Machine Learning in HEP and Multivariate Techniques

- **Machine Learning and Multivariate analyses in HEP**
  - **- Extracted from slides by H. Voss at SOS 2016 and K. Reygers lectures at Heilderbeg Univ.**

What is Machine Learning

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

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

I suggest: forget about 'fancy definitions':

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

Multi-variate Classification

Consider events which can be either signal or background events.

Each event is characterized by *n* observables:

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

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

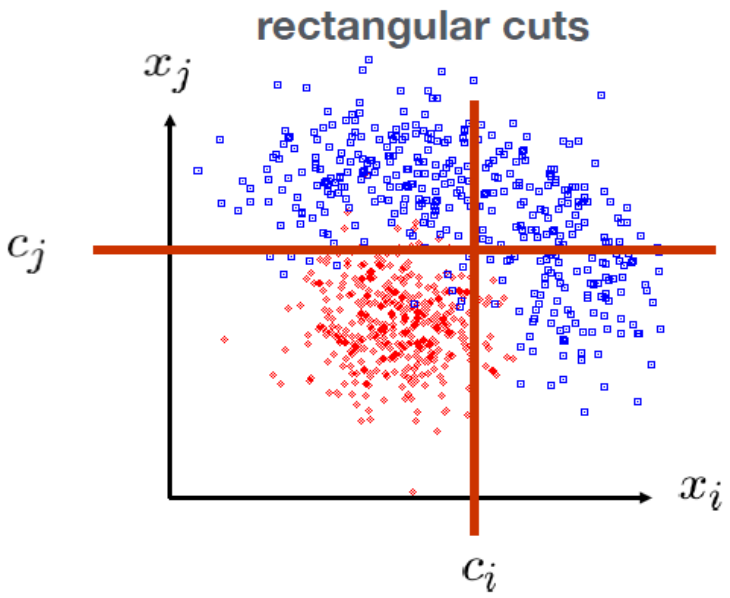
This is usually done by mapping the feature vector to a single variable, i.e., to scalar test statistic:

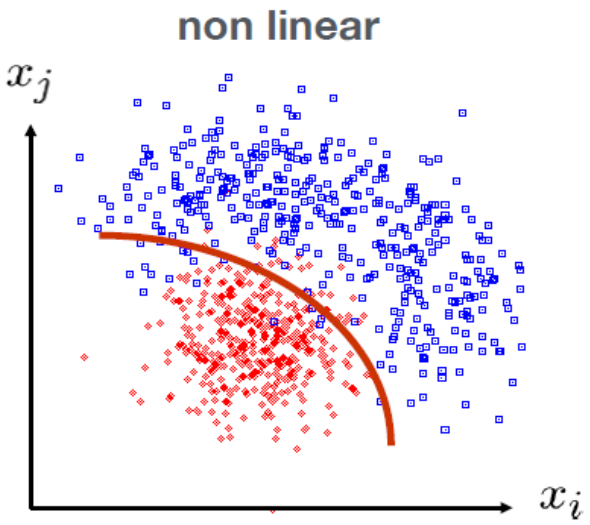
$$\mathbb{R}^n \to \mathbb{R}: \quad y(\vec{x})$$

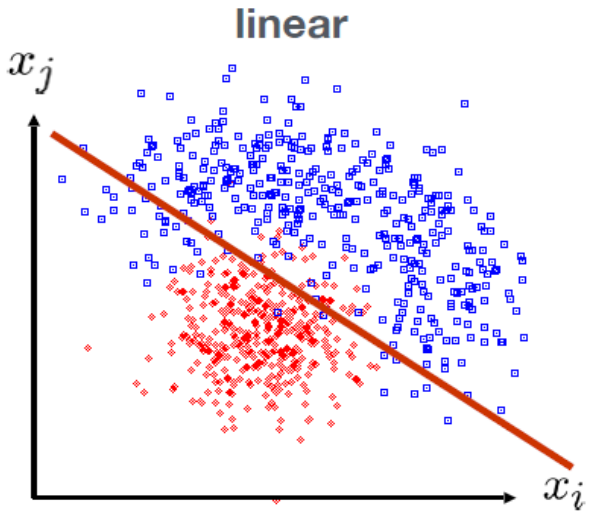
A cut $y > c$ to classify events as signal corresponds to selecting a potentially complicated hyper-surface in feature space. In general superior to classical "rectangular" cuts on the x_i.

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

Classification: Different Approaches



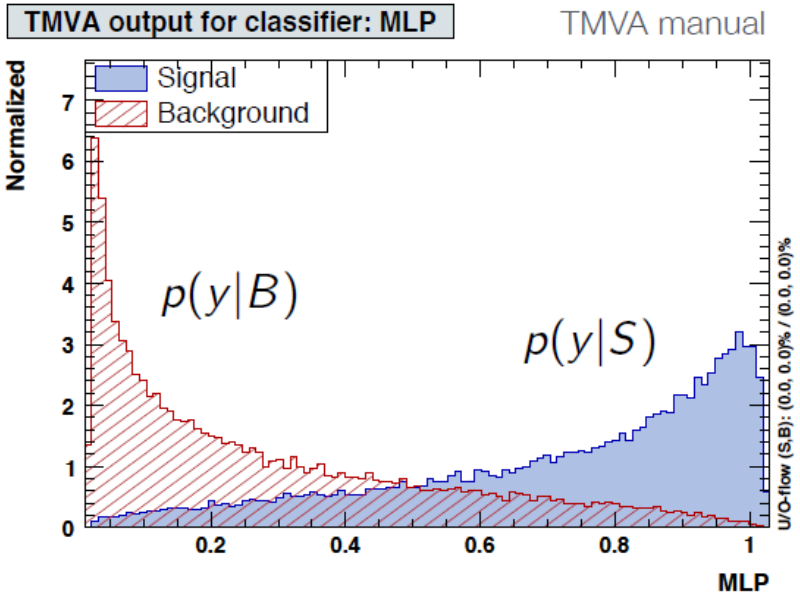




k-Nearest-Neighbor,
Boosted Decision Trees,
Multi-Layer Perceptrons,
Support Vector Machines
...

Signal Probability Instead of Hard Decision

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:

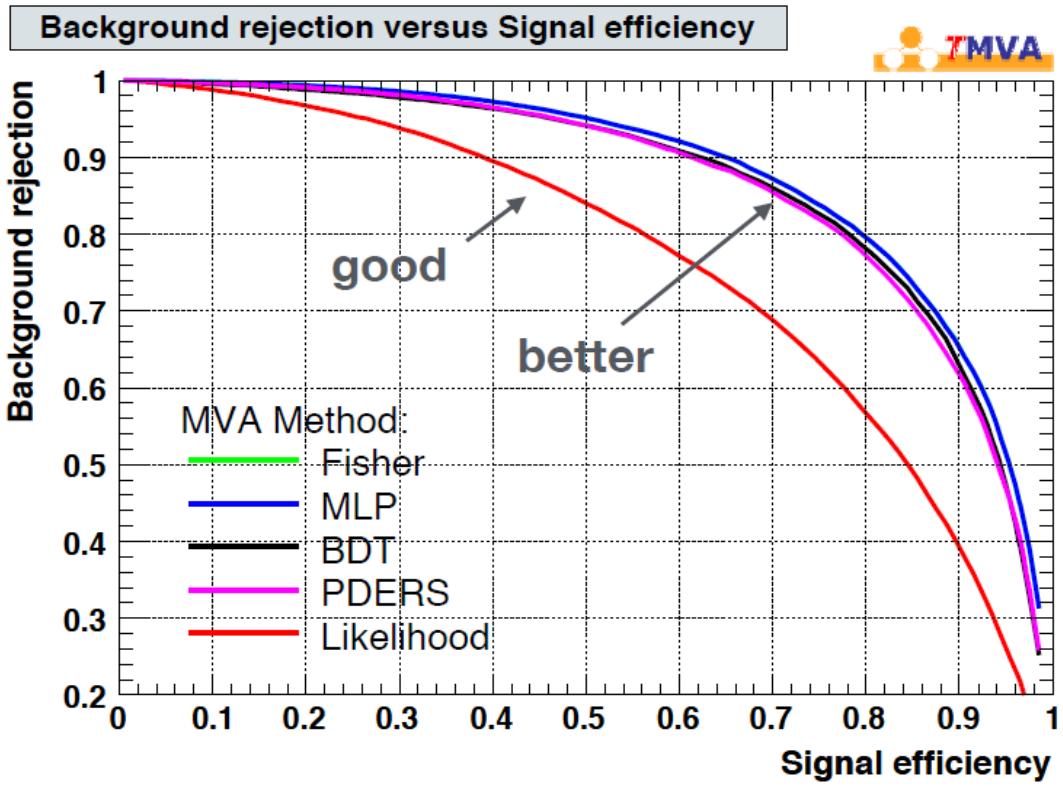
$$P_s(y) \equiv P(S|y) = \frac{p(y|S) \cdot f_s}{p(y|S) \cdot f_s + p(y|B) \cdot (1 - f_s)}, \qquad f_s = \frac{n_s}{n_s + n_b}$$

ROC Curve

Quality of the classification can be characterized by the *receiver operating characteristic* (ROC curve)

$1 - \varepsilon_B$

ε_B: background efficiency



Different Approaches to Classification

Neyman-Pearson lemma states that likelihood ratio provides an optimal test statistic for classification:

$$y(\vec{x}) = \frac{p(\vec{x}|S)}{p(\vec{x}|B)}$$

Problem: the underlying pdf's are almost never known explicitly.

Two approaches:

1. Estimate signal and background pdf's and construct test statistic based on Neyman-Pearson lemma, e.g. Naïve Bayes classifier (= Likelihood classifier)

2. Decision boundaries determined directly without approximating the pdf's (linear discriminants, decision trees, neural networks, …)

General Remarks and Multi-Variate Analyses

MVA Methods

- More effective than classic cut-based analyses
- Take correlations of input variables into account

Important: find good input variables for MVA methods

- Good separation power between S and B
- Little correlations among variables
- No correlation with the parameters you try to measure in your signal sample!

Pre-processing

- Apply obvious variable transformations and let MVA method do the rest
- Make use of obvious symmetries: if e.g. a particle production process is symmetric in polar angle θ use $|\cos\theta|$ and not $\cos\theta$ as input variable
- It is generally useful to bring all input variables to a similar numerical range

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

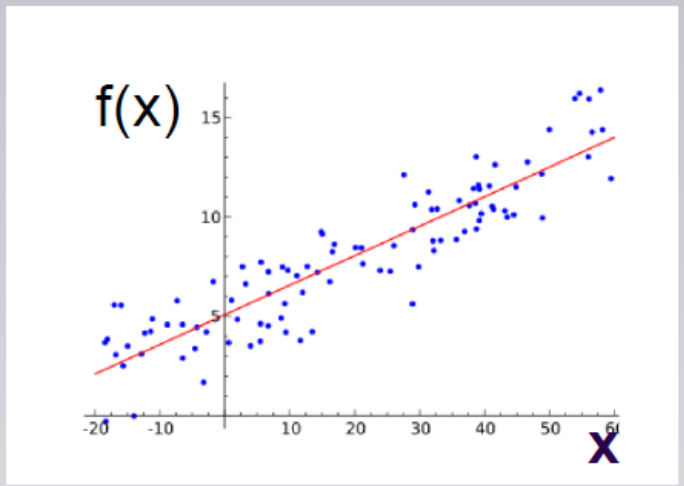
Classifiers and Their Properties

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

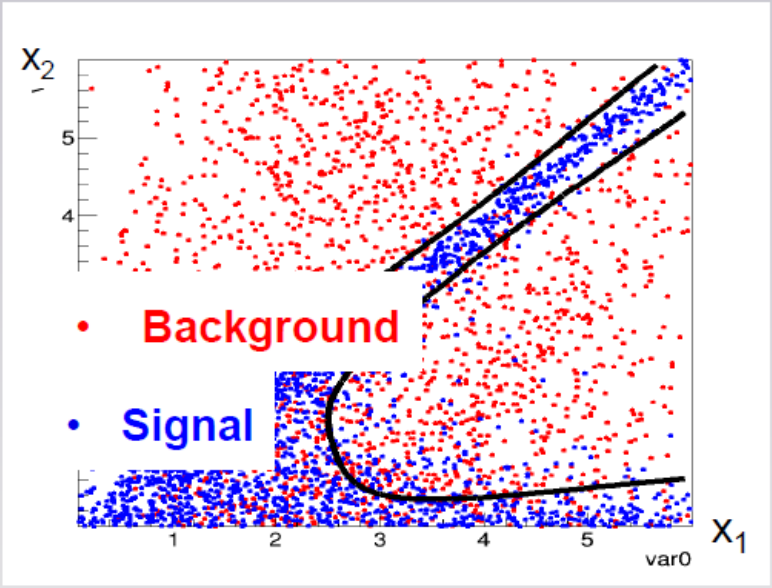
Criteria		Classifiers								
		Cuts	Likeli-hood	PDERS / k-NN	H-Matrix	Fisher	MLP	BDT	RuleFit	SVM
Perfor-mance	no / linear correlations	😐	🙂	🙂	😐	🙂	🙂	😐	🙂	🙂
	nonlinear correlations	😐	☹	🙂	☹	☹	🙂	🙂	😐	🙂
Speed	Training	☹	🙂	🙂	🙂	🙂	😐	☹	😐	☹
	Response	🙂	🙂	☹/😐	🙂	🙂	🙂	😐	😐	😐
Robust-ness	Overtraining	🙂	😐	😐	🙂	🙂	☹	☹	😐	😐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

What are Multivariate Techniques?

→ Many things … starting from "linear regression" …



to multivariate event classification



→ or w/o prior 'analytic' model

→ typically "multivariate"

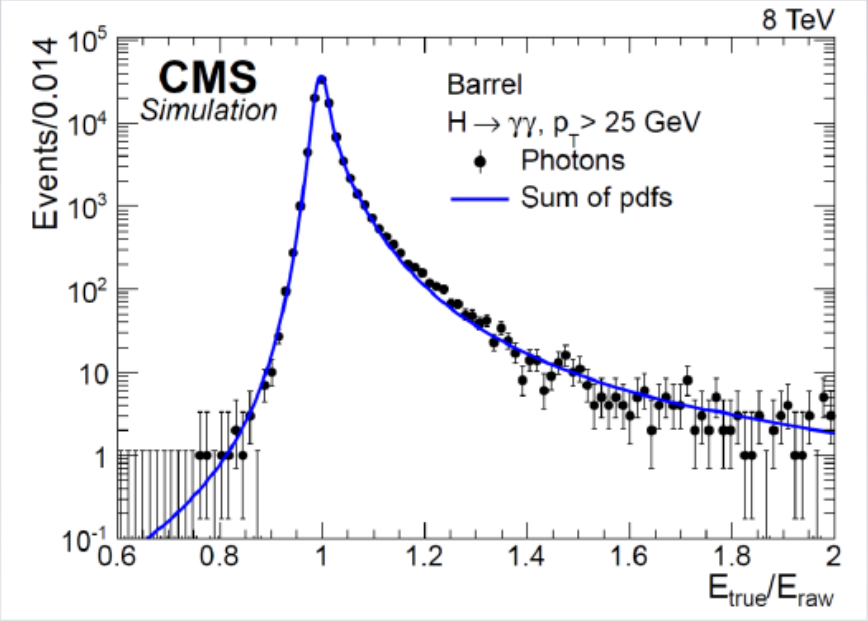
- Parameters depend on the 'joint distribution' $f(x_1, x_2)$

- 'learning from experience' → known data points

Machine Learning - Multivariate Techniques

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

    (e.g. calorimeter calibration/correction function)



MC sample: γ +jets

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

  (energy leakage)
- Pile-up estimators

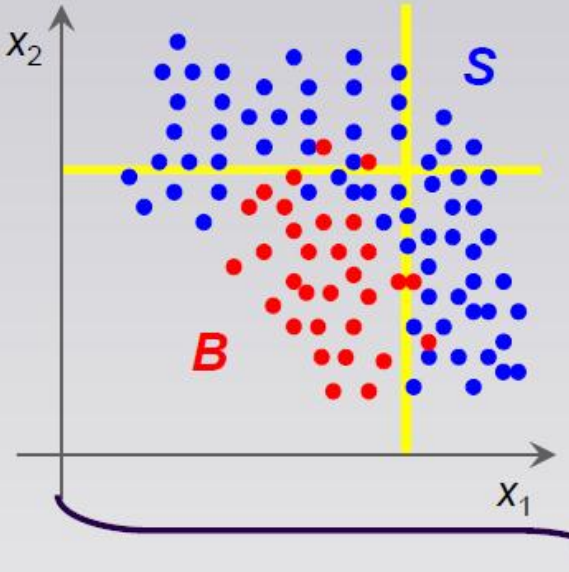
→ predict energy correction (i.e. parameters in crystal-ball: pdf for energy measurement)

Event Classification

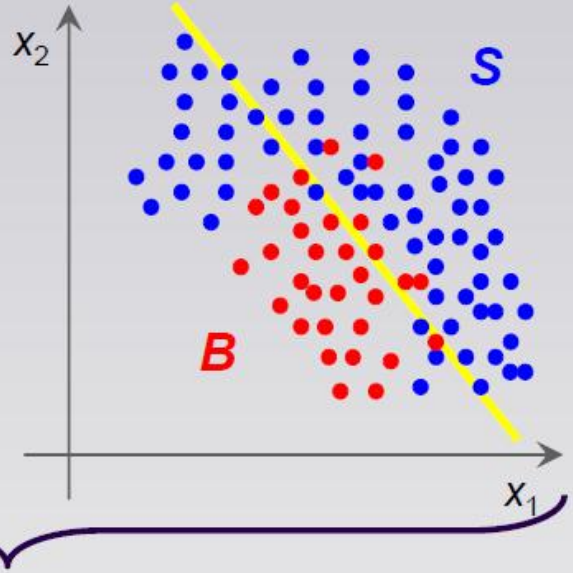
- *Signal* and *Background*
  - discriminating observed variables x_1, x_2, …

  → decision boundary ?

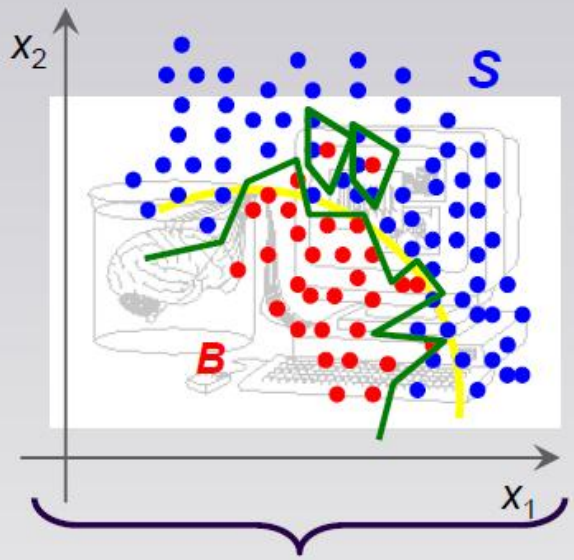
Rectangular cuts?



A linear boundary?



A nonlinear one?

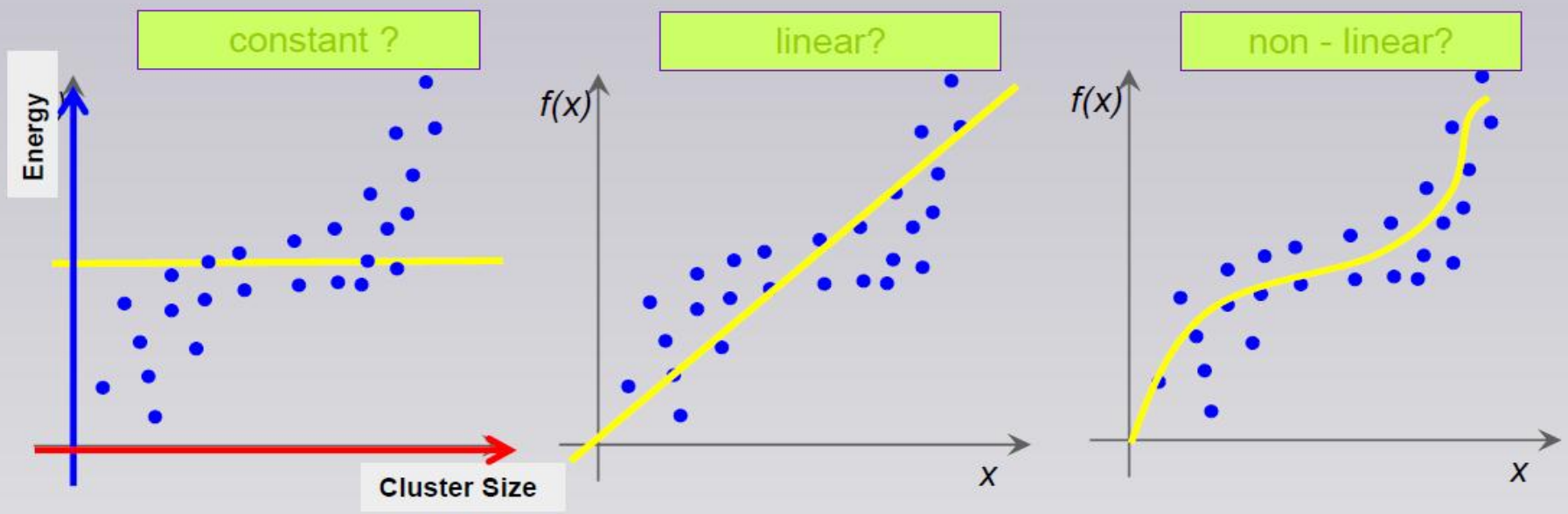


Low variance (stable), high bias methods

High variance, small bias methods

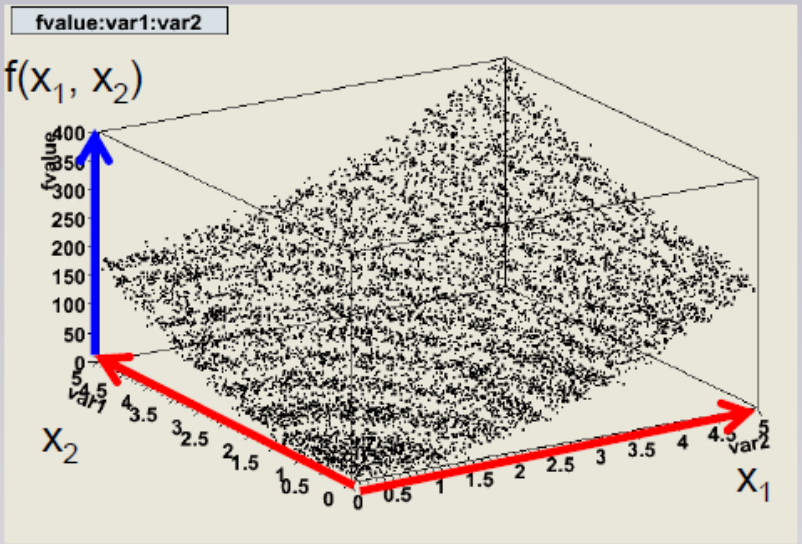
Regression

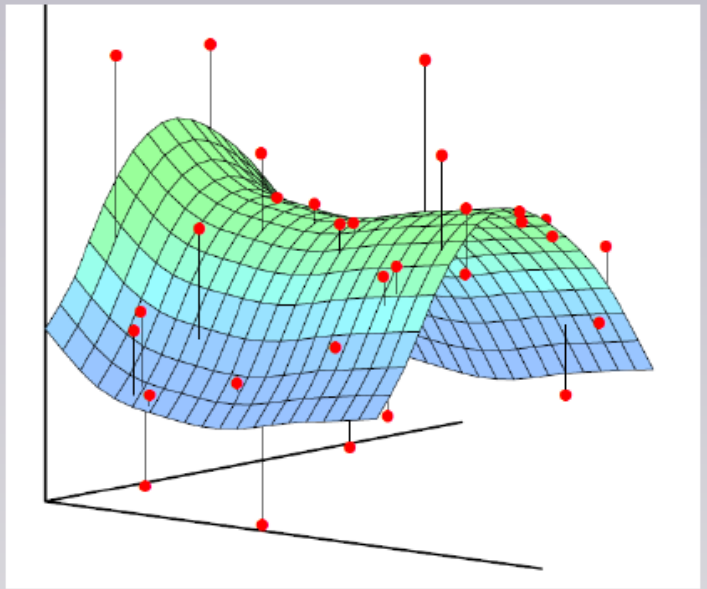
- 'known measurements" → model "functional behaviour"
- e.g. : photon energy as function "D"-variables: ECAL shower parameters + …



- known analytic model (i.e. nth -order polynomial) → Maximum Likelihood Fit)
- no model ?
  → "draw any kind of curve" and parameterize it?
- seems trivial ? → human brain has very good pattern recognition capabilities!
- what if you have **many** input variables?

Regression -> model functional behaviour



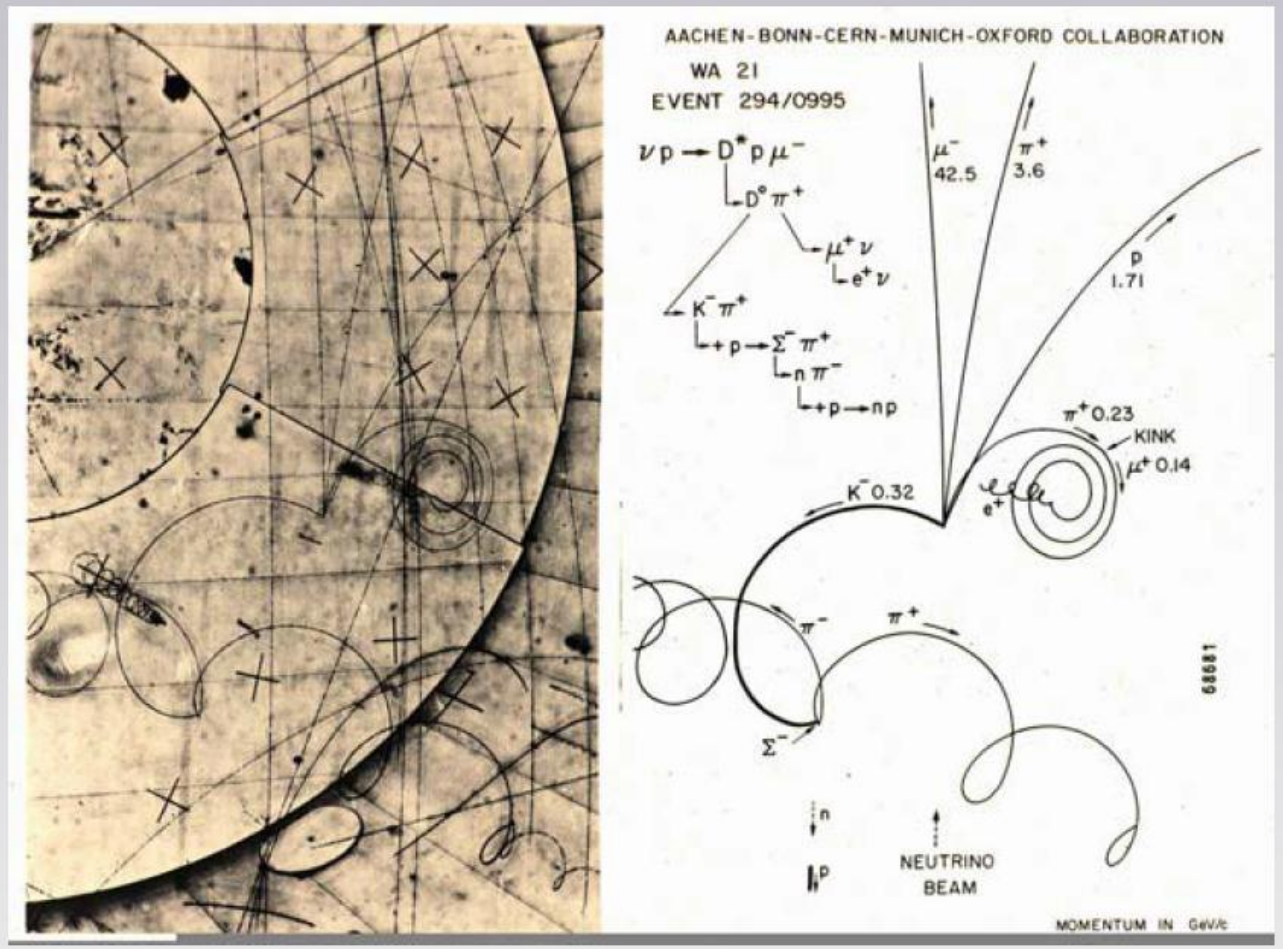


- "standard" regression → fit a known analytic function
  - *e.g.* $f(\mathbf{x}) = ax_1^2+bx_2^2+c$
- BUT most times: don't have a reasonable "model" ? → need something more general:
  - *e.g.* piecewise defined splines, kernel estimators, decision trees to approximate f(x)

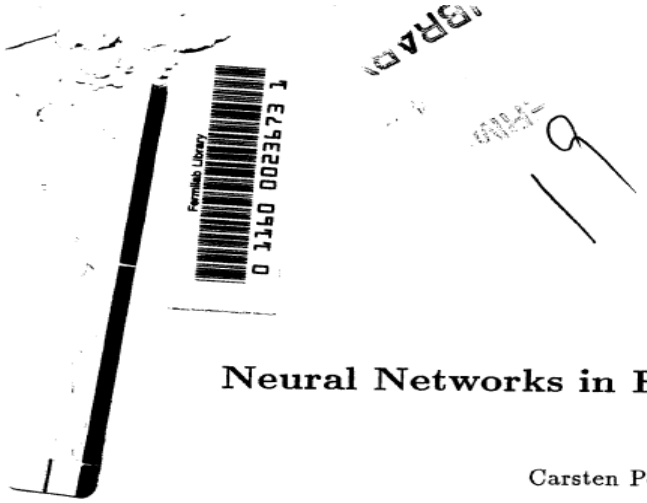
Note: we are not interested in the 'fitted parameter(s)', it is not: "Newton deriving F=m·a"
→ just provide prediction of function values f(x) for new measurements x

HEP: Everything started Multivariate

- intelligent "Multivariate Pattern Recognition" used to identify particles



Machine Learning in HEP



November 1992

LU TP 92-23

Neural Networks in High Energy Physics

Carsten Peterson[1]

Department of Theoretical Physics, University of Lund
Sölvegatan 14A, S-22362 Lund, Sweden

Plenary talk presented at the "Computing in High Energy Physics", September 21 - 25, 1991, Annecy, France

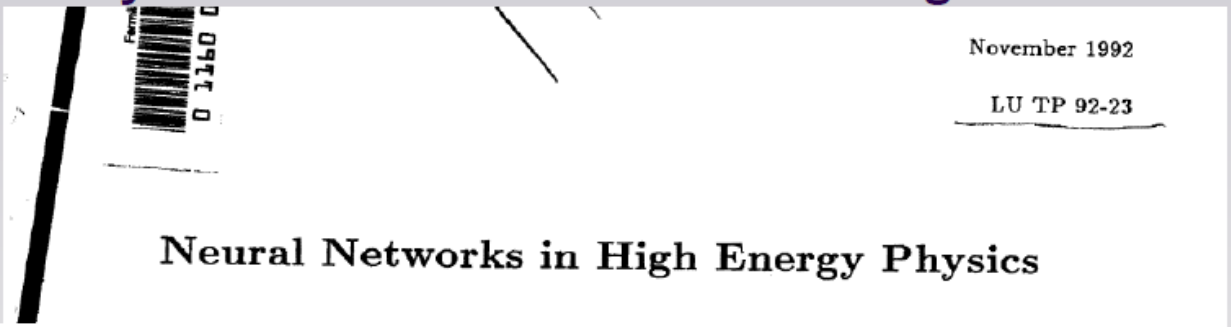
Abstract:

The current status of using artificial neural networks in high energy physics is briefly reviewed. Examples of successful off-line applications for jet identification and tracking are presented. Also, non-classification applications like process control and mass reconstruction are discussed. For classification tasks the approach is demystified by stressing that the output can be interpreted as Bayesian probabilities.

In the optimization domain several approaches to track finding are discussed. In particular template matching approaches are emphazised – the elastic arms algorithm and the rotor model.

Machine Learning in HEP

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



November 1992

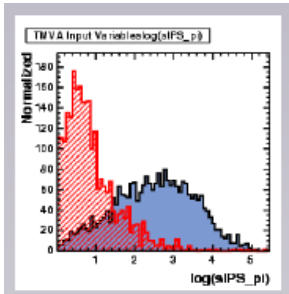
LU TP 92-23

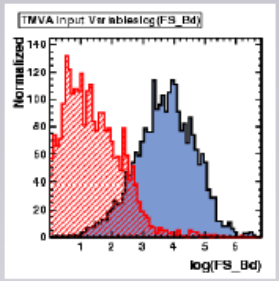
Neural Networks in High Energy Physics

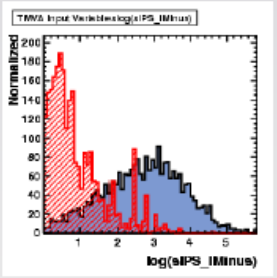
High Energy Physics

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

Event Classification





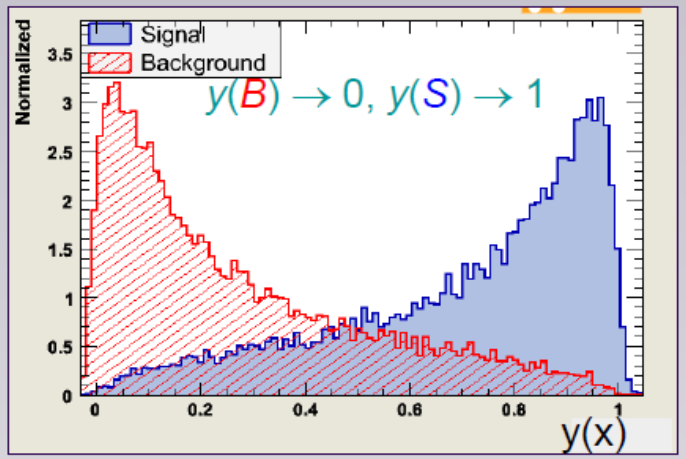


P^D

"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 }

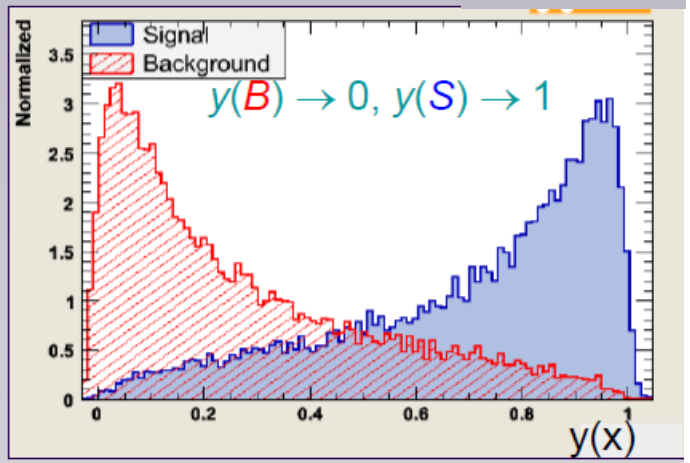
- overlap of $PDF_S(y)$ and $PDF_B(y)$ → separation power , purity
- y(x)=const: surface defining the decision boundary.

Classification ↔ Regression

Classification:

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

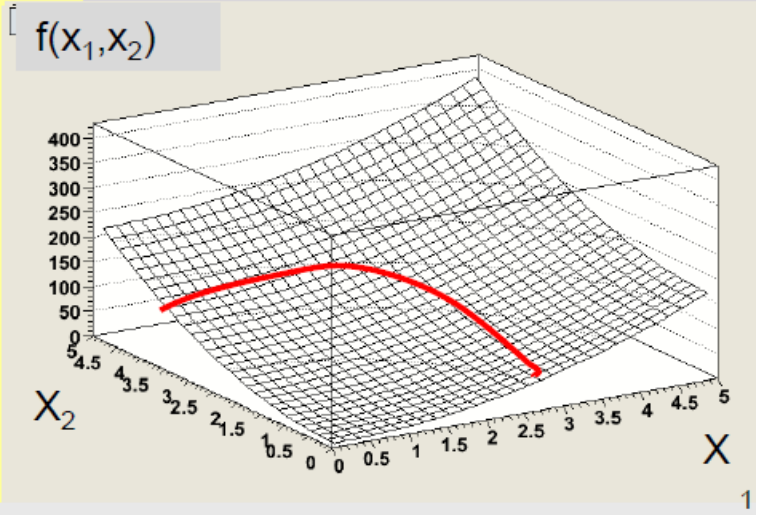
y(x): $R^D \rightarrow R$:



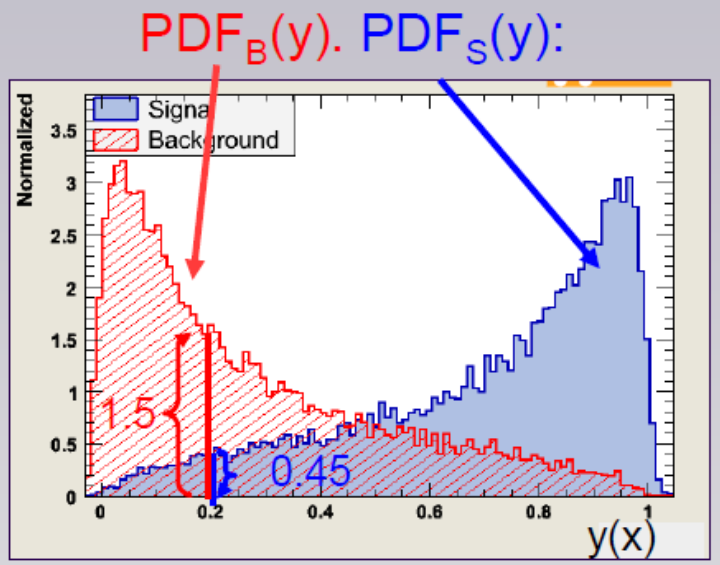
Regression:

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

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



Event Classification



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

→ 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 $PDF_S(y(x)) = 0.45$

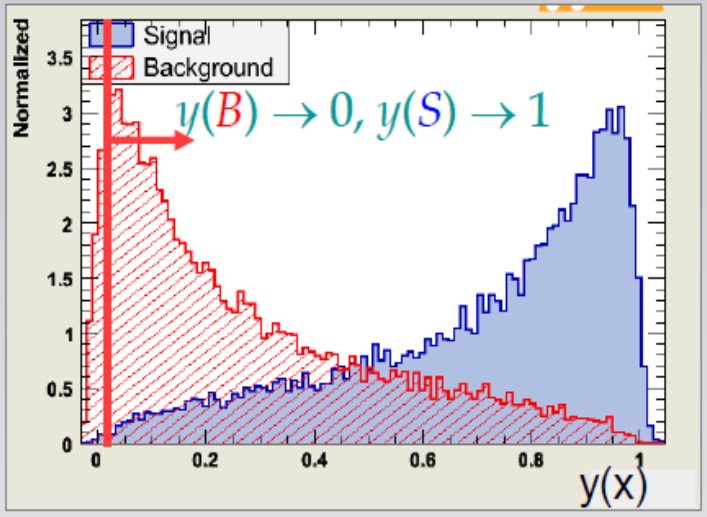
f_S ,f_B : fraction of **S** and **B** in the sample:

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

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

Receiver Operation Characteristic (ROC) curve

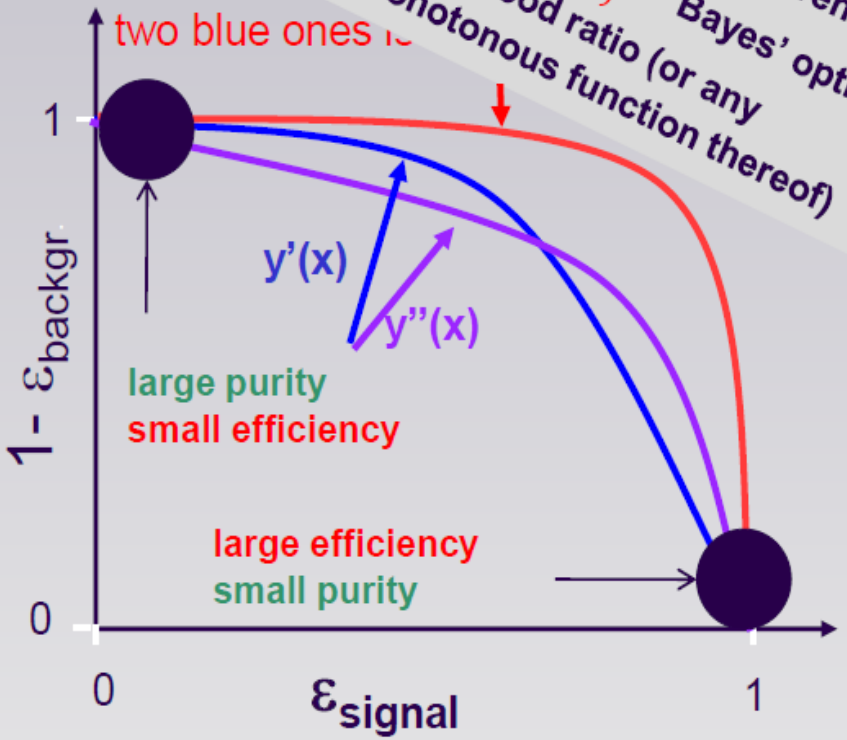
Signal(H_1) /Background(H_0) discrimination:



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

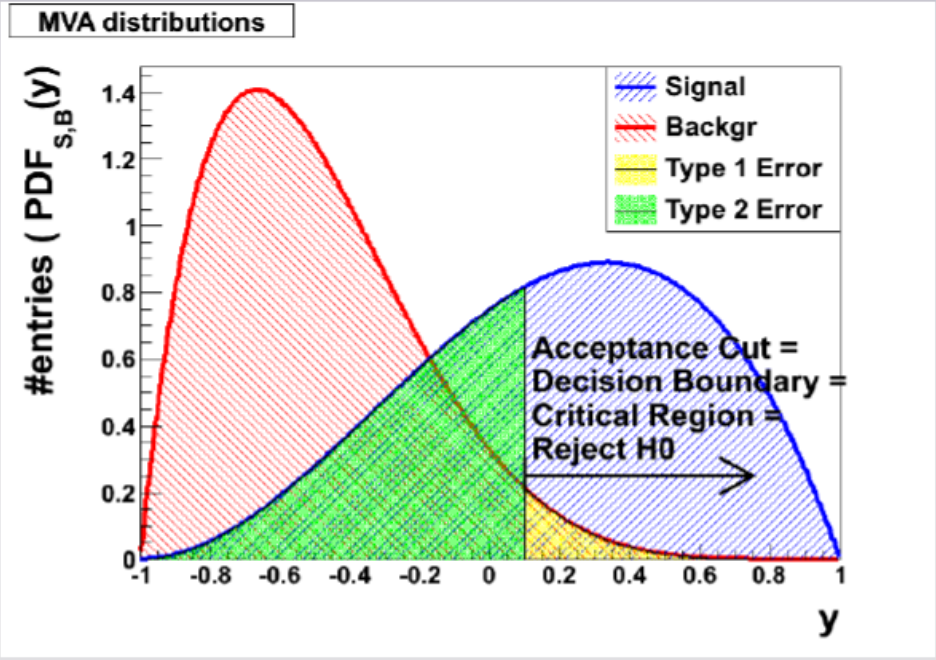
$y(x) = \frac{PDF(x|S)}{PDF(x|B)}$, Bayes' optimal

which one
two blue ones



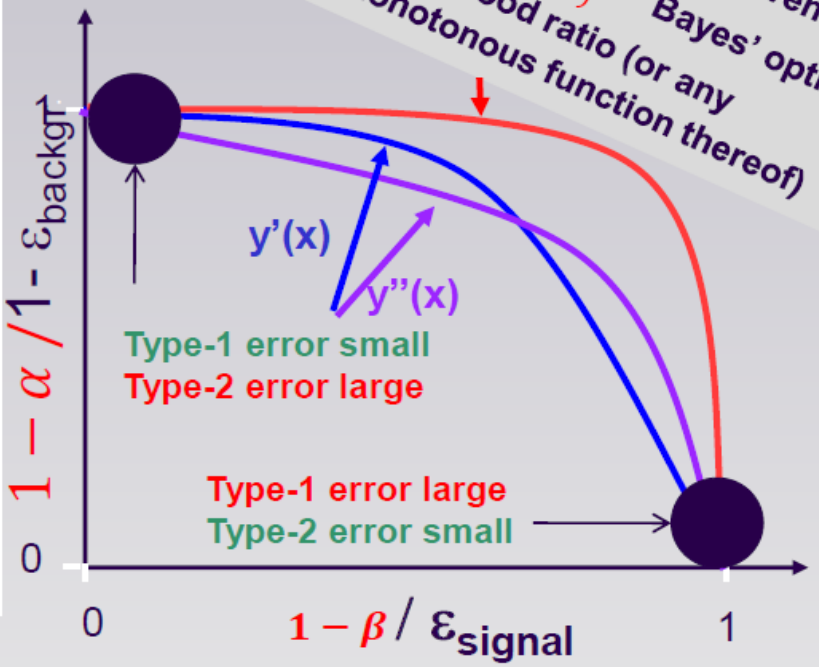
Receiver Operation Characteristic (ROC) curve

Signal(H_1) /Background(H_0)



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

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



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

Event Classification -> finding the mapping function y(x)

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

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

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

- use already classified "events" (e.g. MonteCarlo) to:
  - estimate p(x|S) and p(x|B): (e.g. the differential cross section folded with the detector influences) and use the likelihood ratio
    - → e.g. D-dimensional histogram, Kernel density estimators, …
    - → (generative algorithms)

  OR

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

      (y(x) = const) optimally separate signal from background
    - e.g. Linear Discriminator, Neural Networks, …
    - → (discriminative algorithms)

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

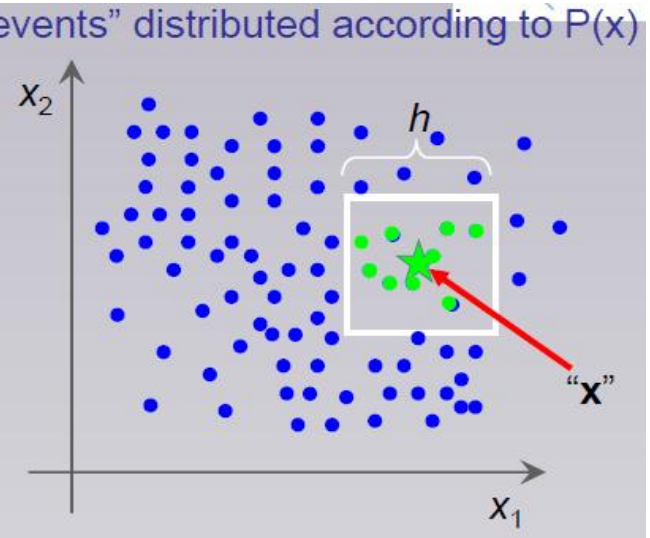
reinforcement-learning:

- 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)

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

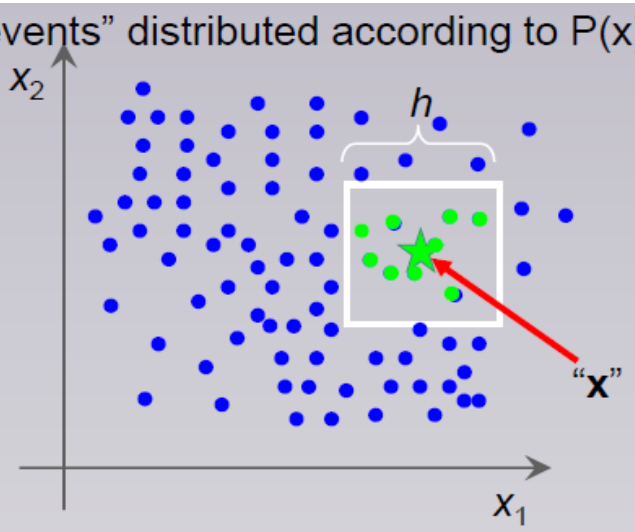
- Classification: Determine
  $PDF_S(x)$ and $PDF_B(x)$
  → likelihood ratio as classifier!

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

→ 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) →

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

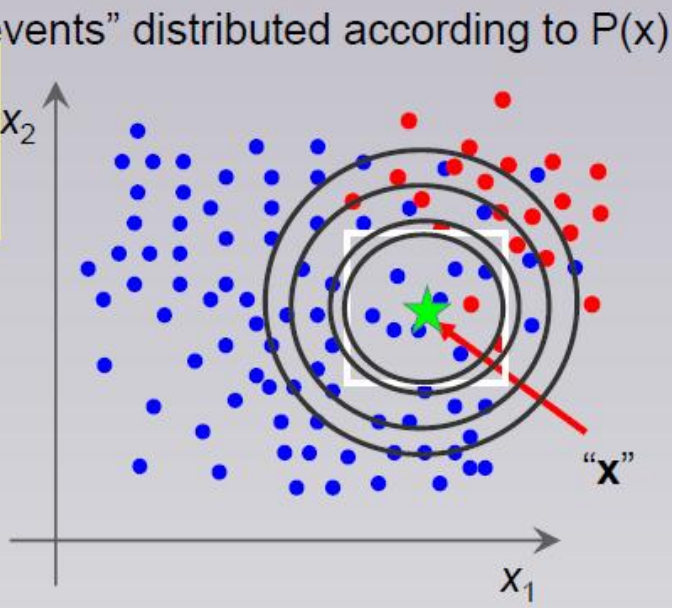
i.e.: the average function value

K-Nearest Neighbour

→kNN : k-Nearest Neighbours

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

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



keep K fixed → variable window size

→automatically 'adapt' resolution to the available data

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

Kernel Density Estimator

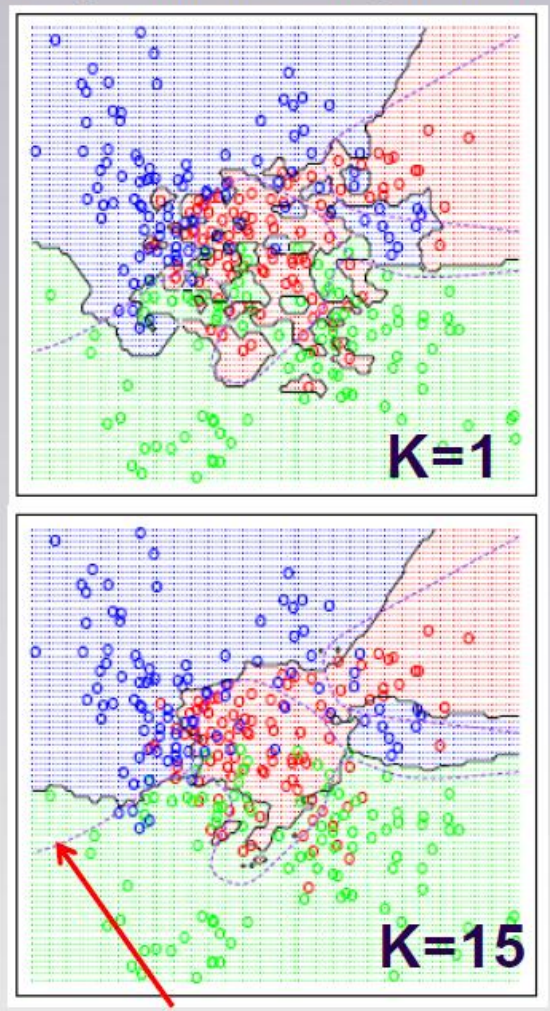
$$P(\mathbf{x}) = \frac{1}{N}\sum_{n=1}^{N} K_h(\mathbf{x} - \mathbf{x}_n)$$: a general probability density estimator using kernel K

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

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

- a drawback of Kernel density estimators:

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



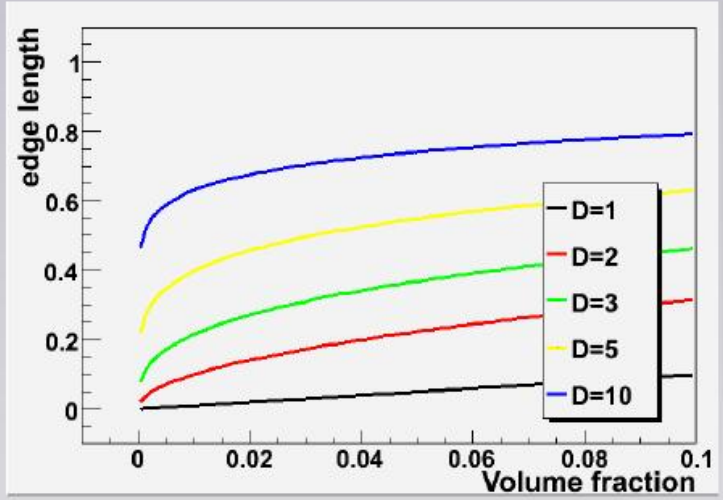
(Elements of statistical learning)

„Curse of Dimensionality”

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)

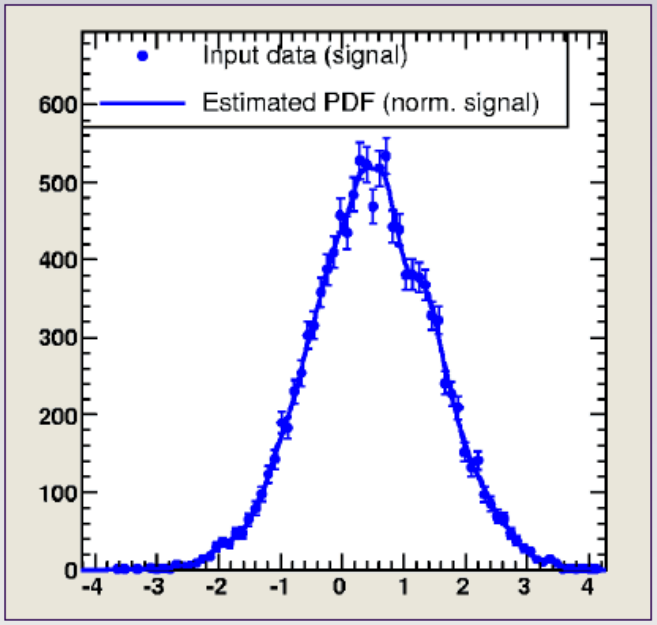
Multivariate Likelihood (k-Nearest Neighbour)
→ estimate the full D-dimensional joint probability density

Naïve Bayesian
→ ignore correlations

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

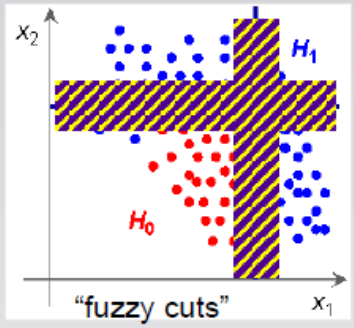
product of marginal PDFs
(1-dim "histograms")

pdf: histogram + smoothing



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

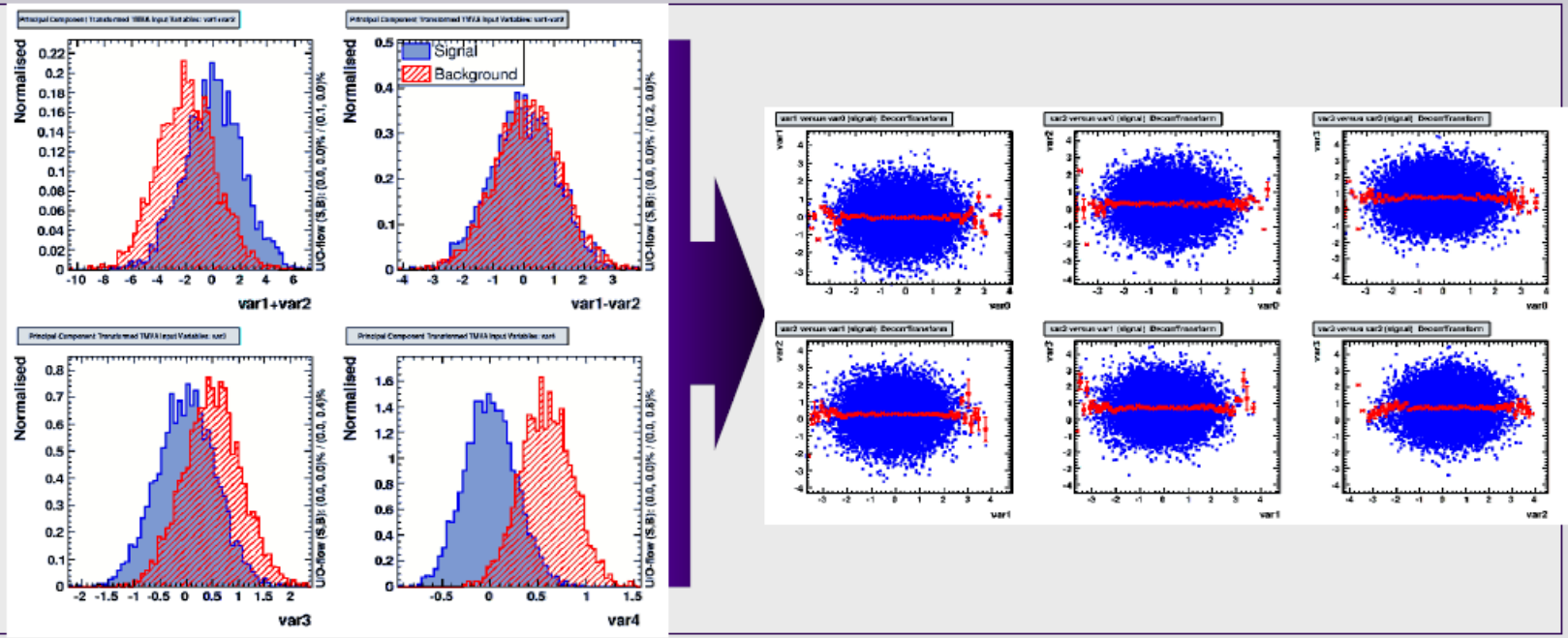
(a very signal like variable may counterweigh another, less signal like variable)



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

De-Correlation

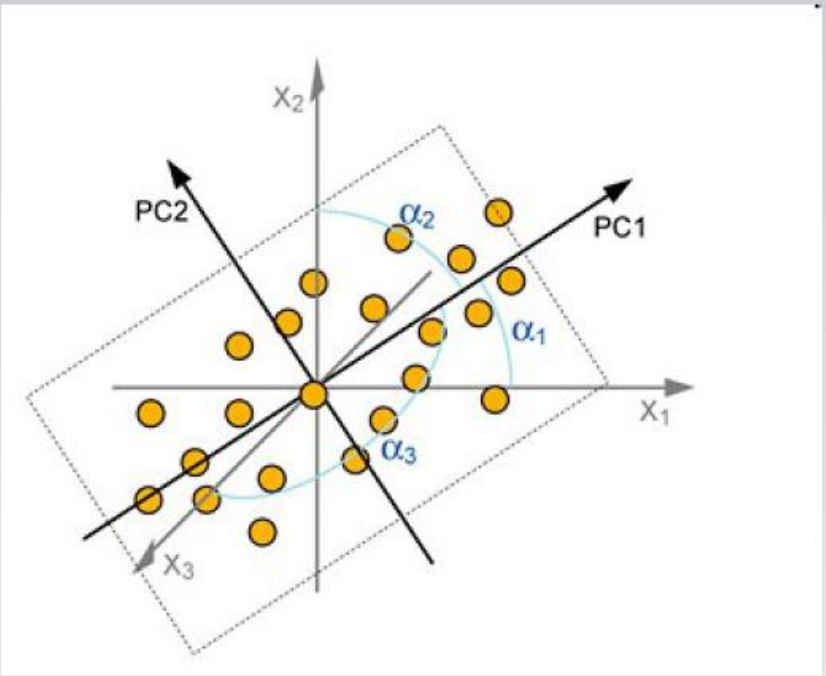
- **Find variable transformation that diagonalises the covariance matrix**
  - Determine *square-root* C' of correlation matrix C, *i.e.*, $C = C'C'$
    - compute C' by diagonalising C: $D = S^T C S \Rightarrow C' = S\sqrt{D}S^T$
    - transformation from original (x) in de-correlated variable space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



Attention: eliminates only linear correlations!!

De-Correlation via PCA (Principal Component Analysis)

- **PCA** (unsupervised learning algorithm)
  - reduce dimensionality of a problem
  - find most dominant features in a distribution
- **Eigenvectors of covariance matrix → "axes" in transformed variable space**
  - large eigenvalue → large variance along the axis (principal component)



→ PCA eliminates correlations!

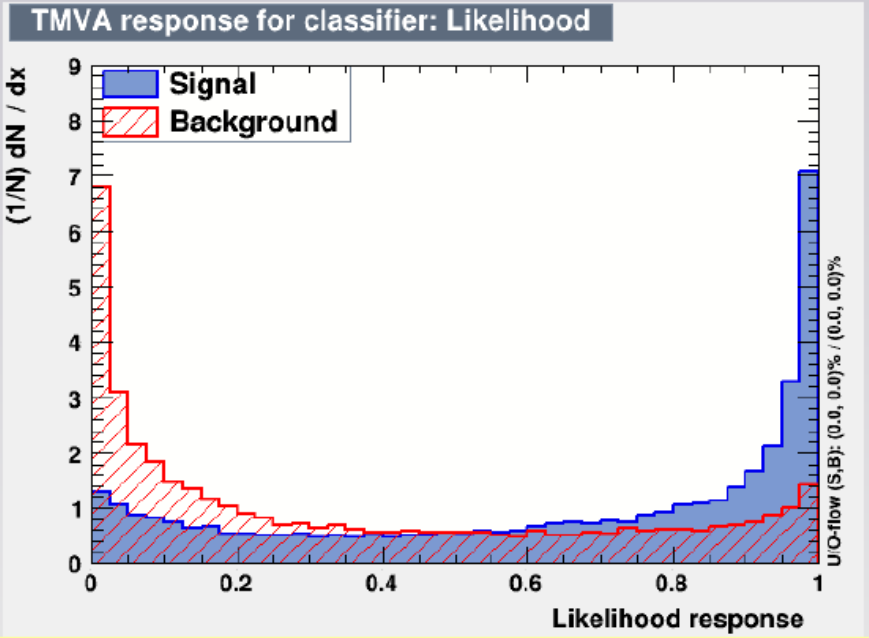
Decorrelation at work

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

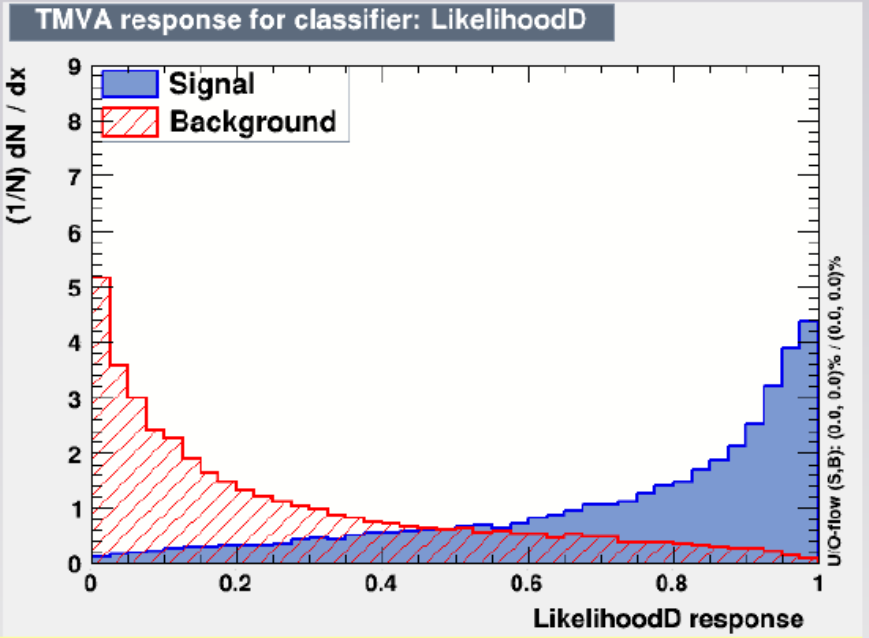
→1-D Likelihood on de-correlated sample give best possible performance

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

correlated variables:



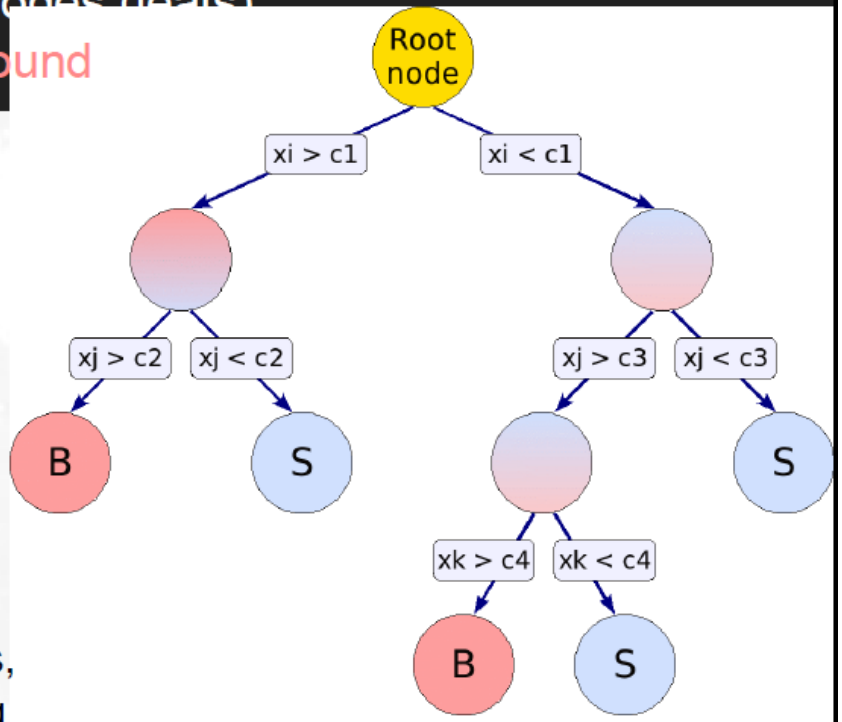
after decorrelation



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

Boosted Decision Trees

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

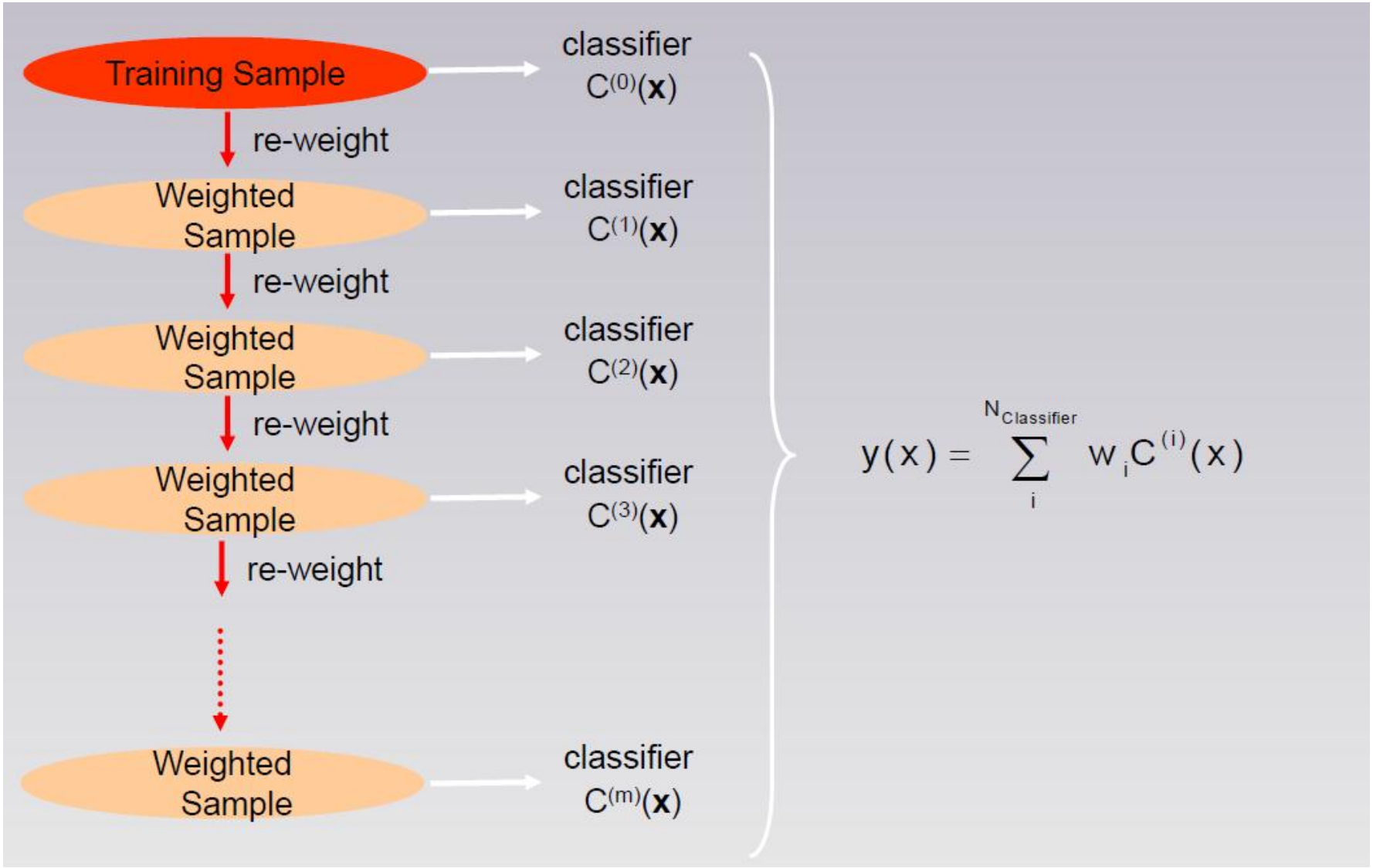


- Each branch → one standard "cut" sequence
  - easy to interpret, visualised
- Disadvatage → very sensitive to statistical fluctuations in training data

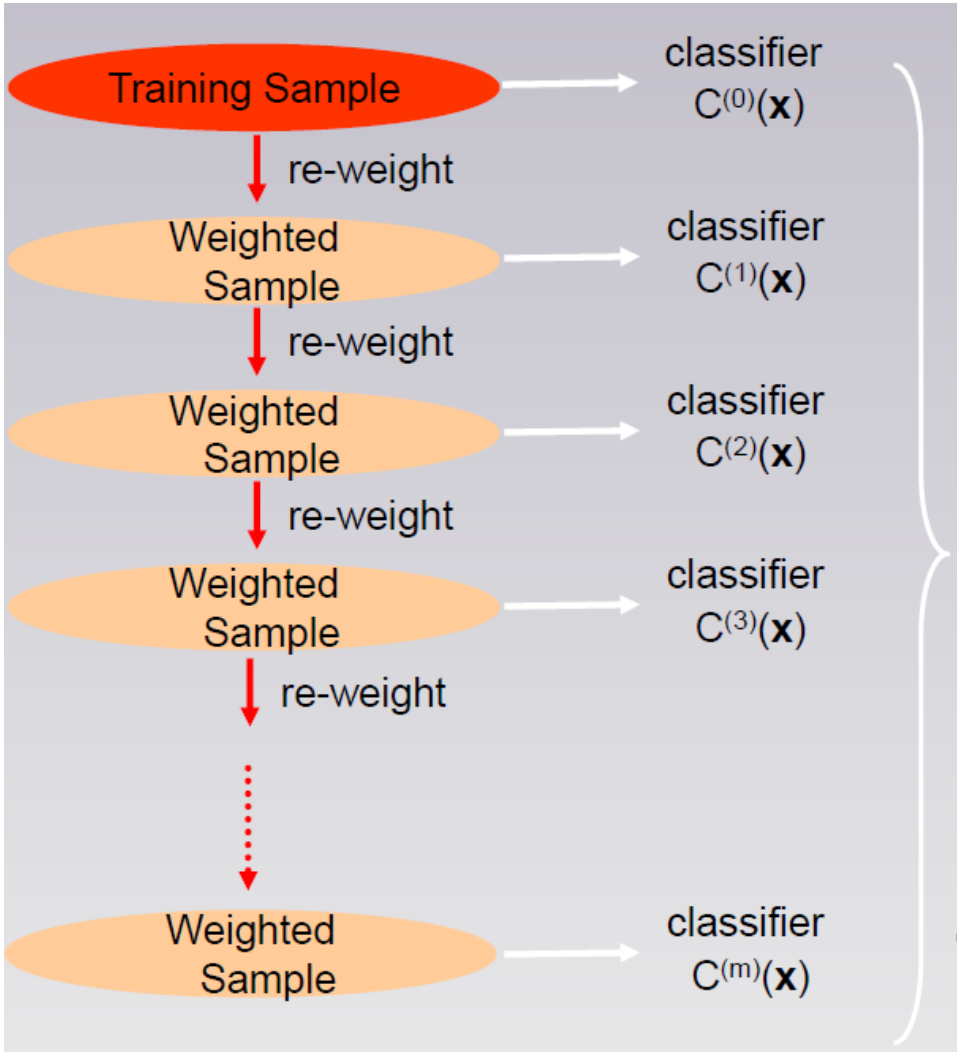
- Boosted Decision Trees (1996): combine a whole forest of Decision Trees, derived from the same sample, e.g. using different event weights.
  - overcomes the stability problem
  - increases performance

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

Boosting


Training Sample
re-weight
Weighted Sample
re-weight
Weighted Sample
re-weight
Weighted Sample
re-weight
Weighted Sample
classifier $C^{(0)}(\mathbf{x})$
classifier $C^{(1)}(\mathbf{x})$
classifier $C^{(2)}(\mathbf{x})$
classifier $C^{(3)}(\mathbf{x})$
classifier $C^{(m)}(\mathbf{x})$
$y(x) = \sum_{i}^{N_{Classifier}} w_i C^{(i)}(x)$

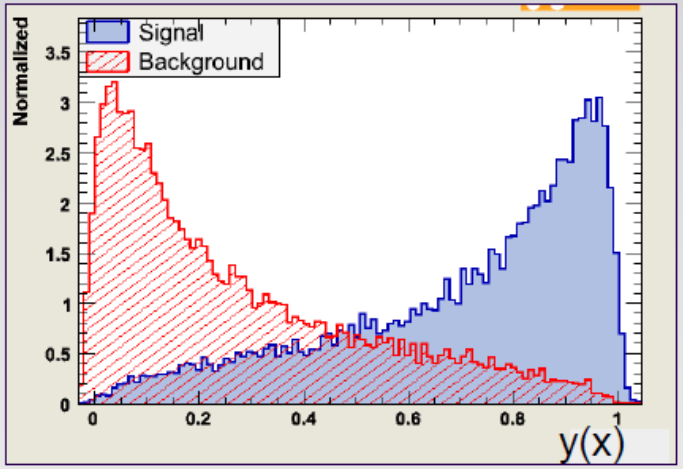
Adaptive Boosting (AdaBoost)



- AdaBoost re-weights events misclassified by previous classifier:

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

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



Boosted Decision Trees

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

MACHINE LEARNING

- Basic terminology
- Classical approaches to prediction
- Bias-variance trade-off
- Introduction to Neural Networks

Some plots from

[4] T. Hastie, R. Tibshirani, J, Friedman, *The Elements of Statistical Learning (2nd ed.)*, Springer Series in Statistics, 2001

Basic terminology

The goal of machine learning is to predict results based on incoming data.

Features (also parameters, or variables): these are the factors for a machine to look at. E.g.: carthesian coordinates, pixel colors, a car mileage, user's gender, stock price, word frequency in the text.

- Quantitative (x = {1.02, 0.21, 0.12, 2})
- Qualitative *discrete* (x = {medium, small, large}) or *categorical* (x={red, blue, green})

Algorithms (also models): Any problem can be solved in different ways. The method you choose affects the precision, performance, and size of the final model.

- If the data is insufficient/inapproriate (e.g. statistically limited or missing important features), even the best algorithm won't help. Pay attention to the accuracy of your results only when you have a good enough dataset.

CLASSICAL MACHINE LEARNING

Data is pre-categorized or numerical

Data is not labeled in any way

SUPERVISED

Predict a category

Predict a number

CLASSIFICATION

«Divide the socks by color»

REGRESSION

«Divide the ties by length»

OUR FOCUS

UNSUPERVISED

Divide by similarity

Find hidden dependencies

Identify sequences

CLUSTERING

«Split up similar clothing into stacks»

ASSOCIATION

«Find what clothes I often wear together»

DIMENSION REDUCTION (generalization)

«Make the best outfits from the given clothes»

Image credit: https://vas3k.com/blog/machine_learning/

Prediction: Least squares

The linear model is one of our most important tools in statistics.

- Given a vector of inputs $X^T = (X_1, X_2, ..., X_p)$, we predict the output Y via

$$\hat{Y} = \hat{\beta}_0 + \sum_{j=1}^{p} X_j \hat{\beta}_j$$

- The term β_0 is the intercept, also known as the *bias* in machine learning

How do we fit the linear model to a set of data?

- The most popular method is the method of least squares: pick the coefficients β to minimize the residual sum of squares (RSS)

$$\mathrm{RSS}(\beta) = \sum_{i=1}^{N} (y_i - x_i^T \beta)^2$$

- RSS(β) is a quadratic function of the parameters, and hence its minimum always exists, but may not be unique.

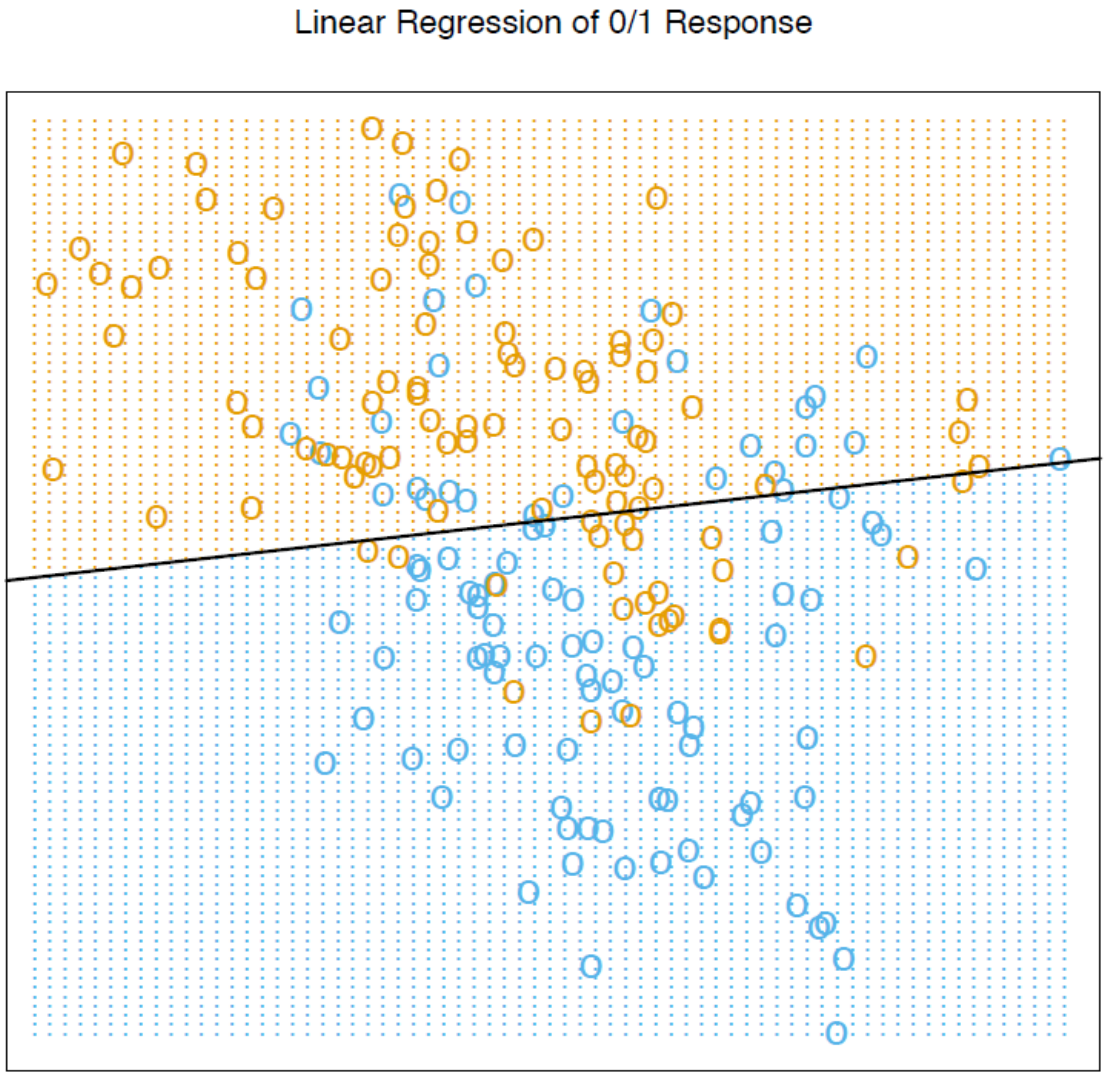
Prediction: Least squares

Data were simulated with outputs being either BLUE or ORANGE.

A linear regression model was fit to the data, used here as *training dataset.*

The fitted values Y are converted in a classification according to

$$\hat{G} = \begin{cases} \texttt{ORANGE} & \text{if } \hat{Y} > 0.5 \\ \texttt{BLUE} & \text{if } \hat{Y} \leq 0.5 \end{cases}$$



Prediction: nearest neighbor classifier

An alternative algorithm for classification is the method of nearest neighbors.

Nearest-neighbor methods use those observations in the training set closest in input space to x to form Y.

The k-nearest neighbor fit for Y is defined as:

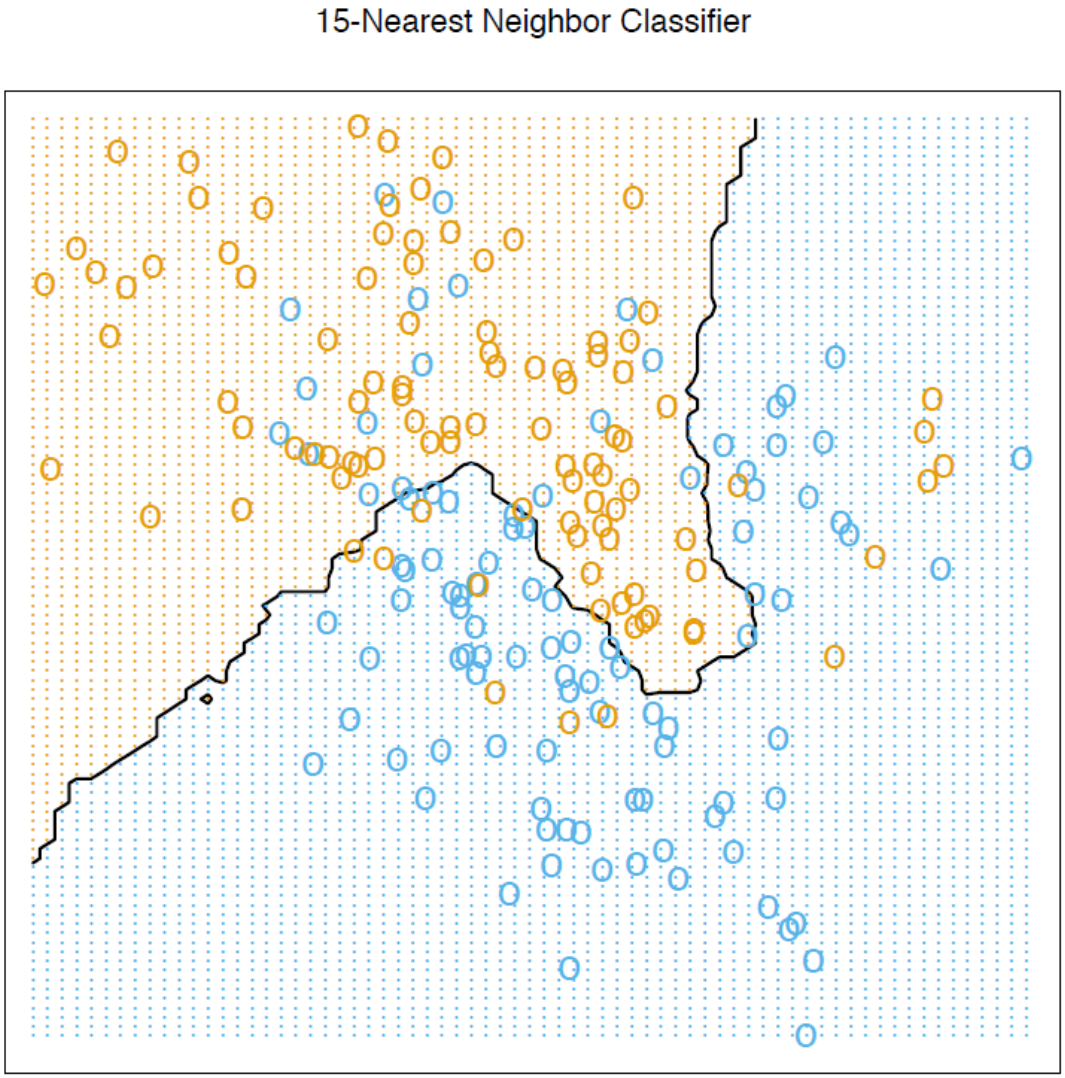
$$\hat{Y}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i$$

where $N_k(x)$ is the neighborhood of x defined by the k closest points x_i in the training sample.

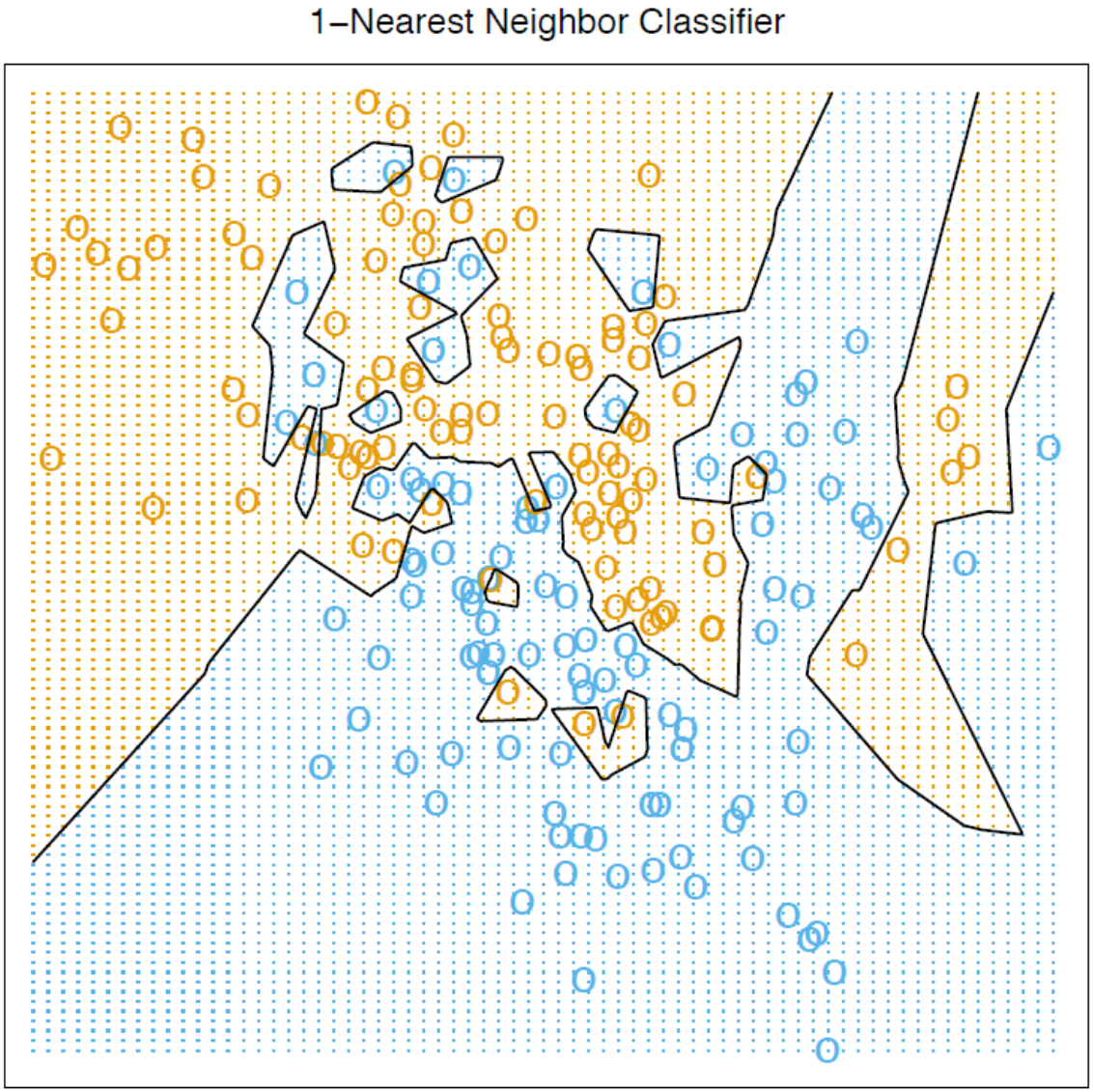
Closeness implies a metric, which in our case we assume is Euclidean distance.

In words, we find the k observations with xi closest to x in input space, and average their responses.

Prediction: nearest neighbor classifier



Perfect classification?



Comparison

To compare the different algorithms, let's define a *loss* (or cost) criterion.

- Here, we can take the rate of misclassifications

In order to compare the performances, let's introduce a second, independent, dataset to evaluate the performance: the *test dataset.*

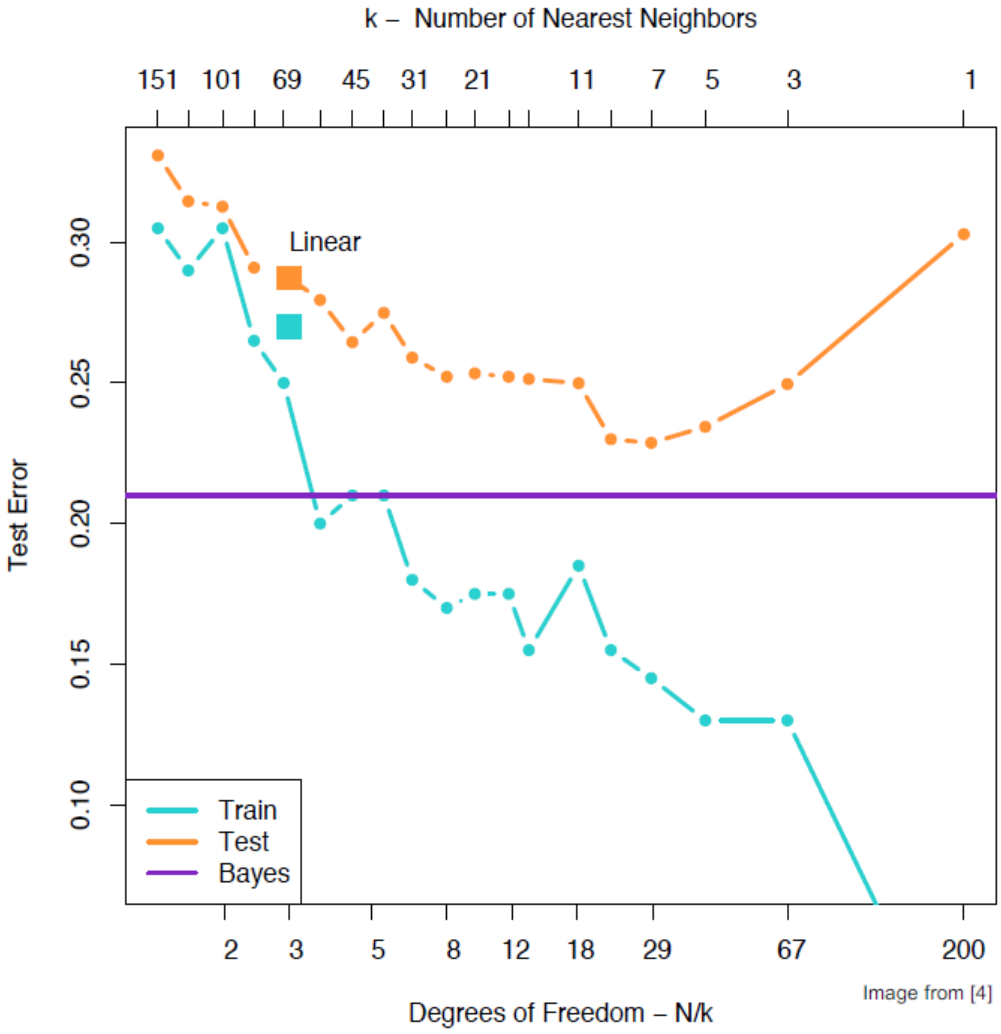


Image from [4]

Bias-variance tradeoff

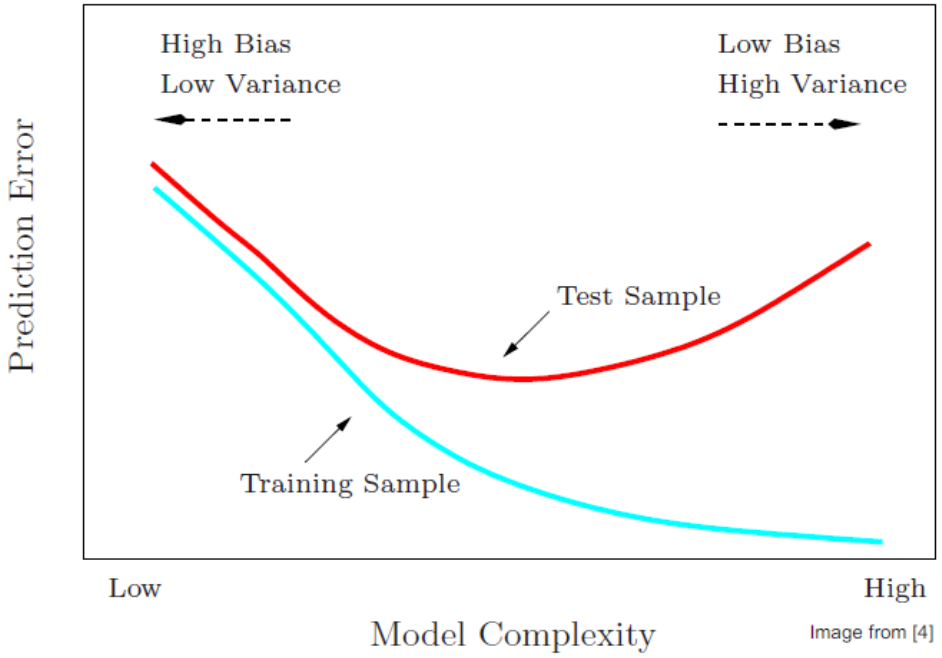


Image from [4]

The training error tends to decrease whenever we increase the model complexity, that is, whenever we fit the data harder.

- With too much fitting, the model adapts itself too closely to the training data, and will not generalize well (i.e., have large test error).
- In contrast, if the model is not complex enough, it will underfit and may have large bias, again resulting in poor generalization.

Where are the neural networks?

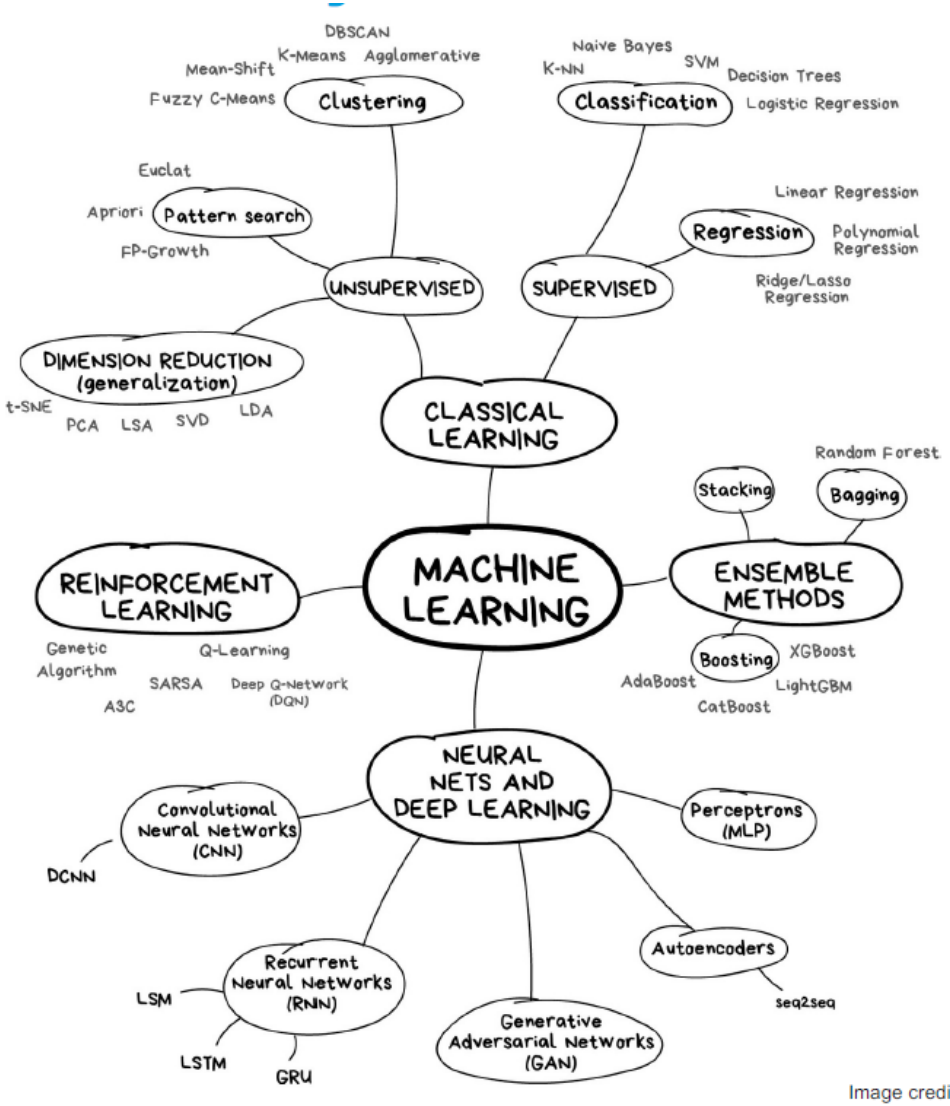


Image credit: https://vas3k.com/blog/machine_learning/

Neural networks

Any neural network is a collection of **neurons** and **connections** between them.

Neuron is a function with a set of inputs and one output. Its task is to take all numbers from its input, apply a function on them and send the result to the output.

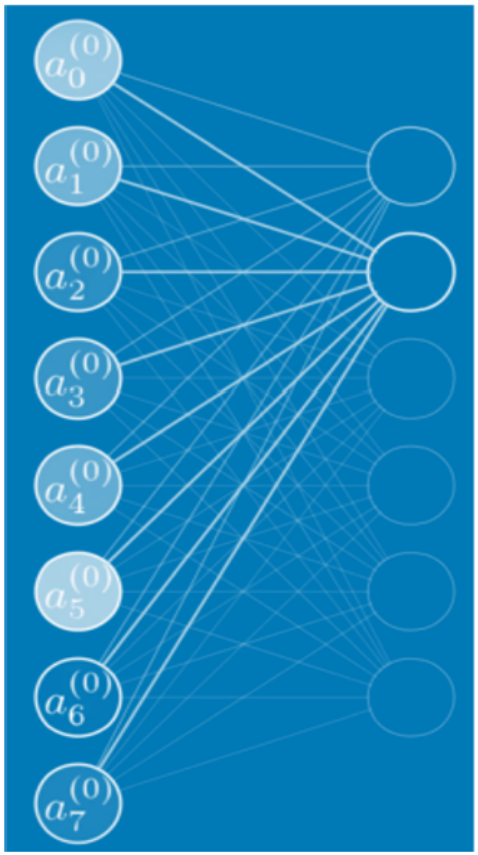
- Example: sum up all numbers from the inputs and if that sum is bigger than N give 1 as a result. Otherwise return zero.

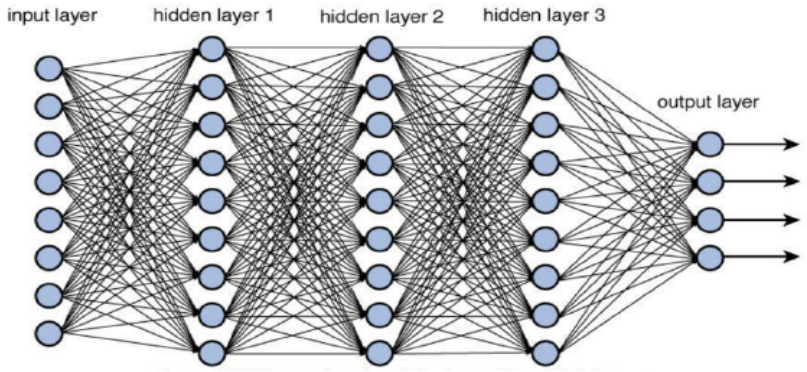
Connections are like channels between neurons. They connect outputs of one neuron with the inputs of another so they can send digits to each other. Each connection has only one parameter the *weight*.

- These weights tell the neuron to respond more to one input and less to another. Weights are adjusted when training — that's how the network learns.

How do NNs work?

How do NNs work?





layer

$a_0^{(1)} = f\,(w_{0,0}\, a_0^{(0)} + w_{0,1}\, a_1^{(0)} + \ldots + w_{0,n}\, a_n^{(0)} + b_0)$

activation function

weights

bias

$$\begin{bmatrix} a_0^{(1)} \\ \ldots \\ a_n^{(1)} \end{bmatrix} = f\left(\begin{bmatrix} w_{0,0} & \cdots & w_{0,n} \\ \vdots & \ddots & \vdots \\ w_{k,0} & \cdots & w_{k,n} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ \ldots \\ a_n^{(0)} \end{bmatrix} + \begin{bmatrix} b_0 \\ \ldots \\ b_n \end{bmatrix} \right)$$

How do NNs learn?

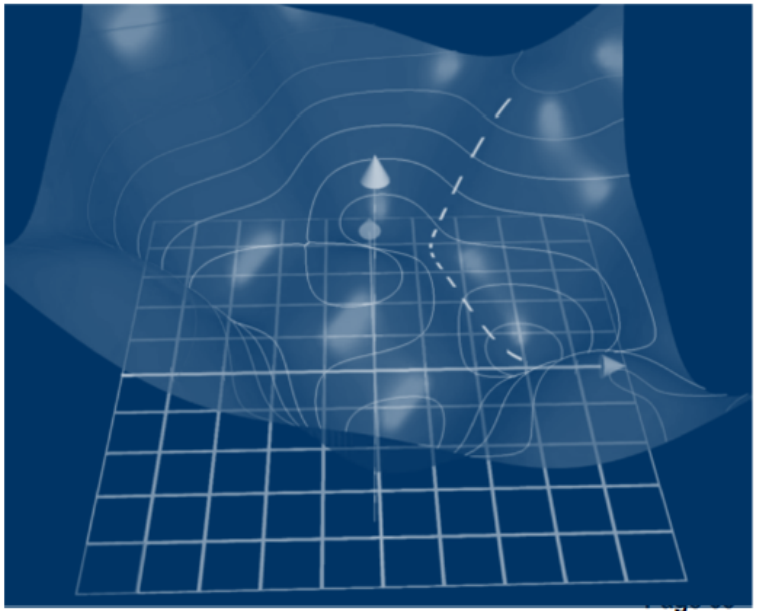
After we constructed a network, our task is to **assign proper weights** so neurons will react correctly to incoming signals.

- define a loss function to measure how far the response is from the truth

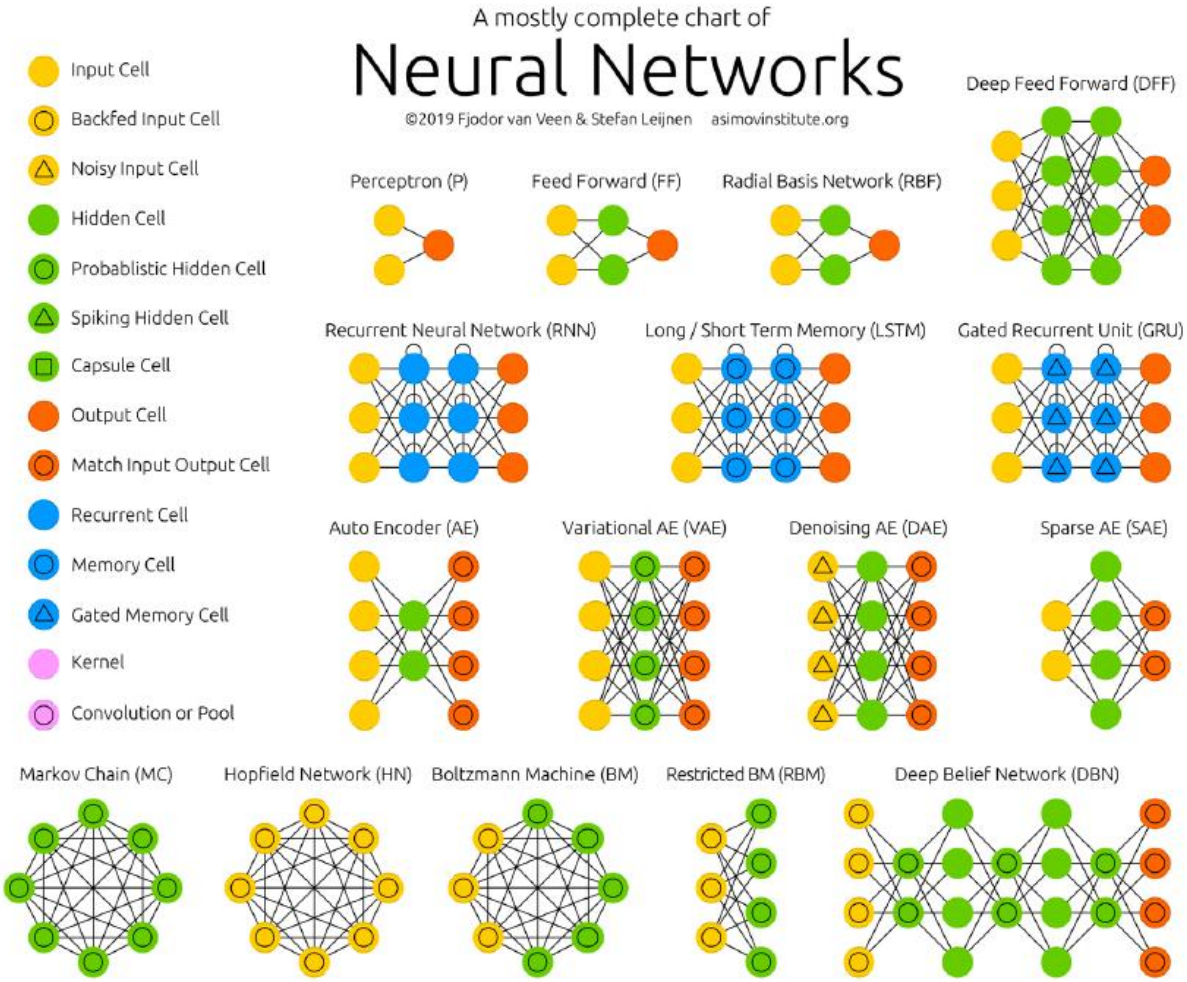
This function is a function of all the weights and biases in the NN (a priori a very large number), and the goal of training is to find its minimum.

- To start with, all weights are assigned randomly.
- After evaluating the NN on the training dataset, we can compute all the per-neuron differences with respect to the correct result.
- Computing the gradient of the loss, gives us a direction in which to tune the weights towards a local minimum

The process of correcting the weights is called backpropagation an error.



How do NNs learn?



There are many more…