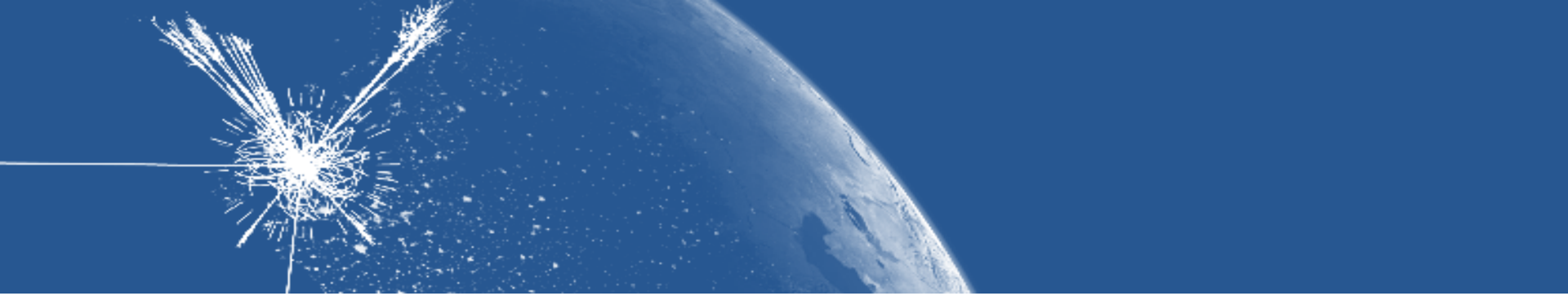
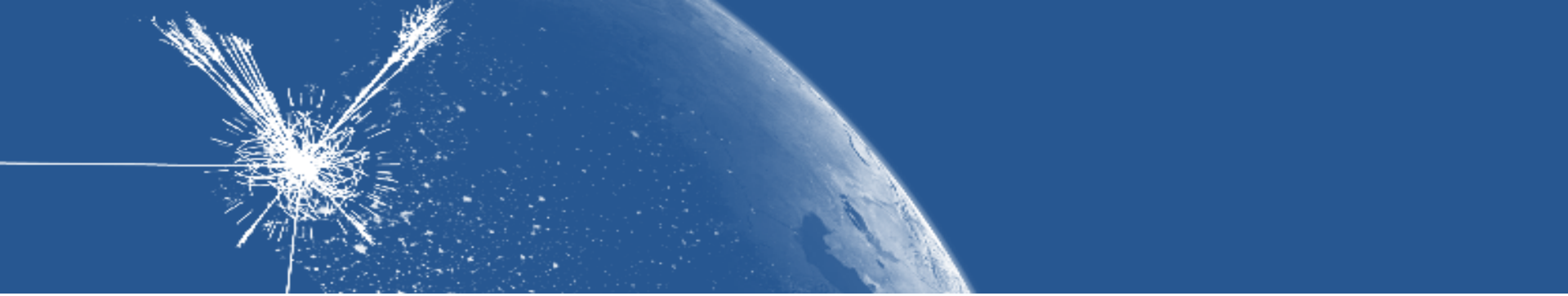


Introduction to probability and statistics (4)

Andreas Hoecker (CERN) & Helge Voss (MPI, Heidelberg)

CERN Summer Student Lecture, 18–21 July 2016



Outline (4 lectures)

1st lecture:

- Introduction
- Probability

2nd lecture:

- Probability axioms and hypothesis testing
- Parameter estimation
- Confidence levels

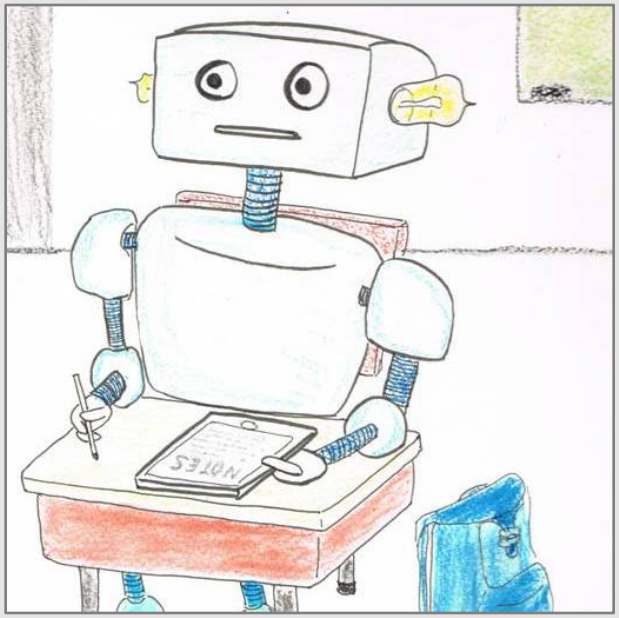
3rd lecture:

- Maximum likelihood fits
- Monte Carlo methods
- Data unfolding

4th lecture:

- Multivariate techniques and machine learning

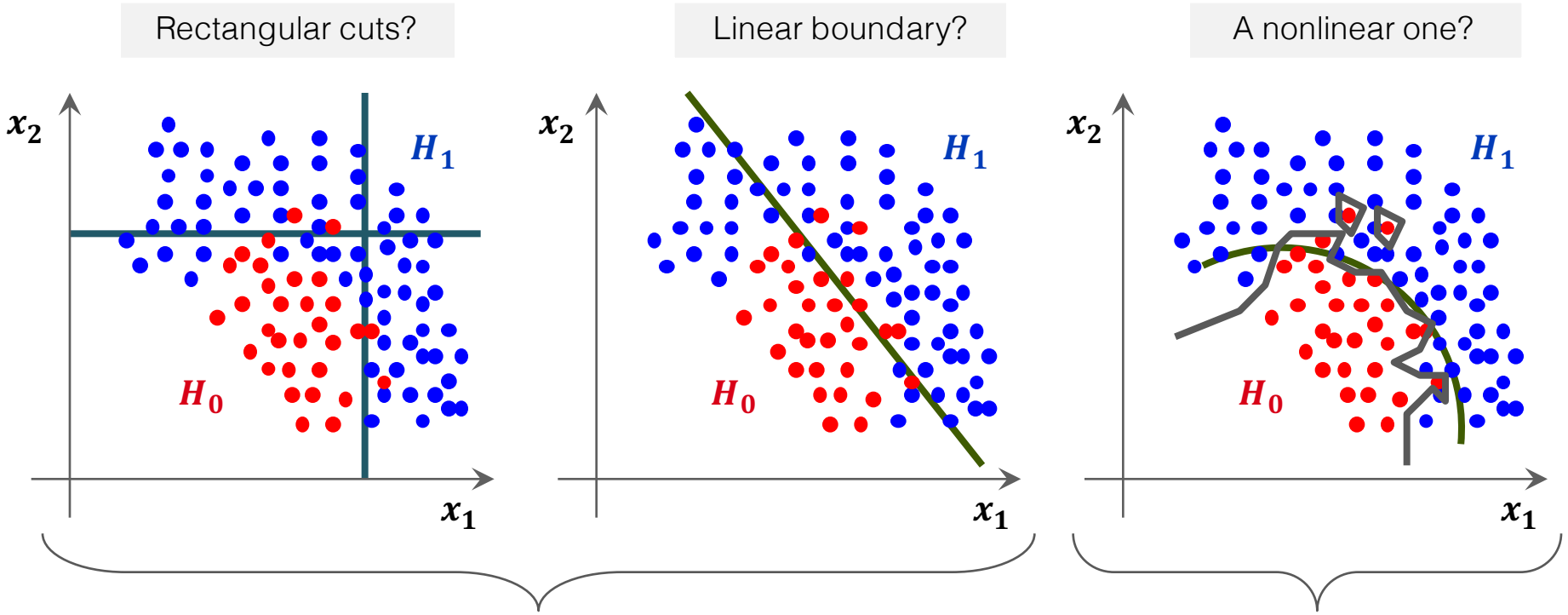
Multivariate techniques and machine learning



Event classification

Suppose data sample with two types of events: H_0, H_1

- We have found discriminating input variables x_1, x_2, …
- What decision boundary should we use to select events of type H_1?

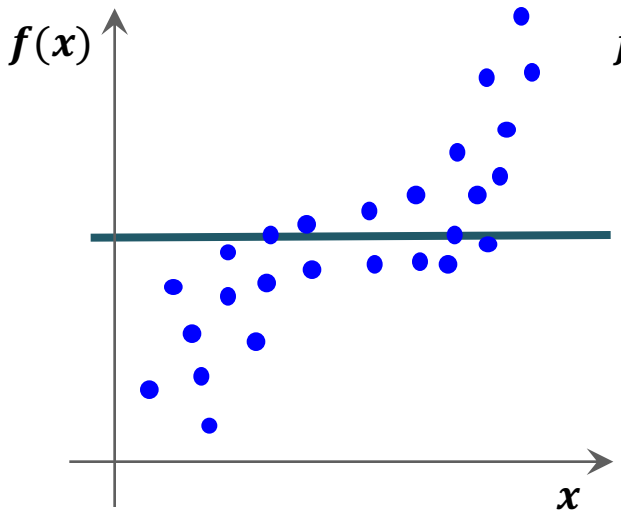


Parameter regression

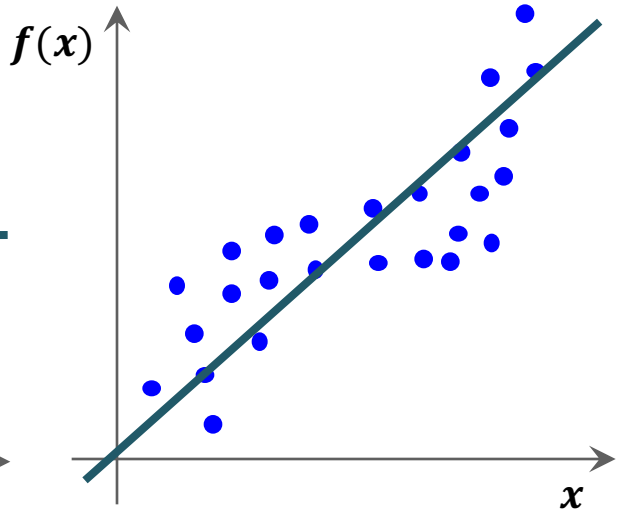
How to estimate a *functional behaviour* from a set of measurements? HEP examples:

- Energy deposit in a the calorimeter, distance between overlapping photons, …
- Entry location of a particle in the calorimeter or on a silicon pad, …

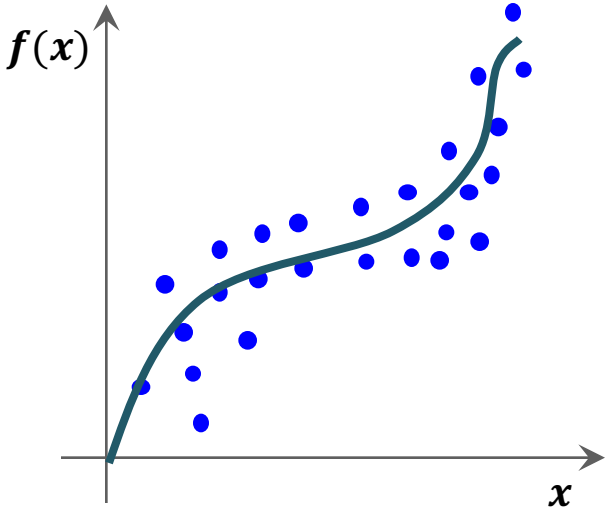
Constant ?



Linear function ?



A non-linear one ?



Looks trivial? What if we have many input variables?

Note: the goal is *not to fit* given data, but to learn $\boldsymbol{f}(\boldsymbol{x})$ vs. $\boldsymbol{x}$ to *predict* ***target*** $\boldsymbol{f}(\boldsymbol{x})$ for new measurements $\boldsymbol{x}$

These are the simplest applications of statistical machine learning (ML). Most particle physics utilisations so far fall into this category

However, there is no limit of use cases for complex machine learning…

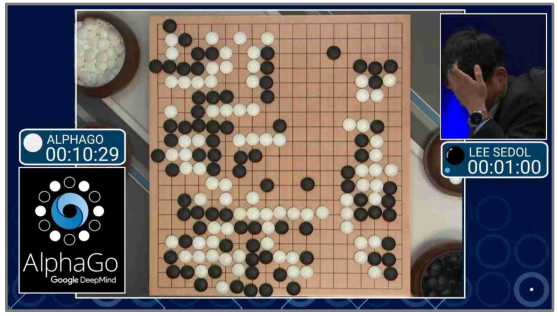
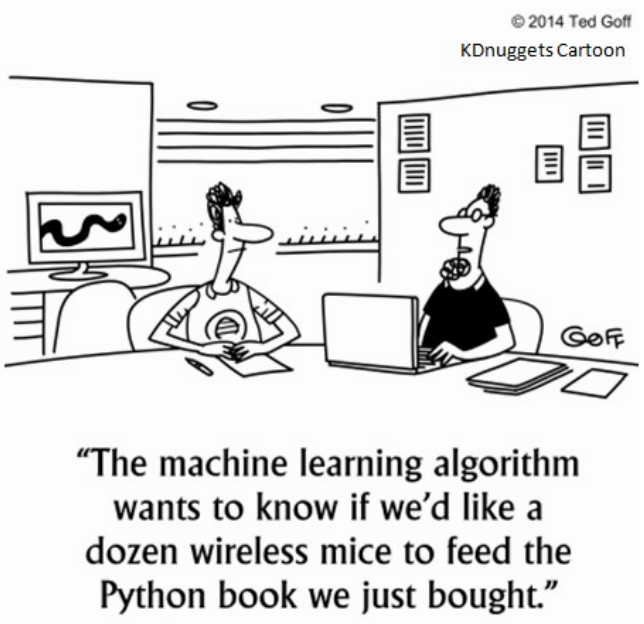


Image from: https://gogameguru.com/alphago-shows-true-strength-3rd-victory-lee-sedol

…as long as the ML algorithms are smart and efficient enough, there is sufficient computing power, and a complete set of training data

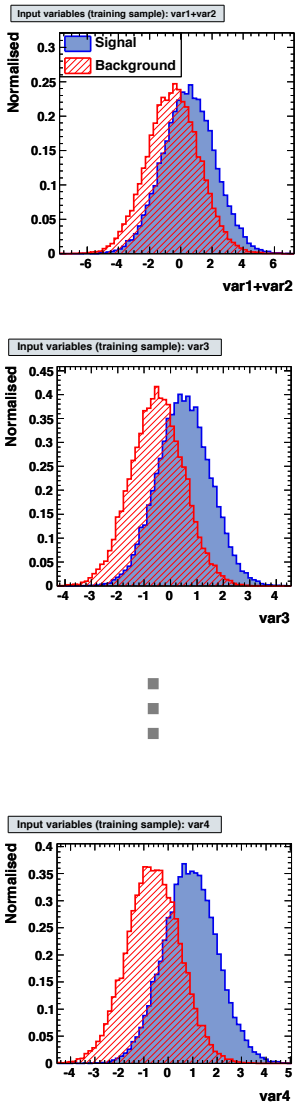
Also in particle physics we can apply ML to more complex problems. Among these: track reconstruction, calibration, tuning

Not so long ago, real-life artificial intelligence used to be like this:



Such things still happen, but the improvements have nevertheless been astounding

$R^D \rightarrow R$ classification



R^D

feature space

$y(x)$ → R

Each event, Signal or Background, has $\boldsymbol{D}$ measured variables

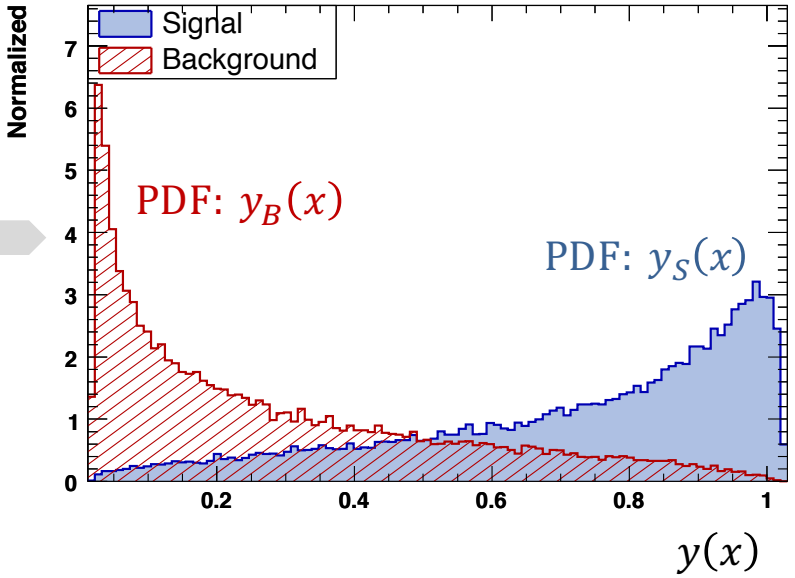
→ Find function that maximises the separation of the two classes

Most general form:

$y = y(x); x \in R^D$

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

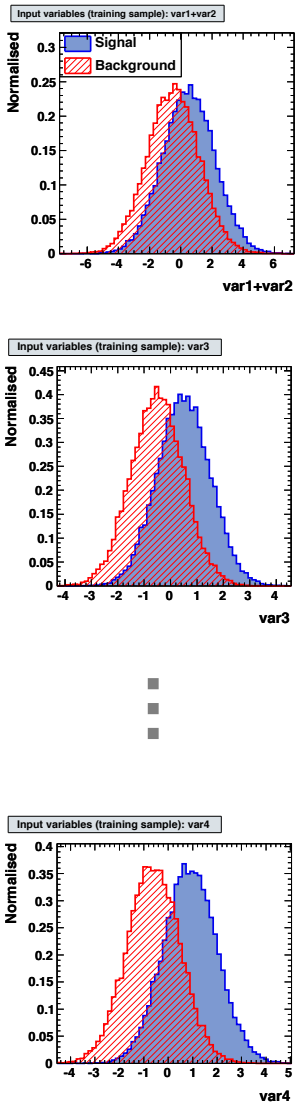
Plotting the resulting $y(x)$ is test statistic:



Separation power:

$$\langle S^2 \rangle = \frac{1}{2} \int \frac{(y_S - y_B)^2}{y_S + y_B} dy$$

$R^D \rightarrow R$ classification



R^D

feature space

$y(x) \rightarrow R$

Each event, Signal or Background, has $\boldsymbol{D}$ measured variables

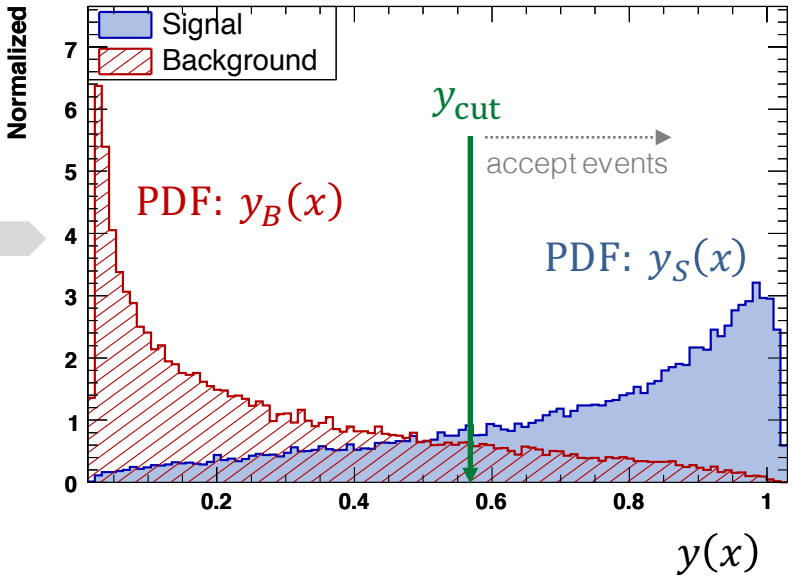
→ Find function that maximises the separation of the two classes

Most general form:

$y = y(x); x \in R^D$

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

Plotting the resulting $y(x)$ is test statistic:



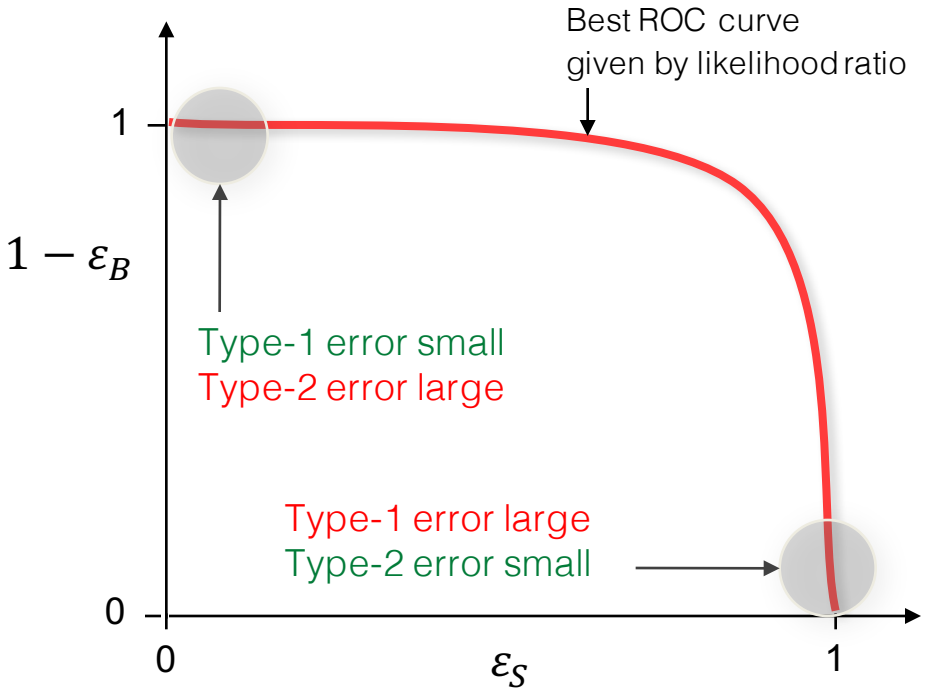
Separation power:

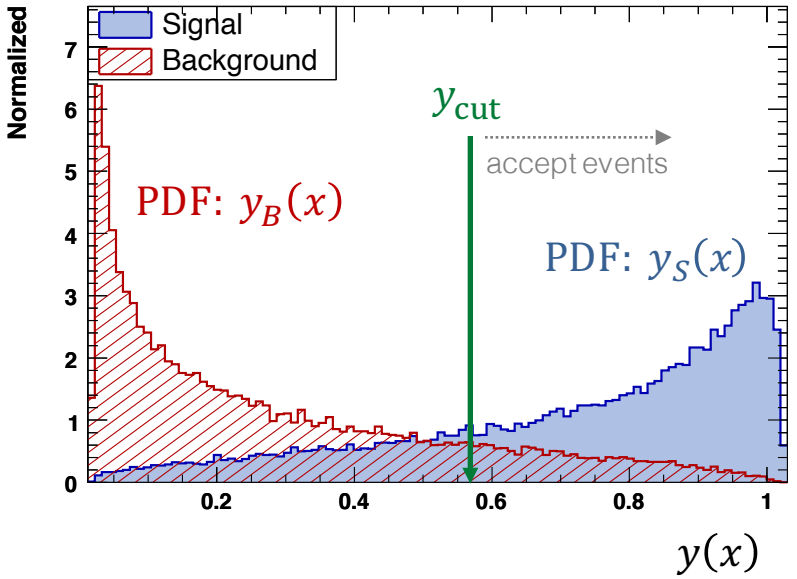
$$\langle S^2 \rangle = \frac{1}{2} \int \frac{(y_S - y_B)^2}{y_S + y_B} dy$$

$R^D \rightarrow R$ classification

Remember the **ROC***curve ? [**Receiver Operation Characteristic*]

- Varying cut value y_{cut} moves working point (efficiency and purity) along the ROC curve
- Best ROC curve given by likelihood ratio (Neyman-Pearson lemma)

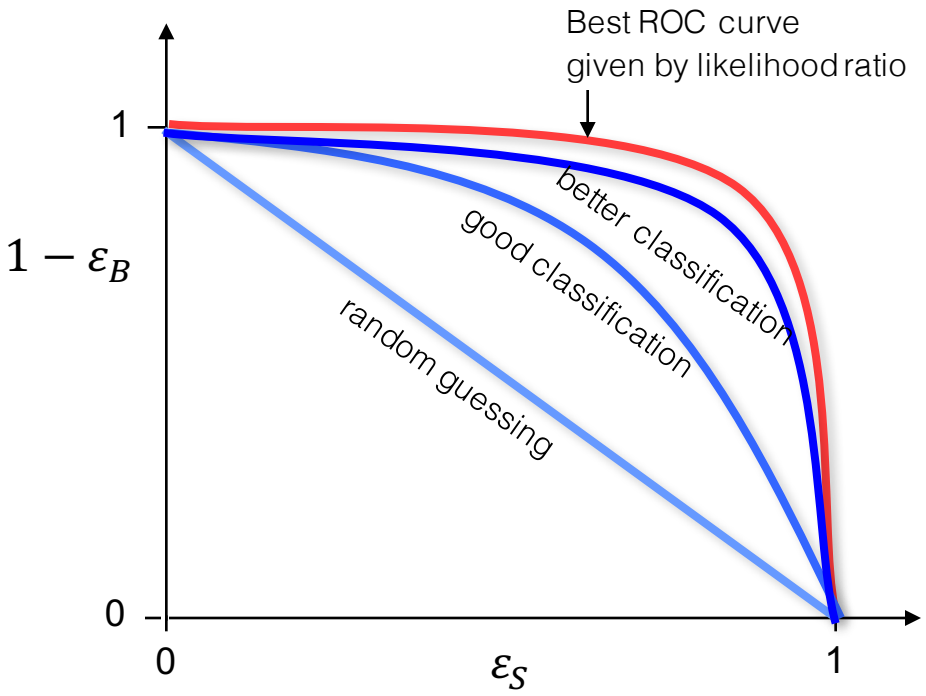


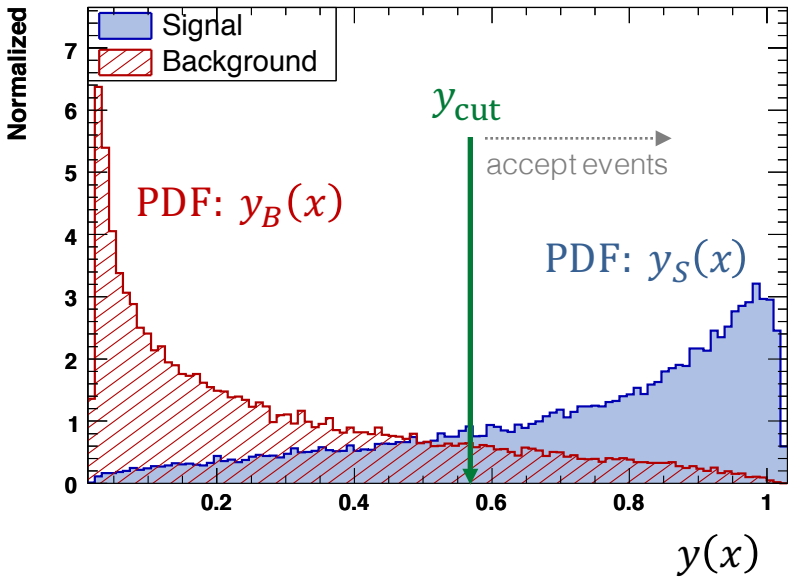


$R^D \rightarrow R$ classification

Remember the **ROC** curve ?

- Varying cut value y_{cut} moves working point (efficiency and purity) along the ROC curve
- Best ROC curve given by likelihood ratio (Neyman-Pearson lemma)





$R^D \rightarrow R$ classification

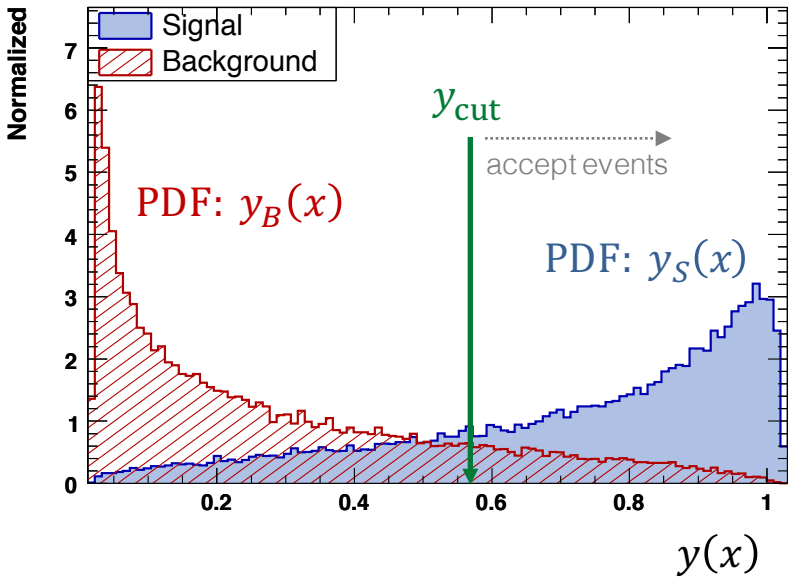
Remember the **ROC** curve ?

- Varying cut value y_{cut} moves working point (efficiency and purity) along the ROC curve
- Best ROC curve given by likelihood ratio (Neyman-Pearson lemma)

Once the ROC curve is known, how to choose the optimal y_{cut} value? This depends on expected abundance of S and B

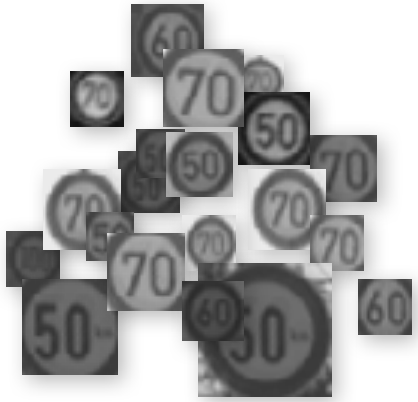
- For measurement of signal cross section: maximise $S/\sqrt{S+B}$
- Discovery of a signal: maximum of $S/\sqrt{B}$
- Precision measurement: high signal purity
- Trigger selection: high efficiency (ε_S) (sometimes high background rejection)

Note that in realistic cases systematic uncertainties also need to be considered in optimisation procedure!

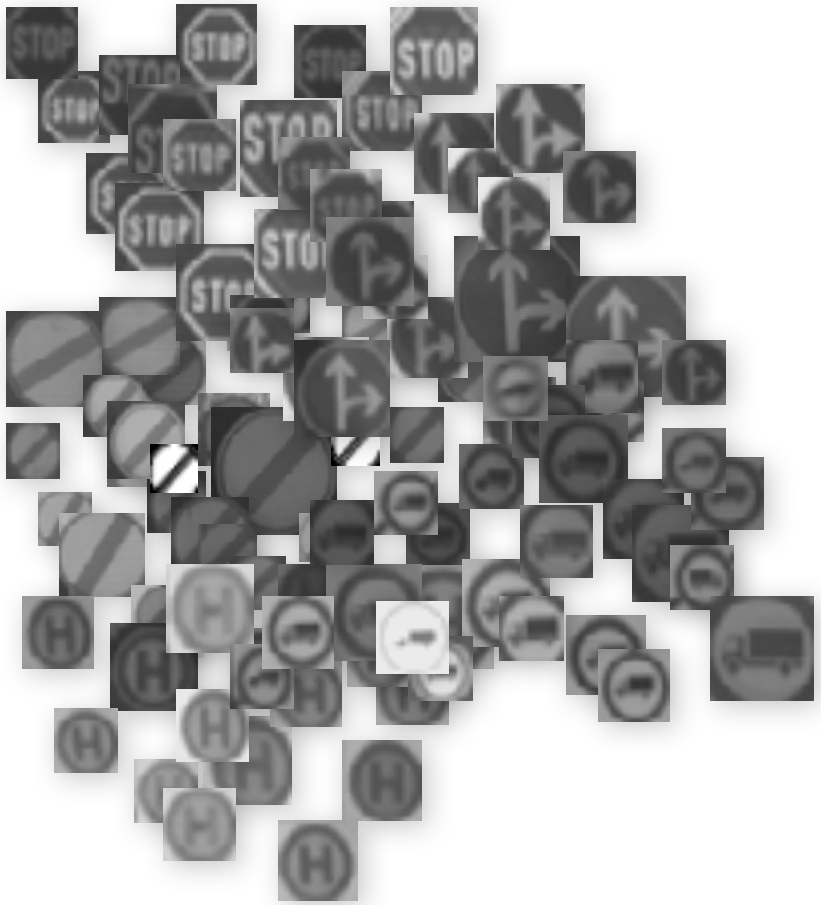


$R^D \rightarrow R^m$ *multi-class* classification

Signal

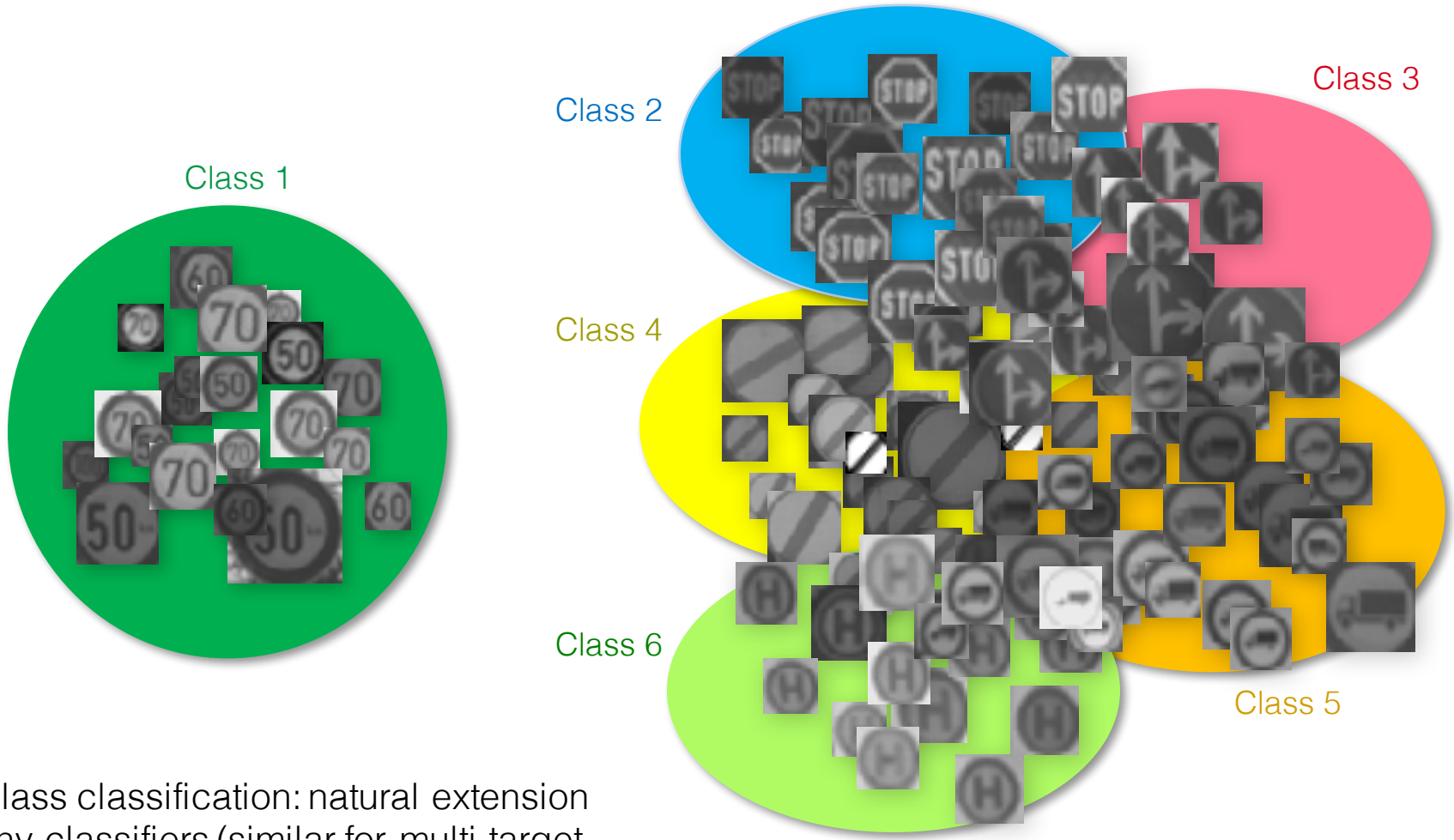


Background



Binary classification: two classes, signal versus background

$R^D \rightarrow R^m$ *multi-class* classification



Multi-class classification: natural extension for many classifiers (similar for multi-target regression)

Realistic event classification / parameter regression

Unfortunately, the true probability densities functions are typically unknown: Neyman-Pearson's lemma doesn't really help us…

Use MC simulation, or more generally a set of known "events" as a **training sample** for the classification / regression problem

- Try to estimate the functional form of the PDFs from which the classification likelihood ratio or regression target can be obtained
  - e.g. *D*-dimensional histogram, Kernel densitiy estimators, MC-based matrix-element methods, …
- Find a discrimination function $y(x)$ and corresponding decision boundary (i.e. a hyperplane in the feature space: $y(x) = \text{const}$) for optimal separation or optimal target fitting
  - e.g. Linear Discriminator, Neural Networks, Boosted Decision, …

→ Supervised machine learning

No magic here, 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 (avoid overtraining), consider trade off between statistical and systematic uncertainties

Machine learning categories

Supervised learning: training with "events" for which outcome ("signal", "background", "regression target", …) is known, eg, from Monte Carlo simulation

Unsupervised learning: no prior knowledge about specific event classes or targets. One could then try, for example, in the given dataset to perform a

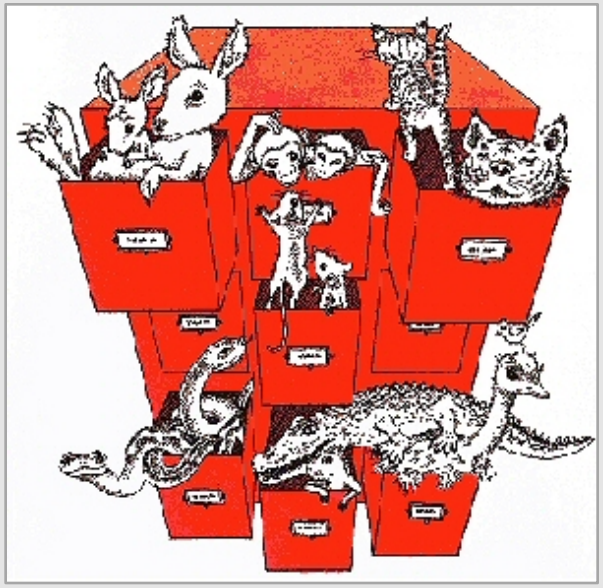
- cluster analysis: if different "groups" are found → class labels
- principal component analysis (PCA): find basis in observable space with biggest hierarchical differences in the variance → infer underlying data structure
- Example: correlation analysis for medical survey: group people and perhaps find common causes for certain diseases (similar for market surveys)

Reinforcement-learning: learn from "success" or "failure" of some "action policy" (example: a robot achieves falls / does not fall, or wins / looses a game, …)

So far, most applications in particle physics (HEP) use supervised learning owing to the good theoretical understanding and power of Monte Carlo simulation → *only form of machine learning discussed in this lecture*

Classification techniques

Discuss also regression where applies

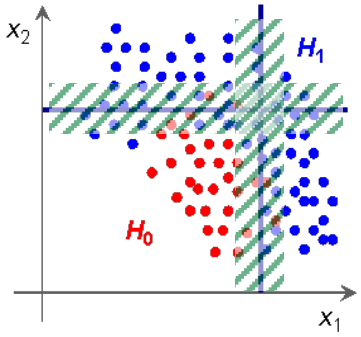


Projective likelihood estimator

Much liked in HEP: normalised one-dimensional probability density estimators for each input variable combined by multiplying the marginal PDFs of all input variables

$$y_{\text{proj-Li}} = \frac{\prod_k p_k^{\text{Signal}}(x_k)}{\prod_k p_k^{\text{Signal}}(x_k) + \prod_k p_k^{\text{Background}}(x_k)}$$

where the products are over all $k = 1 \ldots N_{\text{var}}$ input variables



PDE introduces fuzziness in feature space separation

Ignores correlations between input variables

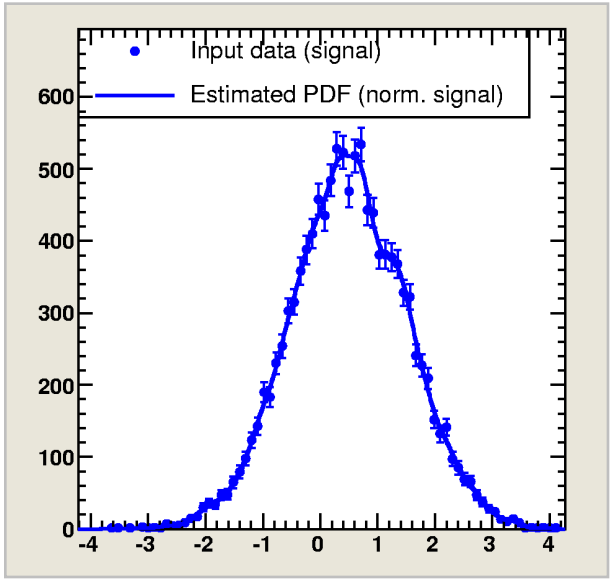
- Optimal approach if correlations are zero
  (or can be eliminated by variable redefinition)
- Otherwise (most realistic cases): significant performance loss

Projective likelihood estimator

Technical challenge: how to estimate the PDF shapes of the input variables to build the projective likelihood estimator ?

May use:

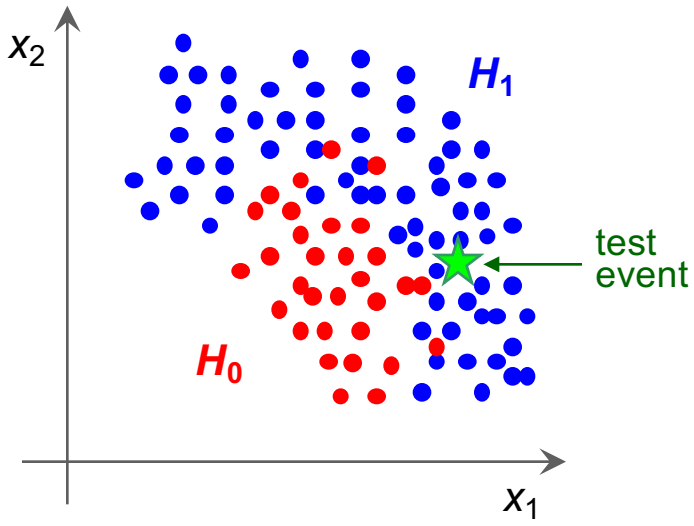
- parametric fitting by function
  → Difficult to automate for arbitrary PDFs
- nonparametric fitting
  → Easy to automate, can create artefacts
- event counting / histogram
  → Automatic, unbiased, but sub-optimal (fluctuations)



Multidimensional likelihood estimator

Overcome limitations of projective method by mapping full feature space: single PDF per event class (eg, signal, background) which spans D dimensions

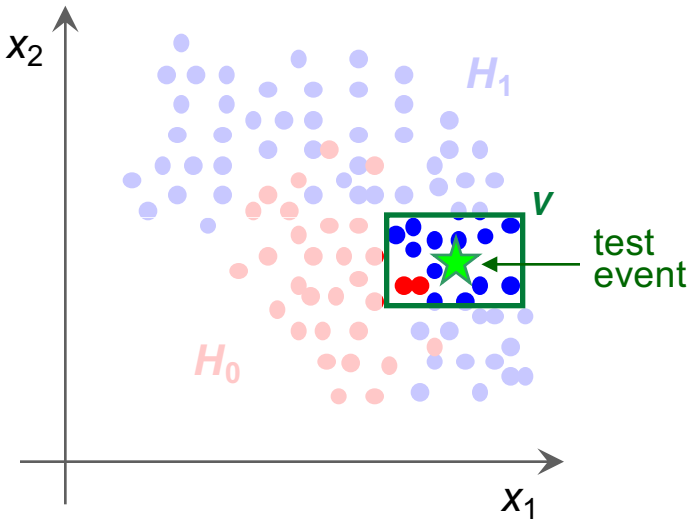
Estimate density of signal / background events using training data sample in "vicinity" of to be classified ("test") event



Multidimensional likelihood estimator

Overcome limitations of projective method by mapping full feature space: single PDF per event class (eg, signal, background) which spans D dimensions

Estimate density of signal / background events using training data sample in "vicinity" of to be classified ("test") event



For example: count number of signal and background events in rectangular volume (V) around test event

Volume size can be adaptive to allow for varying training point density

- Improved estimate within V by using D-dimensional kernel estimators (eg, Gauss)
- Enhance speed of event counting in volume with use of sorted binary tree search
- Regression is similar: take average training target values in vicinity of test event

Multidimensional likelihood estimator

Overcome limitations of projective method by mapping full feature space: single PDF per event class (eg, signal, background), which spans D dimensions

k-Nearest Neighbor Method

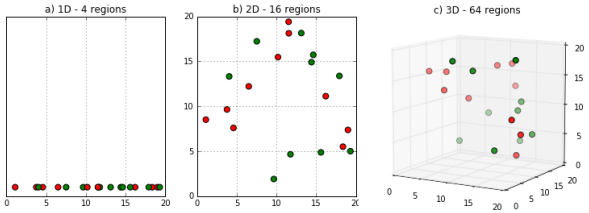
Better than searching within a volume (fixed or floating), count adjacent reference events till statistically significant number reached

- Method intrinsically adaptive
- Very fast search with kd-tree event sorting

- Improved estimate within V by using D-dimensional kernel estimators (eg, Gauss)
- Enhance speed of event counting in volume with use of sorted binary tree search
- Regression is similar: take average training target values in vicinity of test event

Curse of dimensionality

Filling a D-dimensional histogram to get a mapping of the PDF is typically unfeasible due to lack of training events

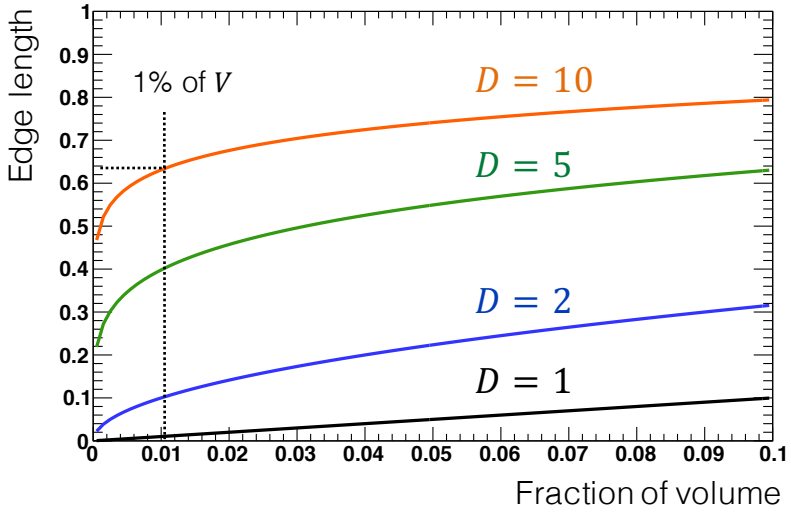


Shortcoming of nearest-neighbour methods:

In higher dimensional cases the idea of looking at "training events" in a reasonably local "vicinity" of the space point to be tested becomes difficult

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

- For a cube of a particular fraction of the volume: $\mathbf{Edge\ length} = (\mathbf{Fraction\ of\ volume})^{1/D}$
- In 10 dimensions: to capture 1% of the volume, 63% of range in each variable needed → that's not "local" anymore !

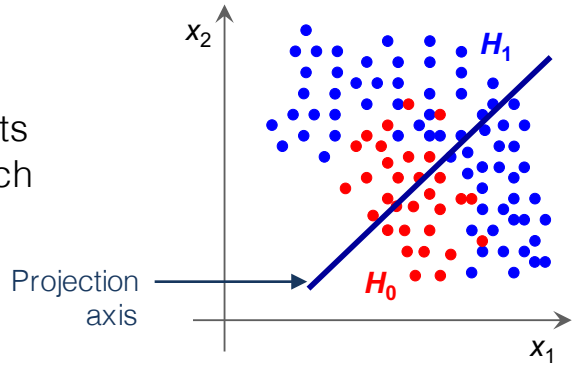


→ Need techniques with better curse of dimensionality

Fisher's linear discriminant analysis (LDA)

Simple and elegant classifier

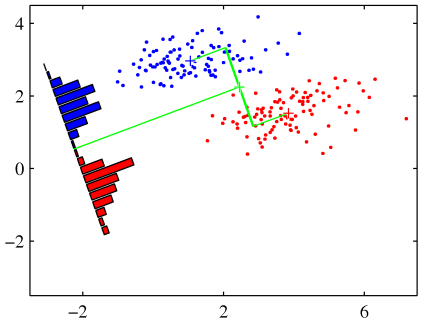
- Determine axis in the feature space such that a projection of events onto this axis pushes signal and background as far away from each other as possible, while confining events of same class in close vicinity to each other



$$y_{\text{Fisher}} = F_0 + \sum_{k=\text{variables}} x_k \cdot F_k$$

F_k = Fisher coefficients

Fisher projection



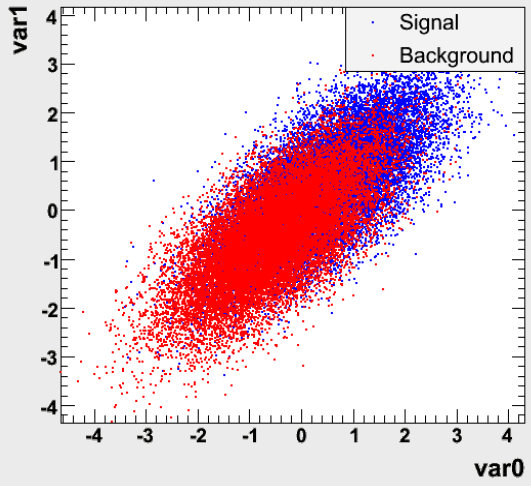
- Fisher coefficients are computed using the signal and background covariance matrices
- Distinct sample means between signal and background are required
- Optimal classifier for linearly correlated Gaussian-distributed variables

Fisher's linear discriminant analysis

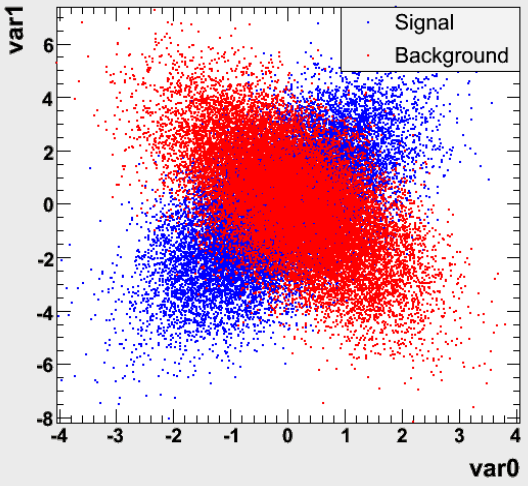
By definition, a linear discriminant can only solve linear problems. Most real-life problems have however some degree of non-linearity

Consider the following two-variable toy examples:

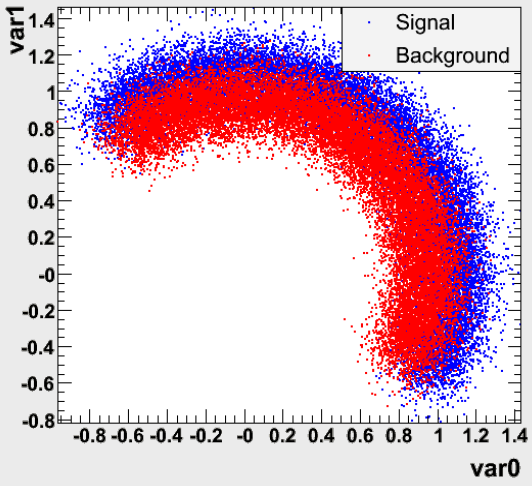
Linear correlations
(same for signal and background)



Cross-linear correlations
(opposite for signal and background)



Circular correlations
(same for signal and background)



Fisher's linear discriminant analysis

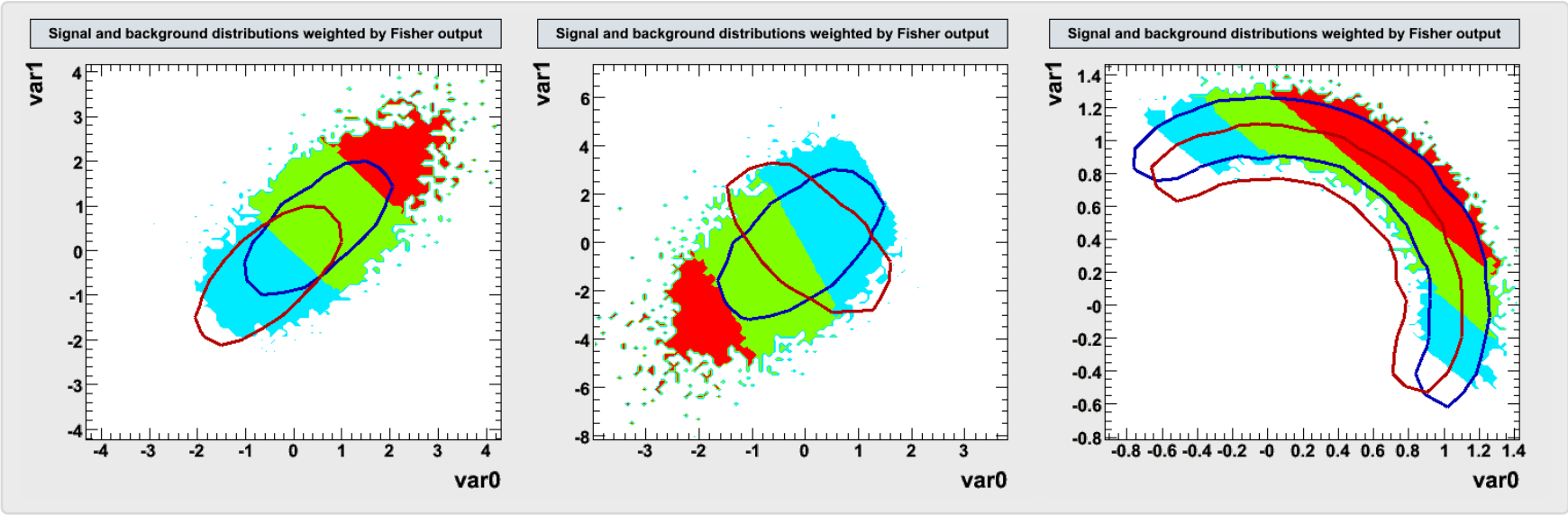
By definition, a linear discriminant can only solve linear problems. Most real-life problems have however some degree of non-linearity

The events are weighted by the *signal-likeness* of the classifier output (red = most signal-line)

Linear correlations
(same for signal and background)

Cross-linear correlations
(opposite for signal and background)

Circular correlations
(same for signal and background)



Optimal performance
(in Neyman-Pearson sense)

Almost zero separation

Poor performance

Non-linear analysis: Artificial Neural Network (NN)

Modification of Fisher discriminant to form arbitrary non-linear decision boundaries

$$y(x) = \text{sigmoid}\left(\sum_{i=1}^{N} w_i h_i(x)\right)$$

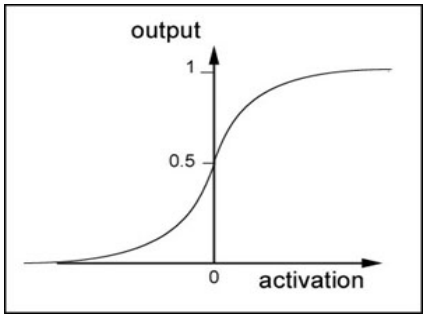
$y(x)$ built from set of "basis" functions $h_i(x)$

$h_i(x)$is sufficiently general (i.e. non linear)

→ can model any function (mathematically proven)

Now we form the following:

$$y(x) = A\left(\sum_{i=1}^{N} w_i \overbrace{A\left(w_{i0} + \sum_{j=1}^{D} w_{ij} x_j\right)}^{h_i}\right)$$



$$A = \frac{1}{1 - e^{-x}}$$

sigmoid "activation" function (other options: tanh, …)

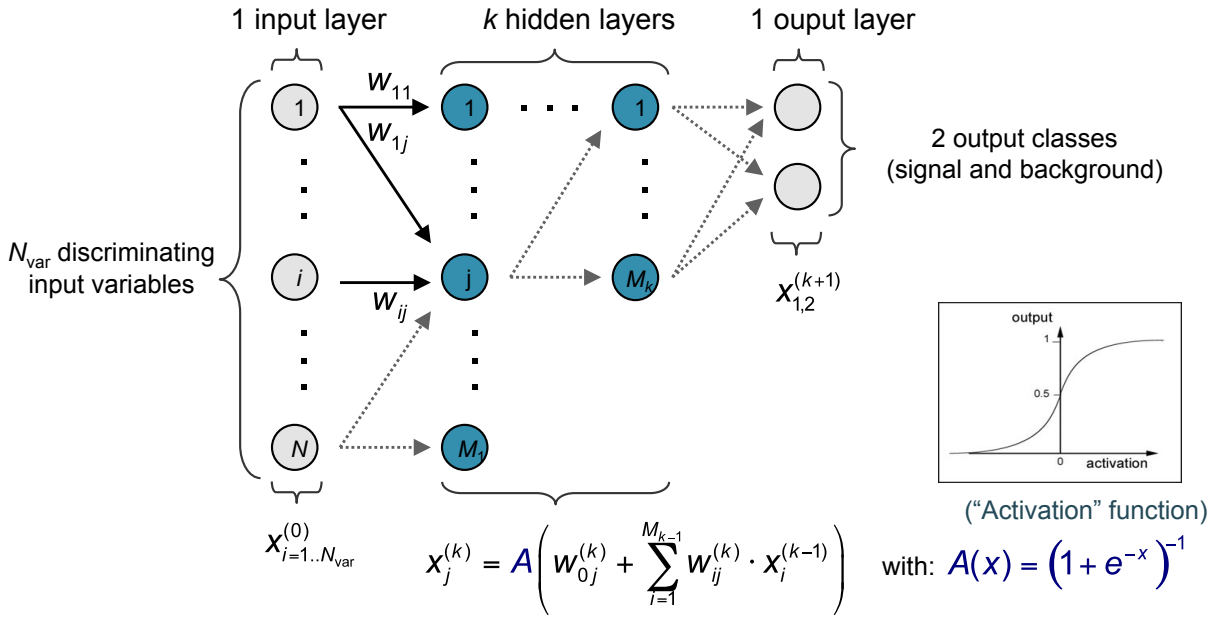
$y(x)$ is:

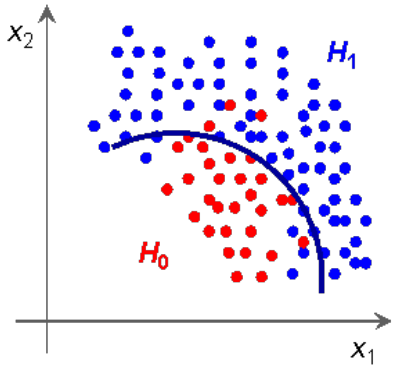
a non linear (sigmoid) function of
a linear combination of
non linear function(s) of
linear combination(s) of
the input data

Ready is the Neural Network. We "only" need to find the appropriate "weights" w_{ij}

Non-linear analysis: Artificial Neural Network

Architecture of feed-forward multilayer perceptron (MLP):





- Nodes in hidden layer represent the “activation functions” whose arguments are linear combinations of input variables → non-linear response to the input
- Output is a linear combination of outputs from 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 network training

NN training (= fit of weights w_{kj}) → minimisation of *loss function* $\boldsymbol{L(w)}$:

Regression :

$$L(w) = \sum_{k=1}^{N_{\text{events}}} L_k(x_k; w) = \frac{1}{2} \sum_{k=1}^{N_{\text{events}}} \left(y_k^{\text{train}} - y(x_k; w) \right)^2$$

y_k^{train}: true value (training data)

$y(x_k; w)$: predicted value (NN output)

Classification: (here *binomial loss* due to 1 vs 0 problem)

$$L(w) = \sum_{k=1}^{N_{\text{events}}} L_k(x_k; w) = - \sum_{k=1}^{N_{\text{events}}} \left(y_k^{\text{train}} \cdot \ln\left(y(x_k; w)\right) + \left(1 - y_k^{\text{train}}\right) \cdot \ln\left(1 - y(x_k; w)\right) \right)$$

where: $y_k^{\text{train}} = \begin{cases} 1, & \text{if } k \text{ signal} \\ 0, & \text{if } k \text{ background} \end{cases}$

Weight fitting, eg, via steepest gradient descent: $w_{ij} \rightarrow w_{ij} - \eta \sum_{k=1}^{N_{\text{events}}} \left(\frac{\partial L_k(x_k; w)}{\partial w_{ij}} \right)$

Neural network training

Weight fitting is *the* art of NN: $y(x)$ and $L_k(x_k; w)$ are highly non-parabolic functions, with narrow valleys and numerous local minima

- Methods to accelerate descent when gradient direction unchanged
- Local minima are mostly not a problem in large networks (finding global minimum represents overtraining)
- Bad critical points are often saddle points

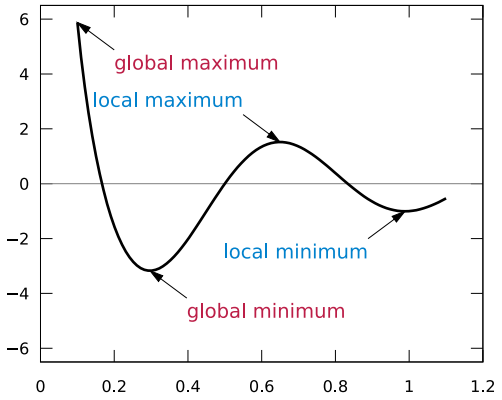


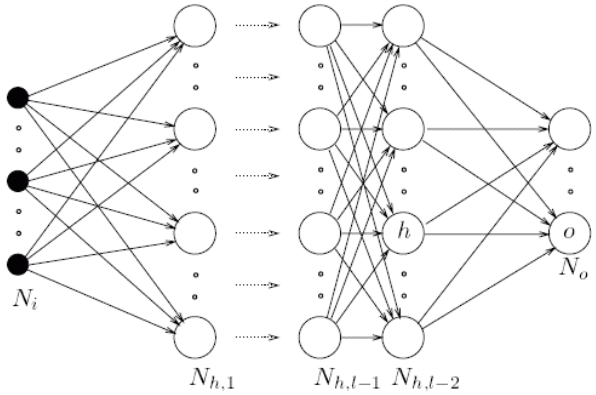
Figure from: https://en.wikipedia.org/wiki/Maxima_and_minima

NNs with many hidden layers used to be impossible to train due to vanishing gradient problem $\partial L_k(x_k; w)/\partial w_{ij} \approx 0$ for all but last layers. Enormous recent progress:

- Layer-wise pre-training using *auto-encoders* (pre-training) or *restricted-Boltzman machines* (RBM)
- Activation functions whose gradient do not vanish
- Smarter random weight initialisation
- Stochastic gradient decent with 'momentum'
- Weight regularisation to reduce network complexity (eg, weight decay, dropout)

Deep neural networks

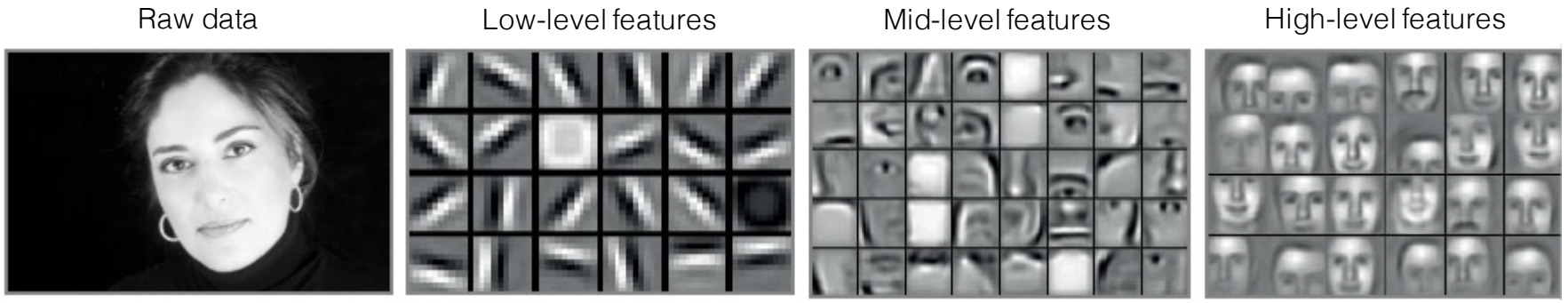
Deep neural network = Artificial neural network with many hidden layers



That's all it means, but it has important consequences

Deep & convolutional neural networks

Many hidden layers (& usage of convolution kernels) allows NN to learn hierarchy of features



https://developer.nvidia.com/deep-learning-course

Getting rid of “hand-crafted features”, revolutionised:

- Image recognition, speech recognition, natural language processing
- Complex automation (eg, self-driving car)

And particle physics ?

- Track fitting (helix pattern recognition)? Event reconstruction from 4-vectors?
- While most high-level MVAs applications are yet simple, there surely are interesting applications for deep NNs

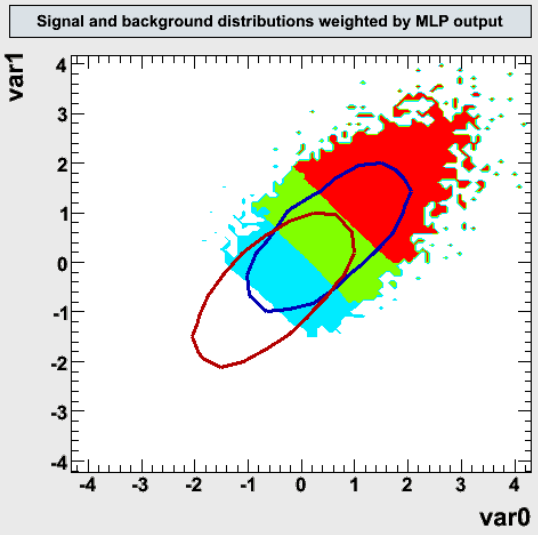
Non-linear analysis: Artificial Neural Network

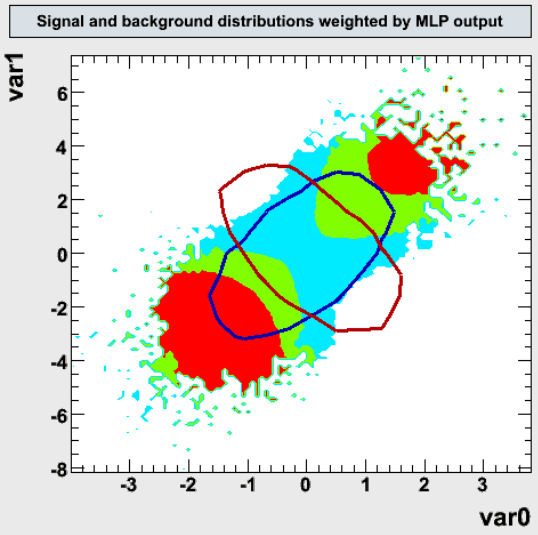
Let's see how well a simple NN deals with the previous two-variable toy examples:

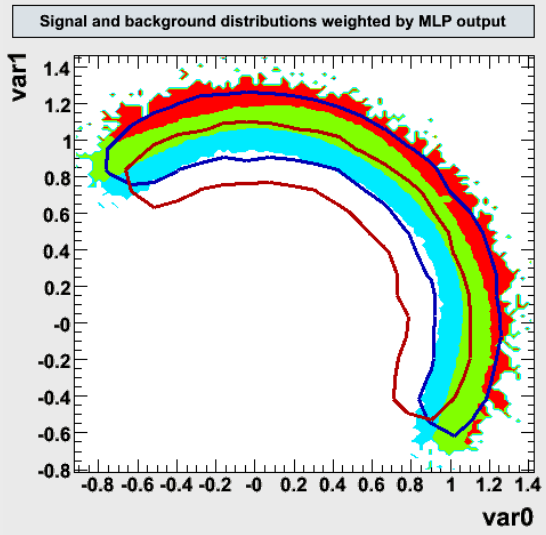
Linear correlations
(same for signal and background)

Cross-linear correlations
(opposite for signal and background)

Circular correlations
(same for signal and background)





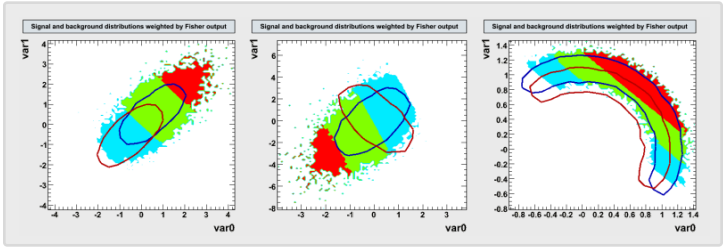


Optimal performance
(in Neyman-Pearson sense)

Optimal performance

Optimal performance

Recall, linear Fisher discriminant case:

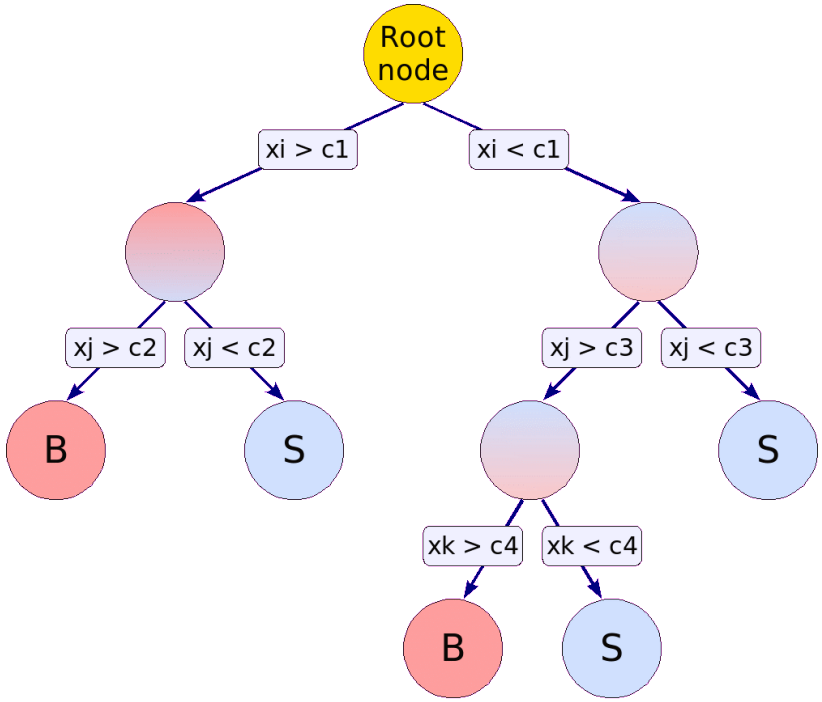


Boosted decision trees (BDT)

Decision tree: sequential application of cuts splits the data into nodes, where the final nodes (leafs) classify an event as signal or background by majority vote

Growing a tree:

- Start with Root node
- Split training sample according to cut on best variable at this node
- Splitting criterion: e.g., maximum "Gini-index": $\mathbf{purity} \cdot (1 - \mathbf{purity})$
- Continue splitting until min. number of events or max. purity reached
- Classify leaf node according to majority of events, or give weight; unknown test events are classified accordingly

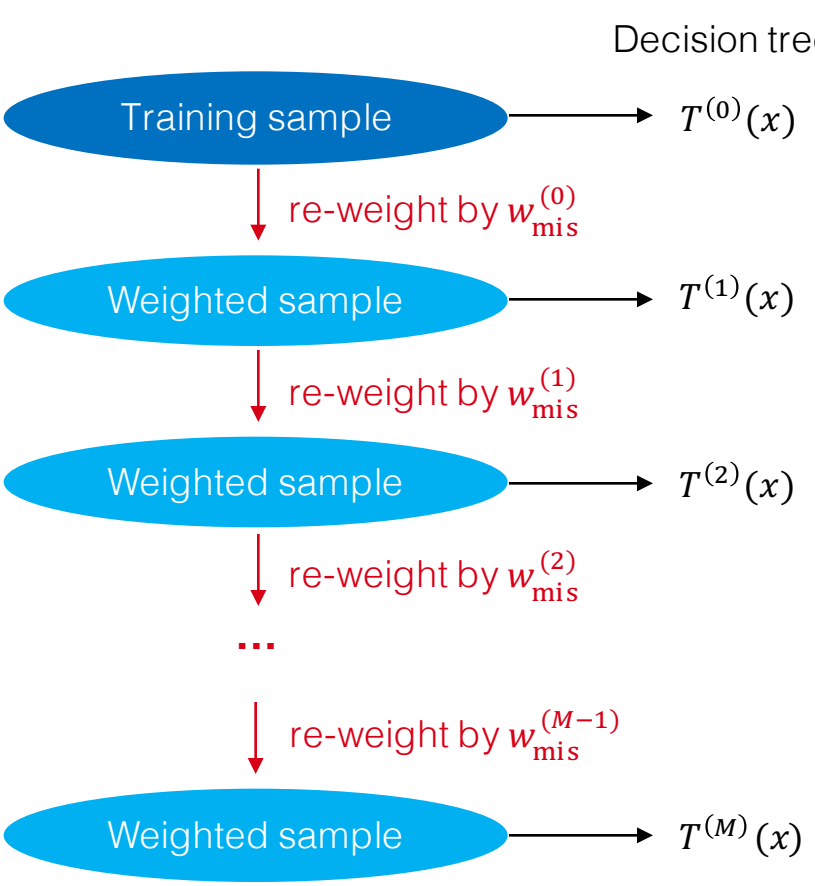


Shortcoming: instability, sensitivity to overtraining

Boosted decision trees (1996): combine many decision trees in *forest*, with differently weighted events in each tree (trees themselves can also be weighted)

(Adaptive) Boosting

Idea: emphasise different features in data sample (eg, hard to classify events)



- **AdaBoost** re-weights events mis-classified by previous classifier by:

$$w_{\text{mis}}^{(i)} = \frac{1 - f_{\text{mis}}^{(i)}}{f_{\text{mis}}^{(i)}}$$

with $f_{\text{mis}} = \dfrac{\text{No. of misclassified events}}{\text{No. of all events}}$

- Final BDT obtained from (weighted) sum over all decision trees:

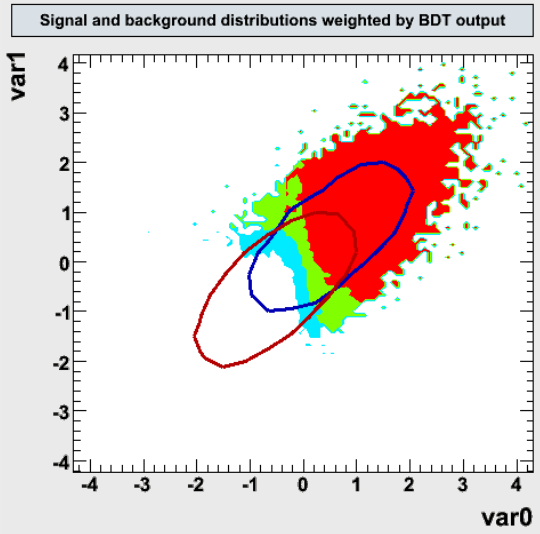
$$y(x) = \sum_{i=1}^{M} \ln(w_{\text{mis}}) \cdot T^{(i)}(x)$$

- Different boosting algorithms: bagging (bootstrap), randomised trees, gradient boost, …. mostly similar performance

Boosted decision trees (BDT)

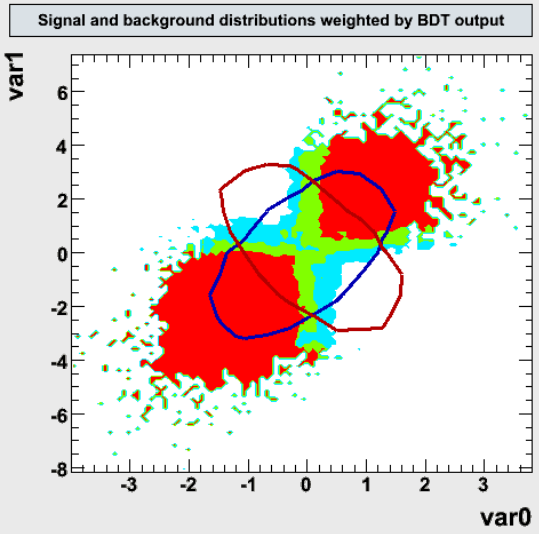
Let's see how well a BDT deals with the previous two-variable toy examples:

Linear correlations
(same for signal and background)



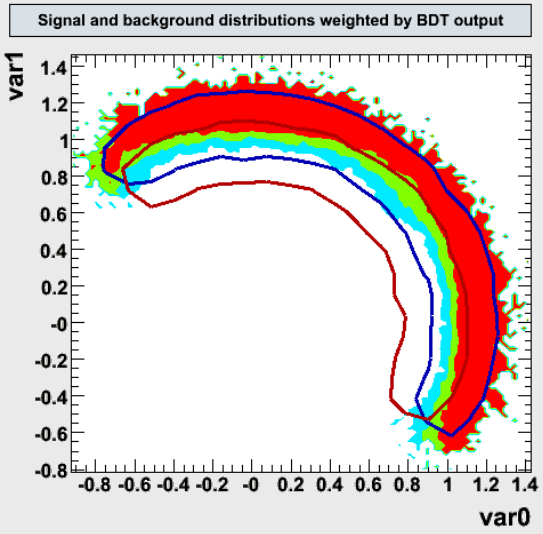
Good performance

Cross-linear correlations
(opposite for signal and background)



Good performance

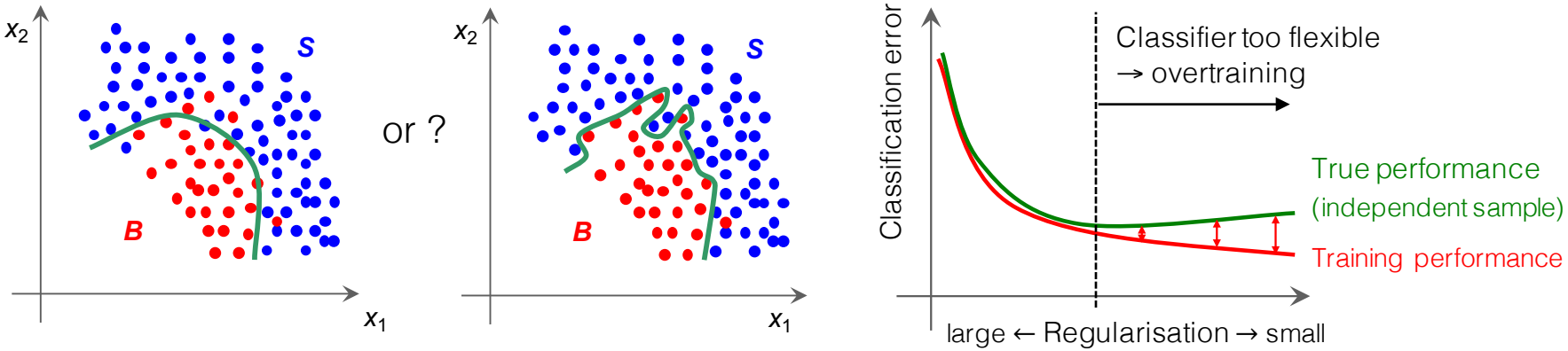
Circular correlations
(same for signal and background)



Good performance

Overtraining

Algorithms with many tuning parameters (large flexibility) can be subject to overtraining, ie, they tune to statistically insignificant information in the training sample



Overtraining produces bias if performance is estimated from training sample (which is not allowed!). If independent performance evaluation is guaranteed, "some" overtraining is not a problem as usually little performance loss

Remedies against overtraining

- Regularisation (pruning, smearing (eg, bagging), weight decay, …)
- Cross-validation (resample training data, see next slide)

Cross validation

Overtraining can be reduced (and algorithm performance improved) by increasing size of training data sample

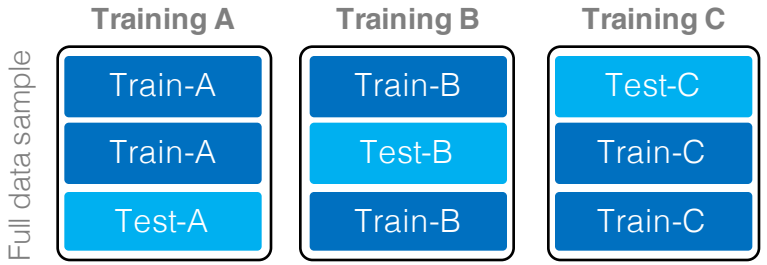
Also: division of dataset into independent "training", "test" and "validation" sample?
→ Not optimal!

Cross validation: divide full data sample into $\boldsymbol{n}$ independent subsamples, eg, $n = 3$

- Train algorithm by using all but the $\boldsymbol{i}$-th sample
- Use $\boldsymbol{i}$-th sample as "test"
- Iterate $\boldsymbol{i}$ through all $\boldsymbol{n}$ samples

→ Allows to use fraction of $(n-1)/n$ of available events for training, and all n for testing

Example for $n = 3$:



To "test" the performance, iterate over all events and use corresponding trainings and test samples

→ **all data used for testing**!

Categories

Multivariate training samples often have distinct sub-populations of data

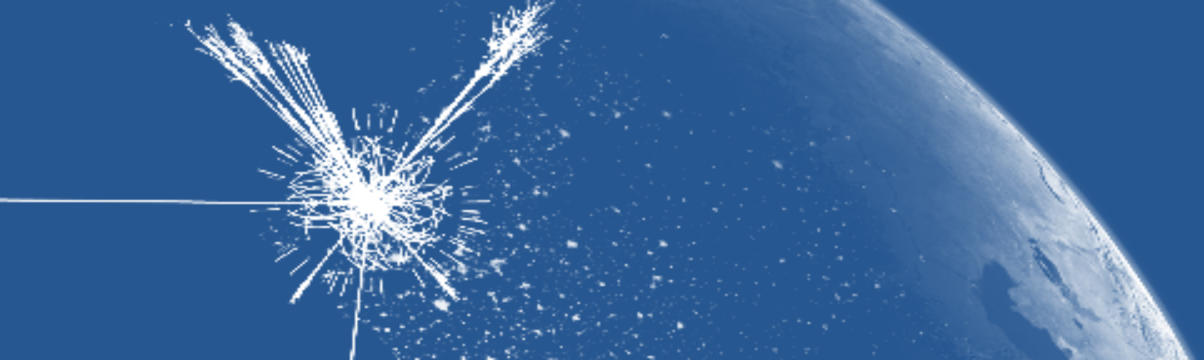
- A detector element may only exist in the barrel, but not in the endcaps
- A variable may have different properties in barrel, overlap, endcap regions

Ignoring this dependence generates correlations between variables that must be learned

- Algorithms such as the projective likelihood, which do not account for correlations, significantly loose performance if the sub-populations are not separated

Categorisation means splitting the data sample into categories defining disjoint ensembles with the (idealised) properties:

- Events belonging to the same category are statistically indistinguishable
- Events belonging to different categories have different properties
- A machine learning algorithm is trained in each category



Summary

Multivariate classification has a long tradition in particle physics to improve the detector performance and to increase the statistical sensitivity of data analyses

The use of multivariate regression in particle physics is relatively new, mainly for calibration purposes, and can certainly be further developed

Multivariate multi-class classification & regression have a large field of application in LHC analyses due to the widely used simultaneous control-region / signal-region fit approach

Boosted decision trees are a particularly popular multivariate method in particle physics due to their overall good performance, simple optimisation and robustness

Machine learning with deep neural networks are the future of the field due to their capability of hierarchical feature organisation allowing to attack complex problems