

The Monte Carlo method

A numerical technique for calculating probabilities & related things using sequences of random numbers.

The usual steps:

- generate sequence $r_1, r_2, \dots, r_m$ uniform in $]0,1[$.
- use them to produce another sequence $x_1, x_2, \dots, x_n$ distributed according to pdf $f(x)$ of interest
- use obtained x values to estimate some property of

$f(x)$, e.g. fraction of x values within $[a,b] = \int_a^b f(x)dx$

$\Rightarrow$ MC calculation sort of integration (at least formally)

Trivial for 1D: $\int_a^b f(x)dx$ obtainable by other methods, but MC more powerful for multi-dimensional problems.

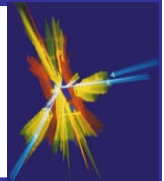
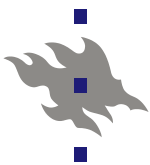
MC x values = "simulated data"

$\rightarrow$ used for testing statistical procedures

MC methods a wide field, actually own field in itself – here focus on the usage of MC for data analysis e.g. determining the statistical (& systematic) uncertainties. In such cases, MCs are used to generate different data distributions. So, let's try to answer the question "how can I generate the type of distribution I need?".

more complete & deeper discussion found in e.g.

- MATR323 "Basics of Monte Carlo simulation"-course by Prof. Flyura Djurabekova in spring 2027 (period III)



goal: to get uniformly distributed values in $]0,1[$ interval.

⇒ "random number generator"

= computer algorithm to generate $r_1, r_2, \dots, r_m$.

e.g. multiplicative linear congruential generator (MLCG)

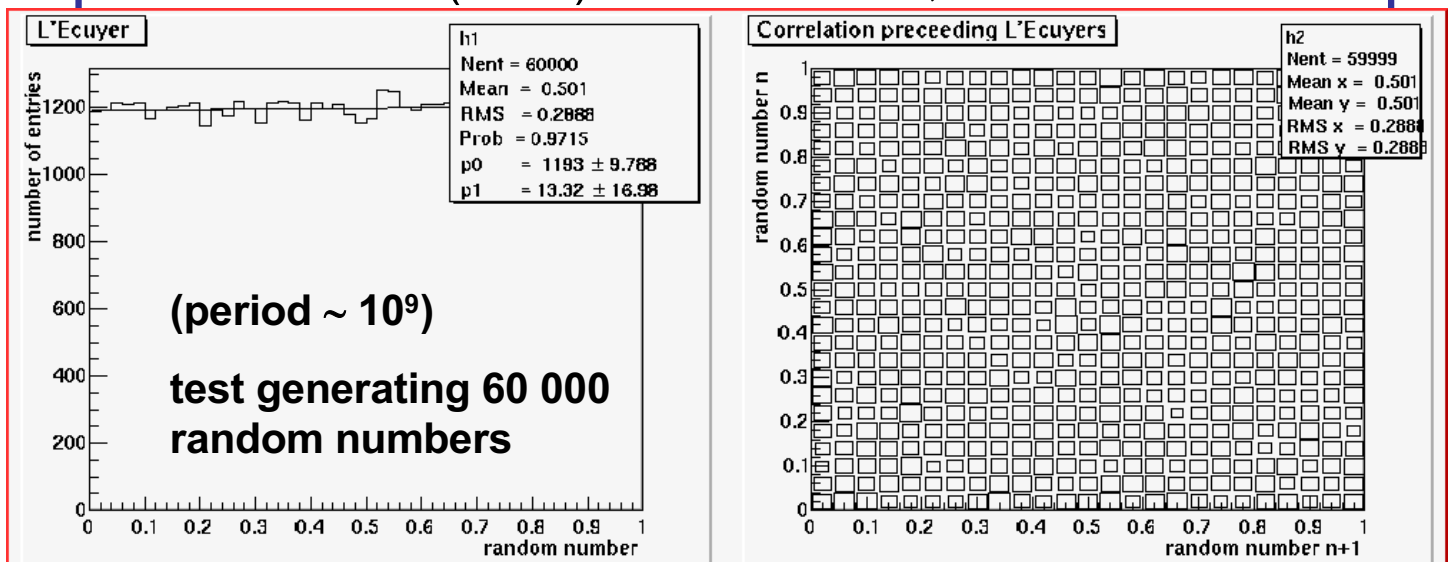
$$n_{i+1} = (an_i) \bmod m, \quad \text{where}$$

n_i = integer, a = **multiplier**, m = **modulus** & n_0 = **seed**.
(NB! mod = modulus (remainder), e.g. $27 \bmod 5 = 2$)

n_i follow periodic sequence in $[1, m-1]$ ⇒

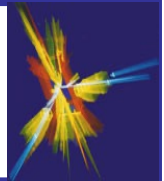
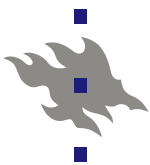
$r_i = n_i / m$ distributed in $]0,1[$.

Choose a & m so that r_i 's pass various tests of randomness: r_i 's uniform in $]0,1[$, succeeding r_i 's uncorrelated & "period" for r_i 's long (maximum = $m-1$) e.g. L'Ecuyer, *Comm. ACM* 31(1988)742: $a = 40692$, $m = 2147483399$



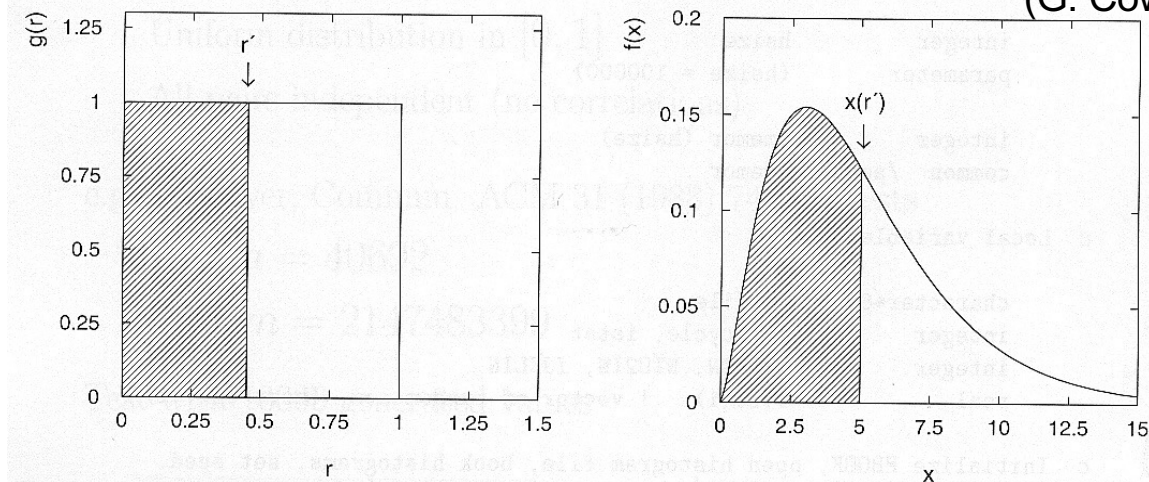
Much better algorithms exist e.g. Mersenne twister, (period $\approx 10^{6000}$) or WELL family (period up to $\approx 10^{45000}$). Many good algorithms included in free program libraries. NB! r_i 's like above in reality **pseudorandom numbers**.

See e.g. F. James, *Comput. Phys. Commun.* 60 (1990) 111



given $r_1, r_2, \dots, r_n$ uniform in $]0,1[$, find $x_1, x_2, \dots, x_n$ which follow $f(x)$ by finding a suitable transformation $x(r)$.

(G. Cowan)



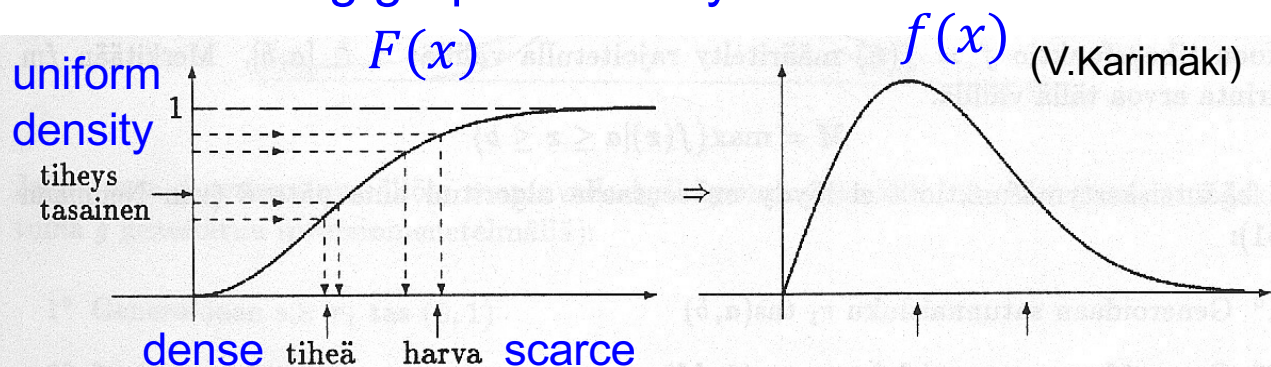
$P(r \leq r') = P(x \leq x(r'))$ i.e.

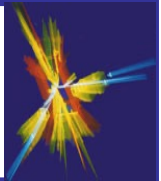
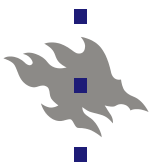
$$\int_{-\infty}^{r'} g(r) dr = r' = \int_{-\infty}^{x(r')} f(x') dx' = F(x(r'))$$

A method that always works when inverse function of cumulative distribution function $F(x)$ can be calculated or put in a table. Then **inverse transform method**:

- sample r from a uniform distribution $]0,1[$
- calculate $x = F^{-1}(r)$

then generated random numbers x that obey pdf $f(x)$. From following graph it is easy to see that it works





Discrete distributions

Inverse transform method for discrete distributions.

p_i = probability for integer i . First have to put in table cumulative distribution function $F_j = \sum_{i=0}^j p_i, j = 0, \dots, N$. If infinite number of possible outcomes i , N to be set so large that $F_N \approx 1$. Generation algorithm for discrete pdf:

(i) sample r from a uniform distribution $]0,1[$

(ii) find k so that $F_{k-1} < r < F_k$

(iii) accept integer $k - 1$

Resulting distribution proportional to probabilities p_i

Example of inverse transform method:

exponential pdf: $f(x; \xi) = \xi^{-1} e^{-x/\xi} \quad (x \geq 0)$

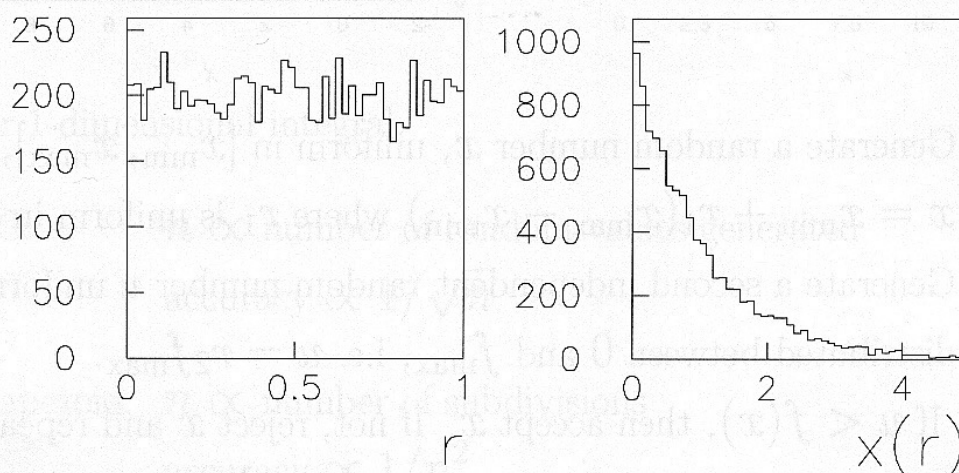
cumulative distribution function:

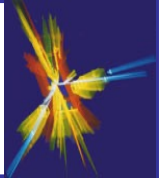
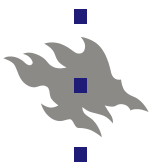
$$F(x) = \int_0^x \xi^{-1} e^{-x'/\xi} dx' = 1 - e^{-x/\xi}$$

assume $r \in]0,1[$, now set $r = F(x)$ & solve for $x(r) \Rightarrow$

$x(r) = -\xi \ln(1 - r)$ (NB! $x(r) = -\xi \ln r$ works also)

(G. Cowan)

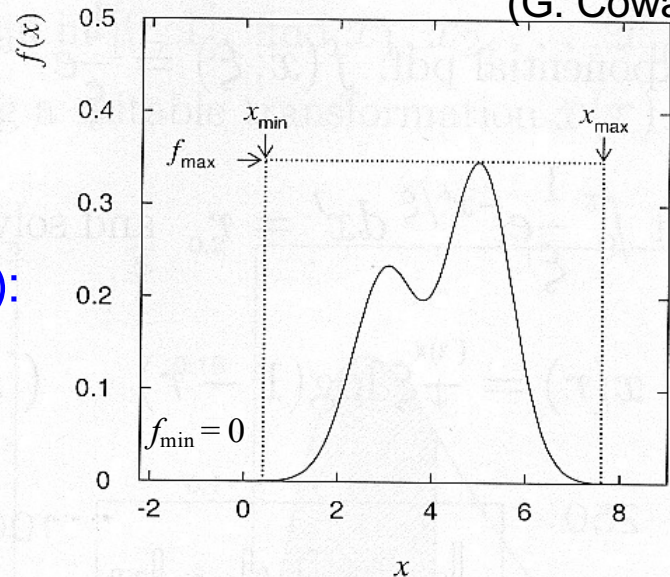




Acceptance-rejection method (von Neumann)

(G. Cowan)

Often analytic solution impossible or very impractical $\Rightarrow$ acceptance-rejection method (or **hit-or-miss**):
enclose pdf in a box
 $f_{\min} = \min(f(x))$,
 $f_{\max} = \max(f(x))$

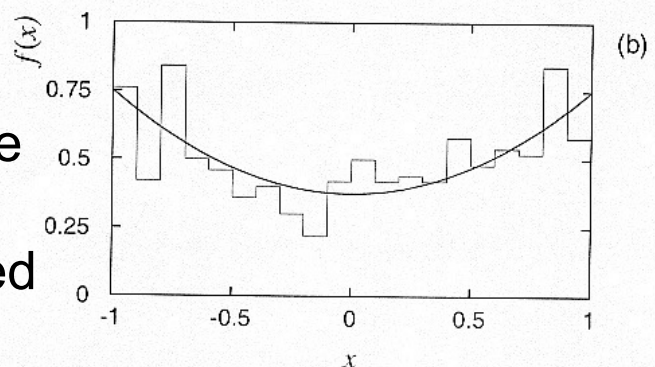
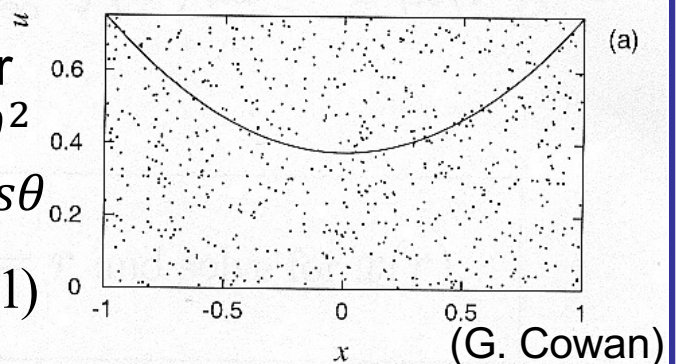


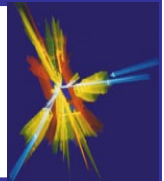
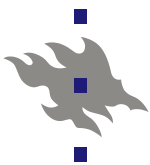
- (i) generate a random number x , uniform in $[x_{\min}, x_{\max}]$, i.e. $x = x_{\min} + r_1(x_{\max} - x_{\min})$, where r_1 uniform in $]0,1[$
- (ii) generate 2nd random number u uniformly distributed between $f_{\min}$ & $f_{\max}$, i.e. $u = f_{\min} + r_2(f_{\max} - f_{\min})$.
- (iii) if $u < f(x)$, then accept x . If not, reject x & repeat.

Example: generate a polar angle distribution $1 + \cos\theta^2$ ($-1 \leq \cos\theta \leq +1$). $x = \cos\theta$

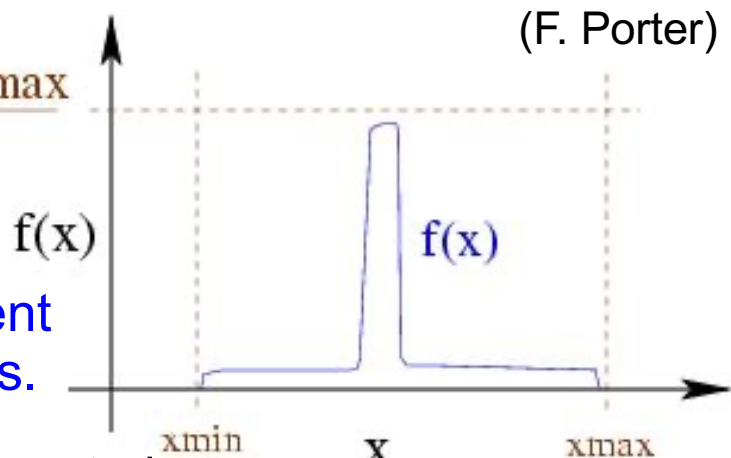
$$f(x) = \frac{3}{8}(1 + x^2) \quad (-1 \leq x \leq 1)$$

$f_{\min} = 0$, $f_{\max} = 3/4$; x values of pairs of random numbers that lie below the curve are accepted. The distribution of the accepted x values shown below.





Acceptance–rejection method simple to apply but efficiency of algorithm depends on area ratio of pdf to enclosing box. Inefficient for "peaky" distributions.



Fraction of trial points accepted:

$$\varepsilon = \frac{\text{Area under curve}}{\text{Area of box}} = \frac{\int_{x_{\min}}^{x_{\max}} f(x) dx}{(f_{\max} - f_{\min})(x_{\max} - x_{\min})}$$

Importance sampling:

To improve acceptance-rejection method efficiency, use importance sampling method. First random numbers generated according to $g(x)$ satisfying $g(x) > f(x)$ all over whole x -interval. x 's generated according to $g(x)/\int g(x)$ (with e.g. inverse transform method), and x accepted if $u < f(x)$, where u random number $]0, g(x)[$, or weighted with a factor $f(x)/g(x)$

e.g. to estimate the integral of the tail of Gaussian distribution. $P(Y > 3)$ if Y Gaussian $N(\mu = 0, \sigma = 1)$: truth ≈ 0.00135 . $h(Y_i) = 1$ if $Y_i > 3$; $h(Y_i) = 0$ if $Y_i \leq 3$

Instead of using the original Gaussian, use an equivalent Gaussian with larger acceptance rate to reduce uncertainty of the estimate and converge much faster.

- Draw an iid sample $Y_1, \dots, Y_{100}$ from a $N(0, 1)$, then the estimator is

$$\text{integral } \hat{I} = \frac{1}{100} \sum_{i=1}^{100} h(Y_i)$$

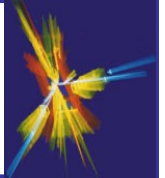
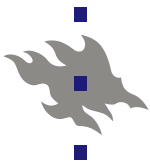
(J. Cisewski)

- Draw an iid sample $Y_1, \dots, Y_{100}$ from a $N(4, 1)$, then the estimator is

$$\hat{I} = \frac{1}{100} \sum_{i=1}^{100} \frac{h(Y_i)f(Y_i)}{g(Y_i)} \quad \text{importance sampling}$$

where f is the density of a $N(0, 1)$ and g is the density of $N(4, 1)$

$N = 10^5$	Expected Value	Variance
Truth	0.00135	0
Monte Carlo	0.00136	1.3×10^{-5}
Importance Sampling	0.00135	9.5×10^{-8}



MC methods often used to "mimic" data in e.g. particle physics, a two-step process: simulation of physics, "**event generator**" & simulation of response of experimental apparatus, "**detector simulation**".

MC event generator:

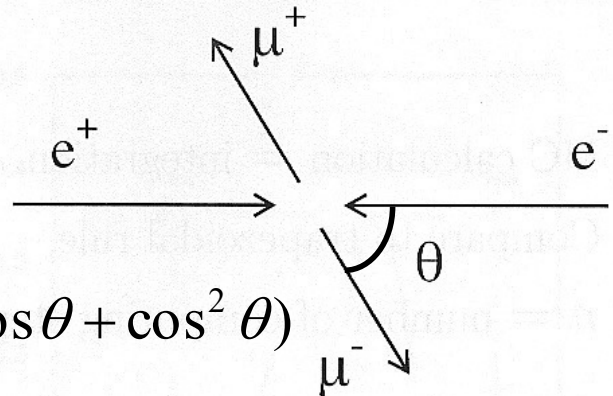
simple example:

$$e^+e^- \rightarrow \mu^+\mu^-$$

generate θ and ϕ :

$$f(\cos \theta; A_{FB}) \propto (1 + \frac{8}{3} A_{FB} \cos \theta + \cos^2 \theta)$$

$$g(\phi) = 1/2\pi$$



In reality implemented into program packages that accounts for all (known) effects (hadronisation, initial and final state radiation, longlived particles etc...)

MC detector simulation (built on e.g. GEANT4):

Input: particle list & momenta from event generator

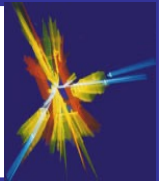
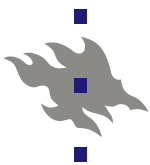
simulate particle interactions with detector material & detector response: multiple Coulomb scattering (generate particle scattering angle), ionization energy loss (generate energy loss dE/dx), electromagnetic & hadronic showers, produce detector signals, electronics response ...

Output: simulated "raw" data $\rightarrow$ input to reconstruction

Usage:

Predict what should be seen at "detector level" given a hypothesis at "generator level". Compare with real data.

- optimize measurement & experiment sensitivity.
- estimate efficiency & purity (expected signal & background)
- simulate measurement many times ("gedanken" experiment)



Standard Gaussian distributed random numbers

cumulative distribution function $F(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-x'^2/2} dx'$

Square evaluation method: Generate numbers in polar coordinates & afterwards transform them to cartesian

$$f(x)f(y)dxdy = re^{-r^2/2}dr \frac{d\phi}{2\pi} = g(r)h(\phi)drd\phi$$

r & ϕ generated according to $g(r)$ & $h(\phi) \Rightarrow$

x & y independent Gaussian distributed random numbers.

Marsaglia polar method: very efficient algorithm for generating Gaussian random numbers

u_1 & u_2 uniform in $]0,1[$

construct $v_1 = 2u_1 - 1$ & $v_2 = 2u_2 - 1$ (uniform in $]-1,1[$),

if $r^2 = v_1^2 + v_2^2 > 1$ start over again, otherwise

$$z_1 = v_1 \sqrt{-2 \ln r^2 / r^2} \quad \text{and} \quad z_2 = v_2 \sqrt{-2 \ln r^2 / r^2}$$

z_1 & z_2 independent Gaussian numbers with $\mu = 0$ & $\sigma = 1$

$z_i' = \mu + \sigma z_i$ Gaussian numbers with mean μ & variance σ^2

Poisson random numbers: efficient algorithm for small ν :

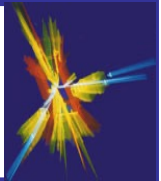
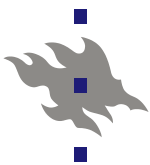
Set $k = 1$ & $A = 1$ at start, then iterate until successful choice

(i) generate u (uniform in $]0,1[$), replace A with uA

(ii) if $A < \exp(-\nu)$, where ν mean of Poisson distribution, accept $n_k = k-1$ & stop.

(iii) replace k by $k+1$ and repeat (i).

for large ν ($> \sim 10$), faster & easier to generate Gaussian random numbers since Poisson distribution $\approx$ Gaussian.



χ^2 random numbers: for even n , generate $n/2$ uniform random numbers u_i ; then $y = -2 \ln \left(\prod_{i=1}^{n/2} u_i \right)$ follows $\chi^2(n)$.
for odd n , generate $(n-1)/2$ uniform numbers u_i & one Gaussian z ; then $y = -2 \ln \left(\prod_{i=1}^{(n-1)/2} u_i \right) + z^2$ follows $\chi^2(n)$.

Binomial random numbers: principle same as for any discrete distribution. Use inverse transform method and tabulate cumulative distribution $F(x)$
Most computer libraries include generators for most common distributions like Gaussian, Poisson, χ^2 etc...

Precision of Monte Carlo methods:

MC = "integration".

compare to trapezoidal rule,
 n = # of computing steps

for 1D integral: MC: $n \propto$
number of accepted random
values, precision $\propto 1/\sqrt{n}$

trapezoid: $n \propto$ number of
subdivisions, precision $\propto 1/n^2$

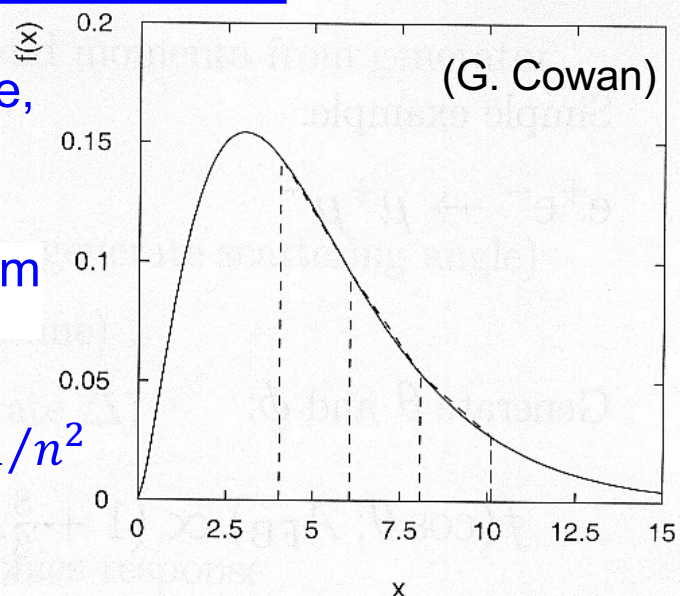
in 1D trapezoid wins!!!

but in d dimensions:

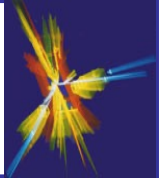
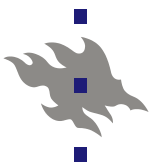
MC: precision $\propto 1/\sqrt{n}$

trapezoid: precision $\propto 1/n^{2/d}$

MC wins for $d > 4$. Gaussian quadrature better than trapezoid but for high enough d , MC always wins!!
(see e.g. F. James, *Rep. Prog. Phys.* 43 (1980) 1145).



← independent of d !



A statistical test:

applied to physics mainly in two different ways

- how to distinguish events of interest, "**signal**", from (a large number of) uninteresting events, "**background**"
- how well a given hypothesis compatible (without any alternative) with observed data, "**goodness-of-fit**" test.

Hypothesis testing:

Statistics related to questions where answers not numerical but logical – a "yes" or "no".

Example: "Is this particle an electron / a signal event?"

Statistics (& science in general) can never in absolute terms say that something is true.

Only falsify ("empirical falsification" Karl Popper 1930).

Answer generally a probability.

This probability refers to ensemble of statements.

Result of a measurement $\bar{x} = (x_1, \dots, x_n)$ follows a pdf in n -dimensional space that depends on variables $x_1, \dots, x_n$ e.g. pdf $f(\bar{x})$ specified by some **hypotheses** $H_0, H_1 \dots$ each having a probability density

$$f(\bar{x} | H_0), f(\bar{x} | H_1) \text{ etc ..}$$

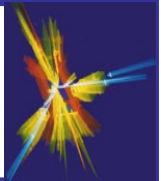
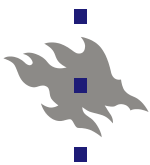
Equivalent to provide a statement & ask if true or not.

null hypothesis: H_0 , hypothesis whose validity tested.

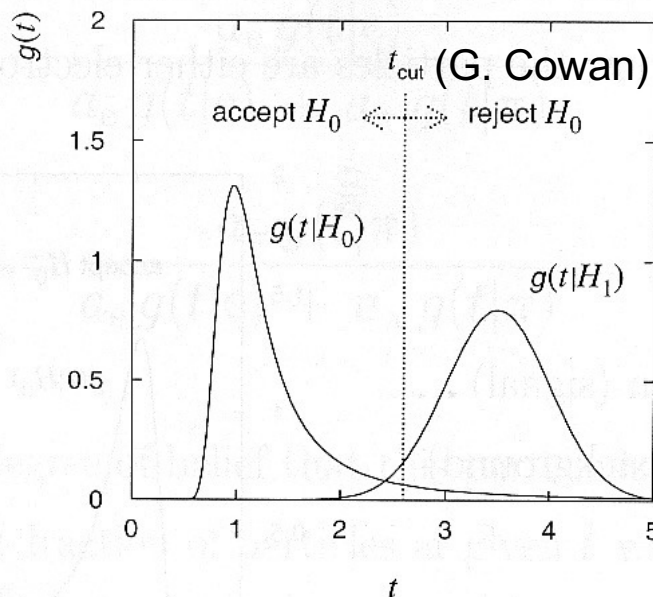
alternative hypothesis: $H_1, H_2 \dots$ to compare with H_0 .

simple hypothesis: $f(\bar{x})$ completely specified.

composite hypothesis: $f(\bar{x}, \theta)$, parameter(s) θ unknown.



Many dimensions awkward $\Rightarrow$ use **test statistic** $t(\bar{x})$ of lower dimension (e.g. 1D) to compactify data, keeping as much hypothesis discrimination power as possible. A test statistic t following pdf's $g(t(\bar{x}) | H_0), g(t(\bar{x}) | H_1)$



Often formulate compatibility between data & various hypotheses in terms of an acceptance or a rejection of null hypothesis H_0

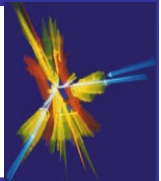
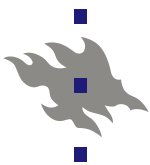
Critical region i.e. where t not likely to occur if H_0 true $t \geq t_{\text{cut}}$ in figure (alternative define **acceptance region**)

If observed value t_{obs} in critical region, reject H_0 , otherwise "accept" (or strictly speaking, not reject !!).

Critical region chosen such that probability for t to be there assuming hypothesis H_0 some small value $\alpha =$ **significance level**.
$$\alpha = \int_{t_{\text{cut}}}^{\infty} g(t | H_0) dt$$

H_0 rejection even if true error of 1st kind (= probability α)
 H_0 acceptance if some other hypothesis true error of 2nd kind (probability β)
$$\beta = \int_{-\infty}^{t_{\text{cut}}} g(t | H_1) dt$$

Can define $1-\beta$ as power to discriminate against alternative hypothesis.



n energy loss measurements for a particle in a particle detector, t = truncated mean of the measurements & suppose all particles either electrons (e) or pions (π).

H_0 = electron (signal)

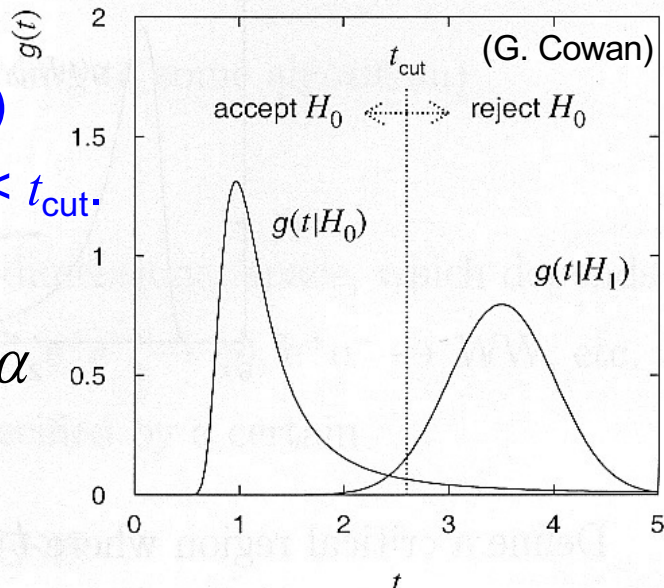
H_1 = pion (background)

select electrons with $t < t_{\text{cut}}$.

selection efficiencies:

$$\varepsilon_e = \int_{-\infty}^{t_{\text{cut}}} g(t | e) dt = 1 - \alpha$$

$$\varepsilon_\pi = \int_{-\infty}^{t_{\text{cut}}} g(t | \pi) dt = \beta$$



Higher t_{cut} : higher e efficiency but more π background.

Lower t_{cut} : lower e efficiency but better sample purity.

For observed value t , probability P , to have an e (or π)?

$$P(e | t) = \frac{a_e g(t | e)}{a_e g(t | e) + a_\pi g(t | \pi)} \quad P(\pi | t) = \frac{a_\pi g(t | \pi)}{a_e g(t | e) + a_\pi g(t | \pi)}$$

a_e (a_π) = fraction of electrons (pions) in sample; $a_e + a_\pi = 1$

Bayesian: degree of belief that particle is an e or a π .

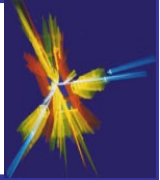
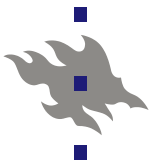
Frequentist: particle fraction at given t which is e or π .

sample purity, p_e , fraction of actual electrons in sample:

$$p_e = \int_{-\infty}^{t_{\text{cut}}} a_e g(t | e) dt / \int_{-\infty}^{t_{\text{cut}}} [a_e g(t | e) + a_\pi g(t | \pi)] dt$$

= electron probability averaged over interval $(-\infty, t_{\text{cut}}]$.

NB! purity depend on (un)known e & π fractions a_e & a_π .



When is discriminant analysis needed?

aim of discriminant analysis:

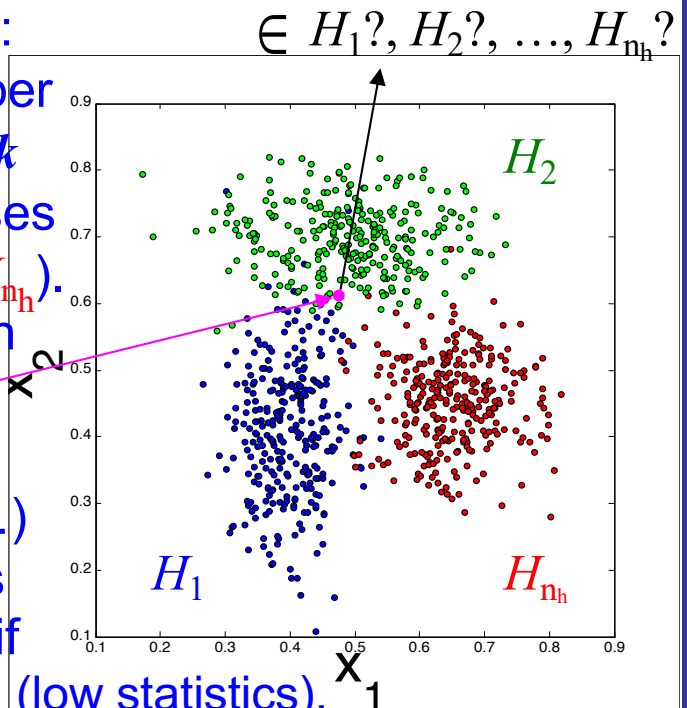
distinguish between a number of distinct hypotheses, H_k , $k = 1, \dots, n_h$.

various hypotheses known *a priori* ($H_1, H_2, \dots, H_{n_h}$).

problem: determine to which hypothesis new element x

(determined *a posteriori* by simulation, measurement,...) belongs.

Careful hypothesis testing required, especially if only few elements available



Discriminant analysis techniques applied in many fields e.g. astrophysics, particle physics, biophysics, imaging,...

A multidimensional test statistic $\bar{t} = (t_1, \dots, t_m)$ and hypotheses H_0 ("signal") & H_1 ("background"). Optimal choice of critical region, i.e. what selection to use?

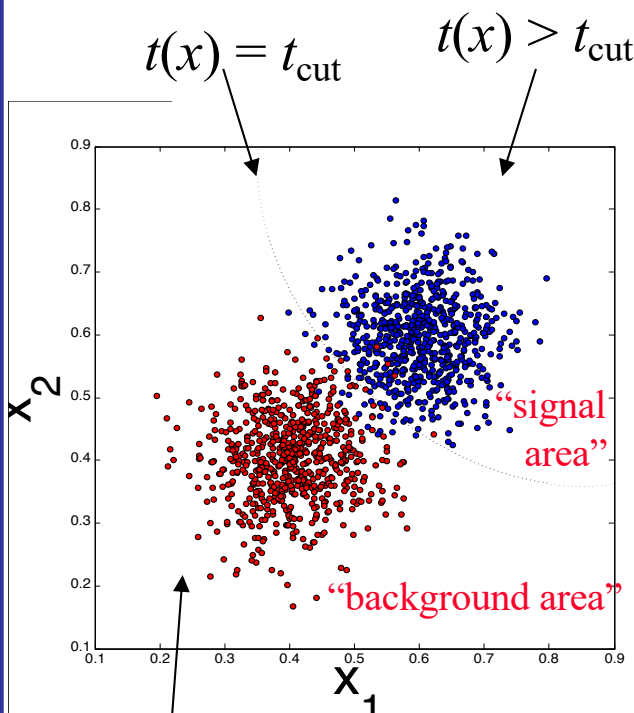
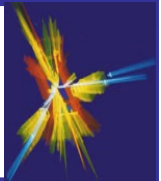
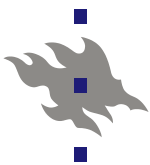
Neyman-Pearson lemma: acceptance region giving highest power (& also highest signal purity) for a given significance level α (or selection efficiency $\varepsilon = 1 - \alpha$):

$$\frac{g(\bar{t} | H_0)}{g(\bar{t} | H_1)} > c,$$

equivalently, optimal test statistic i.e. the **likelihood ratio** for simple hypotheses H_0 & H_1 . Requiring $r > c$ gives maximum significance level (efficiency) for a given power (purity).

$c = \text{constant, fixed by significance level (or selection purity)}$

$$r = \frac{g(\bar{t} | H_0)}{g(\bar{t} | H_1)},$$



A vector $\bar{x} = (x_1, \dots, x_n)$ measured for each event. construct 1D test statistic $t(\bar{x})$ to distinguish between two hypotheses H_0 & H_1 . most optimal choice: the likelihood ratio $t(\bar{x}) = \frac{f(\bar{x} | H_0)}{f(\bar{x} | H_1)}$, but need pdfs analytically $f(\bar{x} | H_0)$ & $f(\bar{x} | H_1)$ (in most cases not possible).

$t(x) < t_{\text{cut}}$

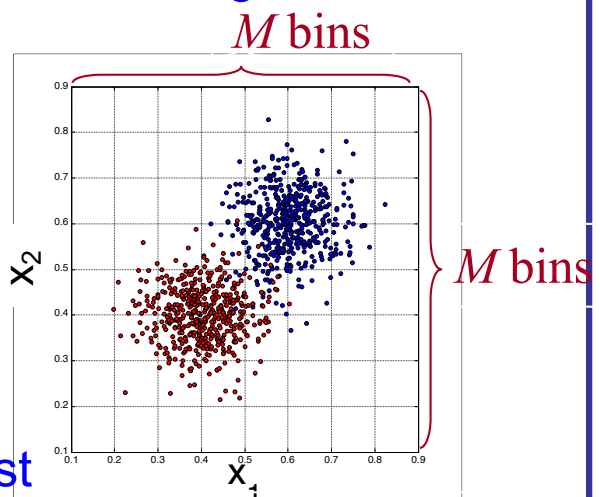
In practice calculated, from MC simulations, where pdf's approximated by multidimensional histograms filled with $\bar{x}$ of each generate event.

method impractical if number of input variables too large since available MC statistics finite.

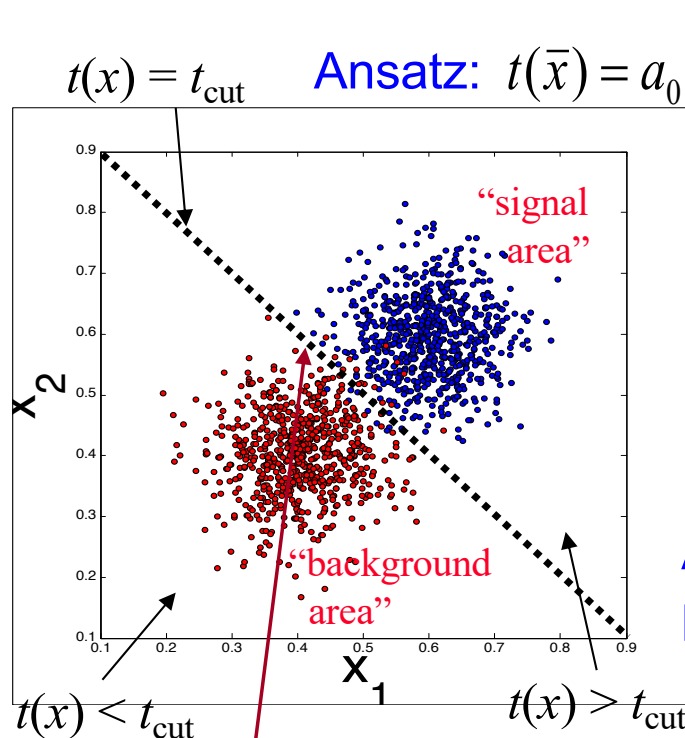
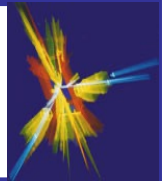
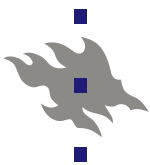
compromise: make Ansatz for functional form of $t(\bar{x})$ with a fewer number of input variables; choose variables (e.g. based on MC) that give best discrimination between H_0 & H_1 .

linear test statistic: e.g. Fisher discriminant – straight "line" cut in $\bar{x}$ -space:

$$t(\bar{x}) = \sum_{i=1}^n a_i x_i = \bar{a}^T \bar{x}$$



Number of parameters to be determined by MC: M^2
nonlinear test statistic: e.g. neural networks – more complicated cut function in $\bar{x}$ -space.



Ansatz: $t(\bar{x}) = a_0 + \sum_{i=1}^n a_i x_i = a_0 + \bar{a}^T \bar{x}$

$t(\bar{x})$ a **Fisher linear discriminant** see e.g.

R.A. Fisher, *Annals of Eugenics* 7 (1936) 179; R.A. Fisher: Contributions to mathematical statistics, 1950.

A choice of $\bar{a}$ gives certain pdf's $g(t | H_0)$ & $g(t | H_1)$

choose a_i 's to maximize “separation” between

$n = 2$, a t_{cut} can be interpreted a straight line in the x_1 - x_2 plane. $g(t | H_0)$ & $g(t | H_1)$

⇒ must define “separation” more exactly !!

Data $\bar{x} = (x_1, \dots, x_n)$ have mean values & covariance matrix

$$(\mu_k)_i = \int x_i f(\bar{x} | H_k) dx_1 \dots dx_n \quad i, j = 1, \dots, n$$

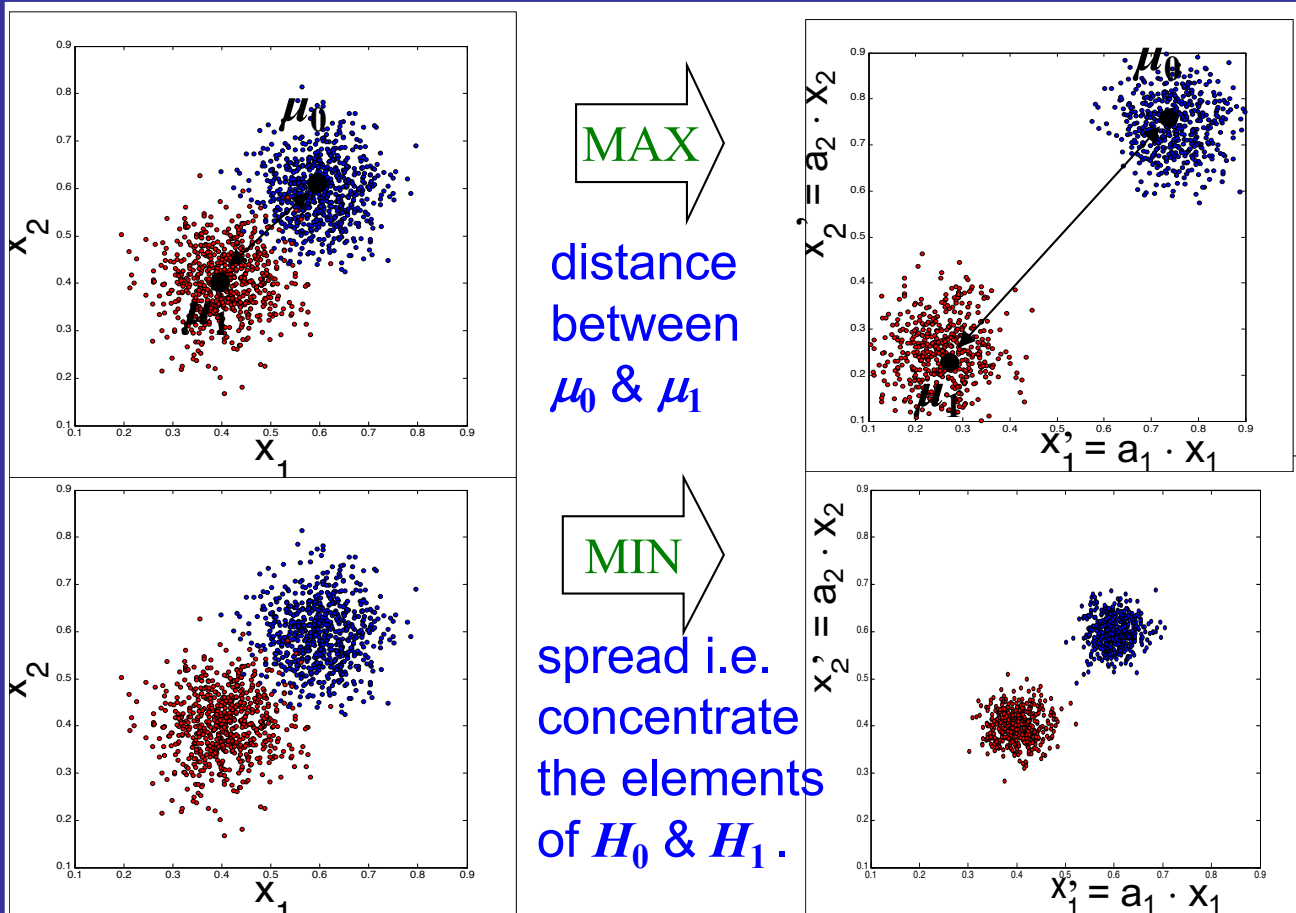
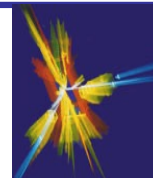
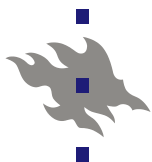
$$(V_k)_{ij} = \int (x - \mu_k)_i (x - \mu_k)_j f(\bar{x} | H_k) dx_1 \dots dx_n \quad k = 0, 1$$

Each hypothesis has certain mean & variance of $t(\bar{x})$

$$\tau_k = \int t(\bar{x}) g(t(\bar{x}) | H_k) dt = \bar{a}^T \bar{\mu}_k$$

$$\Sigma_k^2 = \int (t(\bar{x}) - \tau_k)^2 g(t(\bar{x}) | H_k) dt = \bar{a}^T \mathbf{V}_k \bar{a}$$

To optimize separation: large $|\tau_0 - \tau_1|$, small Σ_0^2 & Σ_1^2 (pdf's tightly concentrated about well separated means).



Fisher defined as a measure of separation (taking both into account): $J(\bar{a}) = (\tau_0 - \tau_1)^2 / (\Sigma_0^2 + \Sigma_1^2)$

Numerator & denominator can be written as

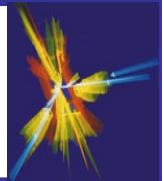
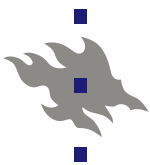
$$(\tau_0 - \tau_1)^2 = \sum_{i,j=1}^n a_i a_j (\mu_0 - \mu_1)_i (\mu_0 - \mu_1)_j = \sum_{i,j=1}^n a_i a_j B_{ij} = \bar{a}^T \mathbf{B} \bar{a}$$

$$\Sigma_0^2 + \Sigma_1^2 = \sum_{i,j=1}^n a_i a_j (\mathbf{V}_0 + \mathbf{V}_1)_{ij} = \bar{a}^T \mathbf{W} \bar{a}$$

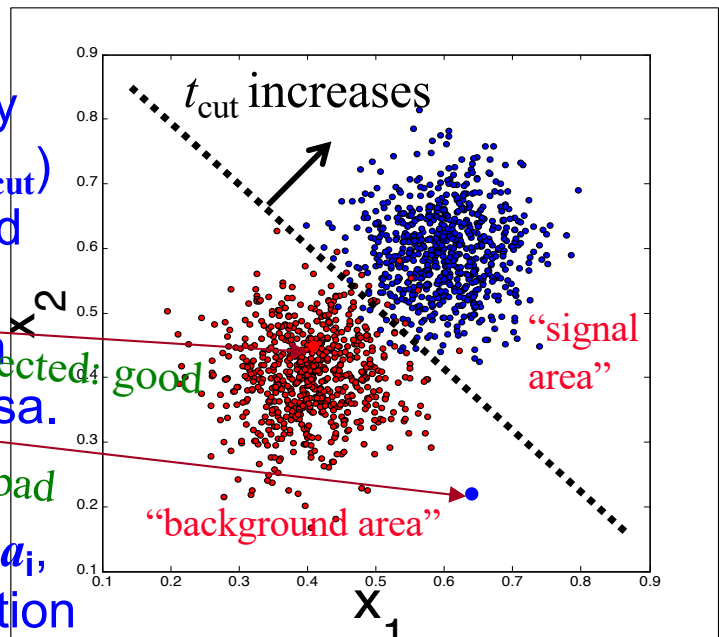
This gives $J(\bar{a}) = \frac{\bar{a}^T \mathbf{B} \bar{a}}{\bar{a}^T \mathbf{W} \bar{a}} = \frac{\text{separation between classes}}{\text{sum of variances within classes}}$

set $\partial J(\bar{a}) / \partial a_i = 0$, for all $i \Rightarrow \bar{a} \propto \mathbf{W}^{-1}(\bar{\mu}_0 - \bar{\mu}_1)$

Above **Fisher's linear discriminant function** & corresponding test statistic a **Fisher discriminant**.



Note coefficients a_i only determined up to arbitrary scale factor. Use scale (t_{cut}) & offset a_0 to fix purity and efficiency. Larger t_{cut} can improve signal purity with efficiency loss & vice-versa.



To determine coefficients a_i , need matrix $\mathbf{W}$ & expectation values $\mu_{(0,1)i}$. Usually estimated from a set of training data e.g. from a MC simulation. Important point being that one doesn't need to determine joint pdf's $f(\bar{x} | H_0)$ & $f(\bar{x} | H_1)$ as n -dimensional histograms, only means & covariances. Instead of determining M^n parameters now only $n(n+1)/2$.

Maximizing $J(\bar{a}) = \frac{(\tau_0 - \tau_1)^2}{(\Sigma_0^2 + \Sigma_1^2)}$ with fixed τ_0 & τ_1 same as

minimizing $\Sigma_0^2 + \Sigma_1^2 = E_0[(t(\bar{x}) - \tau_0)^2] + E_1[(t(\bar{x}) - \tau_1)^2]$

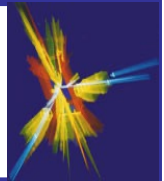
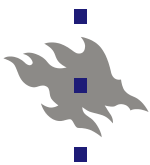
(E_k denotes expectation value under hypothesis H_k)

→ maximizing Fisher's $J(\bar{a})$ kind of **least square** problem. (more about parameter determination using least squares later)

Special Fisher discriminants: $f(\bar{x} | H_k)$ multidimensional Gaussians with same covariance matrix $\mathbf{V}_0 = \mathbf{V}_1 \equiv \mathbf{V}$

$$f(\bar{x} | H_k) = \frac{1}{(2\pi)^{n/2} |\mathbf{V}|^{1/2}} \exp\left[-\frac{1}{2}(\bar{x} - \bar{\mu}_k)^T \mathbf{V}^{-1}(\bar{x} - \bar{\mu}_k)\right], \quad k = 0, 1$$

and the Fisher discriminant: $t(\bar{x}) = a_0 + (\bar{\mu}_0 - \bar{\mu}_1)^T \mathbf{V}^{-1} \bar{x}$



Recall likelihood ratio (maximum efficiency at given purity):

$$r = \frac{f(\bar{x} | H_0)}{f(\bar{x} | H_1)} = \exp\left[\frac{1}{2}(\bar{x} - \bar{\mu}_1)^T \mathbf{V}^{-1}(\bar{x} - \bar{\mu}_1) - \frac{1}{2}(\bar{x} - \bar{\mu}_0)^T \mathbf{V}^{-1}(\bar{x} - \bar{\mu}_0)\right]$$

$$r = \exp\left[(\bar{\mu}_0 - \bar{\mu}_1)^T \mathbf{V}^{-1} \bar{x} - \frac{1}{2} \bar{\mu}_0^T \mathbf{V}^{-1} \bar{\mu}_0 + \frac{1}{2} \bar{\mu}_1^T \mathbf{V}^{-1} \bar{\mu}_1\right] \propto e^t$$

i.e. $t(\bar{x}) \propto \ln r + \text{const.}$ t given by monotonic function of r

$\Rightarrow$ **Fisher discriminant = likelihood ratio i.e. most optimal.**

NB! for non-equal $\mathbf{V}_i$ or non-Gaussian pdfs, no longer true.

Multidimensional Gaussian with equal covariance matrices also gives simple expressions for posterior hypotheses probabilities

e.g.
$$P(H_0 | \bar{x}) = \frac{f(\bar{x} | H_0)\pi_0}{f(\bar{x} | H_0)\pi_0 + f(\bar{x} | H_1)\pi_1} = \frac{1}{1 + (\pi_1/\pi_0)r},$$

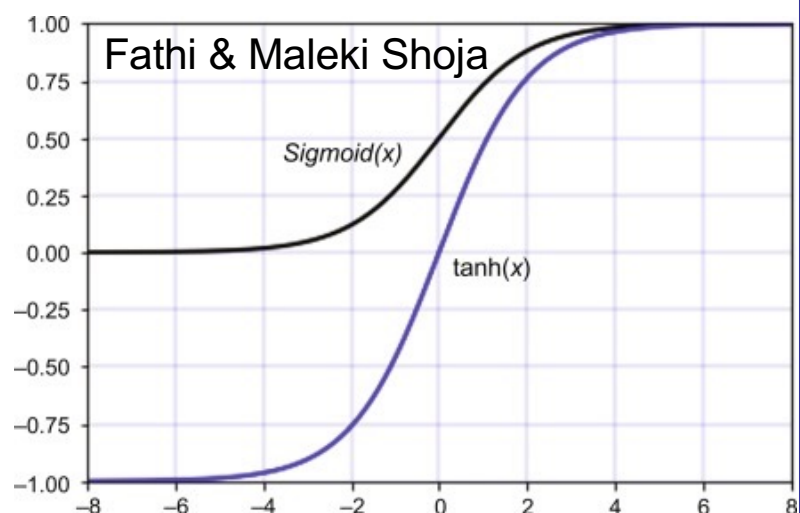
(Bayes' theorem)

where π_0 & π_1 prior probabilities for H_0 & H_1 .

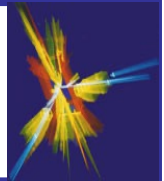
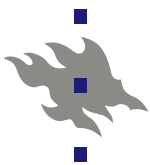
Combining this with above expression for r , one gets

$$P(H_0 | \bar{x}) = \frac{1}{1 + e^{-t}} \equiv s(t) \quad \text{the logistic sigmoid function (see figure below)}$$

Logistic sigmoid a very common neural network **activation function** (= giving the weight of the node in the network). Another common one is the hyperbolic tangent:



$$P(H_0 | \bar{x}) = \tanh(t) = (e^t - e^{-t}) / (e^t + e^{-t})$$

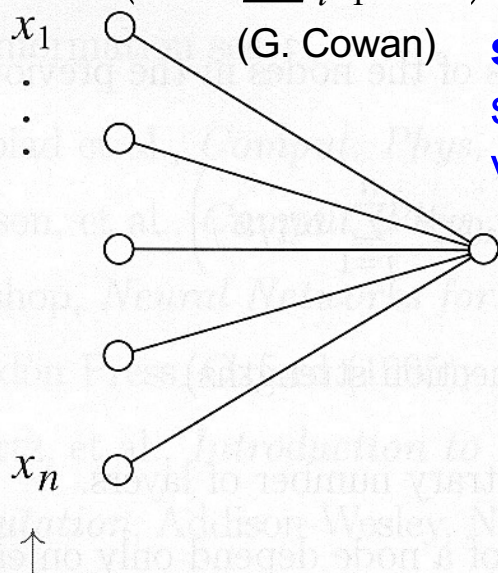


Optimal decision boundary mostly not "straight line" (due to non-equal V_i or non-Gaussian pdfs), use instead nonlinear test statistics: **neural networks, boosted decision trees...**

Artificial Neural Networks (ANN) today used in many fields: medical imaging, pattern recognition, financial forecasting etc... but also in physics. Assume $t(\bar{x})$ to have the form

$$t(\bar{x}) = s\left(a_0 + \sum_{i=1}^n a_i x_i\right) \quad \text{where } s(\cdot) \text{ activation function (often logistic sigmoid or } \tanh(t))$$

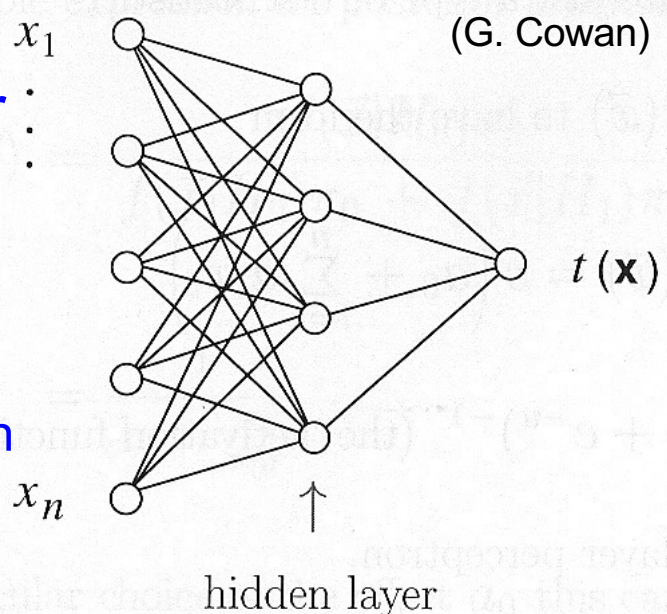
(G. Cowan)

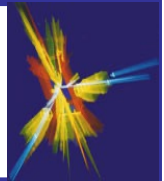
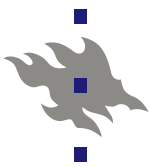


single-layer perceptron a test statistic of this form. $\bar{x}$ input values represented as a set of nodes. $s(\cdot)$ monotonic $\Rightarrow t(\bar{x})$ corresponds to a **linear test statistic**.

input layer

Generalized to **multilayer perceptrons (MLP)** with one or several hidden layers. Each hidden layer has m nodes ($h_1, \dots, h_m$). Usually connection are restricted: value of a given node only depends on nodes in previous layer, a **feed-forward network**.





In a two-layer perceptron output defined by

$$t(\bar{x}) = s\left(a_0 + \sum_{i=1}^m a_i h_i(\bar{x})\right), \text{ where } h_i(\bar{x}) = s\left(w_{i0} + \sum_{j=1}^n w_{ij} x_j\right)$$

$t(\bar{x})$ **non-linear test statistic**, since non-linear function of inputs x_i . a_i and w_i **connection strengths** or **weights**
(n+1)m free parameters for n input parameters. With more nodes, ANN closer to optimal but more parameters to fix.

Parameters determined by minimizing an error function

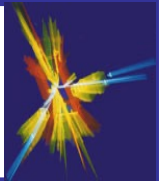
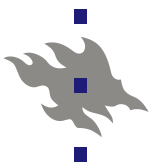
$$\varepsilon = E_0[(t - t^{(0)})^2] + E_1[(t - t^{(1)})^2]$$

where $t^{(0)}$ & $t^{(1)}$ target values e.g. 0 & 1 for logistic sigmoid, equivalent to least square method for Fisher discriminants.

To determine parameters minimizing ε , iterative numerical methods, network training or learning, used. In practice expectation values replaced by means computed from training samples, i.e. MC simulations. Learning starts with random initial values for weights and proceed evaluating function using training data. Weights adjusted to minimize ε by different methods, i.e. error-back-propagation.

Theorem: MLP with a single hidden layer having sufficient number of nodes can approximate arbitrarily well optimal decision boundary; *Leshno et al., Neural networks 6 (1993) 861*

Advantages/disadvantages of using a single hidden layer with many nodes over many hidden layers not known exactly but seems reasonable to assume that many layers have better performance & more stable than a single layer
 $\Rightarrow$ **deep learning** (networks with 10's or 100's of hidden layers)



Nonlinear test statistics

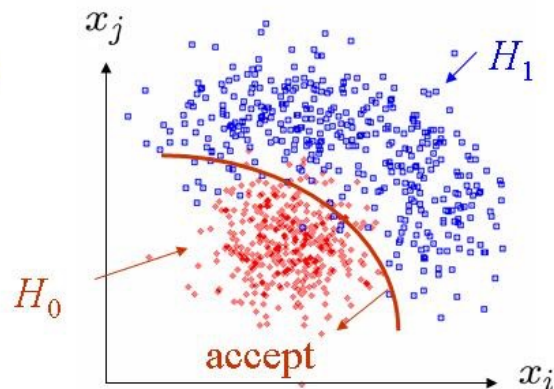
Focus here on supervised learning i.e. answer for each training event known

The optimal decision boundary may not be a hyperplane,

→ nonlinear test statistic $t(\vec{x})$

Multivariate statistical methods are a Big Industry:

- artificial neural networks
- boosted decision trees
- generative AI
- ...

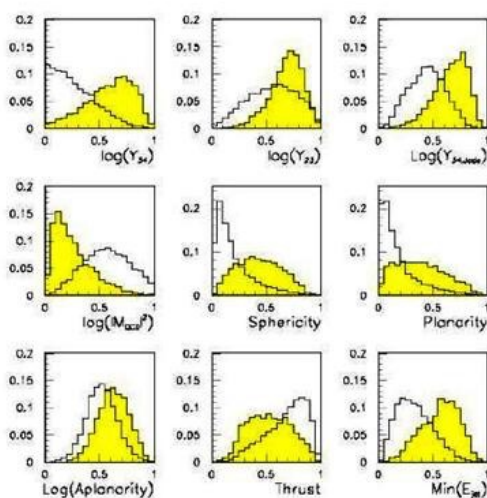


Physics can benefit from progress in Machine learning

Neural network example from LEP II

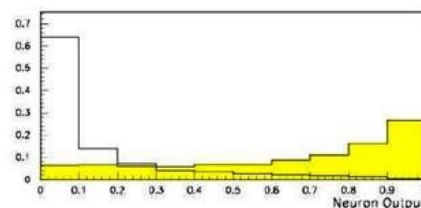
Signal: $e^+e^- \rightarrow W^+W^-$ (often 4 well separated hadron jets) yellow

Background: $e^+e^- \rightarrow q\bar{q}g\bar{g}$ (4 less well separated hadron jets) white



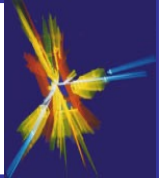
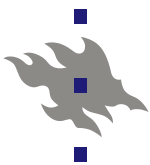
← input variables based on jet structure, event shape, ...
none by itself gives much separation.

Neural network output does better...



(Garrido, Juste and Martinez, ALEPH 96-144)

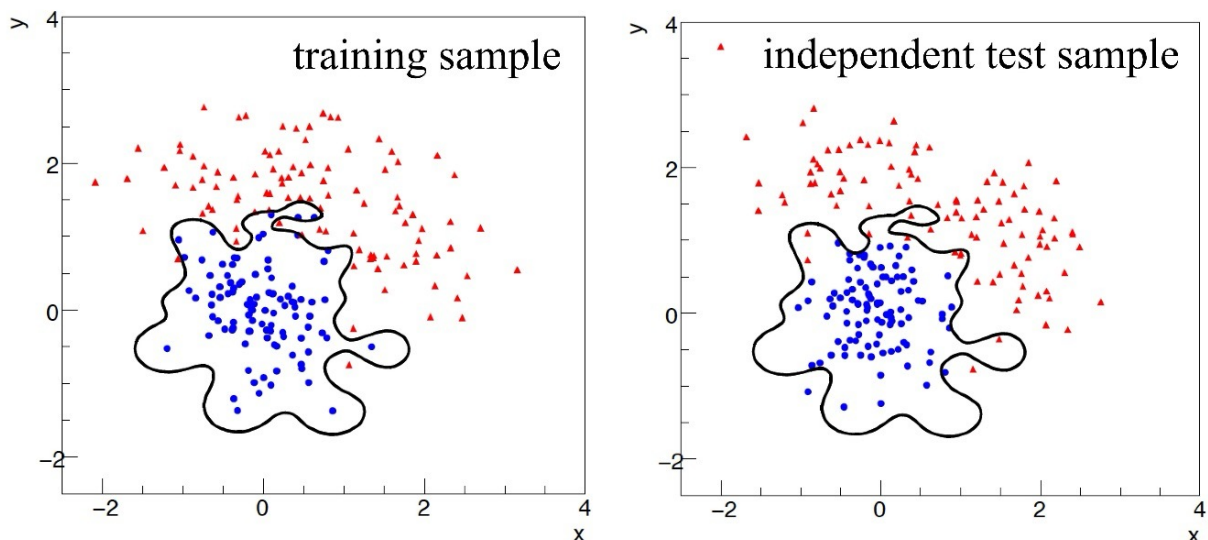
Machine learning in python: sklearn, <https://scikit-learn.org/>



Overtraining

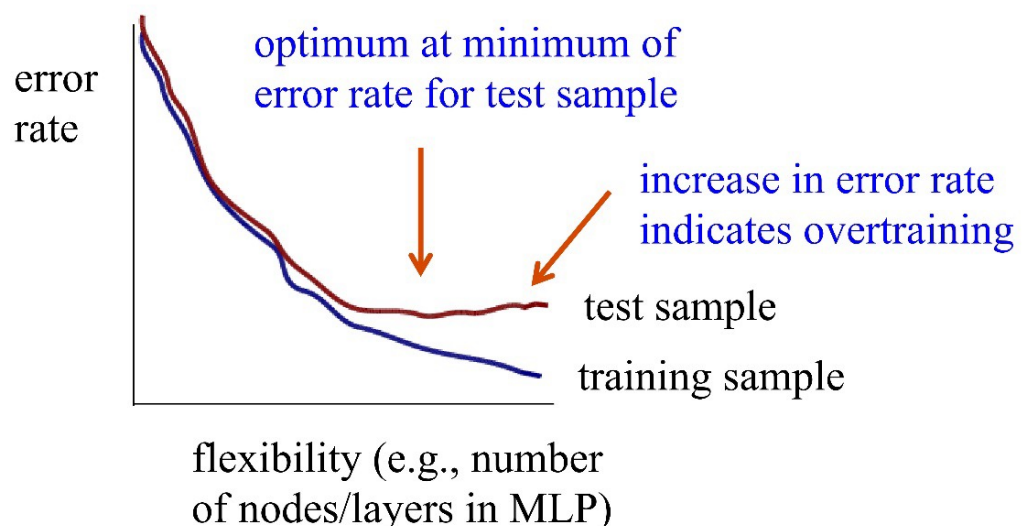
Including more parameters in a classifier makes its decision boundary increasingly flexible, e.g., more nodes/layers for a neural network.

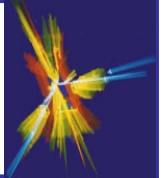
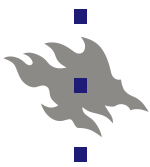
A “flexible” classifier may conform too closely to the training points; the same boundary will not perform well on an independent test data sample (→ “overtraining”).



Monitoring overtraining

If we monitor the fraction of misclassified events (or similar, e.g., error function $E(w)$) for test and training samples, it will usually decrease for both as the boundary is made more flexible:





Decision trees For multiclassification problems can work better in practice than e.g. neural networks

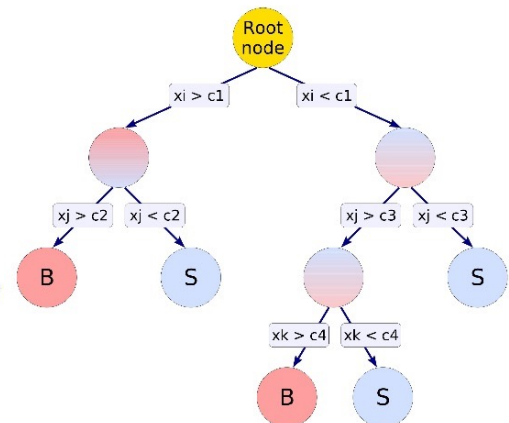
A training sample of signal and background data is repeatedly split by successive cuts on its input variables.

Order in which variables used based on best separation between signal and background.

Iterate until stop criterion reached, based e.g. on purity, minimum number of events in a node.

Resulting set of cuts is a ‘**decision tree**’.

Tends to be sensitive to fluctuations in training sample.



Boosted decision trees

sklearn implementation:
Randomforestregressor

Boosting combines a number classifiers into a stronger one; improves stability with respect to fluctuations in input data.

To use with decision trees, increase the weights of misclassified events and reconstruct the tree.

Iterate → forest of trees (perhaps > 1000). For the m th tree,

$$T_m(\vec{x}) = \begin{cases} 1 & \vec{x} \text{ in signal acceptance region} \\ -1 & \text{otherwise} \end{cases}$$

application:

Jan Welti, PhD
thesis in physics,
HU-P-D235

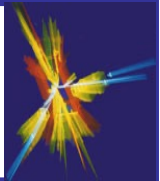
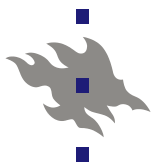
(2017)

Define a score α_m based on error rate of m th tree.

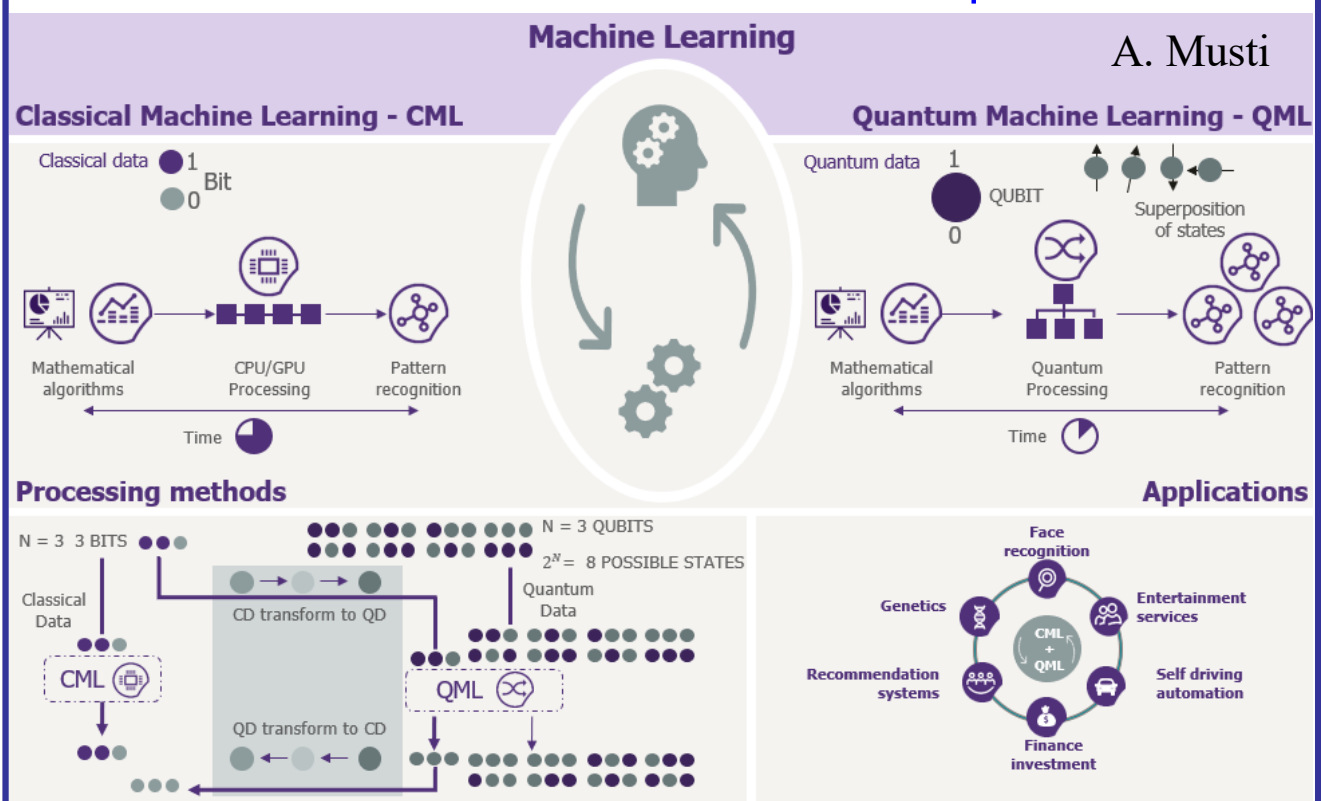
Boosted tree = weighted sum of the trees: $T(\vec{x}) = \sum_m \alpha_m T_m(\vec{x})$

Algorithms: AdaBoost (Freund & Schapire), ϵ -boost (Friedman).

bagging = draw bootstrapped samples from training sample, create tree for each & finally combine them



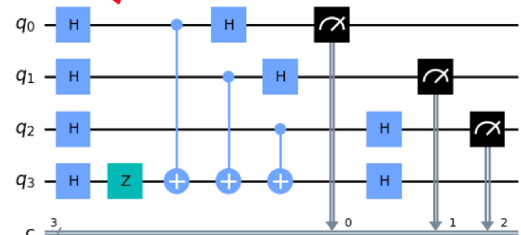
- When data points projected in higher and higher dimensions, becomes harder for a classical computer to deal with it. Even if it does, it takes too much time.
- **Sometimes, classical machine learning algorithms too taxing for classical computers.**
- **Solution: quantum computers?** They use superposition & entanglement to solve problems (potentially) much faster than their classical counterparts.

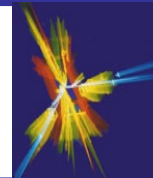
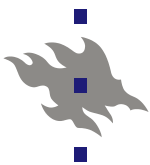


- Operations on qubits are unitary matrices
- One can encode classical data into the circuit by parametrizing these unitary operations
 - one can encode up to 2^n floats into the qubits
- Measurement leads to only an n-bit string (or n floats if running the circuit many times as one usually does)

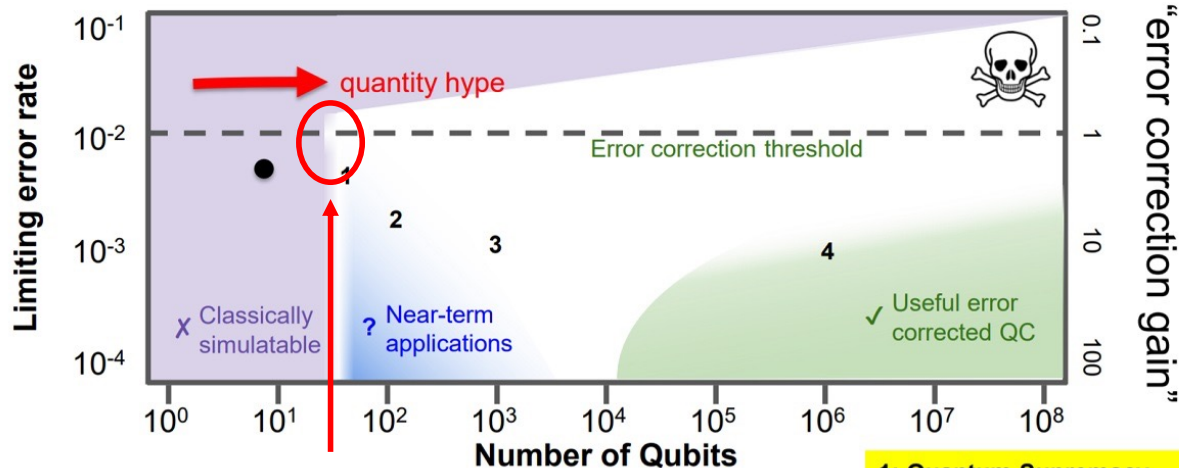


V. Mehtola





Need Both Quality and Quantity



we are somewhere here today

- 1: Quantum Supremacy
- 2: Look for near-term apps
- 3: Error correction
- 4: Full QC

Google strategy

First trial in particle physics in Finland:

V. Mehtola MSc
thesis 2025

quantum kernel
for selecting
Vector Boson
Scattering events
from background



Need sufficient
number of qubits
to remove "noise"
(= limit error rate)
to be competitive
with classical
machine learning.

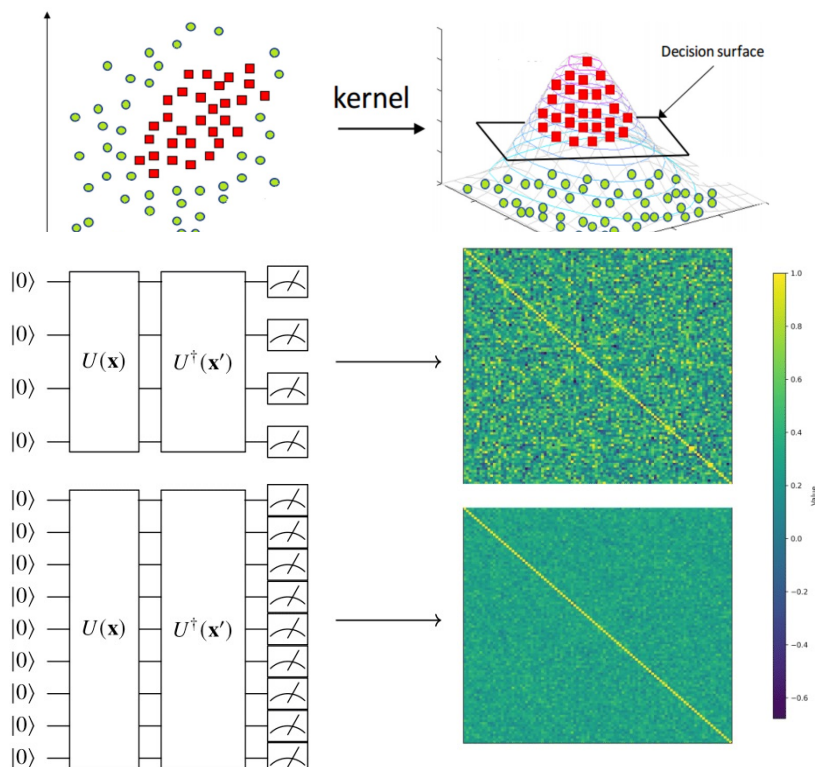
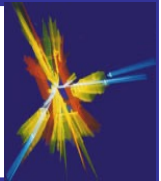
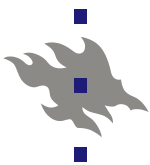


Figure 3: A figure for visualizing the general phenomenon of concentration (exponential or not). On the left, there are two quantum kernel circuits with same feature map scaled to four and nine qubits, respectively. To their right is their respective Gram matrix. For the nine-qubit version, off-diagonal variance is visibly lower. In the case that this scaling is exponential in the number of qubits, the feature map $U(\mathbf{x})$ does not scale well at least for the given data and pre-processing.



A simple example (2D)

Consider two variables, x_1 and x_2 , and suppose we have formulas for the joint pdfs for both signal (s) and background (b) events (in real problems the formulas are usually not available).

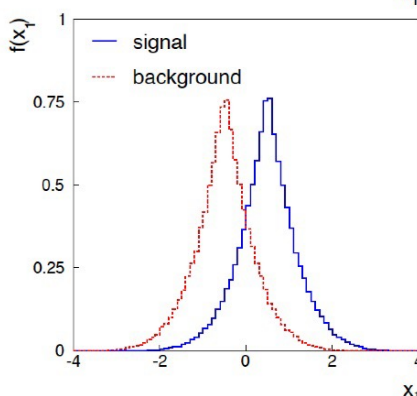
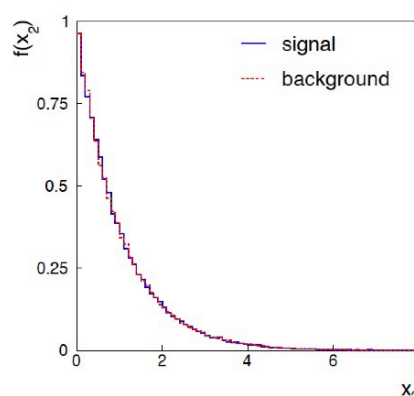
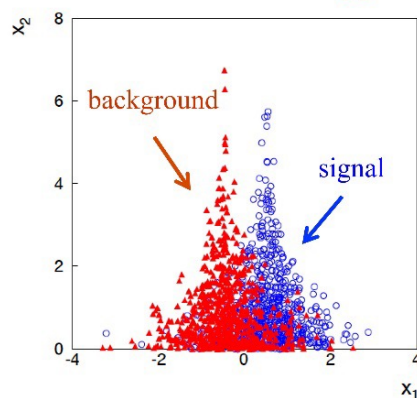
$f(x_1|x_2) \sim$ Gaussian, different means for s/b,
Gaussians have same σ , which depends on x_2 ,
 $f(x_2) \sim$ exponential, same for both s and b,
 $f(x_1, x_2) = f(x_1|x_2)f(x_2)$:

$$f(x_1, x_2|s) = \frac{1}{\sqrt{2\pi}\sigma(x_2)} e^{-(x_1 - \mu_s)^2 / 2\sigma^2(x_2)} \frac{1}{\lambda} e^{-x_2/\lambda}$$

$$f(x_1, x_2|b) = \frac{1}{\sqrt{2\pi}\sigma(x_2)} e^{-(x_1 - \mu_b)^2 / 2\sigma^2(x_2)} \frac{1}{\lambda} e^{-x_2/\lambda}$$

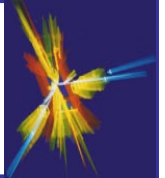
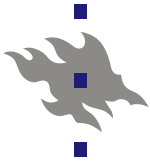
$$\sigma(x_2) = \sigma_0 e^{-x_2/\xi}$$

Joint and marginal distributions of x_1, x_2



Distribution $f(x_2)$ same for s, b.

So does x_2 help discriminate between the two event types?



Likelihood ratio for 2D example

Neyman-Pearson lemma says best critical region is determined by the likelihood ratio:

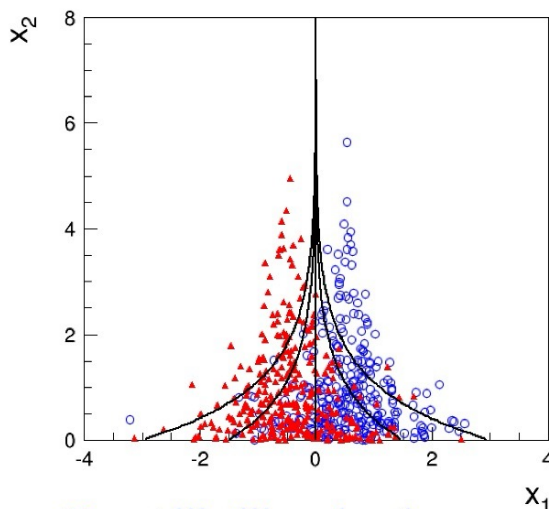
$$t(x_1, x_2) = \frac{f(x_1, x_2|s)}{f(x_1, x_2|b)}$$

Equivalently we can use any monotonic function of this as a test statistic, e.g.,

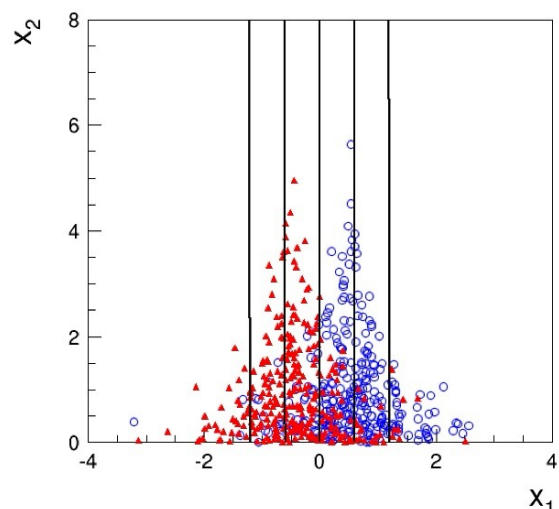
$$\ln t = \frac{\frac{1}{2}(\mu_b^2 - \mu_s^2) + (\mu_s - \mu_b)x_1}{\sigma_0^2 e^{-2x_2/\xi}}$$

Boundary of optimal critical region will be curve of constant $\ln t$, and this depends on x_2 !

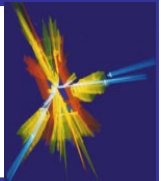
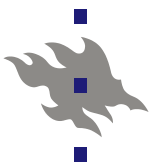
Contours of constant MVA output



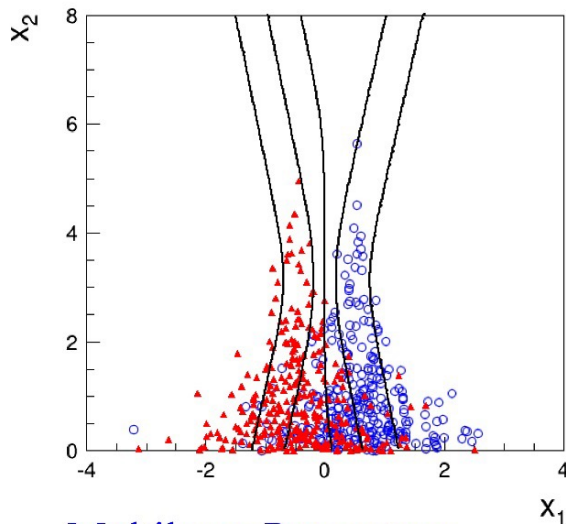
Exact likelihood ratio



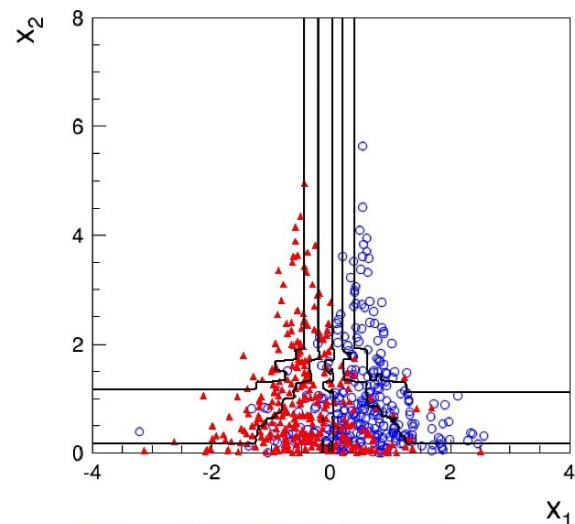
Fisher discriminant



Contours of constant MVA output



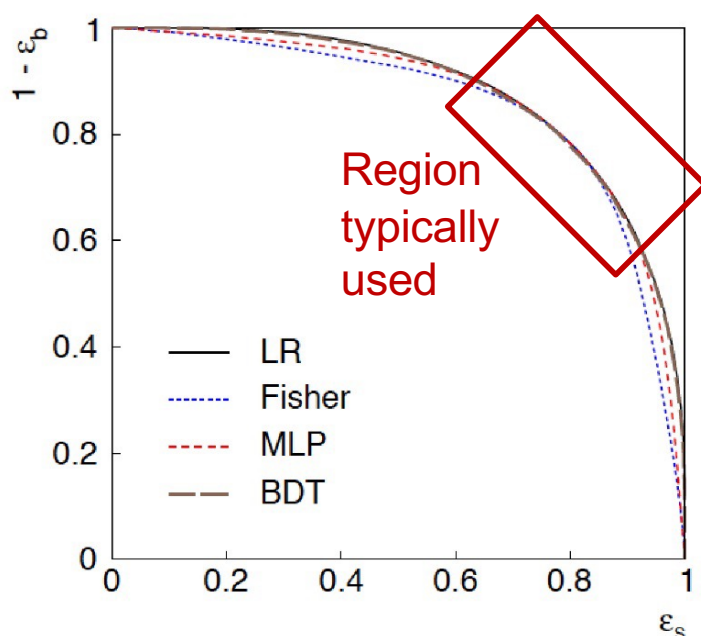
Multilayer Perceptron
1 hidden layer with 2 nodes



Boosted Decision Tree
200 iterations (AdaBoost)

Training samples: 10^5 signal and 10^5 background events

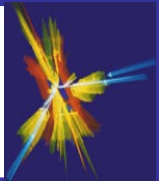
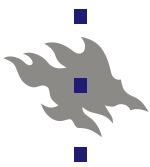
ROC curve



ROC = “receiver operating characteristic” (term from signal processing).

Shows (usually) background rejection ($1 - \epsilon_b$) versus signal efficiency ϵ_s .

Higher curve is better; usually analysis focused on a small part of the curve.



Multivariate analysis discussion

First clean the input of faulty, problematic or inadequate data

For all methods, need to check:

Sensitivity to statistically unimportant variables
(best to drop those that don't provide discrimination);

Level of smoothness in decision boundary (sensitivity
to over-training)

Given the test variable, next step is e.g., select n events and
estimate a cross section of signal: $\hat{\sigma}_s = (n - b)/\varepsilon_s L$

Now need to estimate systematic error...

If e.g. training (MC) data $\neq$ Nature, test variable is not optimal,
but not necessarily biased.

But our estimates of background b and efficiencies would then
be biased if based on MC. (True also for 'simple cuts'.)

But in a cut-based analysis it may be easier to avoid regions
where untested features of MC are strongly influencing the
decision boundary.

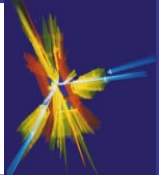
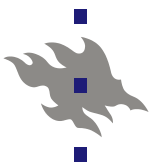
Look at control samples to test joint distributions of inputs.

Try to estimate backgrounds directly from the data (sidebands).

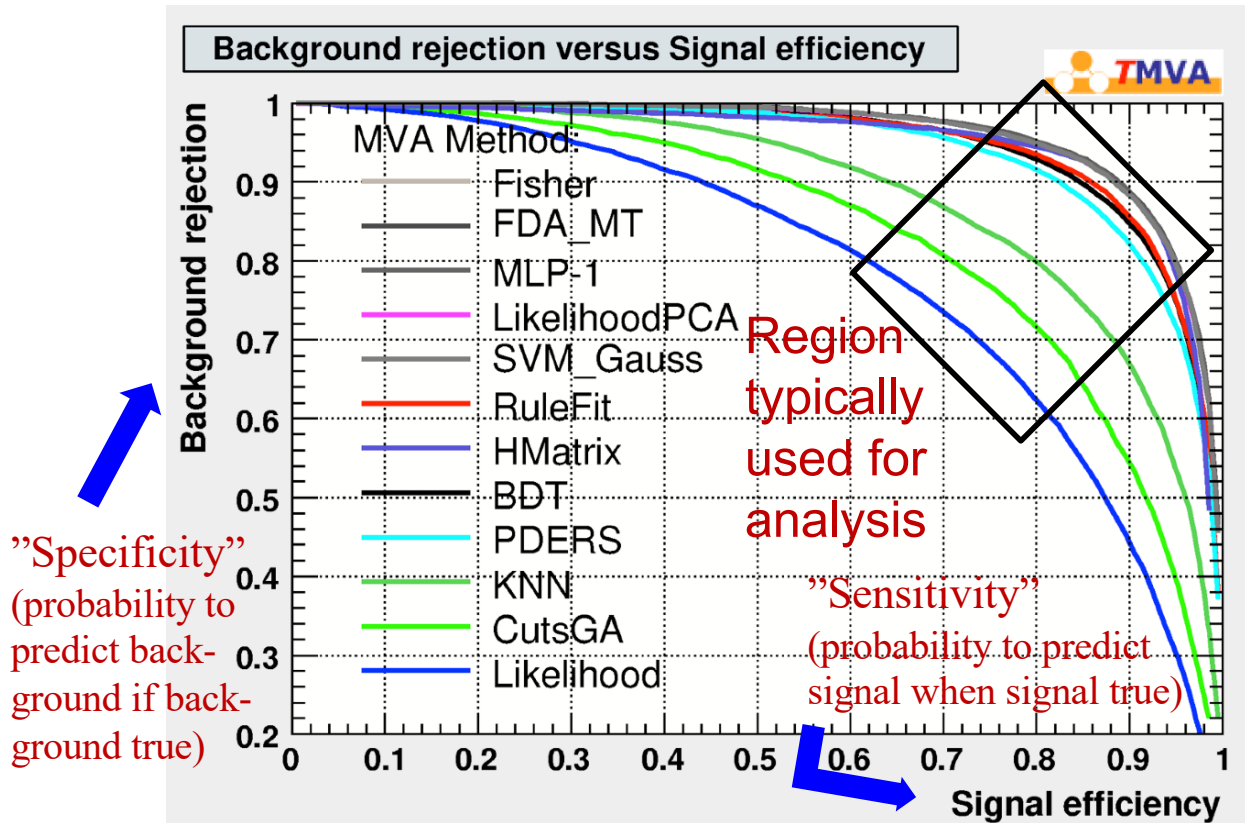
The purpose of the statistical test is often to select objects for
further study and then measure their properties.

Need to avoid input variables that are correlated with the
properties of the selected objects that you want to study.
(Not always easy; correlations may be poorly known.)

Also decide whether "hard" classification (defining explicitly signal
& background regions) is necessary or whether "soft" classification
(that gives simply signal & background probabilities) is sufficient.

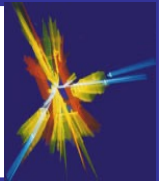
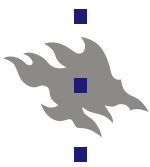


Toolkit for Multivariate Analysis (TMVA) included in ROOT-package enables usage of many techniques simultaneously:



Multivariate/machine learning techniques included:

- Neural networks
- Deep networks
- Multilayer perceptron (MLP)
- Boosted/Bagged decision trees (BDT)
- Support Vector Machine (SVM)
- Fisher and Function discriminant analysis (FDA)
- Multidimensional probability density estimation (PDE) range-search approach (PDERS)
- Multidimensional k-nearest neighbour classifier (KNN)
- Predictive learning via rule ensembles (RuleFit)
- Projective likelihood estimation (PDE approach)
- Rectangular cut optimisation



Resources on multivariate methods

Books:

C.M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006

T. Hastie, R. Tibshirani, J. Friedman, *The Elements of Statistical Learning*, Springer, 2001

R. Duda, P. Hart, D. Stork, *Pattern Classification*, 2nd ed., Wiley, 2001

A. Webb, *Statistical Pattern Recognition*, 2nd ed., Wiley, 2002

PDG review on machine learning:

<https://pdg.lbl.gov/2026/reviews/rpp2026-rev-machine-learning.pdf>

Software for multivariate analysis

TMVA, Höcker, Stelzer, Tegenfeldt, Voss, Voss, physics/0703039

From **tmva.sourceforge.net**, also distributed with ROOT

Variety of classifiers

<https://root.cern.ch/>

Good manual

SKLEARN - Machine Learning in Python

- Simple and efficient tools for predictive data analysis
- Built on NumPy, SciPy, and matplotlib

<https://scikit-learn.org/stable/>

KERAS – Python interface to many different neural network packages <https://keras.io/>

PyTorch – Python interface for neural networks in Torch programme package <https://pytorch.org/>

**Dedicated course at University of Helsinki: DATA11002
Introduction to Machine Learning 5 ECTS (period 2/2026)**