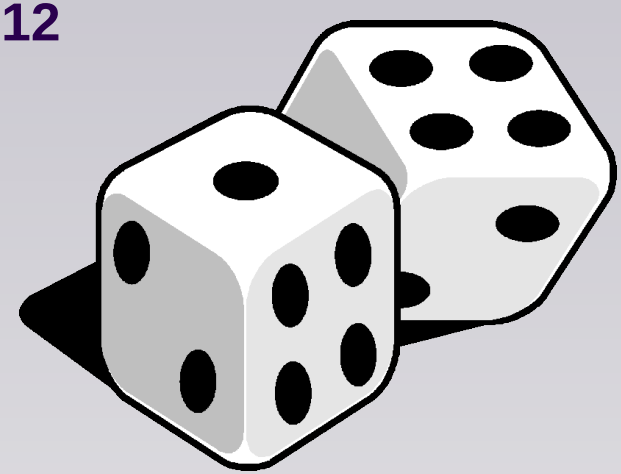
Introduction to Statistics

CERN Summer Student Lecture Program 2012



Helge Voss

… and Machine Learning

(in this last lecture)

Outline

- Why Statistics
- What is Probability :
  - frequentist / Bayesian interpretation
  - Hypothesis testing
    - error types and Neyman-Pearson Lemma, confidence level α and p-value
    - new particle searches – example: Higgs
- Lecture 3
  - Parameter estimation
    - Maximum Likelihood fit
    - χ^2-fit
  - Neyman Confidence belts → Feldman/Cousins …
  - (Monte Carlo Methods (Random numbers/Integration) → see slides)
- **Lecture 4**
  - **Machine Learning / Pattern Recognition**

Outline

- What are Multivariate classification/regression algorithms (MVA)
- Multidimensional Likelihood (kNN : k-Nearest Neighbour)
- Projective Likelihood (naïve Bayes)
- Linear Classifier
- Non linear Classifiers
  - Neural Networks
  - (Support Vector Machines → too bad, no time..)
  - Boosted Decision Trees

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

Literature:

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

Software packages for Mulitvariate Data Analysis/Classification

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

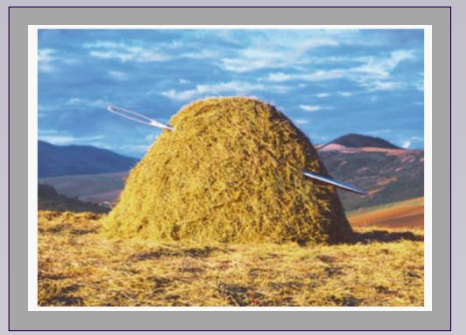
  and many other packages

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

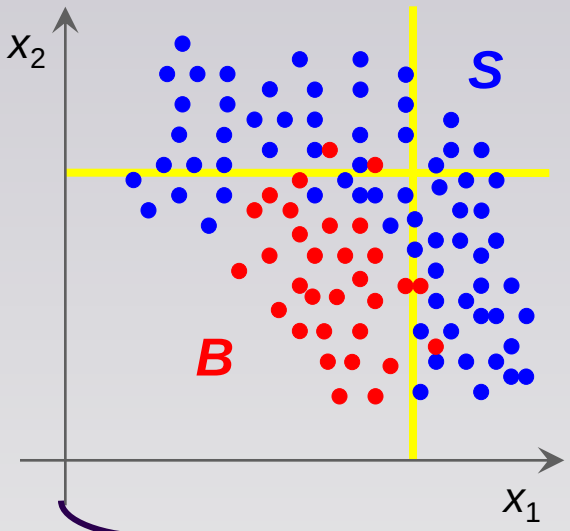
Conferences: PHYSTAT, ACAT,…

Event Classification

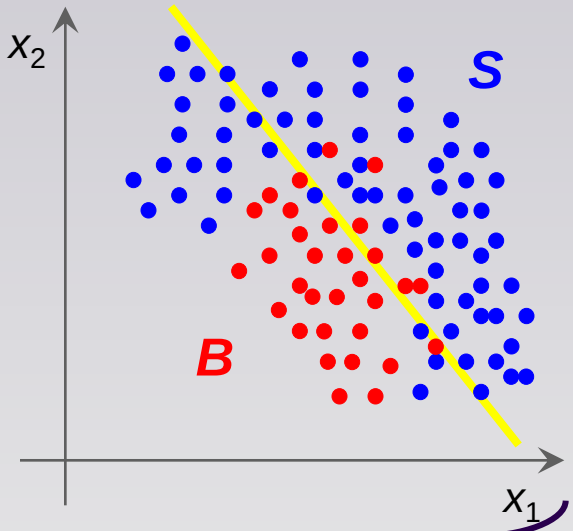
- Discriminate *Signal* from *Background*
  - how to set the decision boundary to select events of type *S* ?
  - we have discriminating variables x_1, x_2, …



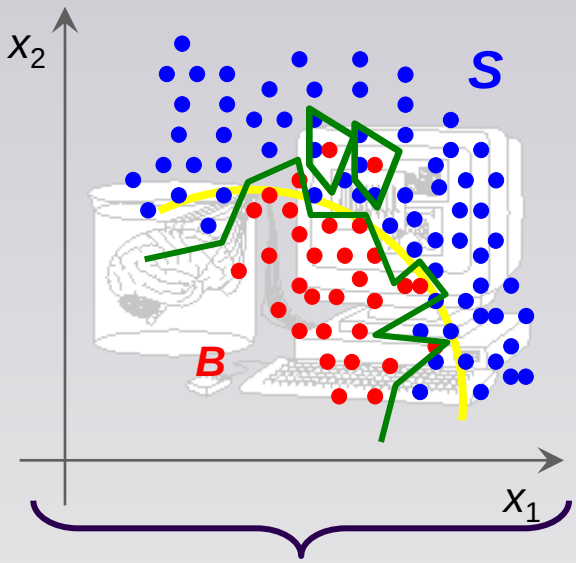
Rectangular cuts?



A linear boundary?



A nonlinear one?



- Which model/class ? Pro and cons ?

Low variance (stable), high bias methods

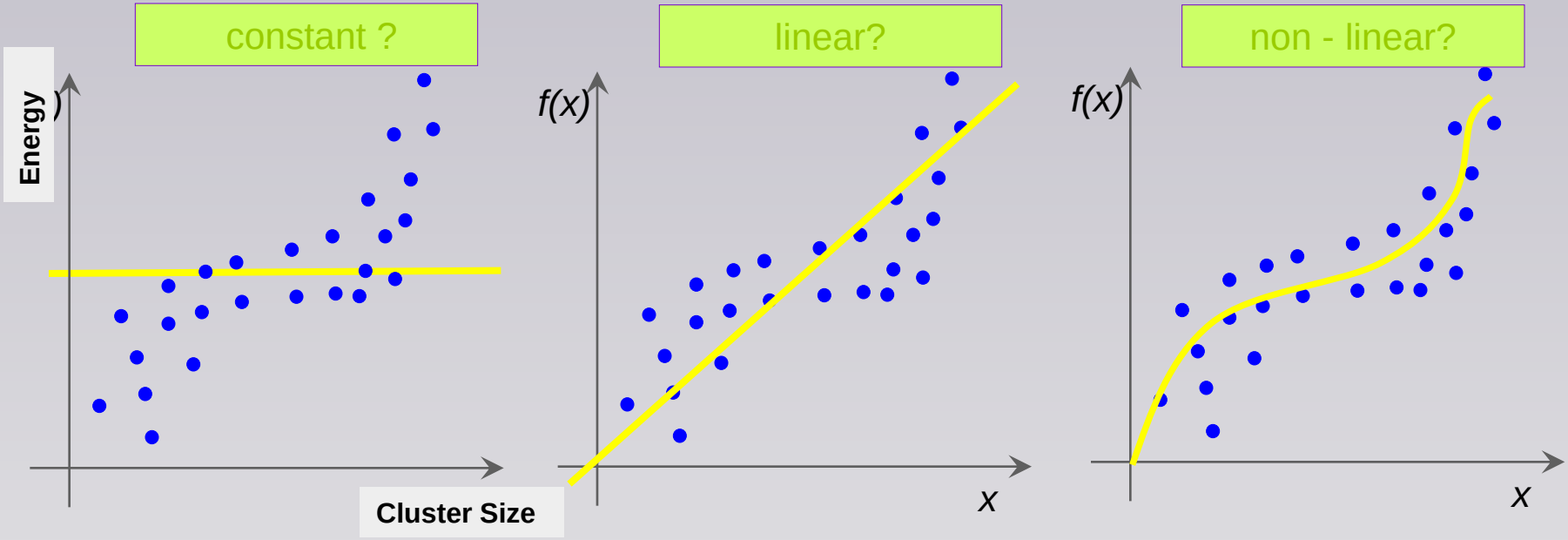
High variance, small bias methods

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

Regression

- estimate a “functional behaviour” from a set of ‘known measurements” ?
- e.g. : “D”-variables that somehow characterize the shower in your calorimeter

  → energy as function of the calorimeter shower parameters .



- if we had an analytic model (i.e. know the function is a n^{th} -order polynomial) than we know how to fit this (i.e. Maximum Likelihood Fit)

  → but what if we just want to “draw any kind of curve” and parameterize it?
- seems trivial ? → The human brain has very good pattern recognition capabilities!
- what if you have **many** input variables?

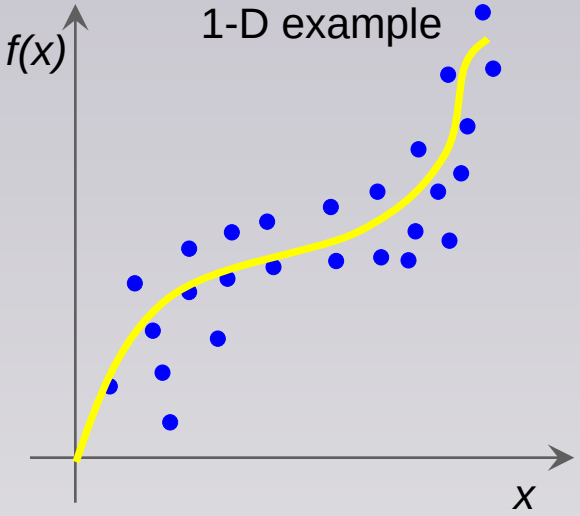
Regression → model functional behaviour

- e.g. "D"-variables that somehow characterize the shower in your calorimeter.
  - Monte Carlo or testbeam

    → data sample with measured cluster observables + known particle energy

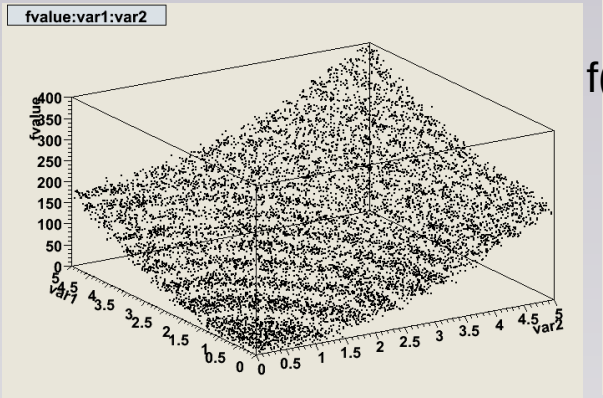
    = calibration function (energy == surface in D+1 dimensional space)

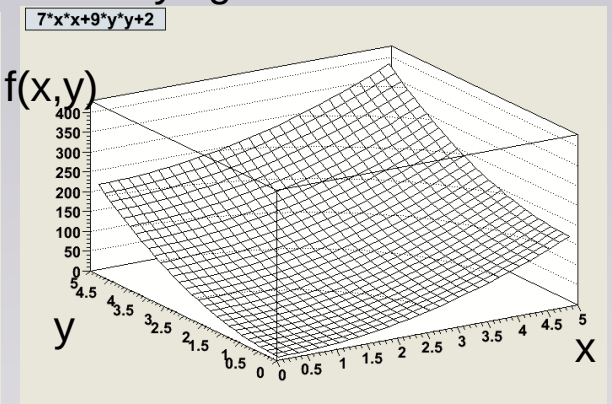
1-D example



2-D example

events generated according: underlying distribution



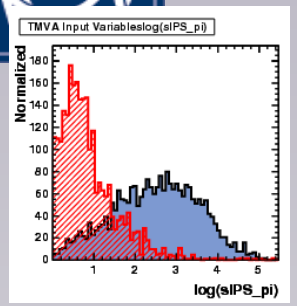


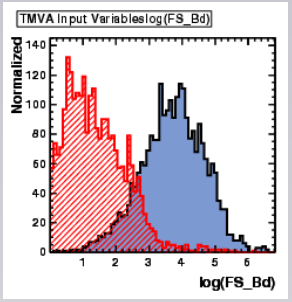
- better known: (linear) regression → fit a known analytic function
  - *e.g.* the above 2-D example → reasonable function would be: $f(x) = ax^2+by^2+c$
- don't have a reasonable "model" ? → need something more general:
  - *e.g.* piecewise defined splines, kernel estimators, decision trees to approximate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

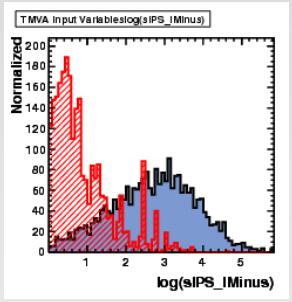
→ NOT in order to "fit a parameter"

→ provide prediction of function value f(x) for new measurements x (where f(x) is not known)

Event Classification







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

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

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

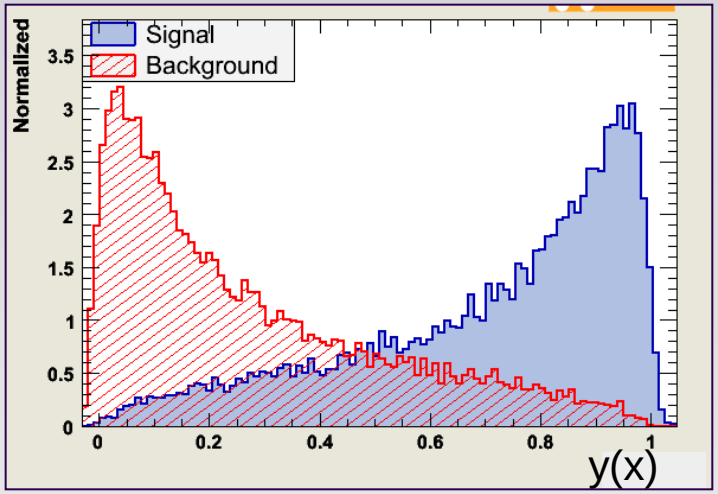
  $\mathbf{P}^D \rightarrow \mathbf{P}$

  most general form

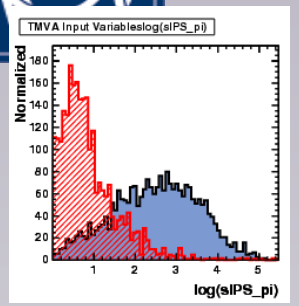
  $y = y(\mathbf{x}); \quad \mathbf{x} \in \mathbf{P}^D$

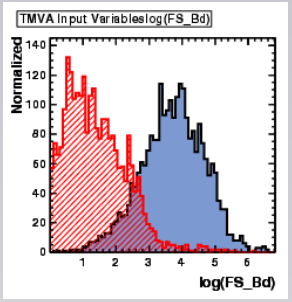
  $\mathbf{x}=\{x_1,\ldots,x_D\}$: input variables

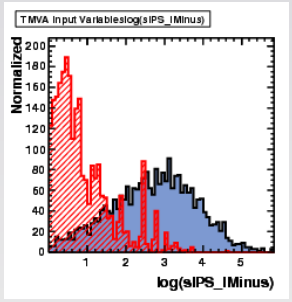
- plotting (historamming) the resulting y(x) values:



Event Classification





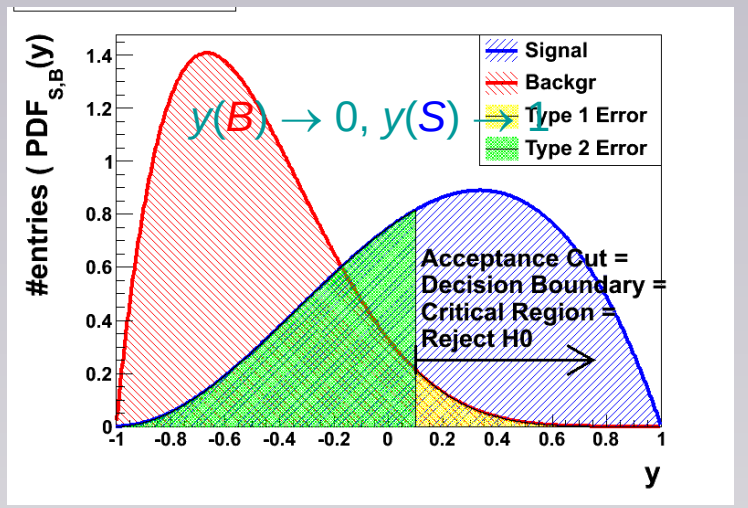


P^D "feature space"

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

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

P

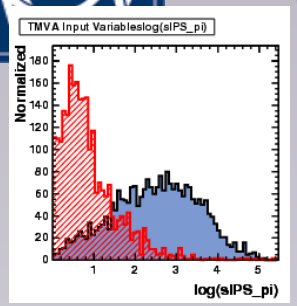


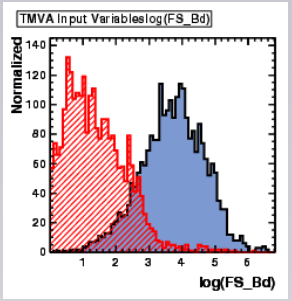
- y(x): "test statistic" in D-dimensional space of input variables
- distributions of y(x): $PDF_S(y)$ and $PDF_B(y)$
- used to set the selection cut!
- → efficiency and purity

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

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

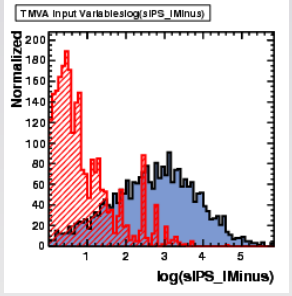
Classification ↔ Regression





P^D

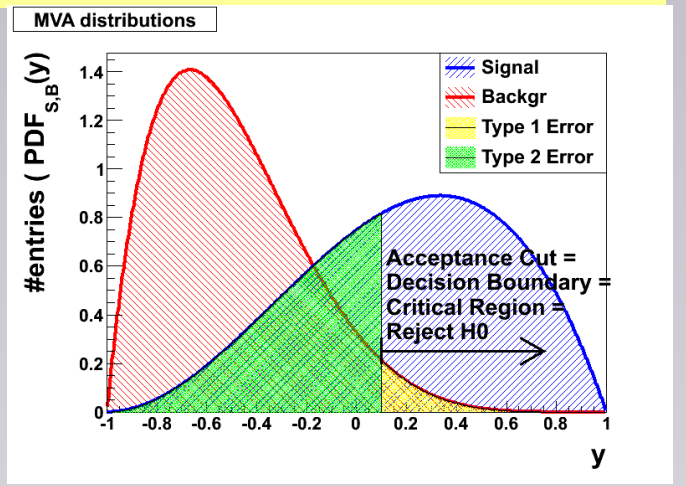
"feature space"



Classification:

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

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

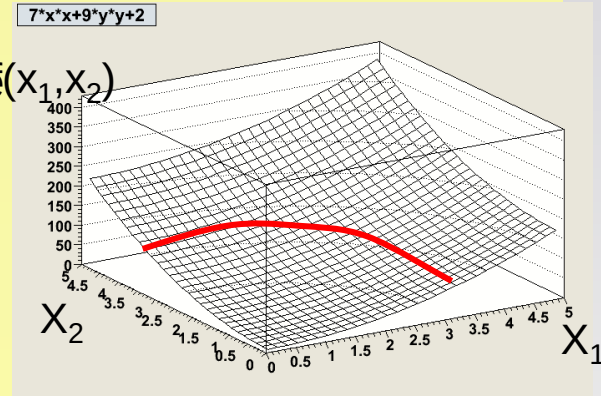


y(x): $R^D \rightarrow R$: → P

Regression:

- Each event has "D" measured variables + one function value (*e.g.* cluster shape variables in the ECAL + particles energy)
- y(x): $R^D \rightarrow R$ "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.



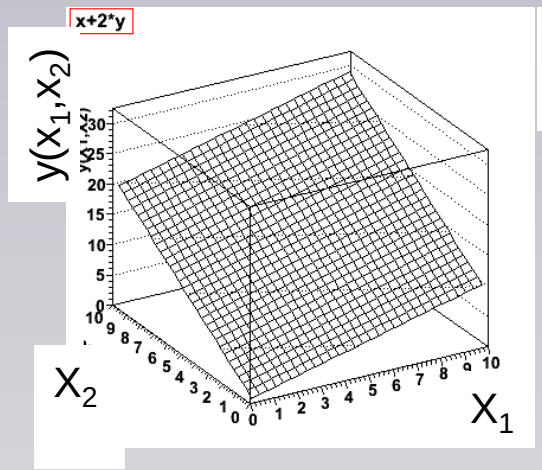
Classification and Regression Visualisation in 2D

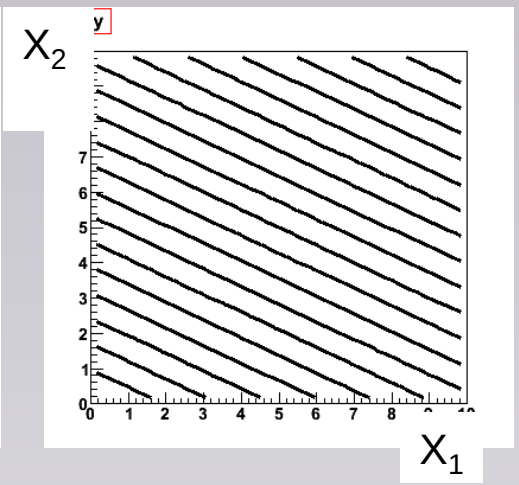
Test Statistic y(x):

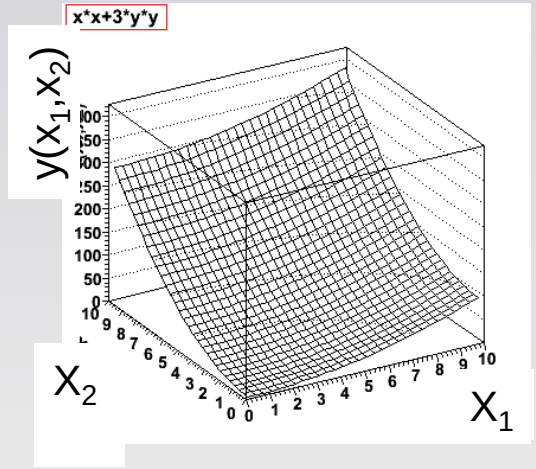
→ function of the the input variables :

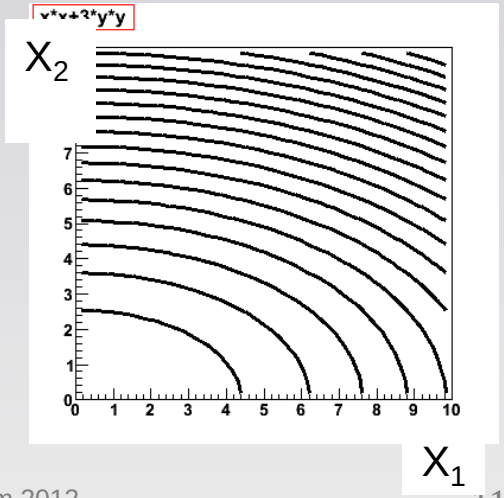
→ Classification:
y(x)=const → decision boundaries !

→ Regression:
y(x) = your target function



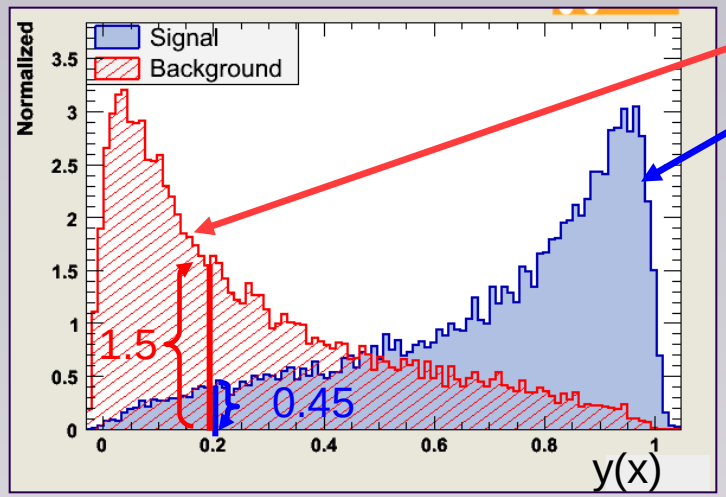






Event Classification

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



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

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

Probability densities for background and signal

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

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

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

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

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

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

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

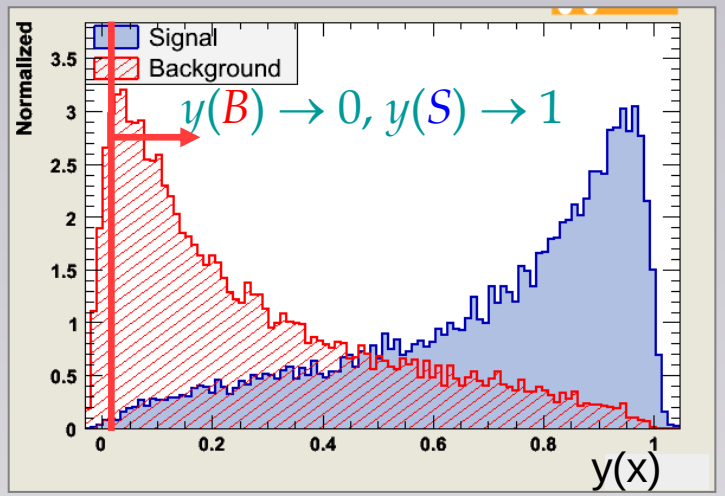
Posterior probability

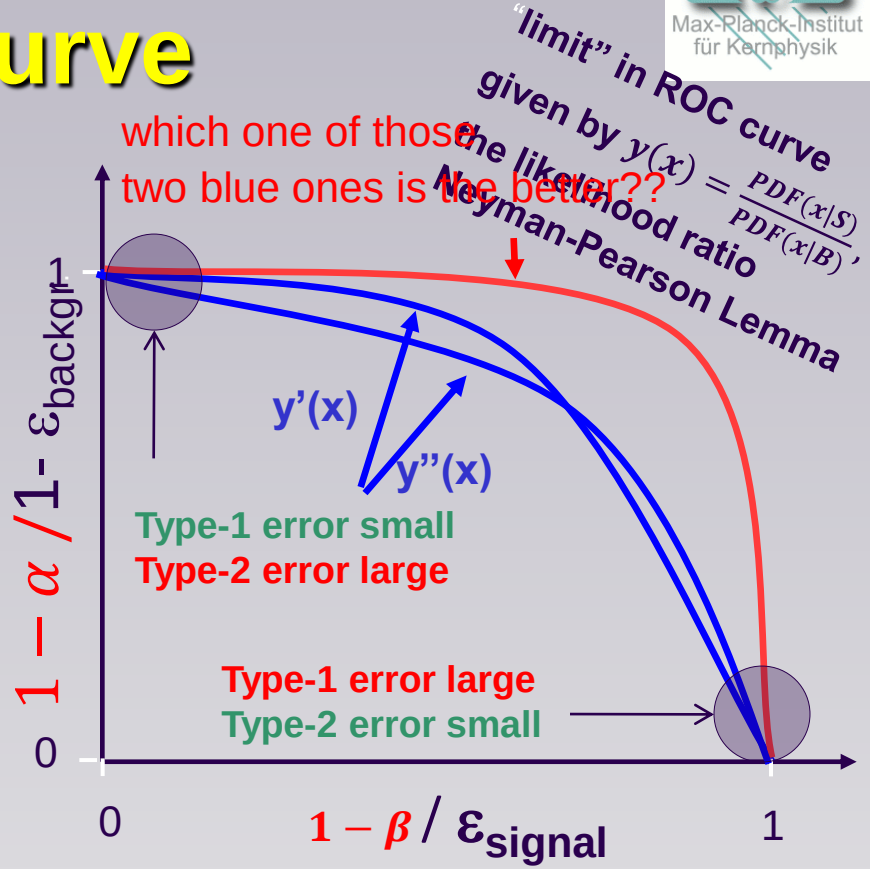
Overall probability density to observe the actual measurement y(x). *i.e.* $P(y) = \sum_{\text{Classes}} P(y \mid \text{Class})P(\text{Class})$

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

Receiver Operation Characteristic (ROC) curve

Signal(H_1) /Background(H_0) discrimination:





Signal(H_1) /Background(H_0) :

- **Type 1 error: reject H_0 although true → background contamination**
  - Significance α: background sel. efficiency **1– α: background rejection**
- **Type 2 error: accept H_0 although false → loss of efficiency**
  - Power: **1– β signal selection efficiency**

MVA and Machine Learning

- Finding y(x) : $R^n \rightarrow R$
  - given a certain type of model class y(x)
  - "automatically" using "known" or "previously solved" events
    - i.e. learn from known "patterns"
  - such that y(x):
    - separates well Signal from Background in training data
    - (regression: fits well the target function for training events
    - … AND in new events → predictions

  → supervised machine learning

- Of course… there's no magic, we still need to:
  - choose the discriminating variables
  - choose the class of models (linear, non-linear, flexible or less flexible)
  - tune the "learning parameters" → bias vs. variance trade off
  - check generalization properties
  - consider trade off between statistical and systematic uncertainties

Event Classification

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

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

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

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

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

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

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

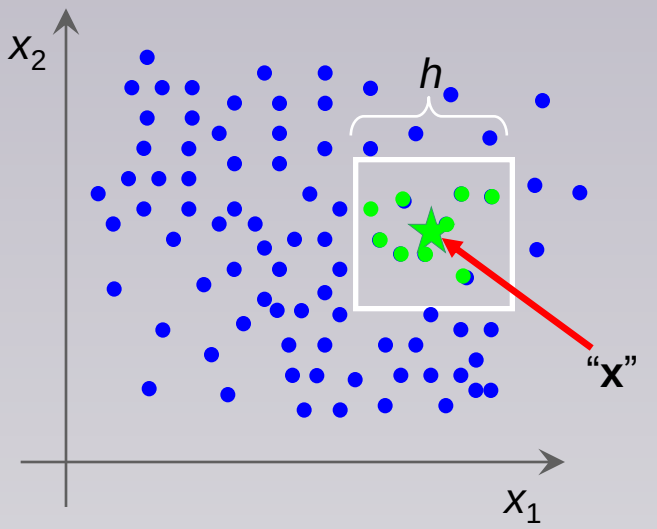
    → e.g. Linear Discriminator, Neural Networks, …

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

K- Nearest Neighbour

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

"events" distributed according to P(x)



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

k(u): is called a Kernel function

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

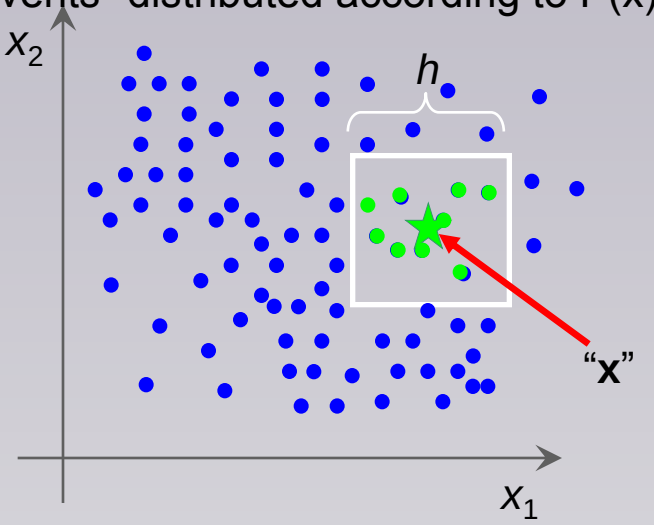
- Classification: Determine
  $PDF_S(x)$ and $PDF_B(x)$
  → likelihood ratio as classifier!

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

→ Kernel Density estimator of the probability density

Nearest Neighbour and Kernel Density Estimator

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



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

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

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

- 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

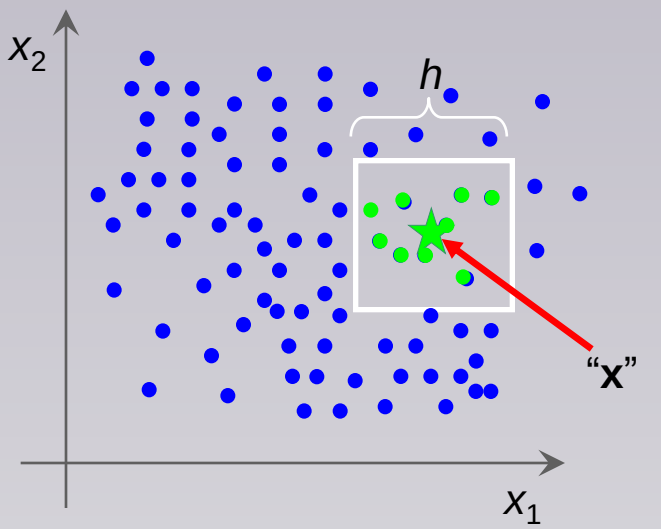
Nearest Neighbour and Kernel Density Estimator

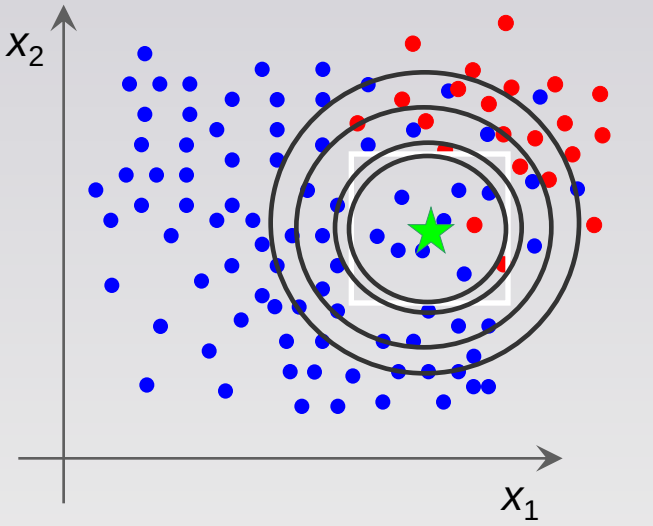
- estimate probability density P(x) in D-dimensional space:
- The only thing at our disposal is our "training data"
- Say we want to know P(x) at "this" point "x"
- One expects to find in a volume V around point "**x**"
  $N^* \int_V P(x)dx$ events from a dataset with N events
- For the chosen a rectangular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

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

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

- Kernel Density Estimator: replace "window" by "smooth" kernel function → weight events by distance

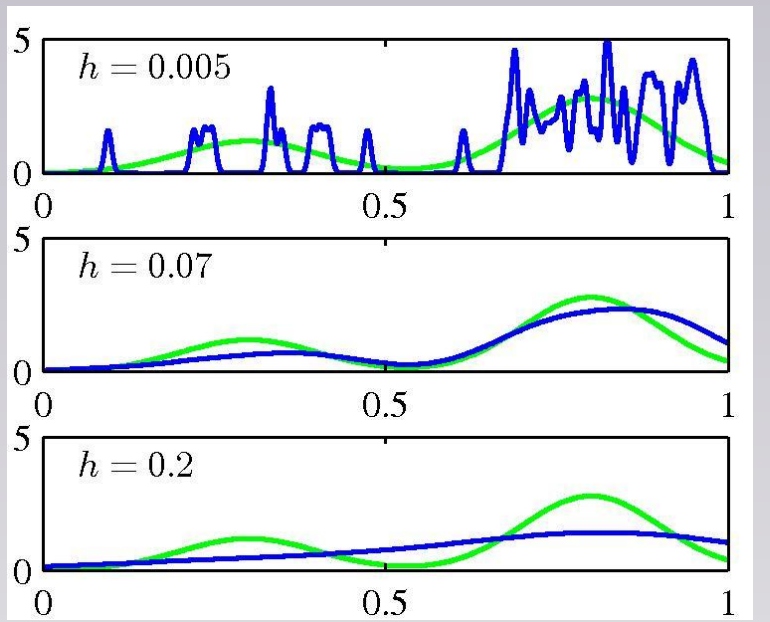




Kernel Density Estimator

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

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



(Christopher M.Bishop)

- which metric for the Kernel (window)?
  - 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

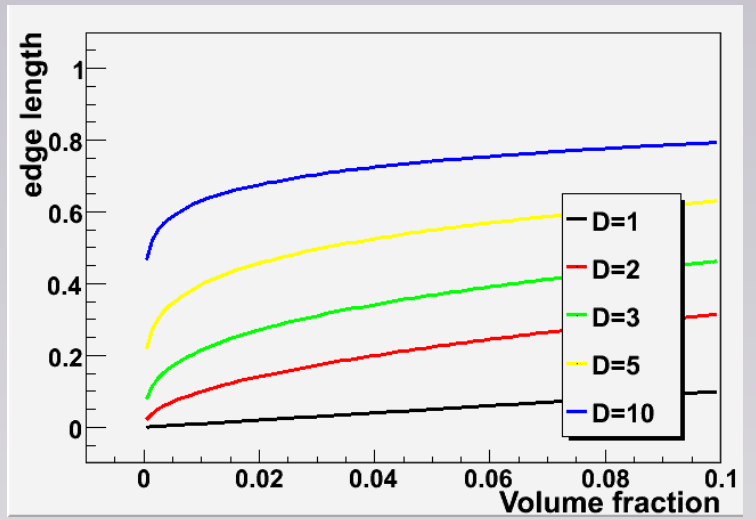
"Curse of Dimensionality"

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

We all know:

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

Shortcoming of nearest-neighbour strategies:



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

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

for a cube of a particular fraction of the volume:

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

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

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

Naïve Bayesian Classifier (projective Likelihood Classifier)

Multivariate Likelihood (k-Nearest Neighbour)

→ estimate the full D-dimensional joint probability density

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

product of marginal PDFs

(1-dim "histograms")

PDFs

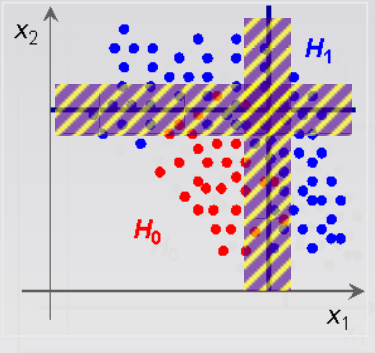
discriminating variables

Likelihood ratio for event *event*

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

Classes: signal, background types

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



PDE introduces fuzzy logic

Classifier Training and Loss-Function

- kNN,Likelihood → estimate underlying PDF in D- and 1- dimension
  → exploit Neyman Pearson lemma
  → limitations: curse of dimensionality and correlations

- Alternative: provide a set of "basis" functions (or model):
  - $y(x) = \sum w_i h_i(x)$
  - adjust parameters w_i → optimally separating hyperplane (surface)
    → called "training

- optimally separating → minimum in expectation value of a Loss function:
  $L(y_{true}, y(x))$ penalizes prediction errors in training

  - $E[L] = E[(y_{true} - y(x))^2]$ squared error loss (regression) → **minimize**
  - $E[L] = E[|y_{true} - y(x)|]$ misclassification error (classification)

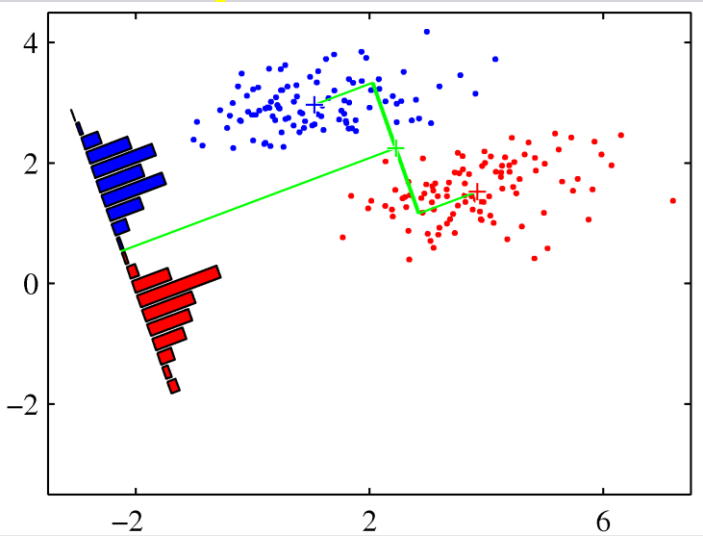
where: regression: y_{true} the functional value of training events
classification: y_{true} =1 for signal, =0 (-1) background

Linear Discriminant

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

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

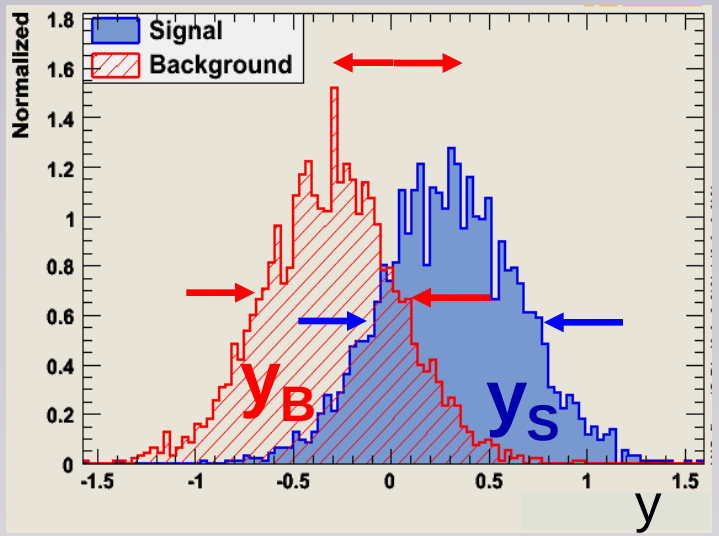
i.e. any linear function of the input variables: → linear decision boundaries



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

Fisher's Linear Discriminant

$$y(\vec{x} = \{x_1, \ldots, x_D\}) \Rightarrow y(\vec{x}, \vec{w}) \Rightarrow \vec{w}\,\vec{x}_0 + \sum_{i=1}^{D} w_i x_i$$



determine the "weights" w that do "best"

- Maximise "separation" between the S and B

→ minimise overlap of the distribution y_S and y_B

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

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

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

note: these quantities can be calculated from the training data

Neural Networks

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

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

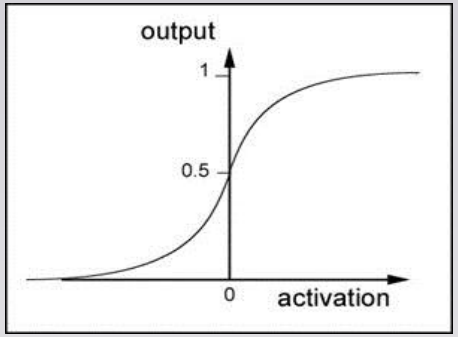
- Think of $h_i(x)$ as a set of “basis” functions
- If h(x) is sufficiently general (i.e. non linear), a linear combination of “enough” basis function should allow to describe any possible discriminating function y(x)

there are also mathematical proves for previous statement.

$h_i(\mathbf{x})$

Imagine you chose do the following:

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



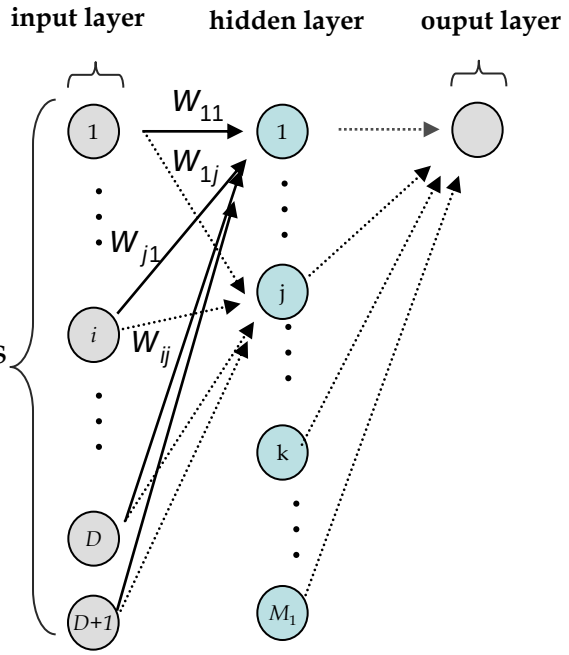
$A(x) = \frac{1}{1+e^{-x}}$:
the sigmoid function

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

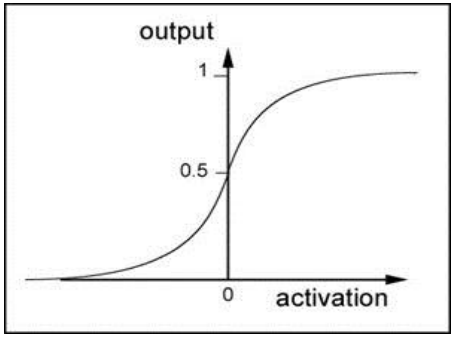
Ready is the Neural Network
Now we “only” need to find the appropriate “weights” w

Neural Networks: Multilayer Perceptron MLP

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



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



"Activation" function e.g. sigmoid:

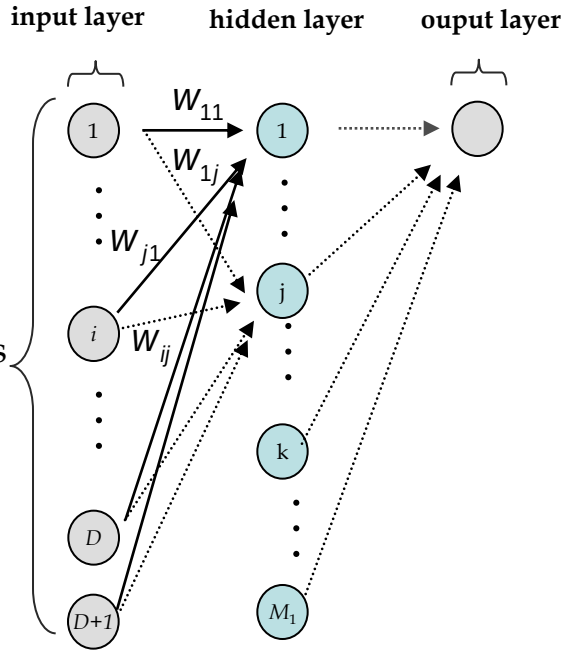
$$A(x) = \left(1+e^{-x}\right)^{-1}$$

or tanh
or …

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

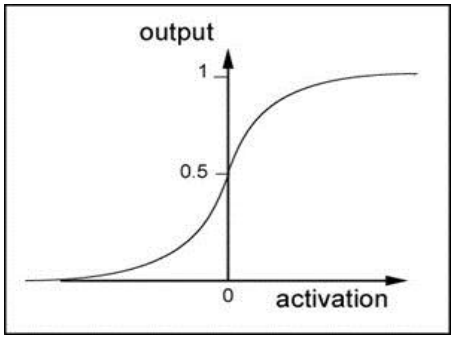
Neural Networks: Multilayer Perceptron MLP

try to "interpret" the formula as a Neural Network:



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

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



"Activation" function e.g. sigmoid:

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

or tanh
or …

nodes→neurons
links(weights)→synapses

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

Neural Network Training

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

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

how do we adjust ?

- minimize Loss function:

$$L(w) = \sum_{i}^{\text{events}} (y(x_i) - y(C))^2$$

where

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

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

predicted event type — **true event type**

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

→one global overall fit not efficient/reliable

→back propagation (learn from experience, gradually adjust your resonse)

→online learning (update event by event)

→batch learning (update after seeing the whole sample)

Neural Network Training back-propagation

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

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

- for weights connected to output nodes

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

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

- for weights not connected to output nodes

  … a bit more complicated formula

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

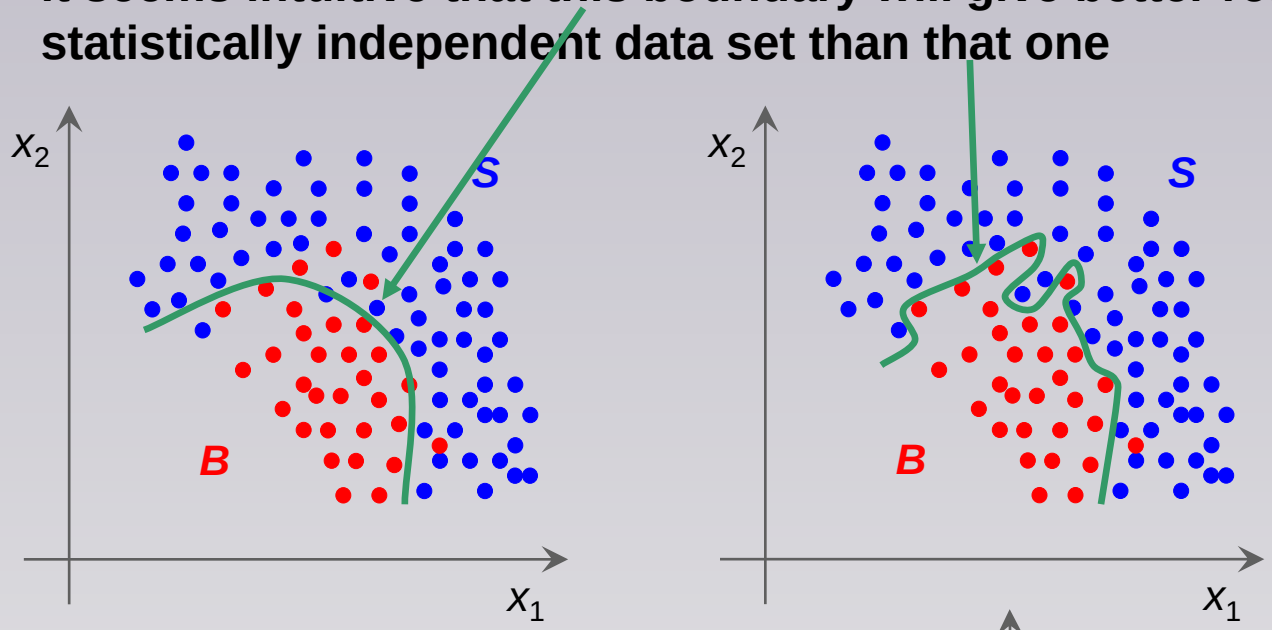
- training is repeated n-times over the whole training data sample. how often ??

  → early stopping: traditional way to avoid overtraining

  → there are also other "regularisation"

Overtraining

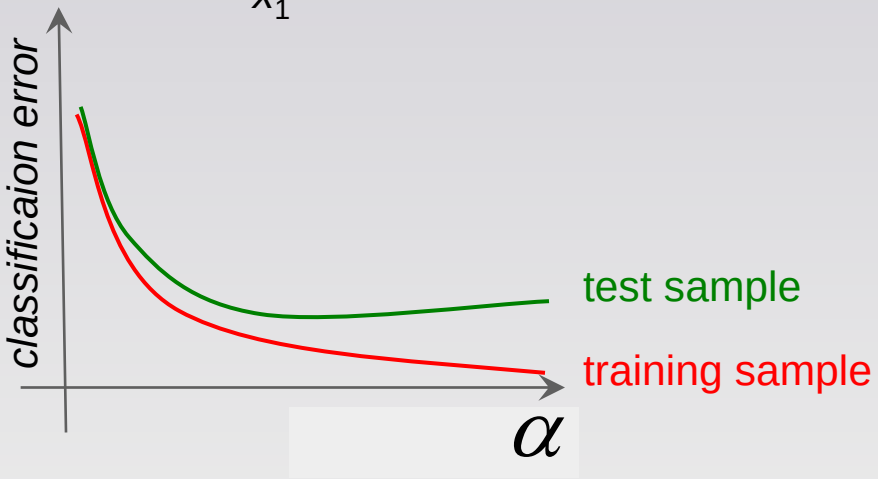
- training: n-times over all training data **how often ??**
- it seems intuitive that this boundary will give better results in another statistically independent data set than that one



→ e.g. stop training before you learn statistical fluctuations in the data

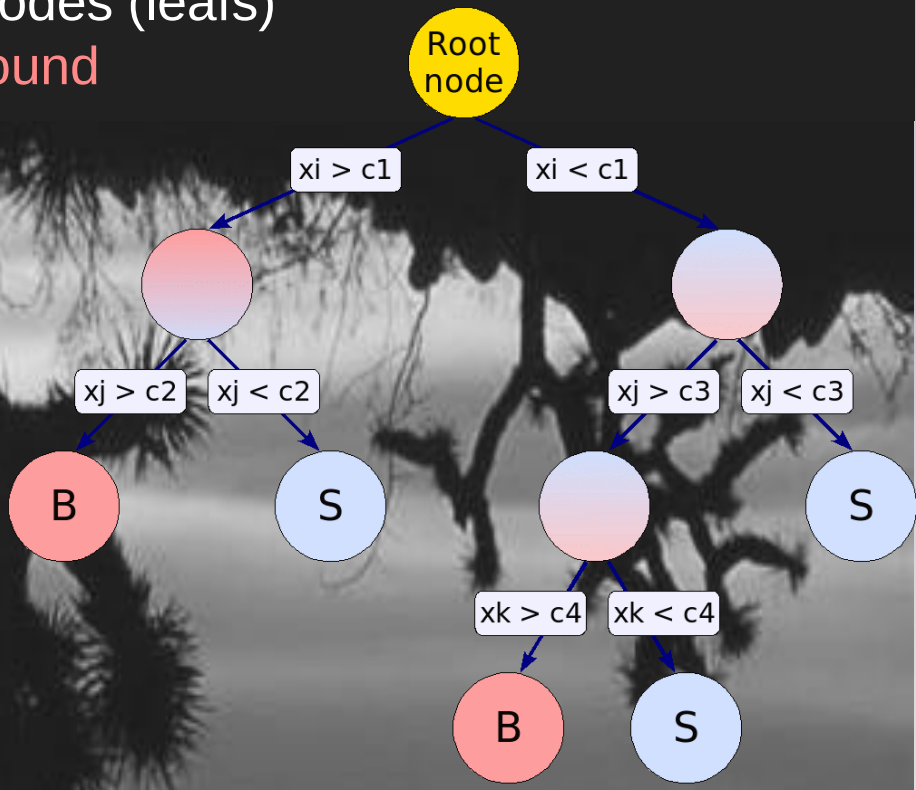
→ verify on independent "test" sample

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



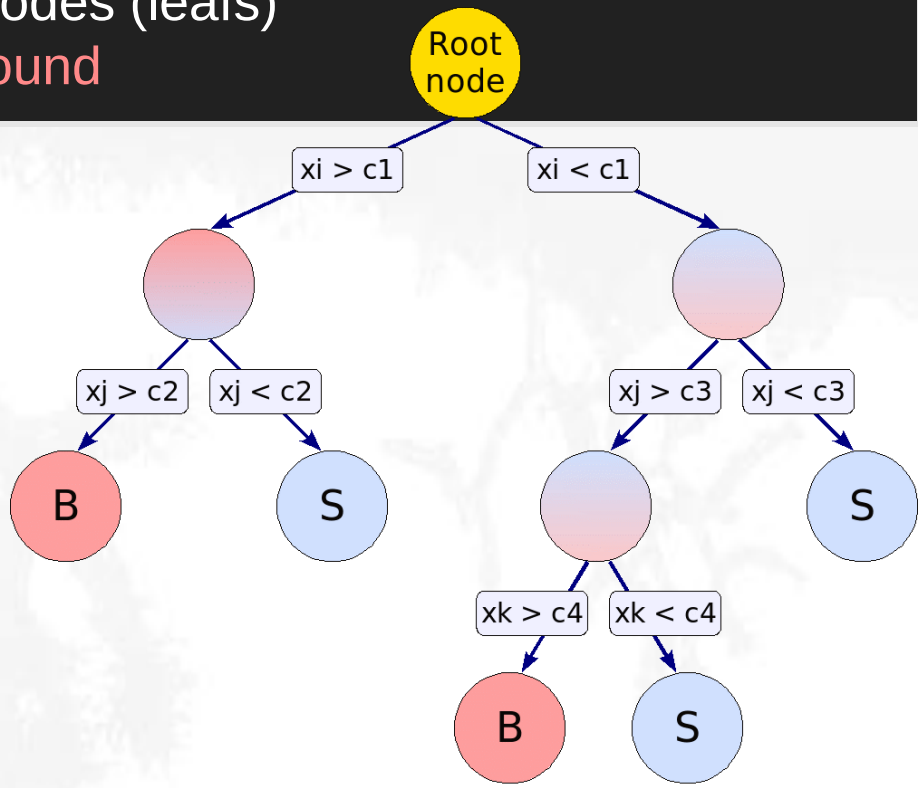
Boosted Decision Trees

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



Boosted Decision Trees

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

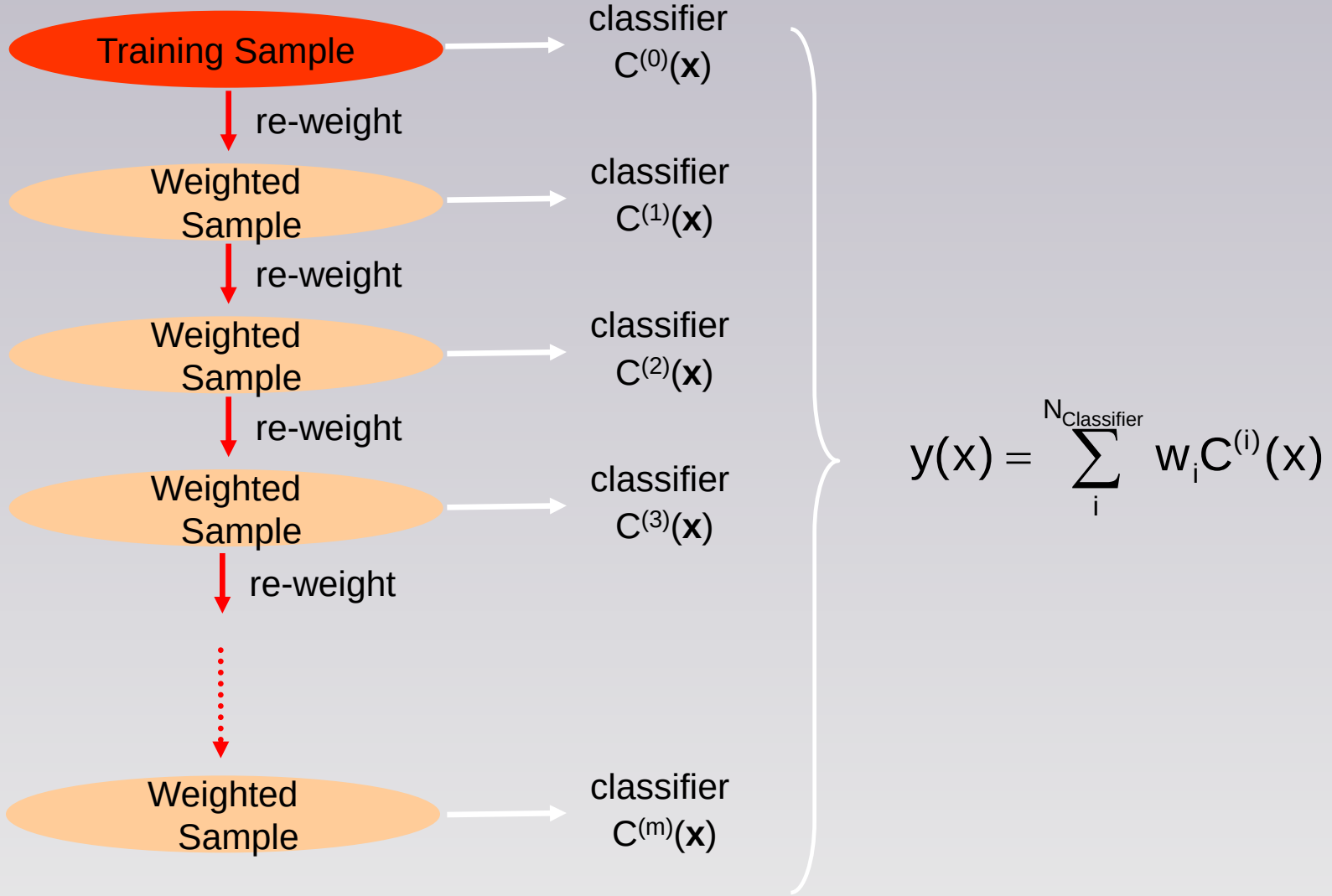


- Each branch → one standard "cut" sequence
  - easy to interpret, visualised
  - independent of monotonous variable transformations, immune against outliers
  - weak variables are ignored (and don't (much) deteriorate performance)
- Disadvatage → very sensitive to statistical fluctuations in training data

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

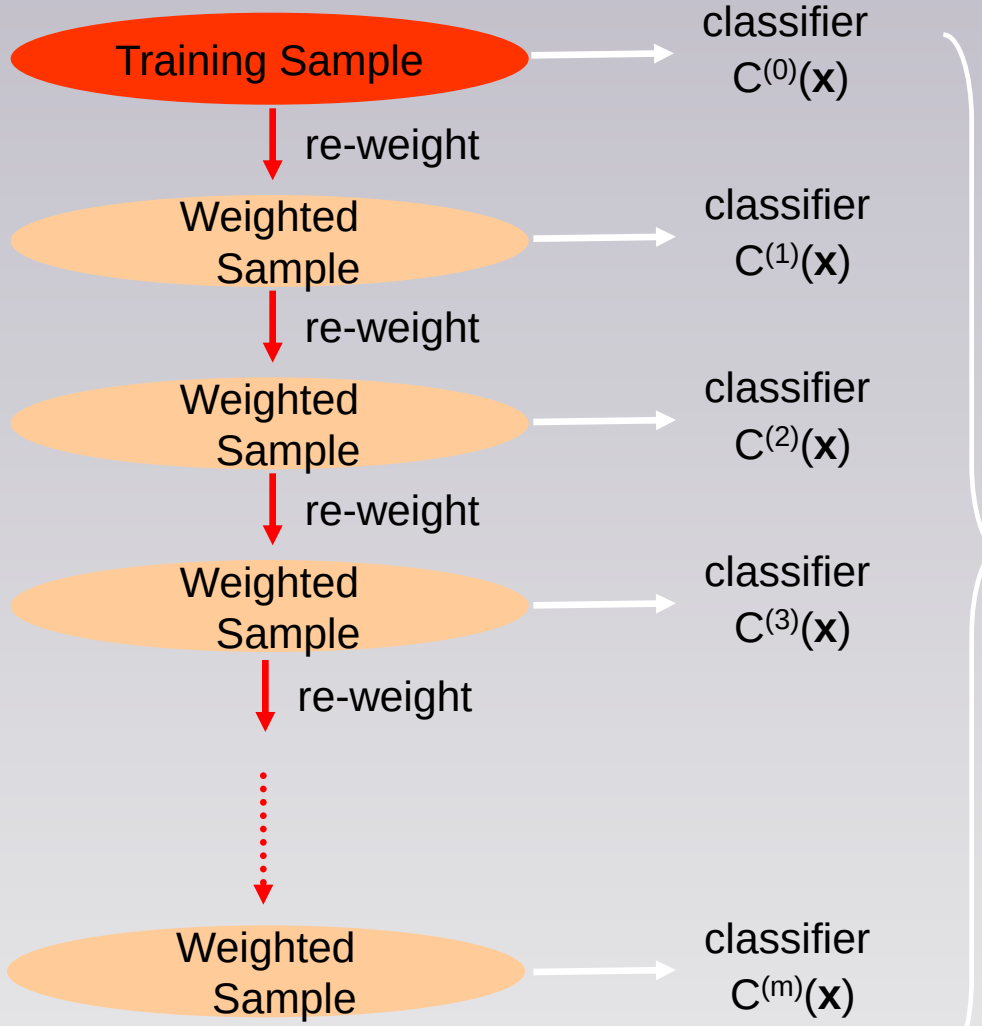
→ became popular in HEP since MiniBooNE, B.Roe et.a., NIM 543(2005)

Boosting



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

Adaptive Boosting (AdaBoost)



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

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

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

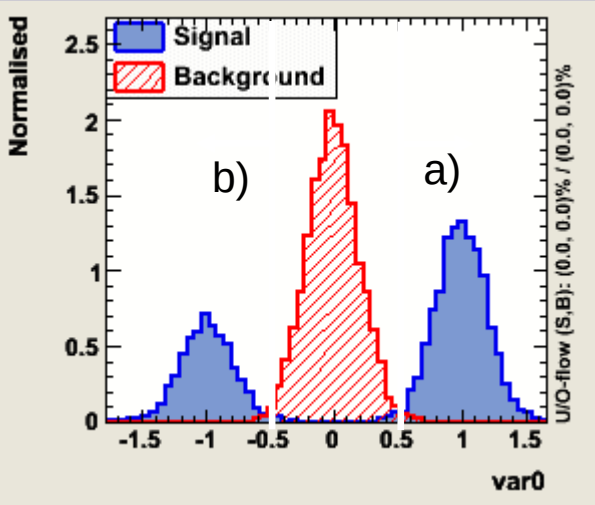
- AdaBoost weights the classifiers also using the error rate of the individual classifier according to:

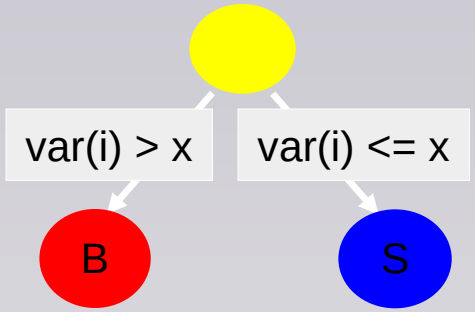
$$y(x) = \sum_{i}^{N_{Classifier}} \log\left(\frac{1-f_{err}^{(i)}}{f_{err}^{(i)}}\right) C^{(i)}(x)$$

AdaBoost: A simple demonstration

The example: (somewhat artificial…but nice for demonstration) :

- Data file with three “bumps”
- Weak classifier (i.e. one single simple “cut” ↔ decision tree stumps)





a) Var0 > 0.5 → ε_{sig}=66% ε_{bkg} ≈ 0% misclassified events in total 16.5%

or

b) Var0 < -0.5 → ε_{sig}=33% ε_{bkg} ≈ 0% misclassified events in total 33%

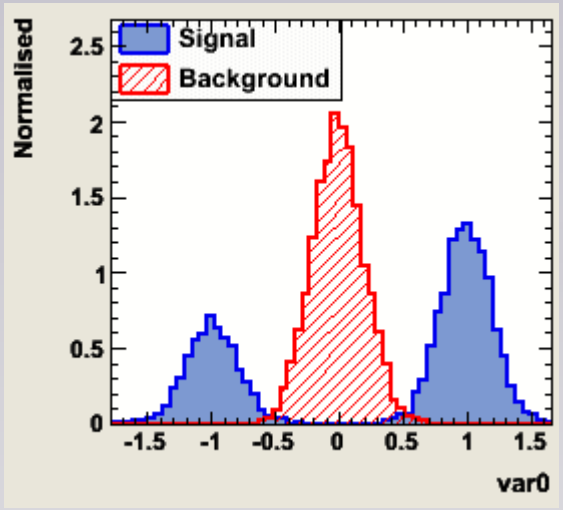
the training of a single decision tree stump will find “cut a)”

AdaBoost: A simple demonstration

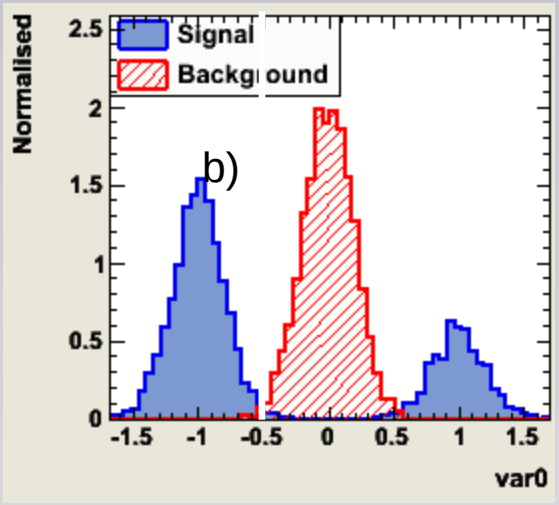
The first “tree”, choosing cut a) will give an error fraction: err = 0.165

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

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

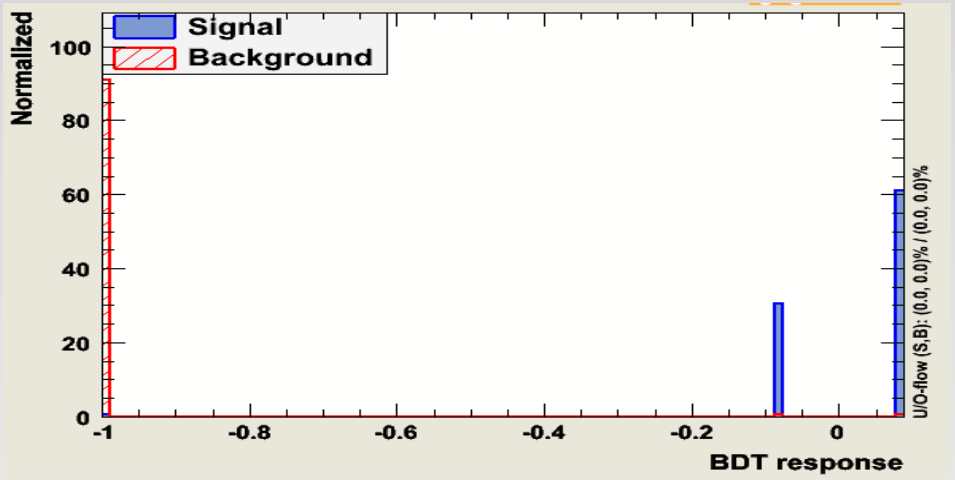


re-weight



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

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



Bagging and Randomised Trees

- Bagging:
  - combine trees grown from "bootstrap" samples

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

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

- or any "combination" of Bagging/Randomising/Boosting
- These combined classifiers work surprisingly well, are very stable and almost perfect "out of the box" classifiers

General Advice for (MVA) Analyses

- There is no magic in MVA-Methods:
  - no need to be too afraid of “black boxes” → they are not sooo hard to understand
  - you typically still need to make careful tuning and do some “hard work”
  - no “artificial intelligence” … just “fitting decision boundaries” in a given model
- The most important thing at the start is finding good observables
  - good separation power between S and B
  - little correlations amongst each other
  - no correlation with the parameters you try to measure in your signal sample!
- Think also about possible combination of variables
  - this may allow you to eliminate correlations
    - rem.: you are MUCH more intelligent than what the algorithm will do
- Apply pure preselection cuts and let the MVA only do the difficult part.
- “Sharp features should be avoided” → numerical problems, loss of information when binning is applied
  - simple variable transformations (i.e. log(variable)) can often smooth out these areas and allow signal and background differences to appear in a clearer way
- Treat regions in the detector that have different features “independent”
  - can introduce correlations where otherwise the variables would be uncorrelated!

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 a region(s) 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 !!

MVA and Systematic Uncertainties

- minimize “systematic” uncertainties

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

→ hard to “translate”: try to:

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

→ 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 this

    → look at “projections” of input variables

    → + the combined MVA test statistic “y(x)” !

Summary

- **Multivariate Classifiers (Regressors) → 1 dimensional test statistic y(x) and y(x)>c defines decision boundary**

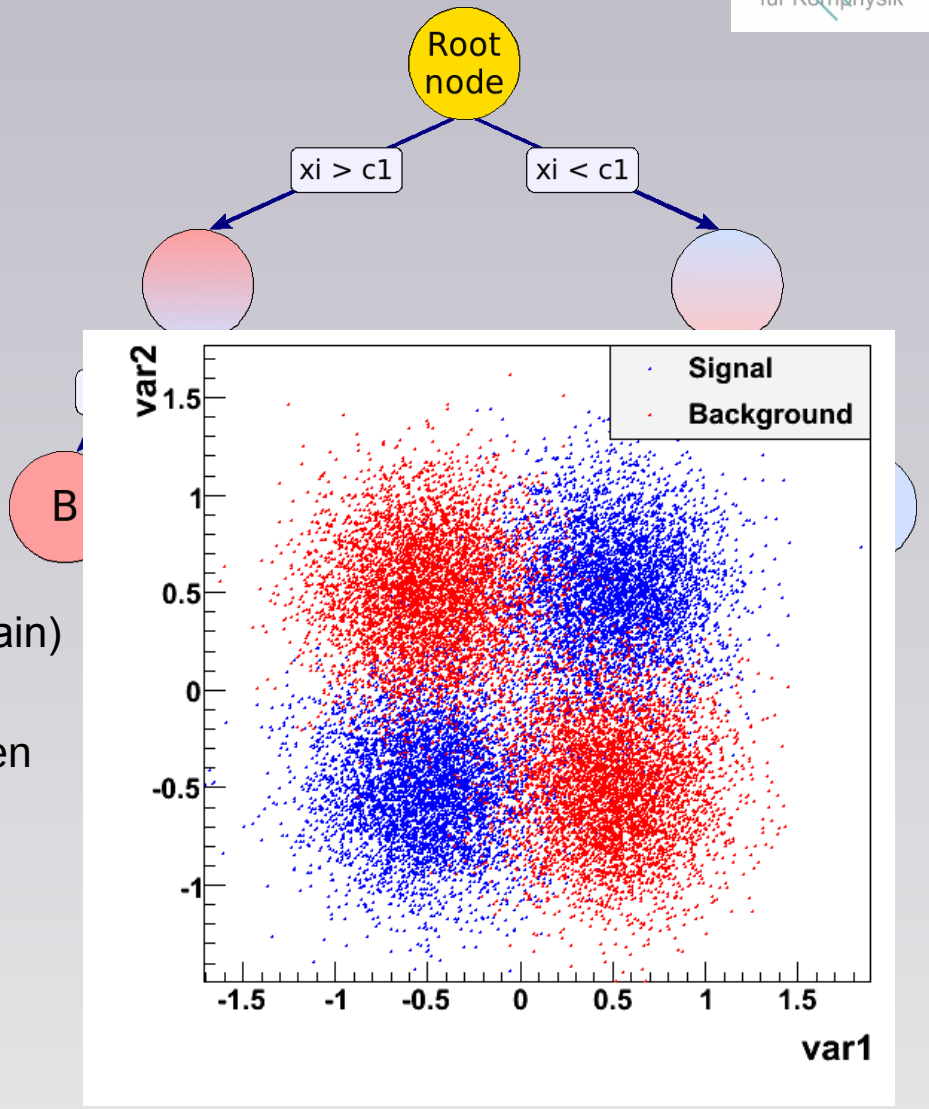
- **Mulit-dimensional (and projective) Likelihood**
  - **estimate the PDF and exploint Neyman-Pearsons Lemma: best test statistic is the Likelihood ratio**

- **Other classifiers "fit" a decision boundary "model"**
  - **Linear: Linear Classifier (e.g. Fisher Discriminant)**
  - **Non-Linear**
    - Neural Network
    - Boosted Decision Trees
    - (Support Vector Machines) → very nice but hard to explain in 5min…
- **No "magic" or "intelligence" … just fitting !**
- **Once one understands what "they are" you know**
  - **systematic uncertainties don't lie in the training !!**
  - **estimate them similar as you'd do in classical cuts**

Backup and Left overs…

Growing a Decision Tree

- training sample at the root node
- split training sample into two
  - variable and cut with best separation gain
- continue splitting until:
  - minimal #events per node
  - maximum number of nodes
  - maximum depth specified
  - (a split doesn't give a minimum separation gain) → not a good idea → see "chessboard"
    - Decision trees: grow large tree and then 'prune'
    - Boosted Decision tree: early stopping
- leaf-nodes classify S,B according to the majority of events or give a S/B probability



"A Statistical View of Boosting" (Friedman 1998 et.al)

Abstract:

…. For the two-class problem, boosting can be viewed as an approximation to additive modeling on the logistic scale using maximum Bernoulli likelihood as a criterion. We develop more direct approximations and show that they exhibit nearly identical results to boosting. Direct multi-class generalizations based on multinomial likelihood are derived that exhibit performance comparable to other recently proposed multi-class ….

- Boosted Decision Trees: two different interpretations
  - give events that are "difficult to categorize" more weight and average afterwords the results of all classifiers that were obtained with different weights
  - see each Tree as a "basis function" of a possible classifier →
    - boosting or bagging is just a mean to generate a set of "basis funciton"
    - linear combination of basis functions gives final classifier or: final classifier is an expansion in the basis functions.

$$y(\vec{\alpha}, x) = \sum_{\text{tree}} \alpha_i T_i(x)$$

    - every "boosting" algorithm can be interpreted as optimising in a "greedy stagewise" manner (*i.e.* from the current point in the optimisation –*e.g.building of the decision tree forest*- one chooses the parameters for the next boost step (weights) such that one moves a long the steepest gradient of the loss function)
    - AdaBoost: "exponential loss function" = exp($-y_0 y(\alpha,x)$) where y_0=-1 (bkg), y_0=1 (signal)

Gradient Boost

- Gradient Boost is a way to implement “boosting” with arbitrary “loss functions” by approximating “somehow” the gradient of the loss function

- AdaBoost: Exponential loss $\exp(-y_0 y(\alpha,x))$ → theoretically sensitive to outliers

- Binomial log-likelihood loss $\ln(1 + \exp(-2y_0 y(\alpha,x))$ → more well behaved loss function,

Support Vector Machines

- Neural Networks are complicated by finding the proper optimum “weights” for best separation power by “wiggly” functional behaviour of the piecewise defined separating hyperplane

- KNN (multidimensional likelihood) suffers disadvantage that calculating the MVA-output of each test event involves evaluation of ALL training events

- If Boosted Decision Trees in theory are always weaker than a perfect Neural Network

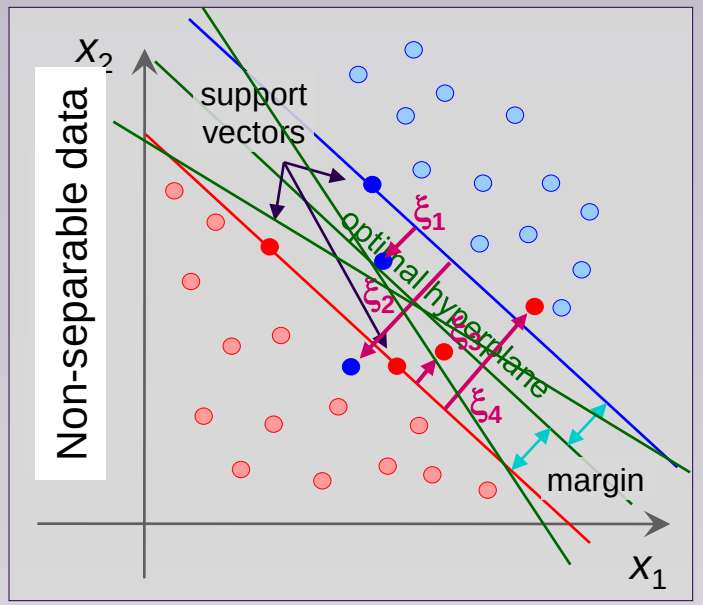
Support Vector Machine

- 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

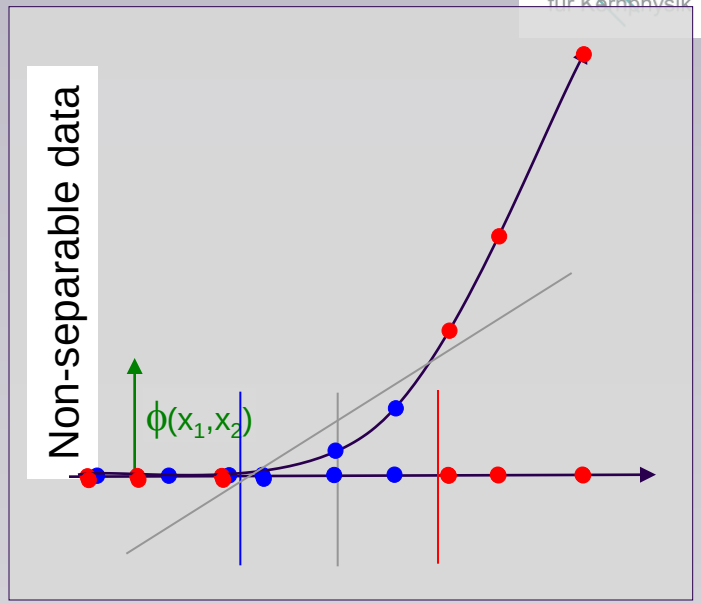
- hyperplane that separates **S** from **B**
  - Linear decision boundary
  - Best separation: maximum distance (margin) between closest events (*support*) to hyperplane
  - If data non-separable add *misclassification cost* parameter $C \cdot \Sigma_i \xi_i$ to minimisation function
  - **Solution of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

→ **quadratic minimisation problem**



Support Vector Machines

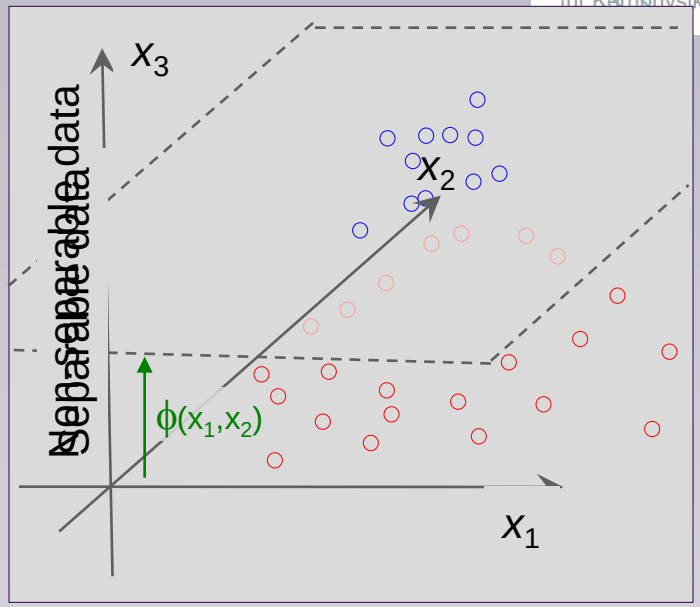
- hyperplane that separates **S** from **B**
  - Linear decision boundary
  - Best separation: maximum distance (margin) between closest events (*support*) to hyperplane
  - If data non-separable add *misclassification cost* parameter $C \cdot \Sigma_i \xi_i$ to minimisation function
  - **largest margin - 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 ponts: $\vec{x}\ and\ \vec{y}$ “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

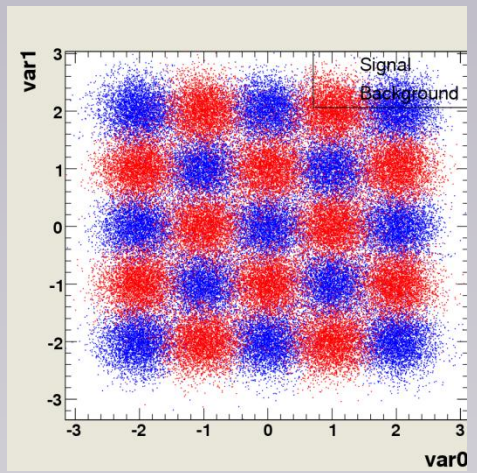
→ full separation of “any” event configuration with decision boundary along coordinate axis

→ well, that would of course be: overtraining

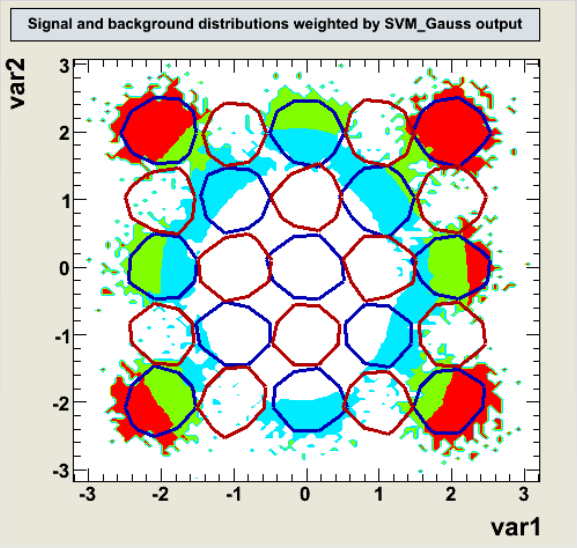
Support Vector Machines

SVM: the Kernel size parameter:

example: Gaussian Kernels



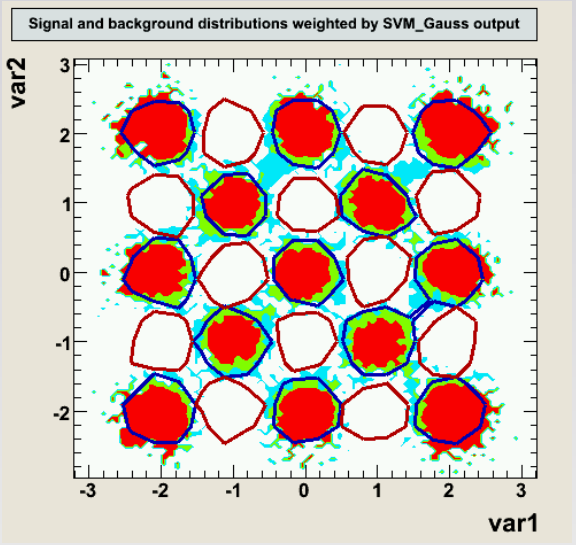
- Kernel size (σ of the Gaussian) choosen too large: → not enough "flexibility" in the underlying transformation



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

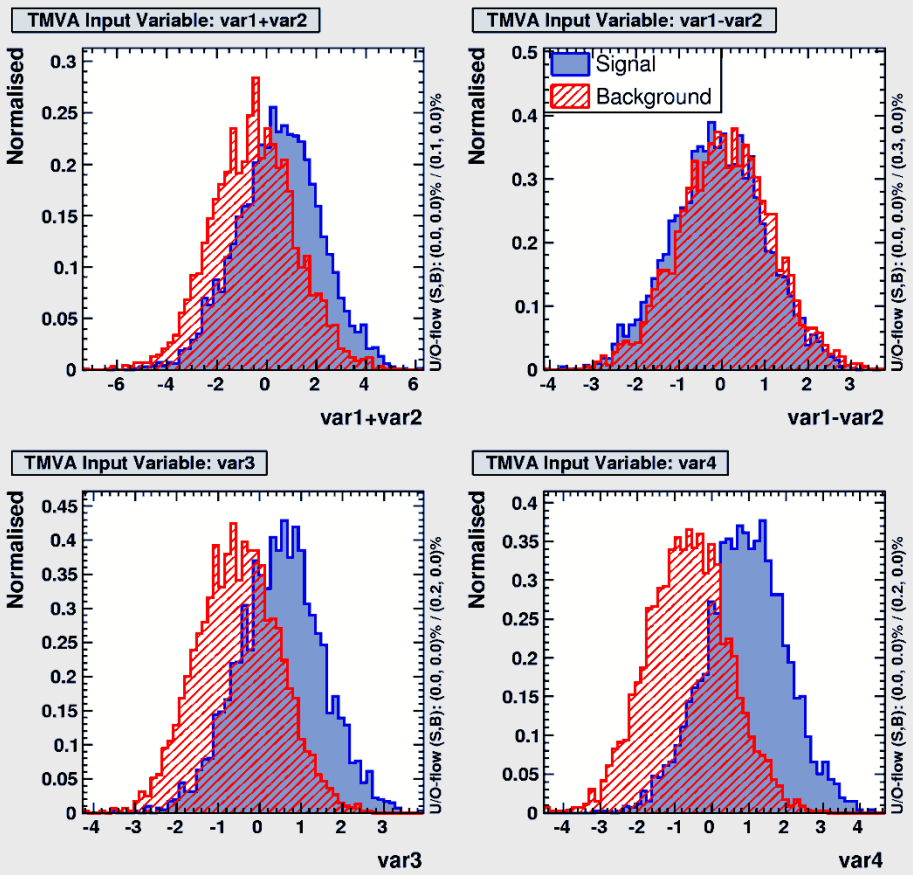
colour code:

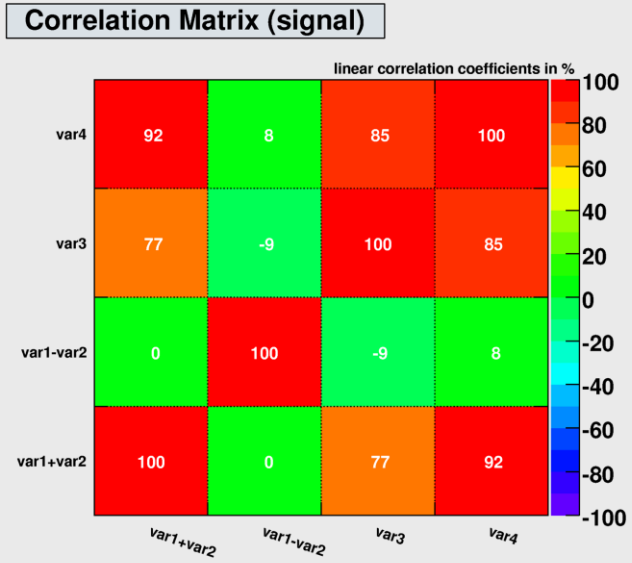
Red → large signal probability:



What if there are correlations?

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

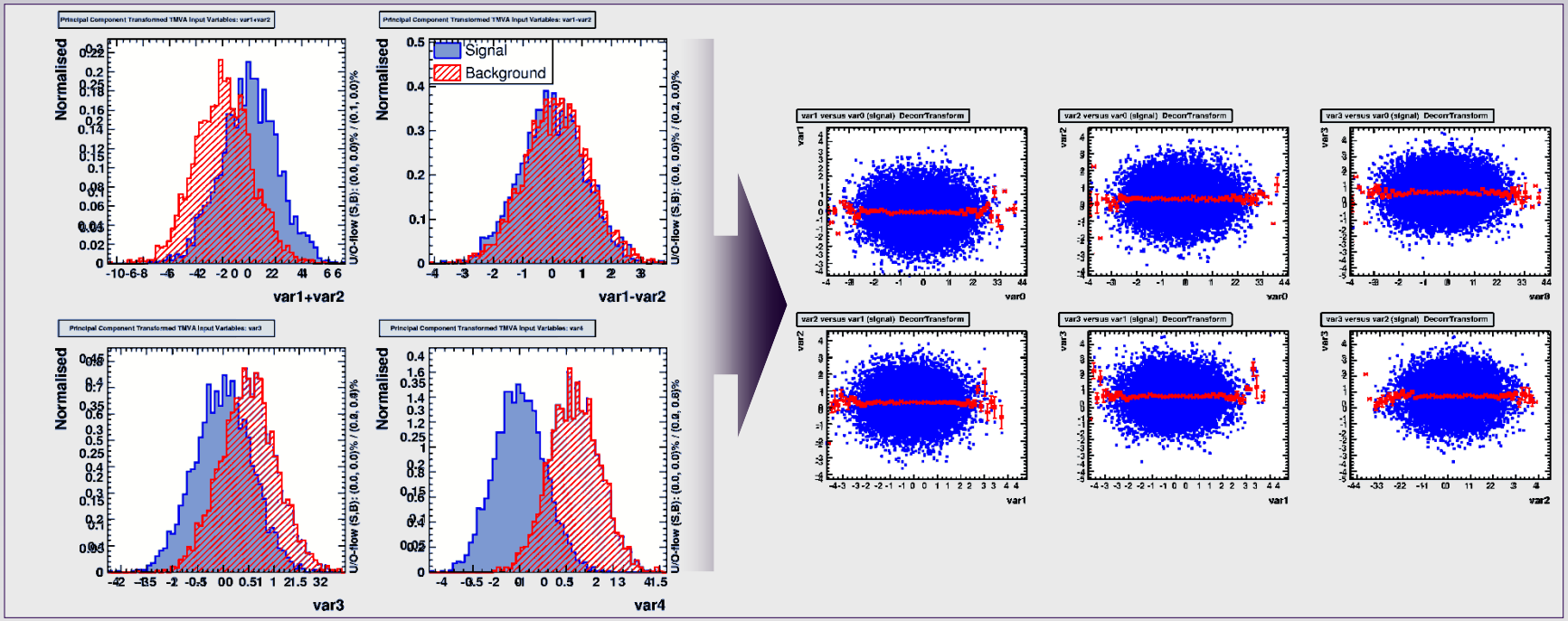




→ pre-processing: choose set of linear transformed input variables for which $C_{ij} = 0$ $(i\neq j)$

Decorrelation

- **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!!