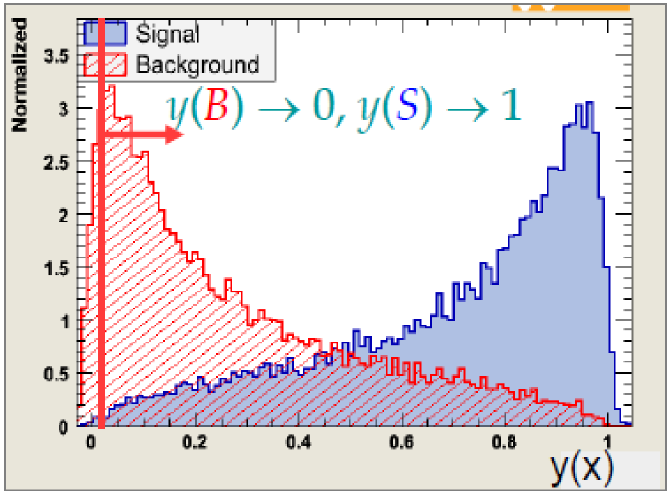
Signal(H_1) /Background(H_0) discrimination:

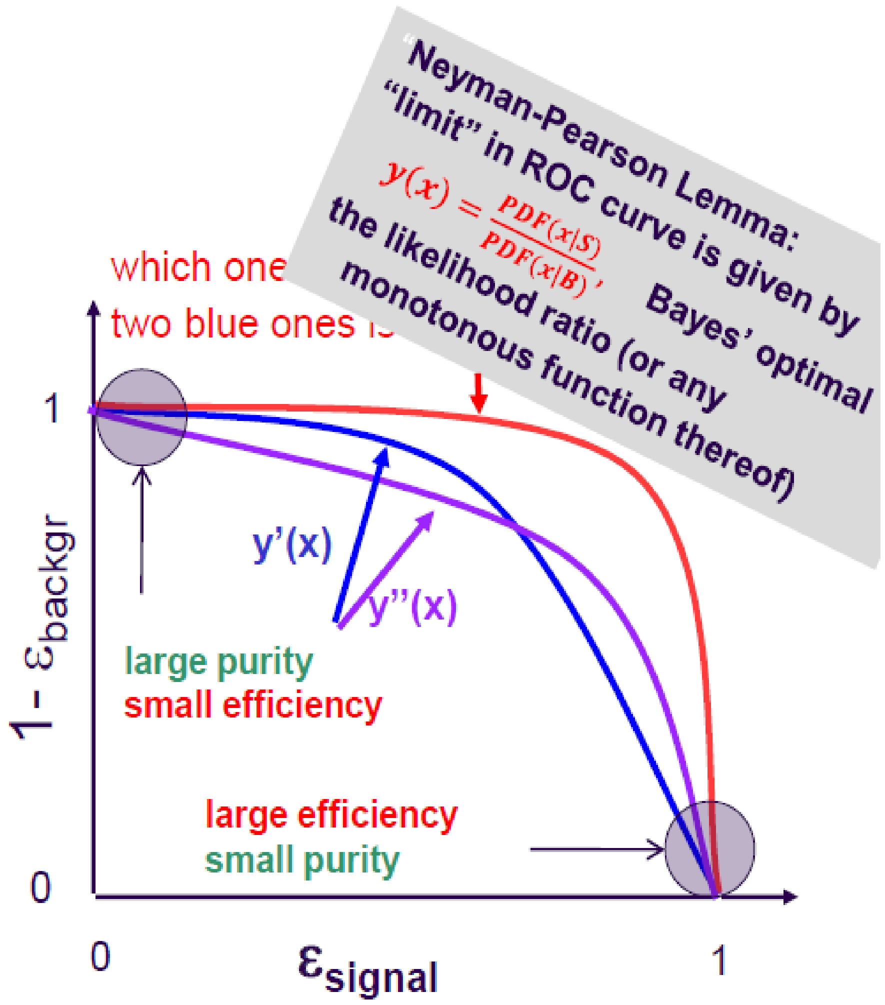




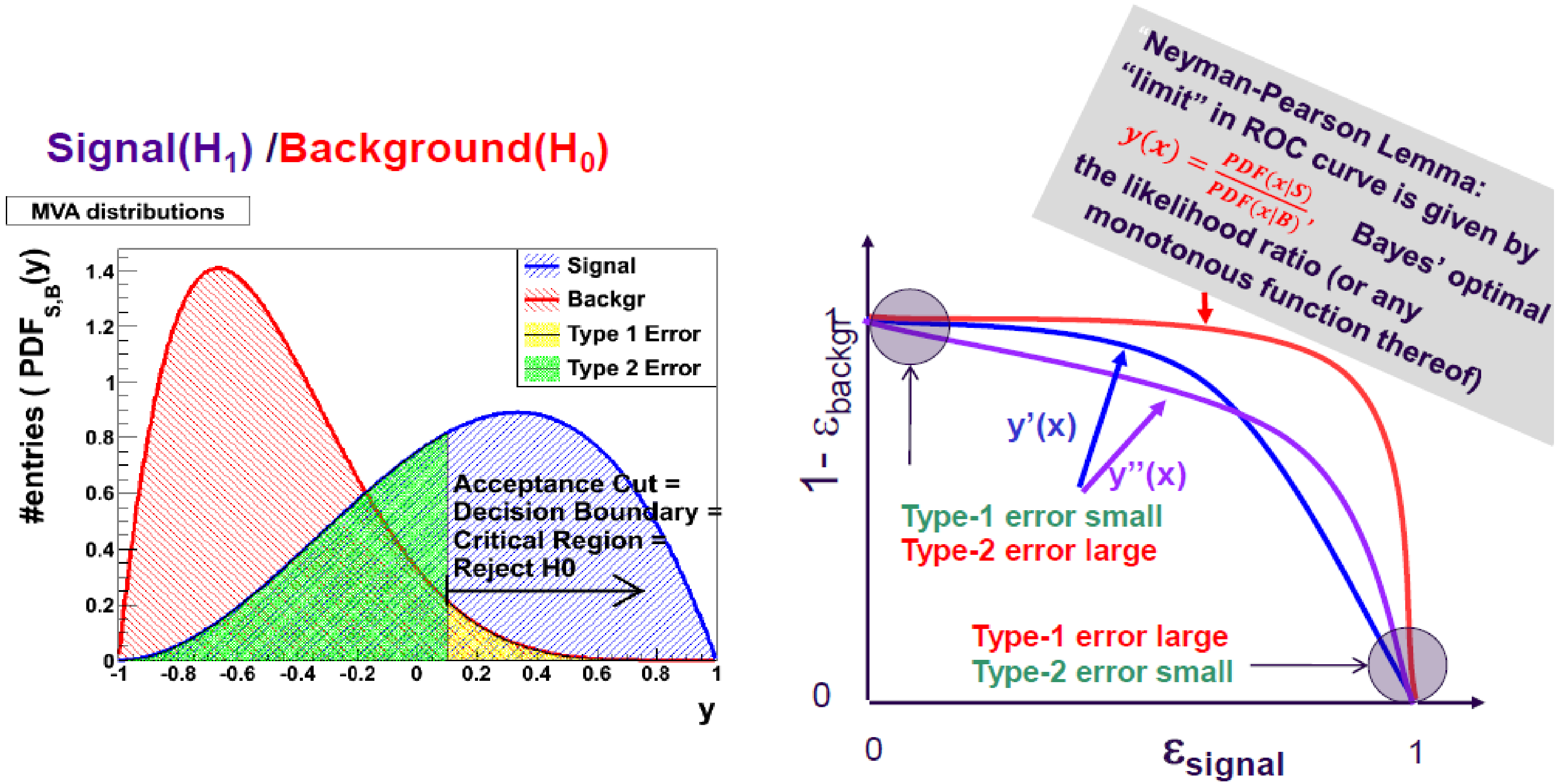
Receiver Operation Characteristic (ROC) curve

Signal(H_1) /Background(H_0) discrimination:





Receiver Operation Characteristic (ROC) curve



- 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)

  <u>OR</u>

  - 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

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”

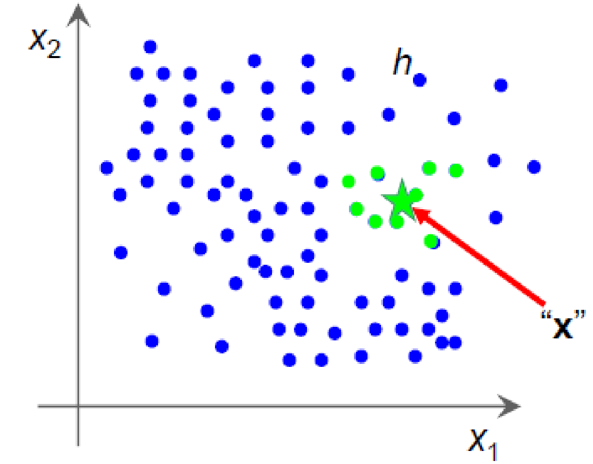
(i.e. a robot achieves his goal or does not / falls or does not fall/ wins or looses the game)

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:

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

k(u): is called a Kernel function:

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

- Classification: Determine

$PDF_S(x)$ and $PDF_B(x)$

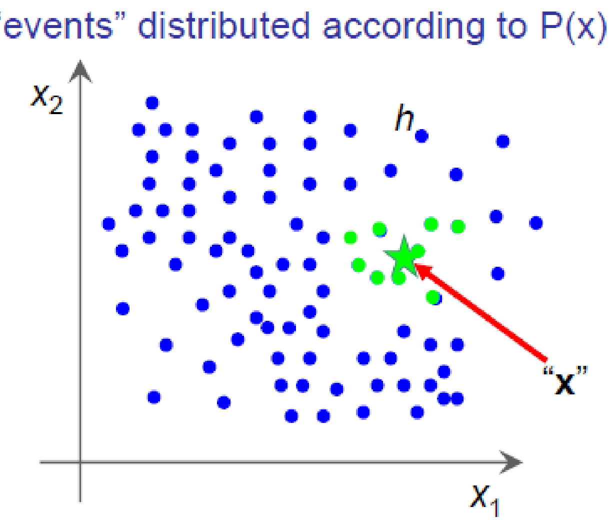
→ likelihood ratio as classifier!

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

→ 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



→ 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) →

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

i.e.: the average function value

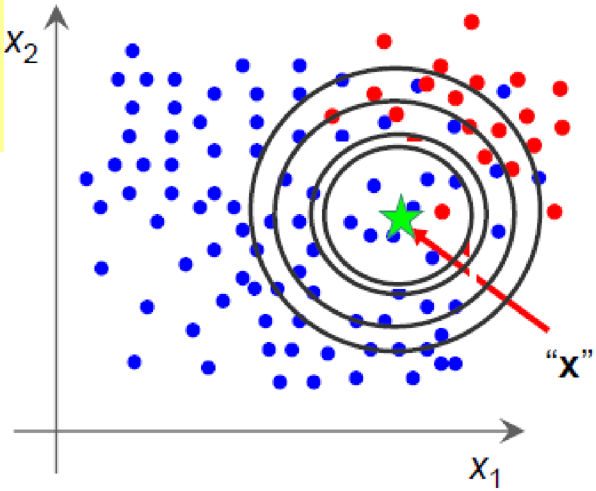
K- Nearest Neighbour

→ kNN : k-Nearest Neighbours

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

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

"events" distributed according to P(x)



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

$$P(\mathbf{x}) = \frac{1}{N}\sum_{n=1}^{N} K_h(\mathbf{x} - \mathbf{x}_n)$$

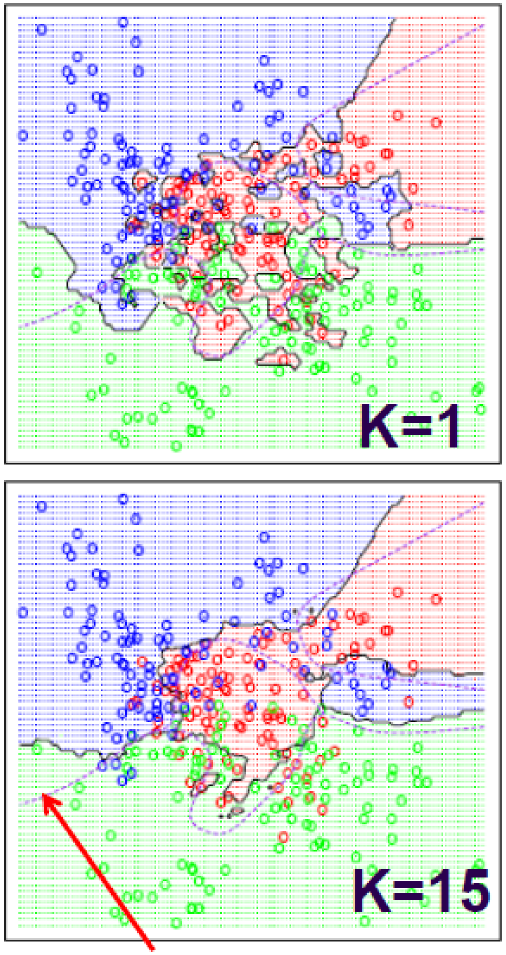
: a general probability density estimator using kernel K

- 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)

Bayes’ optimal decision boundary

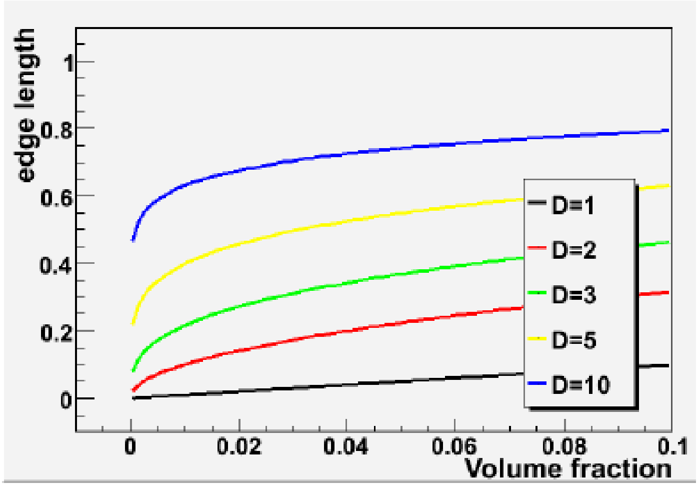
„Curse of Dimensionality"

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

We all know:

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

Shortcoming of nearest-neighbour strategies:



- 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

Naive Bayesian Classifier (Projective Likelihood Classifier)

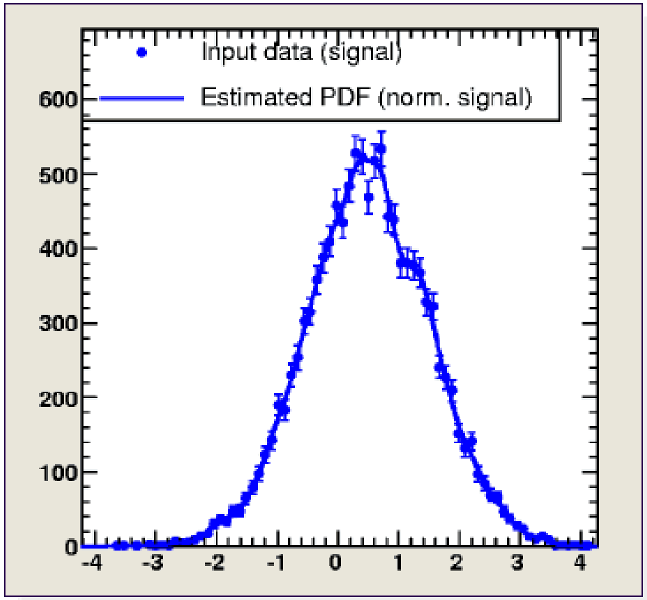
Multivariate Likelihood (k-Nearest Neighbour)
→ estimate the full D-dimensional joint probability density

Naïve Bayesian
→ ignore correlations

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

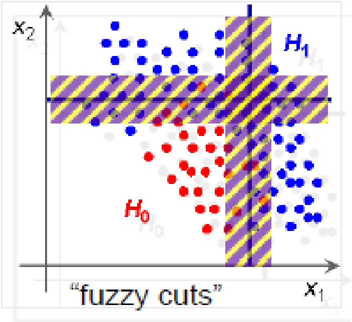
product of marginal PDFs (1-dim "histograms")

pdf: histogram + smoothing



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

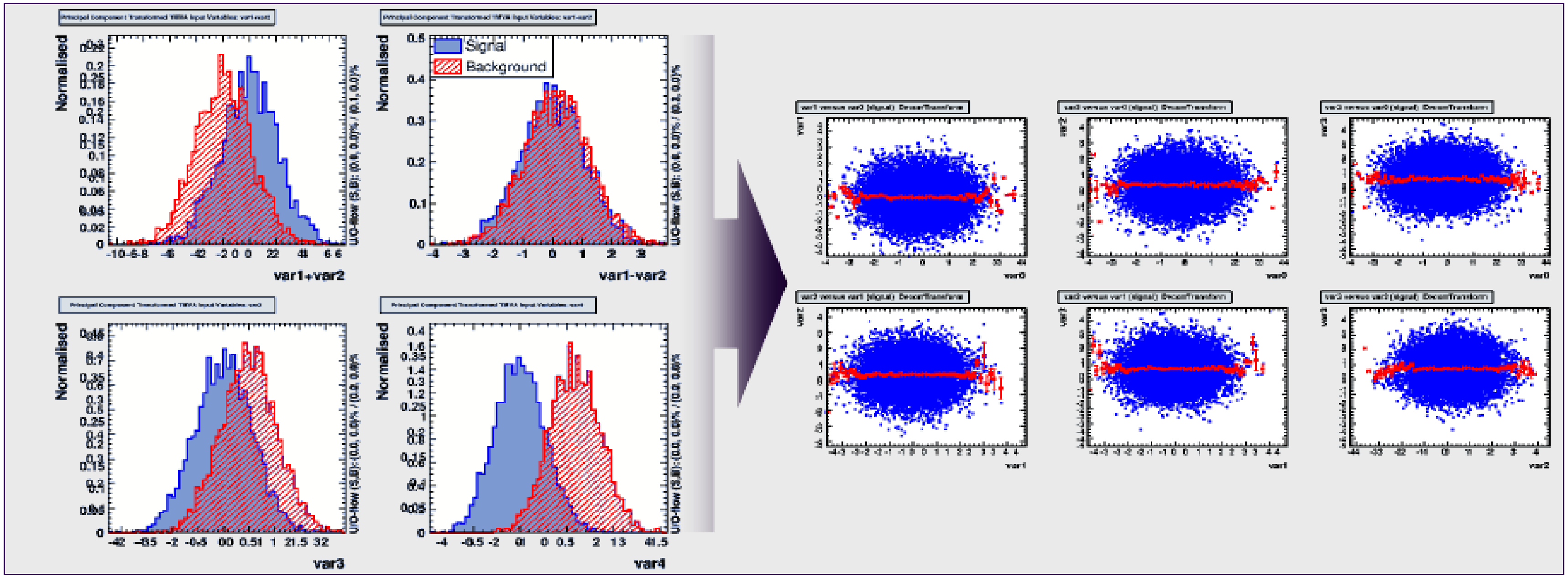
(a very signal like variable may counterweigh another, less signal like variable)



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

De-Correlation

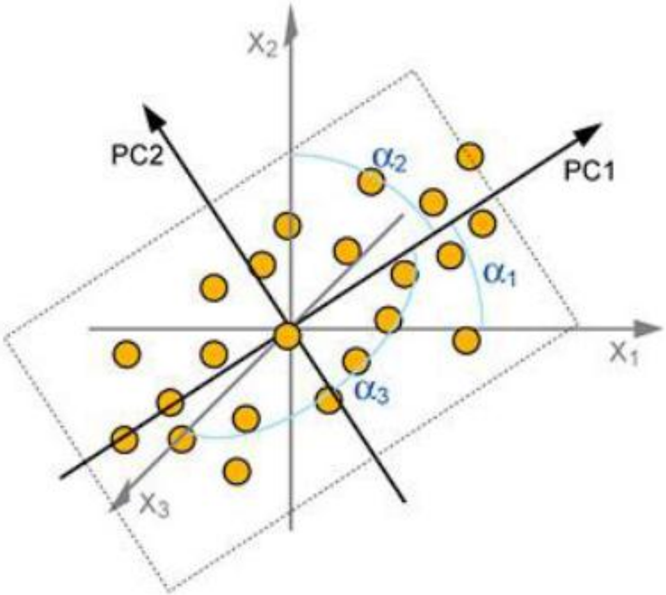
- **Find variable transformation that diagonalises the covariance matrix**
  - Determine *square-root C ′* of correlation matrix C, *i.e.*, C = C ′C ′
    - compute C ′ by diagonalising C: $D = S^T C S \Rightarrow C' = S\sqrt{D}S^T$
    - transformation from original (x) in de-correlated variable space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



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!

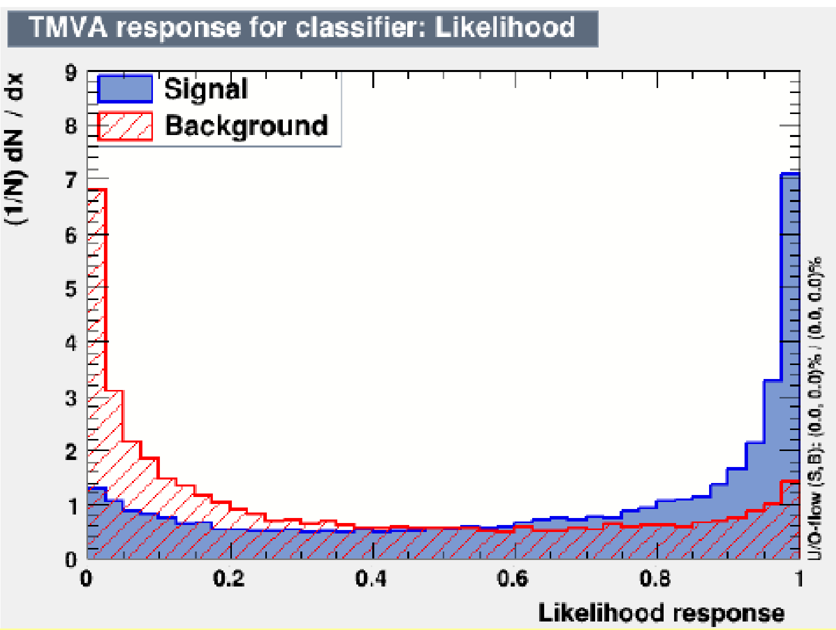
Decorrelation at Work

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

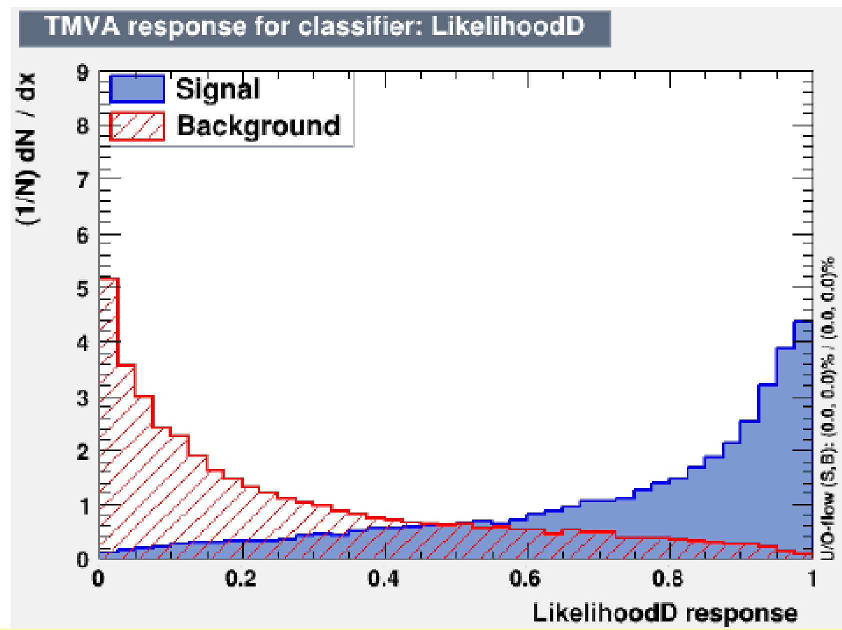
→ 1-D Likelihood on de-correlated sample give best possible performance

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

correlated variables:



after decorrelation

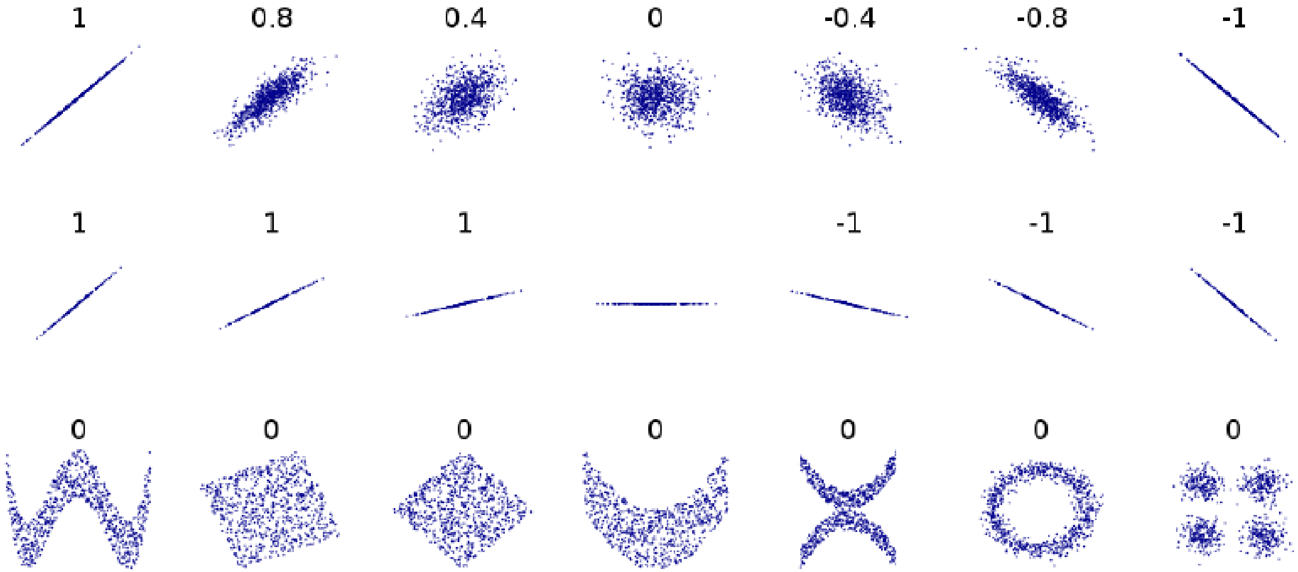


Watch out! Things might look very different for non-linear correlations!

Correlation Coefficients

'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 likelihood ratio (or a monotonic function thereof)

→ use as 'event weights'

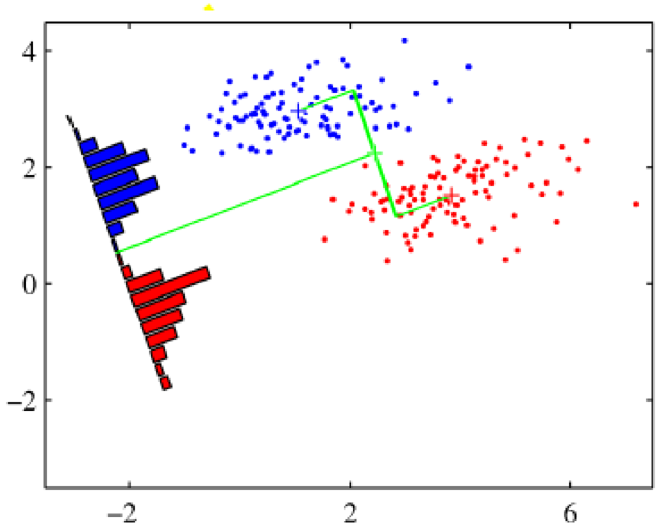
arXiv:1506.02169 for a 'more theoretical' analysis

Linear Discriminant

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

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

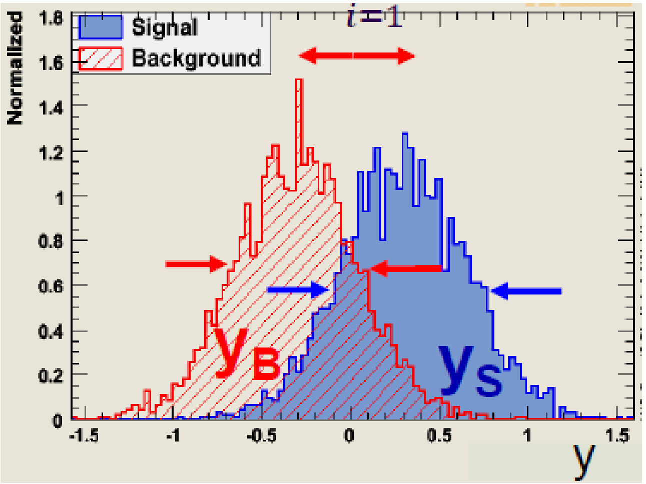
i.e. any linear function of the input variables: → linear decision boundaries



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

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

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

note: these quantities can be calculated from the training data

Linear Discriminant and non linear correlations

assume the following non-linear correlated data:

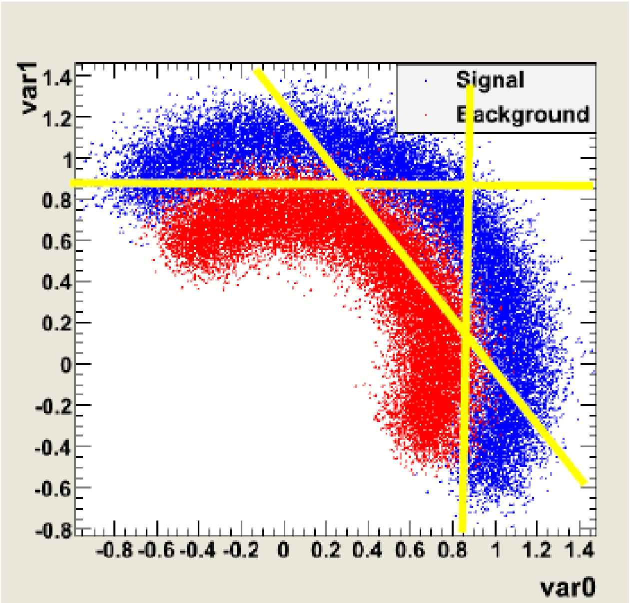
- the Linear discriminant obviously doesn't do a very good job here:

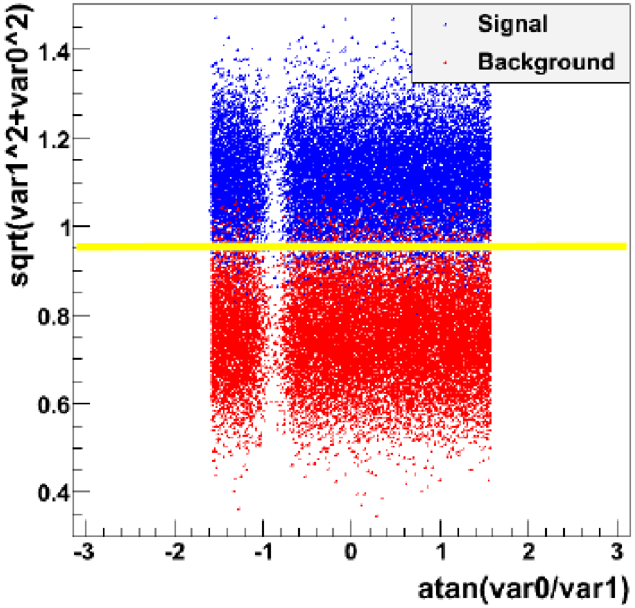
- Of course, these can easily be de-correlated:

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

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

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





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:

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

- Classification:

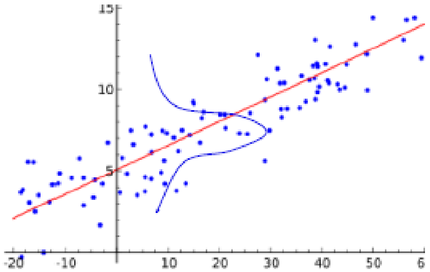
$$\rightarrow E[L] = E[y_i^{train} \log(y(x_i)) + (1 - y_i^{train}) \log(1 - y(x_i))]$$ binomial loss

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

<u>classification:</u> 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



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

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

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

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

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

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

Logistic Regression *

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

Fisher Discriminant:

→ equivalent to Linear Discriminant with 'squared loss function'

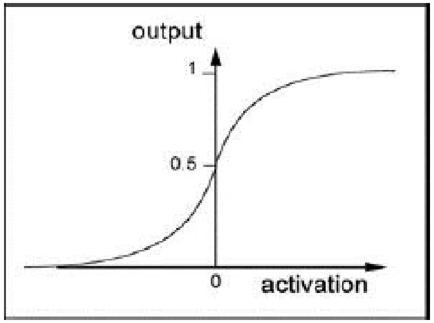
→ build a linear classifier that maximizes 'binomial loss':

- → 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

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

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

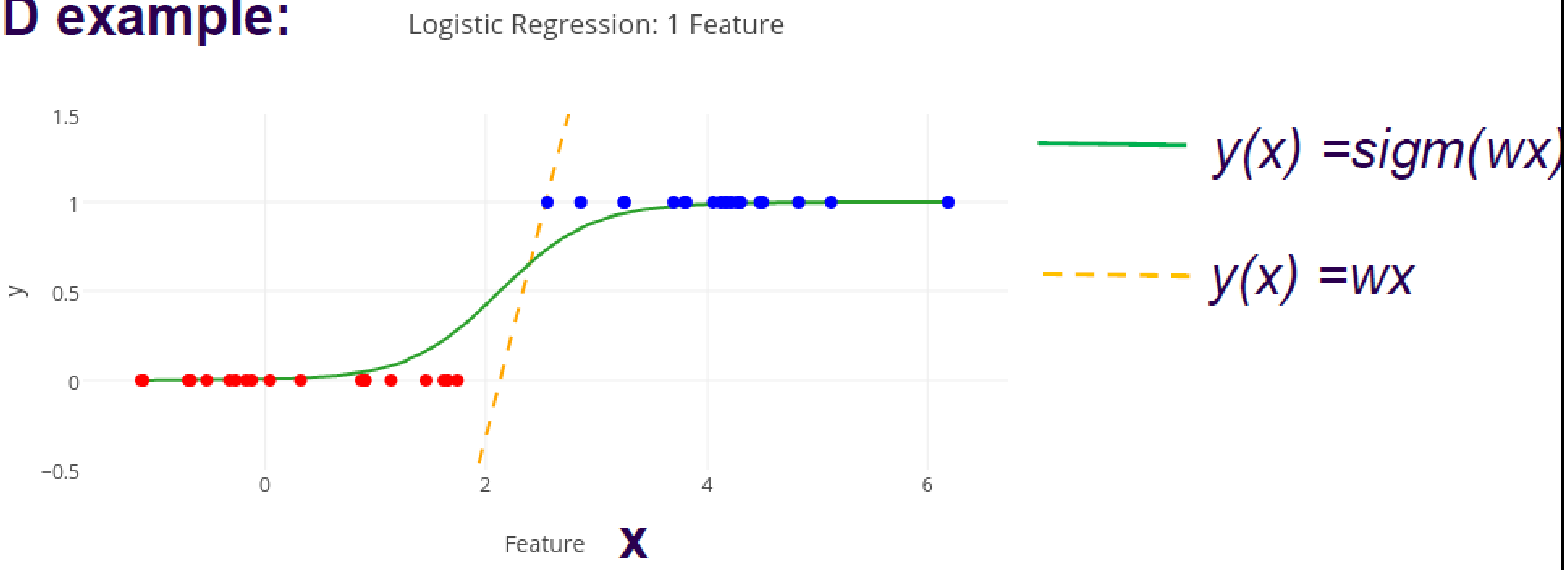


Note: Now y(x) has a 'probability' interpretation. y(x) of the Fisher discriminant was 'just' a discriminator.

Logistic Regression

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

1D example:

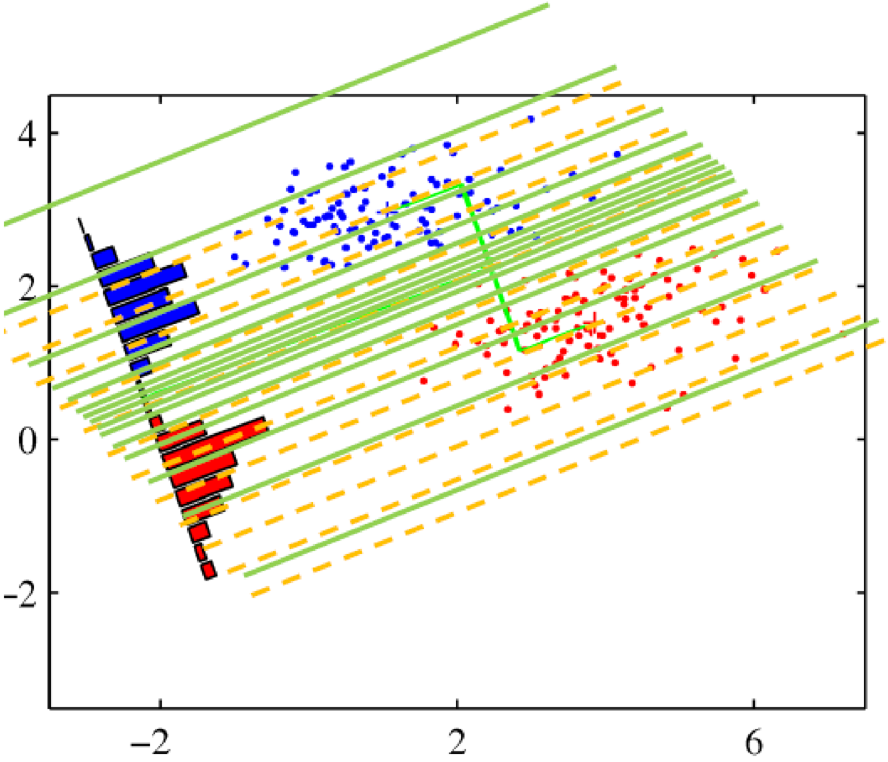


Note: decision boundaries are still 'linear', just the 'contour lines' (y(x)=const) are non-linear, parametrizing the probability of the event being y=0 or y=1 as 'distance' from the boundary….

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

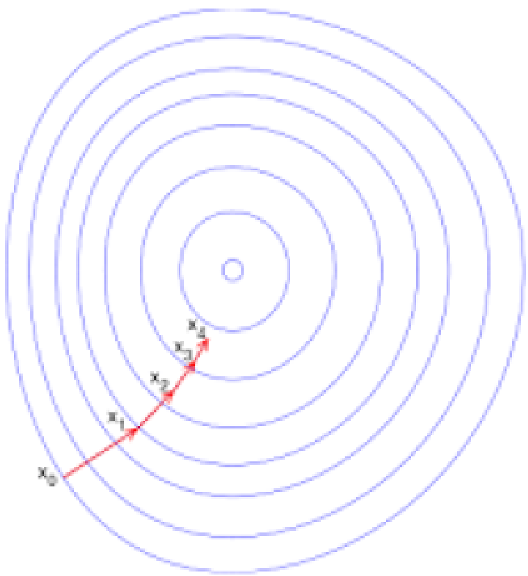
(Stochastic) Gradient Decent SDG

minimize the "loss function" → "*W*" ?

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

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

learning rate



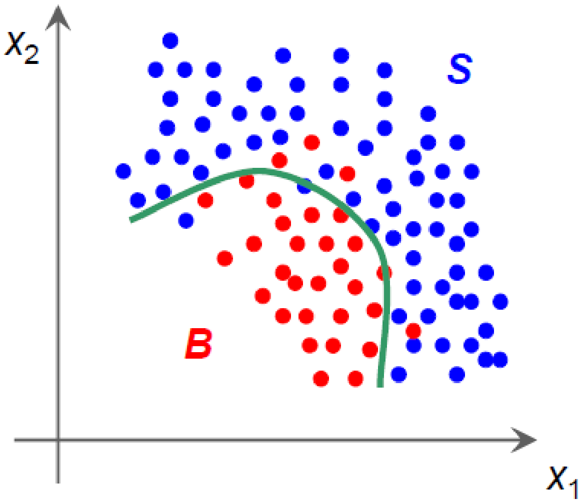
$$W \rightarrow W - \eta \frac{\partial E(L)}{\partial w}$$: gradient decent

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

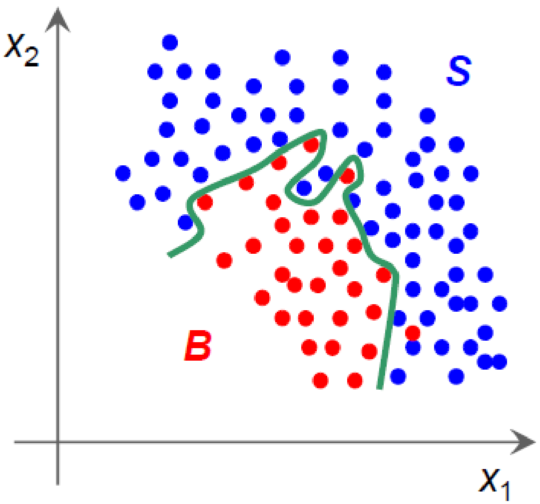
$$W \rightarrow W - \eta \frac{\partial L}{\partial w}$$: stochastic gradient decent

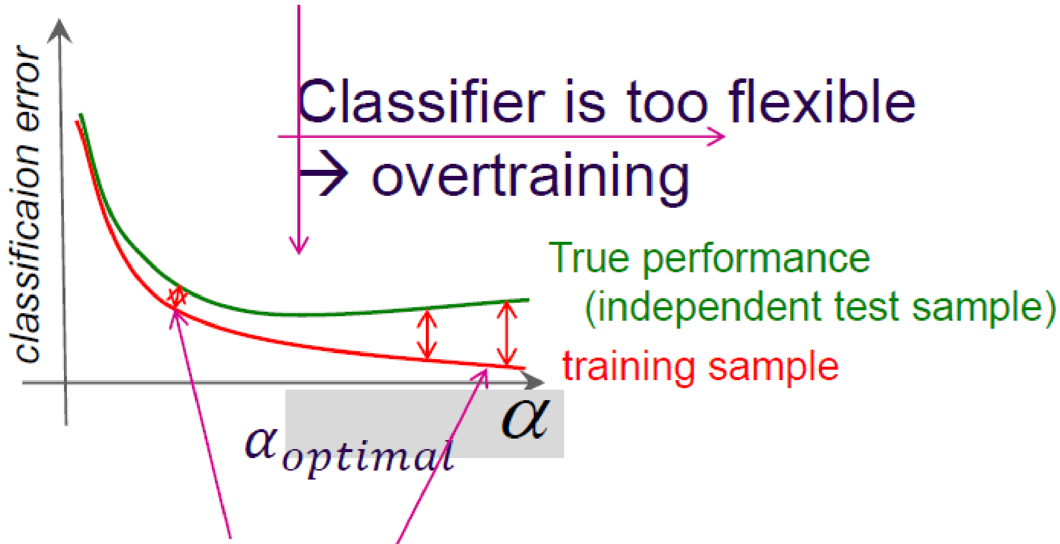
mostly: something in between → mini-batches

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

Regulatisation

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

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

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

$$= \sum_i^{events} \log(P(y_i^{train}|y(x_i)) + \log\left(p(w)\right)$$

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"

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? ☹

<u>Cross Validation:</u> 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

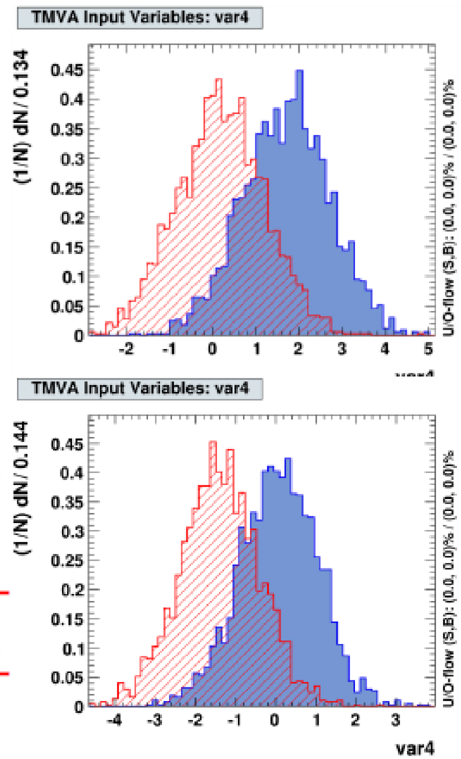
MVA Categories

- **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!

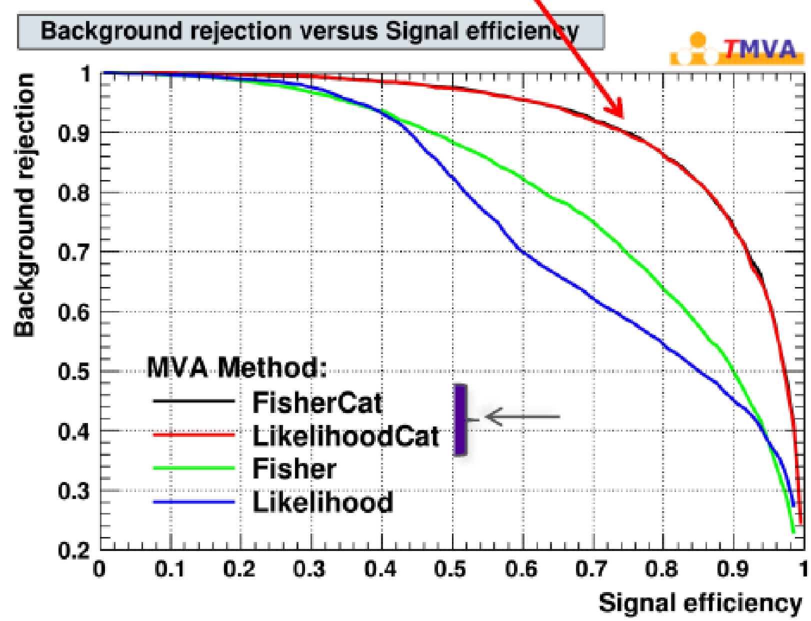
Example: var4 depends on some variable "eta"

|eta| > 1.3

|eta| < 1.3



Recover optimal performance after splitting into categories



About Systematic Errors

- 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 !!

Classifiers and Their Properties

H. Voss, Multivariate Data Analysis and Machine Learning in High Energy Physics
http://tmva.sourceforge.net/talks.shtml

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

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!