Statistical Data Analysis
Stat 2: Monte Carlo Method, Statistical Tests

London Postgraduate Lectures on Particle Physics;

University of London MSci course PH4515

Glen Cowan
Physics Department
Royal Holloway, University of London
g.cowan@rhul.ac.uk
www.pp.rhul.ac.uk/~cowan

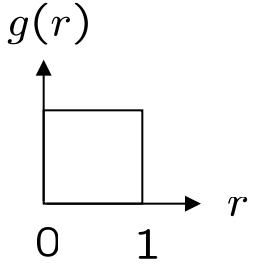
Course web page:

www.pp.rhul.ac.uk/~cowan/stat_course.html

The Monte Carlo method

What it is: a numerical technique for calculating probabilities and related quantities using sequences of random numbers.

The usual steps:



(1) Generate sequence $r_1, r_2, \ldots, r_m$ uniform in [0, 1].

(2) Use this to produce another sequence $x_1, x_2, \ldots, x_n$ distributed according to some pdf $f(x)$ in which we're interested (x can be a vector).

(3) Use the x values to estimate some property of $f(x)$, e.g., fraction of x values with $a < x < b$ gives $\int_a^b f(x)\,dx$.

→ MC calculation = integration (at least formally)

MC generated values = 'simulated data'

→ use for testing statistical procedures

Random number generators

Goal: generate uniformly distributed values in [0, 1].

Toss coin for e.g. 32 bit number... (too tiring).

→ 'random number generator'

= computer algorithm to generate $r_1, r_2, \ldots, r_n$.

Example: multiplicative linear congruential generator (MLCG)

$n_{i+1} = (a\, n_i) \bmod m$, where

n_i = integer

a = multiplier

m = modulus

n_0 = seed (initial value)

N.B. mod = modulus (remainder), e.g. 27 mod 5 = 2.

This rule produces a sequence of numbers $n_0, n_1, \ldots$

Random number generators (2)

The sequence is (unfortunately) periodic!

Example (see Brandt Ch 4): $a = 3$, $m = 7$, $n_0 = 1$

$$n_1 = (3 \cdot 1) \bmod 7 = 3$$
$$n_2 = (3 \cdot 3) \bmod 7 = 2$$
$$n_3 = (3 \cdot 2) \bmod 7 = 6$$
$$n_4 = (3 \cdot 6) \bmod 7 = 4$$
$$n_5 = (3 \cdot 4) \bmod 7 = 5$$
$$n_6 = (3 \cdot 5) \bmod 7 = 1 \quad \leftarrow \text{ sequence repeats}$$

Choose a, m to obtain long period (maximum = $m - 1$); m usually close to the largest integer that can represented in the computer.

Only use a subset of a single period of the sequence.

Random number generators (3)

$r_i = n_i/m$ are in [0, 1] but are they 'random'?

Choose a, m so that the r_i pass various tests of randomness:

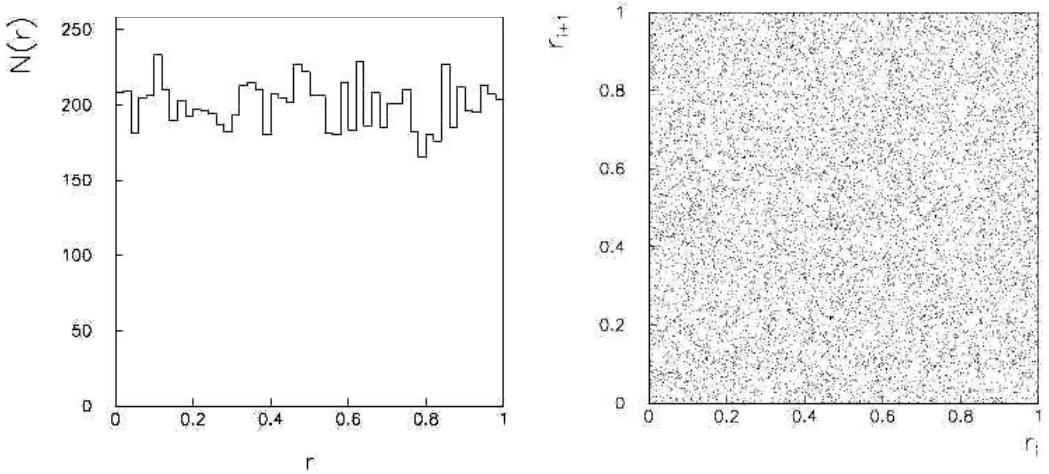
uniform distribution in [0, 1],

all values independent (no correlations between pairs),

e.g. L'Ecuyer, Commun. ACM **31** (1988) 742 suggests

$a = 40692$

$m = 2147483399$

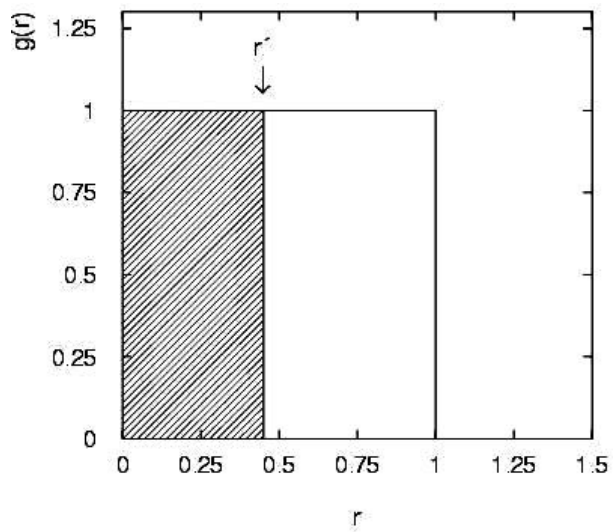


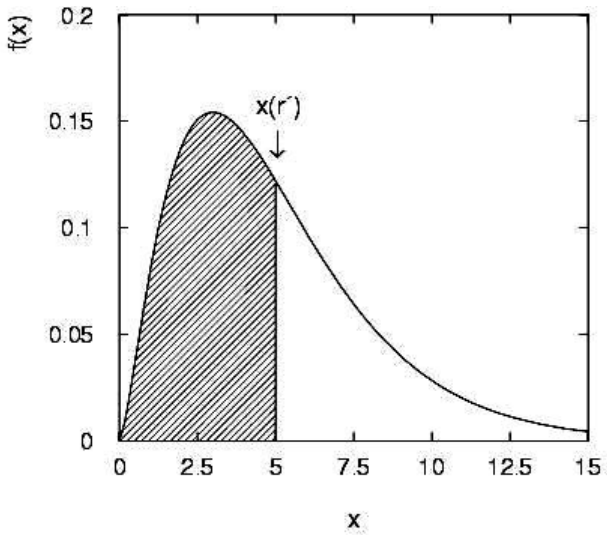
Far better generators available, e.g. `TRandom3`, based on Mersenne twister algorithm, period = $2^{19937} - 1$ (a "Mersenne prime").

See F. James, Comp. Phys. Comm. 60 (1990) 111; Brandt Ch. 4

The transformation method

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





Require: $P(r \leq r') = P(x \leq x(r'))$

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

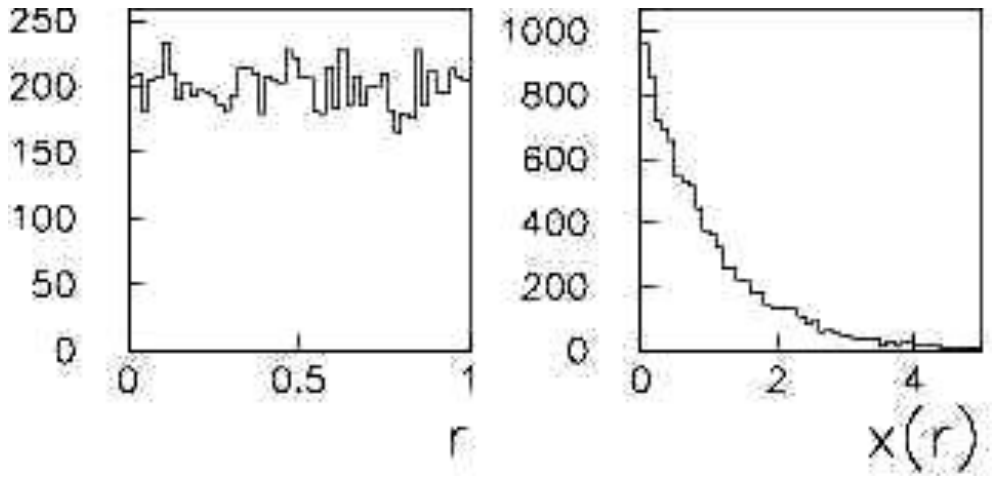
That is, set $F(x) = r$ and solve for $x(r)$.

Example of the transformation method

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

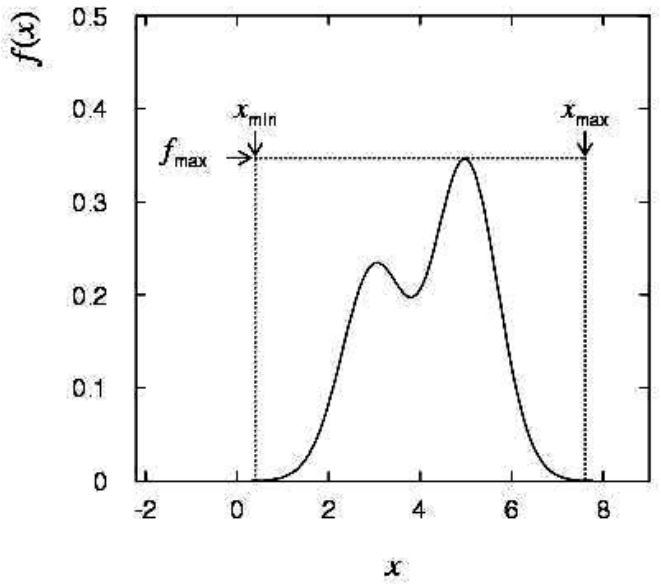
Set $\int_0^x \frac{1}{\xi} e^{-x'/\xi}\, dx' = r$ and solve for $x\,(r)$.

$\rightarrow \quad x(r) = -\xi \ln(1-r) \qquad (\, x(r) = -\xi \ln r$ works too.)



The acceptance-rejection method

Enclose the pdf in a box:



(1) Generate a random number x, uniform in $[x_{\text{min}}, x_{\text{max}}]$, i.e. $x = x_{\text{min}} + r_1(x_{\text{max}} - x_{\text{min}})$, r_1 is uniform in [0,1].

(2) Generate a 2nd independent random number u uniformly distributed between 0 and f_{max}, i.e. $u = r_2 f_{\text{max}}$.

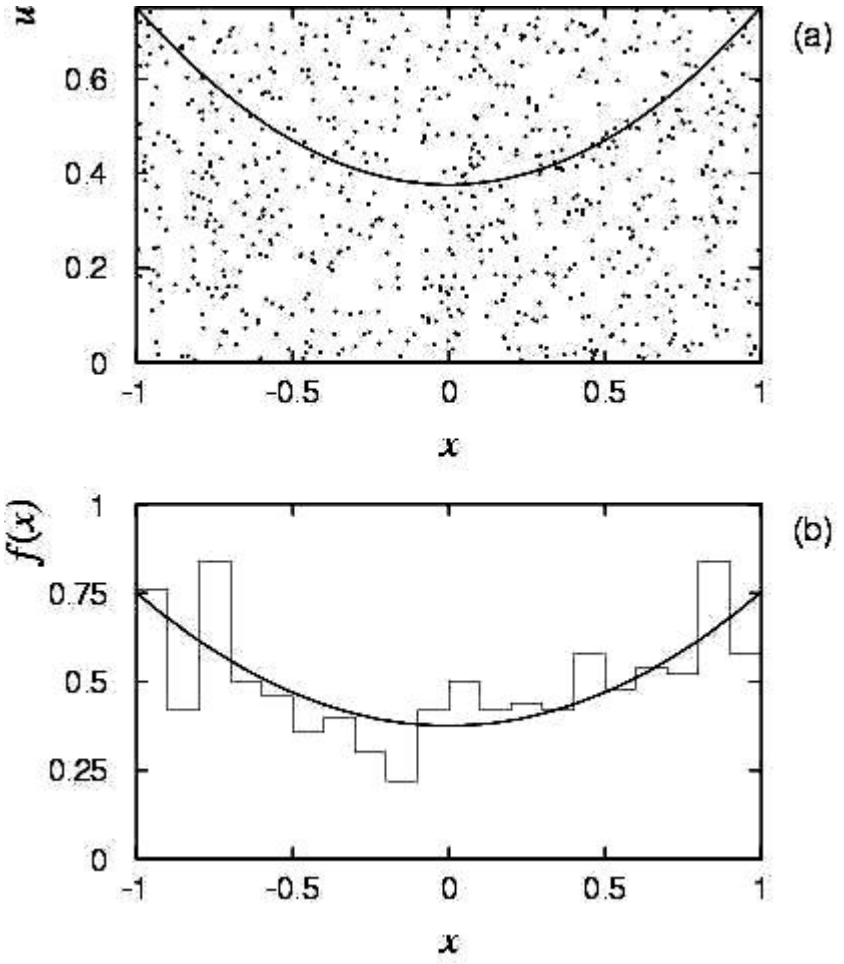
(3) If $u < f(x)$, then accept x. If not, reject x and repeat.

Example with acceptance-rejection method

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

$$(-1 \leq x \leq 1)$$

If dot below curve, use x value in histogram.

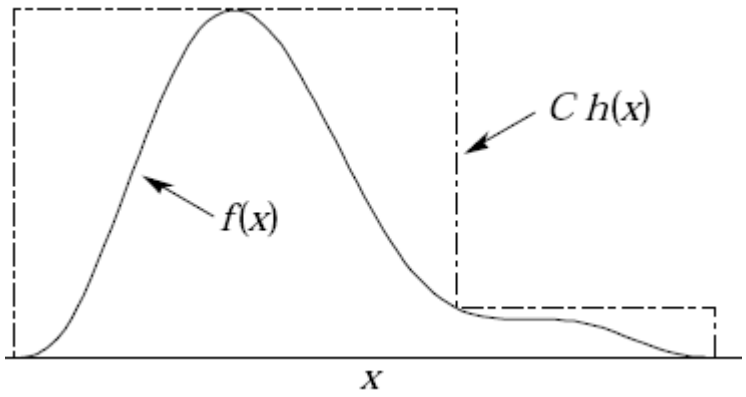


Improving efficiency of the acceptance-rejection method

The fraction of accepted points is equal to the fraction of the box's area under the curve.

For very peaked distributions, this may be very low and thus the algorithm may be slow.

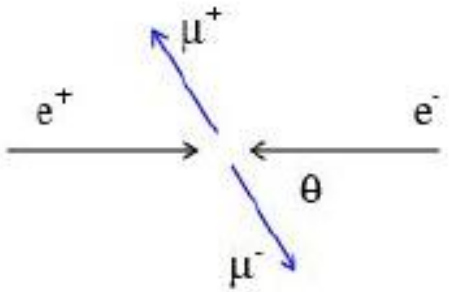
Improve by enclosing the pdf $f(x)$ in a curve $C\,h(x)$ that conforms to $f(x)$ more closely, where $h(x)$ is a pdf from which we can generate random values and C is a constant.



Generate points uniformly over $C\,h(x)$.

If point is below $f(x)$, accept x.

Monte Carlo event generators



Simple example: $e^+e^- \to \mu^+\mu^-$

Generate $\cos\theta$ and ϕ:

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

$$g(\phi) = \frac{1}{2\pi} \quad (0 \leq \phi \leq 2\pi)$$

Less simple: 'event generators' for a variety of reactions:

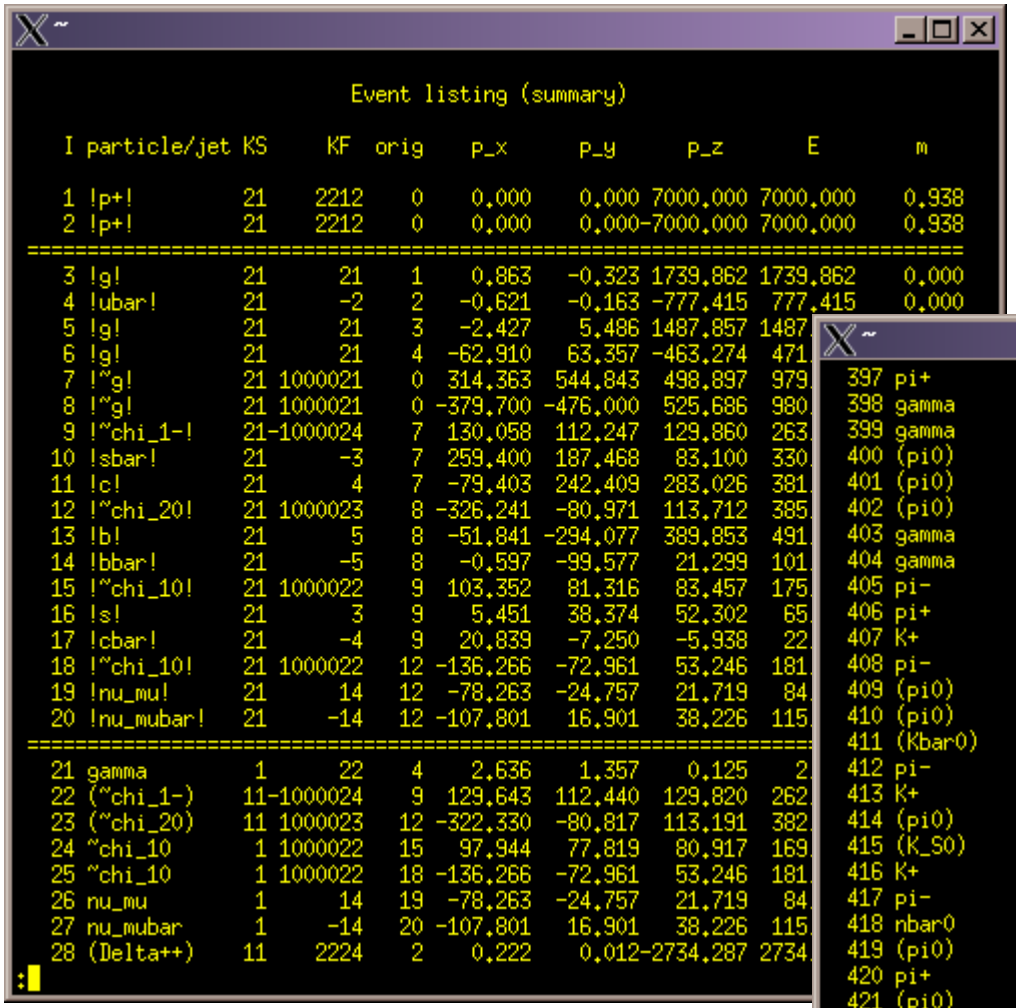
$e^+e^- \to \mu^+\mu^-$, hadrons, ...

$pp \to$ hadrons, D-Y, SUSY,...

e.g. PYTHIA, HERWIG, ISAJET...

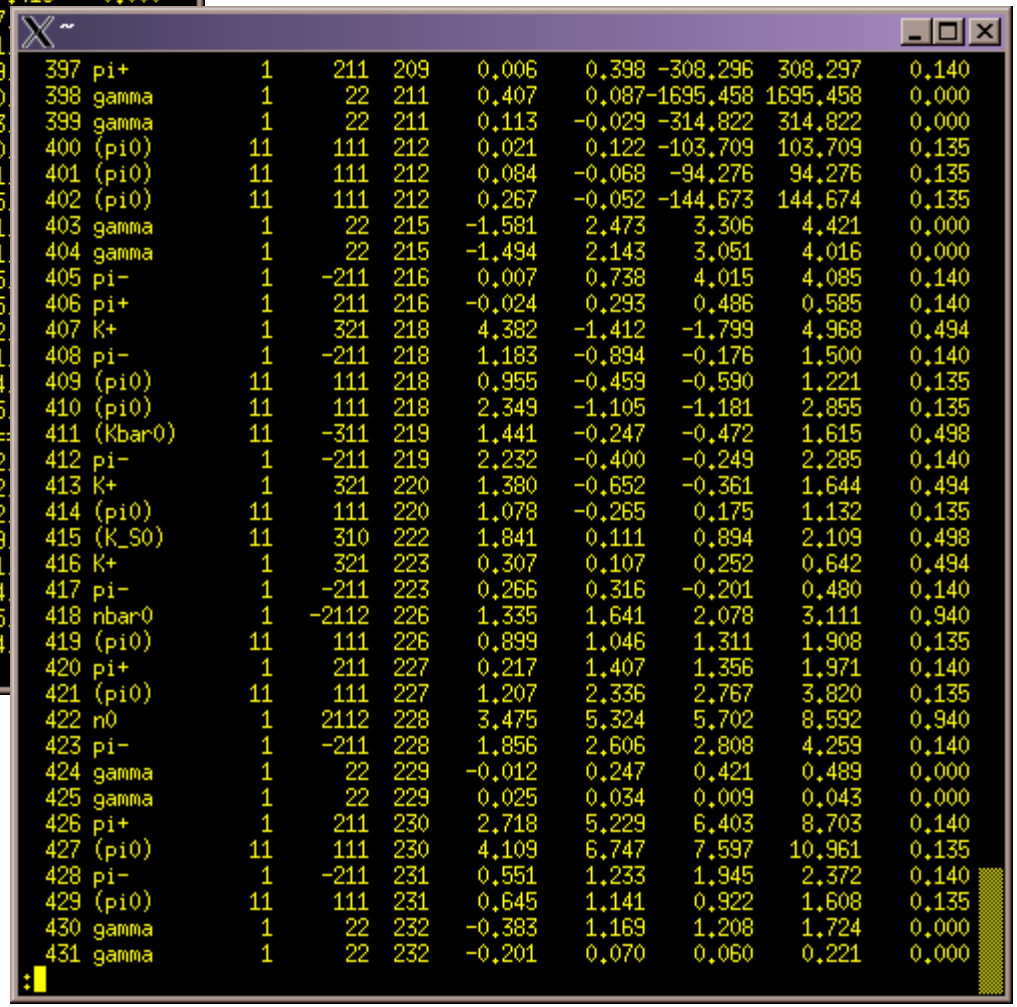
Output = 'events', i.e., for each event we get a list of generated particles and their momentum vectors, types, etc.

A simulated event



Event listing (summary)

I	particle/jet	KS	KF	orig	p_x	p_y	p_z	E	m
1	!p+!	21	2212	0	0.000	0.000	7000.000	7000.000	0.938
2	!p+!	21	2212	0	0.000	0.000	-7000.000	7000.000	0.938
3	!g!	21	21	1	0.863	-0.323	1739.862	1739.862	0.000
4	!ubar!	21	-2	2	-0.621	-0.163	-777.415	777.415	0.000
5	!g!	21	21	3	-2.427	5.486	1487.857	1487	
6	!g!	21	21	4	-62.910	63.357	-463.274	471	
7	!~g!	21	1000021	0	314.363	544.843	498.897	979	
8	!~g!	21	1000021	0	-379.700	-476.000	525.686	980	
9	!~chi_1-!	21	-1000024	7	130.058	112.247	129.860	263	
10	!sbar!	21	-3	7	259.400	187.468	83.100	330	
11	!c!	21	4	7	-79.403	242.409	283.026	381	
12	!~chi_20!	21	1000023	8	-326.241	-80.971	113.712	385	
13	!b!	21	5	8	-51.841	-294.077	389.853	491	
14	!bbar!	21	-5	8	-0.597	-99.577	21.299	101	
15	!~chi_10!	21	1000022	9	103.352	81.316	83.457	175	
16	!s!	21	3	9	5.451	38.374	52.302	65	
17	!cbar!	21	-4	9	20.839	-7.250	-5.938	22	
18	!~chi_10!	21	1000022	12	-136.266	-72.961	53.246	181	
19	!nu_mu!	21	14	12	-78.263	-24.757	21.719	84	
20	!nu_mubar!	21	-14	12	-107.801	16.901	38.226	115	
21	gamma	1	22	4	2.636	1.357	0.125	2	
22	(~chi_1-)	11	-1000024	9	129.643	112.440	129.820	262	
23	(~chi_20)	11	1000023	12	-322.330	-80.817	113.191	382	
24	~chi_10	1	1000022	15	97.944	77.819	80.917	169	
25	~chi_10	1	1000022	18	-136.266	-72.961	53.246	181	
26	nu_mu	1	14	19	-78.263	-24.757	21.719	84	
27	nu_mubar	1	-14	20	-107.801	16.901	38.226	115	
28	(Delta++)	11	2224	2	0.222	0.012	-2734.287	2734	



397	pi+	1	211	209	0.006	0.398	-308.296	308.297	0.140
398	gamma	1	22	211	0.407	0.087	-1695.458	1695.458	0.000
399	gamma	1	22	211	0.113	-0.029	-314.822	314.822	0.000
400	(pi0)	11	111	212	0.021	0.122	-103.709	103.709	0.135
401	(pi0)	11	111	212	0.084	-0.068	-94.276	94.276	0.135
402	(pi0)	11	111	212	0.267	-0.052	-144.673	144.674	0.135
403	gamma	1	22	215	-1.581	2.473	3.306	4.421	0.000
404	gamma	1	22	215	-1.494	2.143	3.051	4.016	0.000
405	pi-	1	-211	216	0.007	0.738	4.015	4.085	0.140
406	pi+	1	211	216	-0.024	0.293	0.486	0.585	0.140
407	K+	1	321	218	4.382	-1.412	-1.799	4.968	0.494
408	pi-	1	-211	218	1.183	-0.894	-0.176	1.500	0.140
409	(pi0)	11	111	218	0.955	-0.459	-0.590	1.221	0.135
410	(pi0)	11	111	218	2.349	-1.105	-1.181	2.855	0.135
411	(Kbar0)	11	-311	219	1.441	-0.247	-0.472	1.615	0.498
412	pi-	1	-211	219	2.232	-0.400	-0.249	2.285	0.140
413	K+	1	321	220	1.380	-0.652	-0.361	1.644	0.494
414	(pi0)	11	111	220	1.078	-0.265	0.175	1.132	0.135
415	(K_S0)	11	310	222	1.841	0.111	0.894	2.109	0.498
416	K+	1	321	223	0.307	0.107	0.252	0.642	0.494
417	pi-	1	-211	223	0.266	0.316	-0.201	0.480	0.140
418	nbar0	1	-2112	226	1.335	1.641	2.078	3.111	0.940
419	(pi0)	11	111	226	0.899	1.046	1.311	1.908	0.135
420	pi+	1	211	227	0.217	1.407	1.356	1.971	0.140
421	(pi0)	11	111	227	1.207	2.336	2.767	3.820	0.135
422	n0	1	2112	228	3.475	5.324	5.702	8.592	0.940
423	pi-	1	-211	228	1.856	2.606	2.808	4.259	0.140
424	gamma	1	22	229	-0.012	0.247	0.421	0.489	0.000
425	gamma	1	22	229	0.025	0.034	0.009	0.043	0.000
426	pi+	1	211	230	2.718	5.229	6.403	8.703	0.140
427	(pi0)	11	111	230	4.109	6.747	7.597	10.961	0.135
428	pi-	1	-211	231	0.551	1.233	1.945	2.372	0.140
429	(pi0)	11	111	231	0.645	1.141	0.922	1.608	0.135
430	gamma	1	22	232	-0.383	1.169	1.208	1.724	0.000
431	gamma	1	22	232	-0.201	0.070	0.060	0.221	0.000

PYTHIA Monte Carlo
pp → gluino-gluino

Monte Carlo detector simulation

Takes as input the particle list and momenta from generator.

Simulates detector response:

multiple Coulomb scattering (generate scattering angle),
particle decays (generate lifetime),
ionization energy loss (generate Δ),
electromagnetic, hadronic showers,
production of signals, electronics response, ...

Output = simulated raw data → input to reconstruction software:

track finding, fitting, etc.

Predict what you should see at 'detector level' given a certain hypothesis for 'generator level'. Compare with the real data.

Estimate 'efficiencies' = #events found / # events generated.

Programming package: **GEANT**

Hypotheses

A hypothesis H specifies the probability for the data, i.e., the outcome of the observation, here symbolically: x.

x could be uni-/multivariate, continuous or discrete.

E.g. write $x \sim f(x|H)$.

x could represent e.g. observation of a single particle, a single event, or an entire "experiment".

Possible values of x form the sample space S (or "data space").

Simple (or "point") hypothesis: $f(x|H)$ completely specified.

Composite hypothesis: H contains unspecified parameter(s).

The probability for x given H is also called the likelihood of the hypothesis, written $L(x|H)$.

Definition of a test

Goal is to make some statement based on the observed data x as to the validity of the possible hypotheses.

Consider e.g. a simple hypothesis H_0 and alternative H_1.

A test of H_0 is defined by specifying a critical region W of the data space such that there is no more than some (small) probability α, assuming H_0 is correct, to observe the data there, i.e.,

$$P(x \in W \,|\, H_0) \leq \alpha$$

If x is observed in the critical region, reject H_0.

α is called the size or significance level of the test.

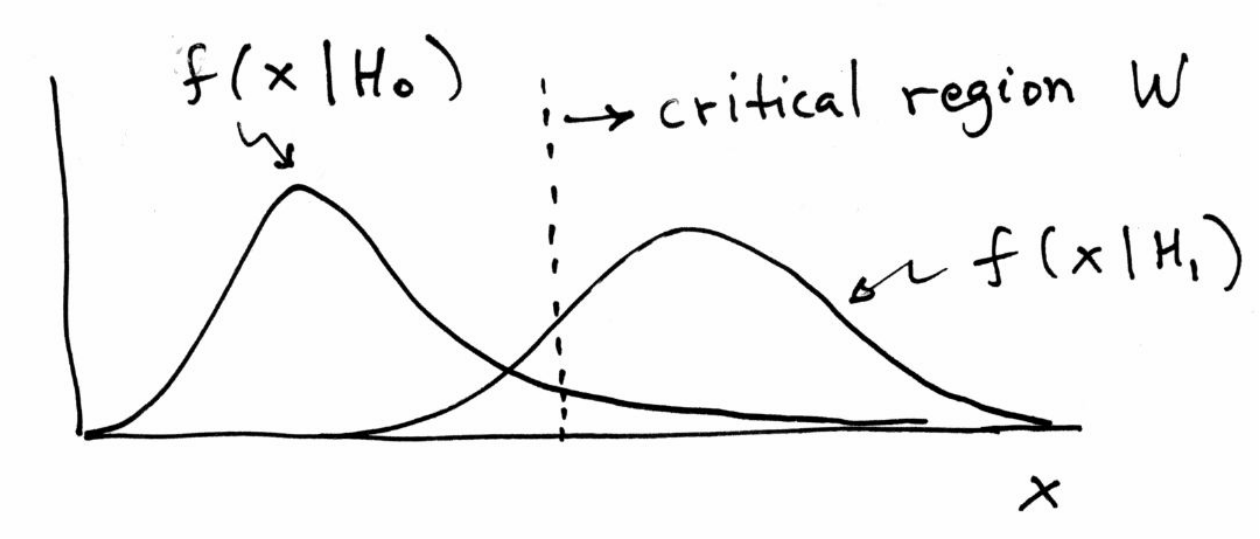
Critical region also called "rejection" region; complement is acceptance region.

Definition of a test (2)

But in general there are an infinite number of possible critical regions that give the same significance level α.

So the choice of the critical region for a test of H_0 needs to take into account the alternative hypothesis H_1.

Roughly speaking, place the critical region where there is a low probability to be found if H_0 is true, but high if H_1 is true:



Rejecting a hypothesis

Note that rejecting H_0 is not necessarily equivalent to the statement that we believe it is false and H_1 true. In frequentist statistics only associate probability with outcomes of repeatable observations (the data).

In Bayesian statistics, probability of the hypothesis (degree of belief) would be found using Bayes' theorem:

$$P(H|x) = \frac{P(x|H)\pi(H)}{\int P(x|H)\pi(H)\,dH}$$

which depends on the prior probability $\pi(H)$.

What makes a frequentist test useful is that we can compute the probability to accept/reject a hypothesis assuming that it is true, or assuming some alternative is true.

Type-I, Type-II errors

Rejecting the hypothesis H_0 when it is true is a Type-I error.

The maximum probability for this is the size of the test:

$$P(x \in W \,|\, H_0) \leq \alpha$$

But we might also accept H_0 when it is false, and an alternative H_1 is true.

This is called a Type-II error, and occurs with probability

$$P(x \in \mathrm{S} - W \,|\, H_1) = \beta$$

One minus this is called the power of the test with respect to the alternative H_1:

$$\text{Power} = 1 - \beta$$

Choosing a critical region

To construct a test of a hypothesis H_0, we can ask what are the relevant alternatives for which one would like to have a high power.

Maximize power wrt H_1 = maximize probability to reject H_0 if H_1 is true.

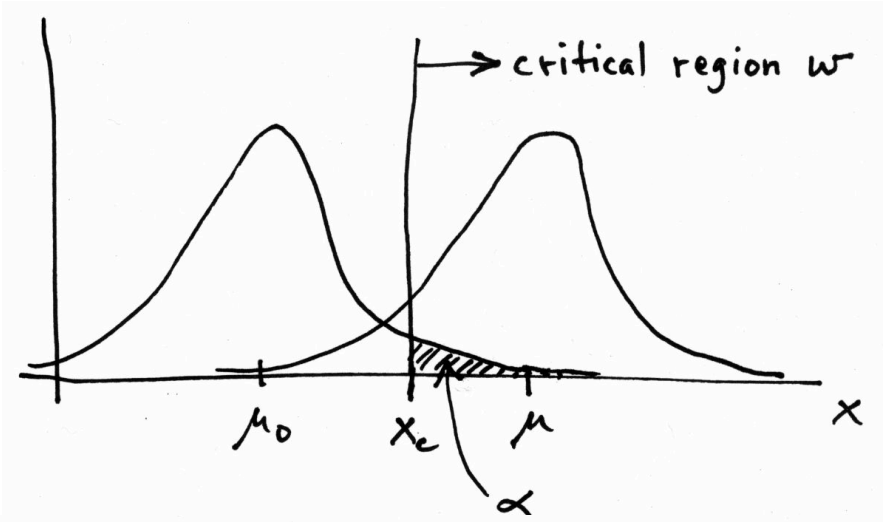
Often such a test has a high power not only with respect to a specific point alternative but for a class of alternatives.
E.g., using a measurement $x \sim \text{Gauss}\,(\mu, \sigma)$ we may test

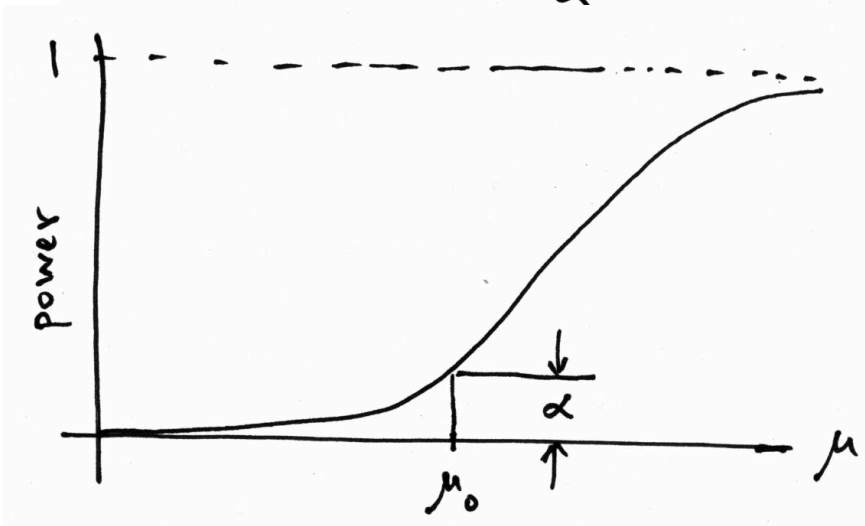
$$H_0 : \mu = \mu_0 \text{ versus the composite alternative } H_1 : \mu > \mu_0$$

We get the highest power with respect to any $\mu > \mu_0$ by taking the critical region $x \geq x_c$ where the cut-off x_c is determined by the significance level such that

$$\alpha = P(x \geq x_c | \mu_0).$$

Test of $\mu = \mu_0$ vs. $\mu > \mu_0$ with $x \sim \text{Gauss}(\mu,\sigma)$





Standard Gaussian cumulative distribution

$$\alpha = 1 - \Phi\left(\frac{x_c - \mu_0}{\sigma}\right)$$

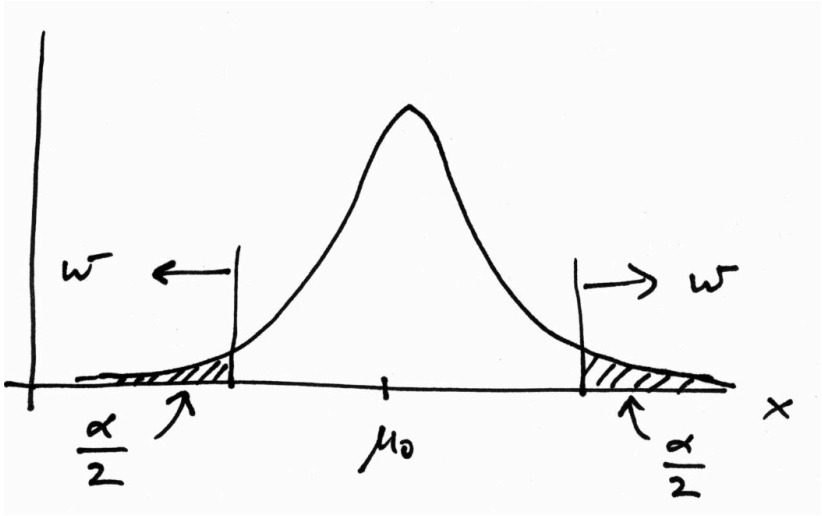
$$x_c = \mu_0 + \sigma\Phi^{-1}(1-\alpha)$$

Standard Gaussian quantile

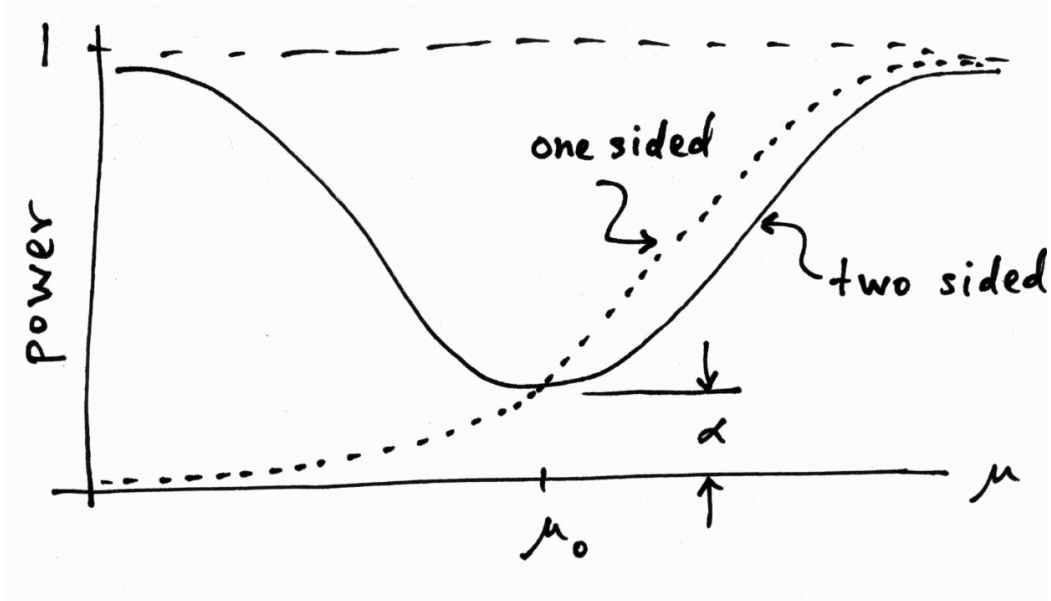
$$\text{power} = 1 - \beta = P(x > x_c|\mu) =$$

$$1 - \Phi\left(\frac{\mu_0 - \mu}{\sigma} + \Phi^{-1}(1-\alpha)\right)$$

Choice of critical region based on power (3)



But we might consider $\mu < \mu_0$ as well as $\mu > \mu_0$ to be viable alternatives, and choose the critical region to contain both high and low x (a two-sided test).



New critical region now gives reasonable power for $\mu < \mu_0$, but less power for $\mu > \mu_0$ than the original one-sided test.

No such thing as a model-independent test

In general we cannot find a single critical region that gives the maximum power for all possible alternatives (no "Uniformly Most Powerful" test).

In HEP we often try to construct a test of

H_0 : Standard Model (or "background only", etc.)

such that we have a well specified "false discovery rate",

α = Probability to reject H_0 if it is true,

and high power with respect to some interesting alternative,

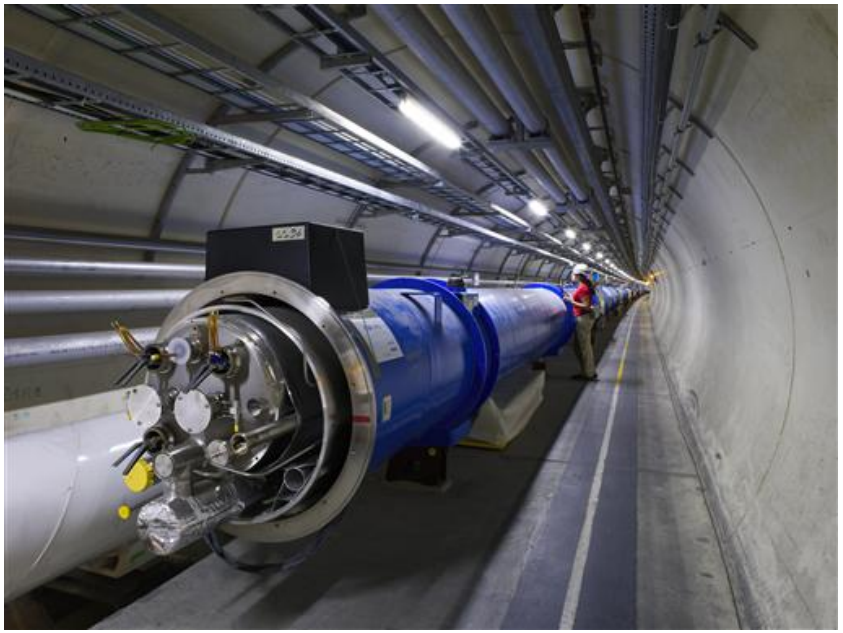
H_1 : SUSY, Z', etc.

But there is no such thing as a "model independent" test. Any statistical test will inevitably have high power with respect to some alternatives and less power with respect to others.

Example setting for statistical tests: the Large Hadron Collider

Counter-rotating proton beams in 27 km circumference ring

pp centre-of-mass energy 14 TeV

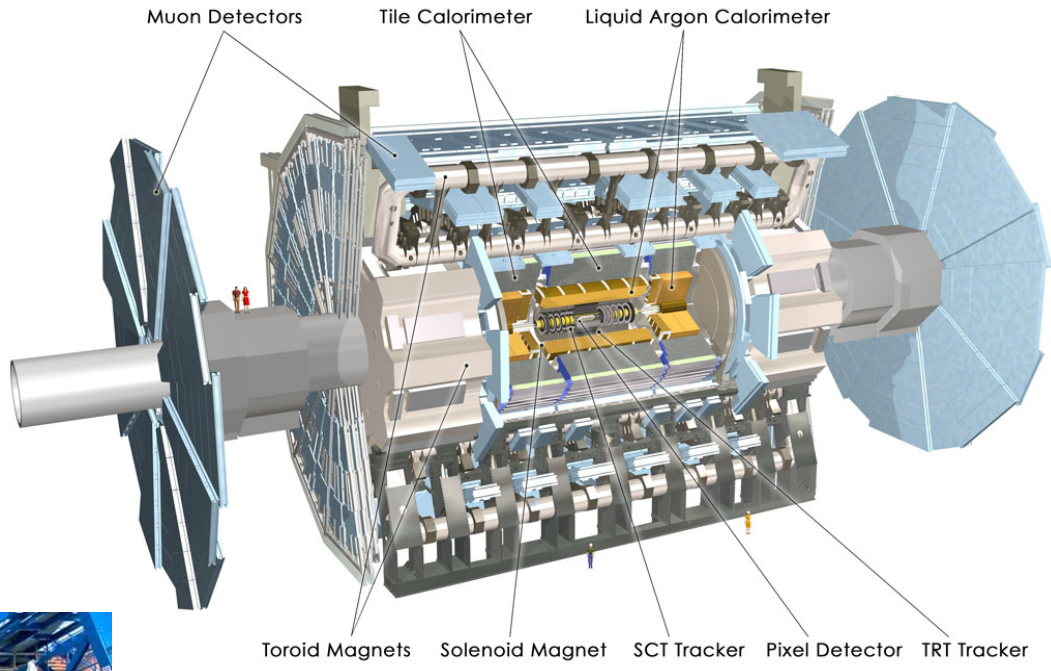


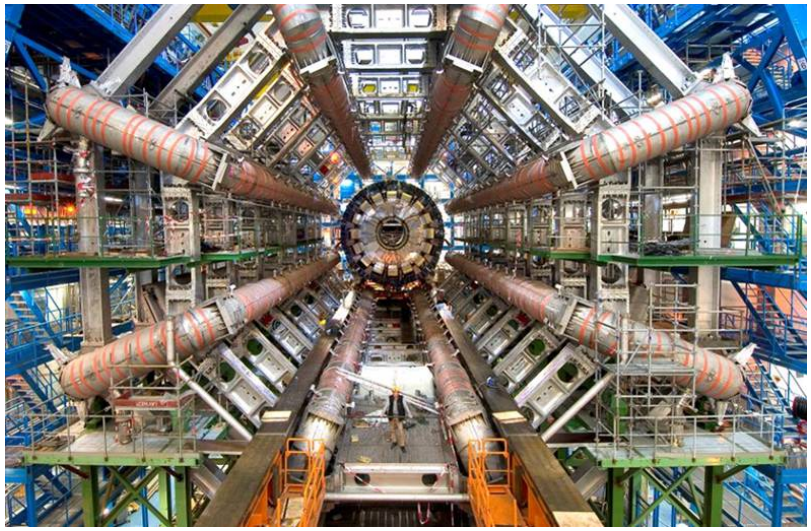
Detectors at 4 pp collision points:

ATLAS
CMS ← general purpose
LHCb (b physics)
ALICE (heavy ion physics)

The ATLAS detector

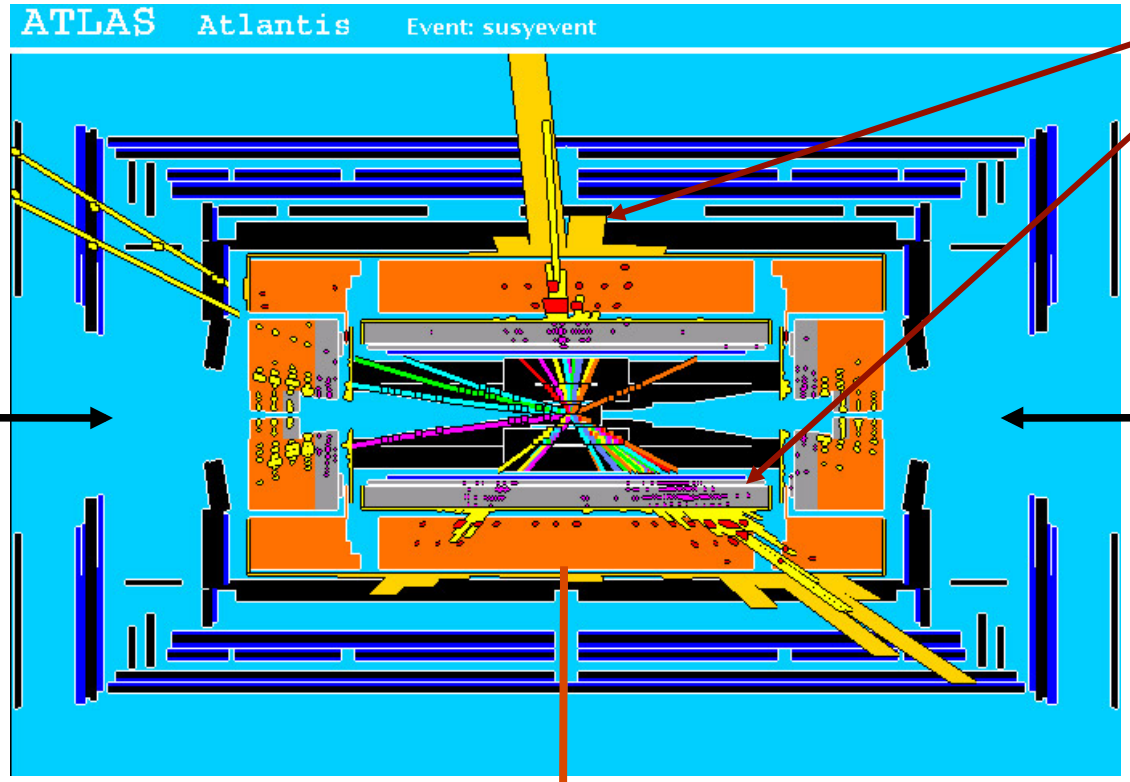
2100 physicists
37 countries
167 universities/labs





25 m diameter
46 m length
7000 tonnes
~10^8 electronic channels

A simulated SUSY event



high p_T muons

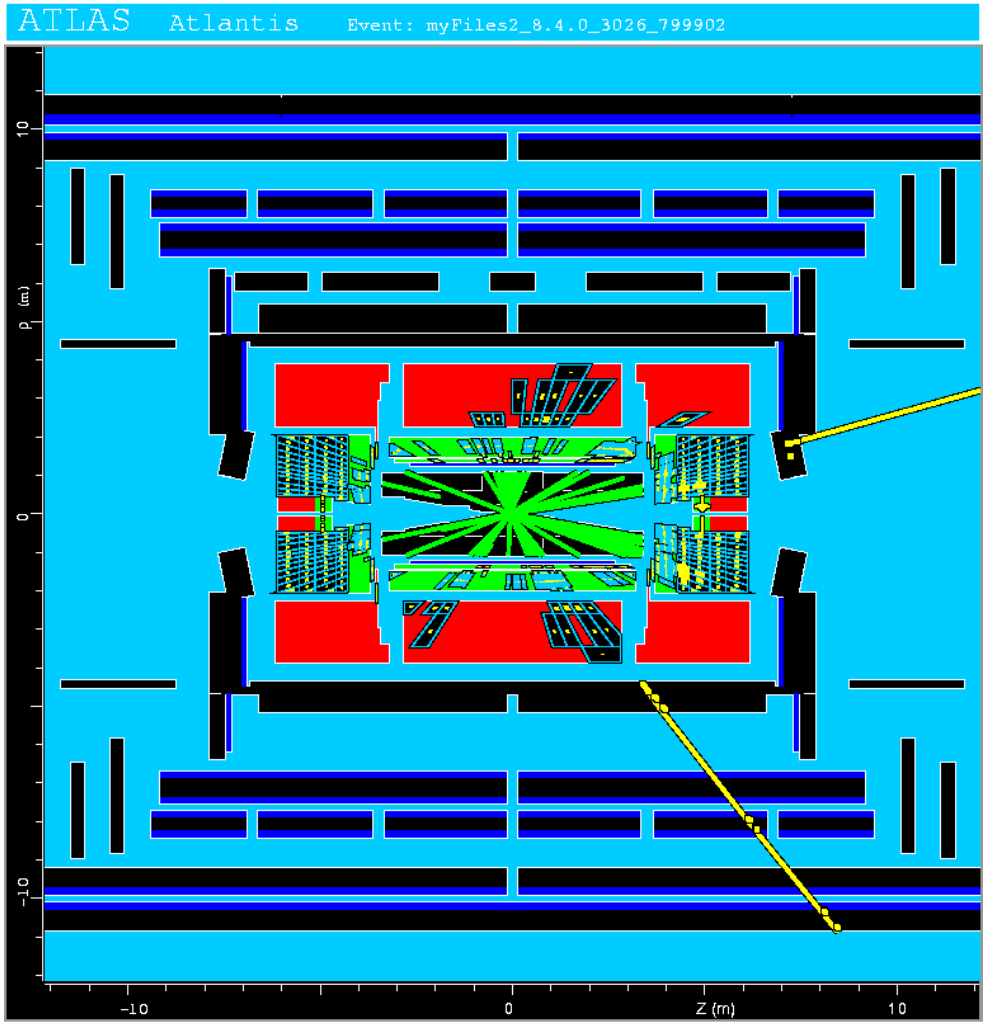
high p_T jets of hadrons

p

p

missing transverse energy

Background events



This event from Standard Model ttbar production also has high p_T jets and muons, and some missing transverse energy.

→ can easily mimic a SUSY event.

Statistical tests (in a particle physics context)

Suppose the result of a measurement for an individual event is a collection of numbers $\vec{x} = (x_1, \ldots, x_n)$

x_1 = number of muons,

x_2 = mean p_T of jets,

x_3 = missing energy, ...

$\vec{x}$ follows some n-dimensional joint pdf, which depends on the type of event produced, i.e., was it

$$pp \to t\bar{t} \ , \quad pp \to \tilde{g}\tilde{g} \ , \ldots$$

For each reaction we consider we will have a hypothesis for the pdf of $\vec{x}$, e.g., $f(\vec{x}|H_0)$, $f(\vec{x}|H_1)$, etc.

E.g. call H_0 the background hypothesis (the event type we want to reject); H_1 is signal hypothesis (the type we want).

Selecting events

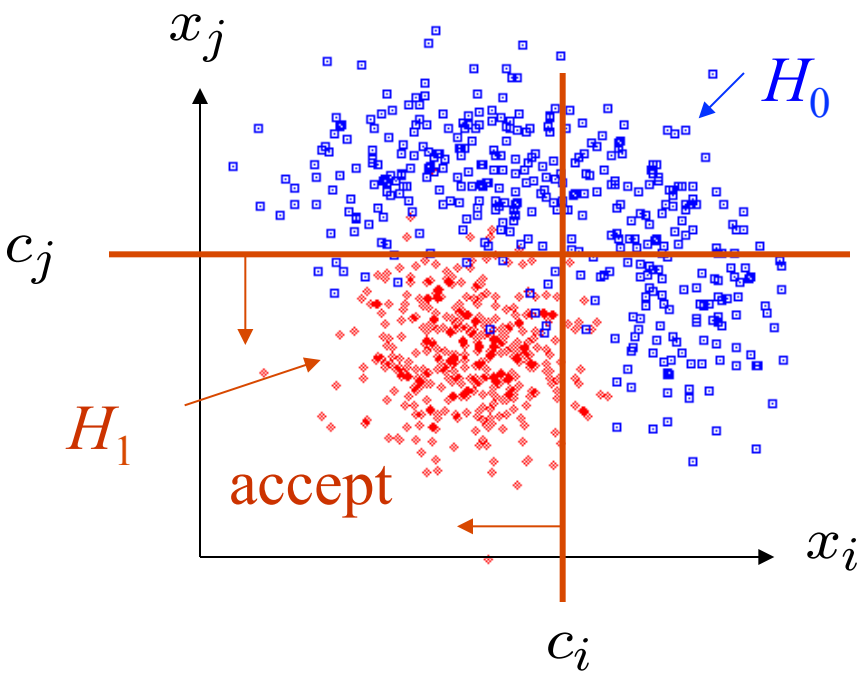
Suppose we have a data sample with two kinds of events, corresponding to hypotheses H_0 and H_1 and we want to select those of type H_1.

Each event is a point in $\vec{x}$ space. What 'decision boundary' should we use to accept/reject events as belonging to event types H_0 or H_1?

Perhaps select events with 'cuts':

$$x_i < c_i$$

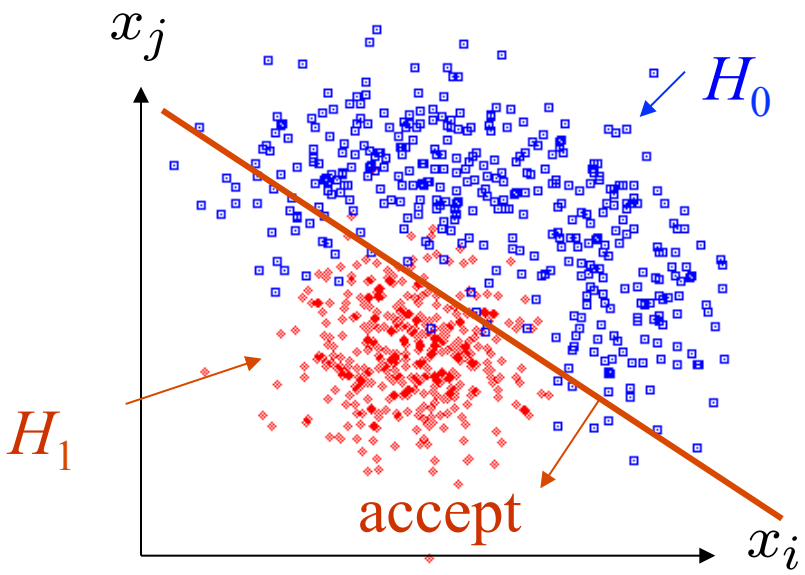
$$x_j < c_j$$



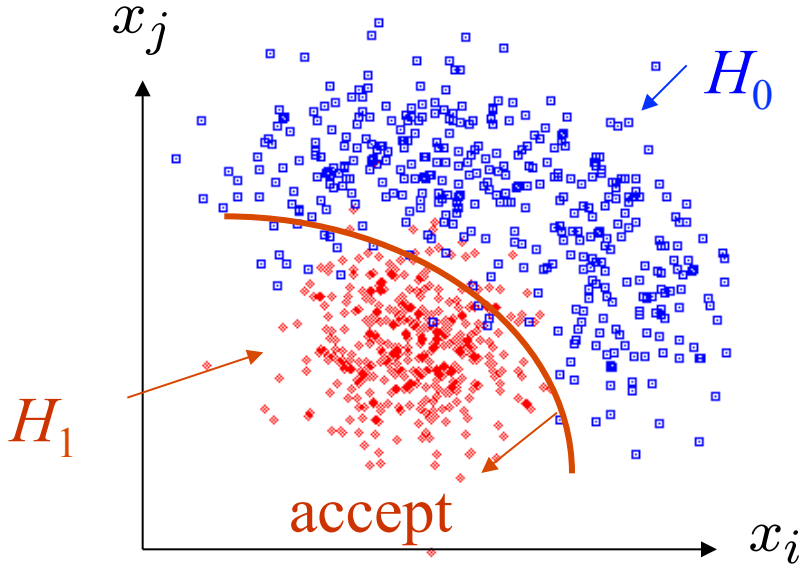
Other ways to select events

Or maybe use some other sort of decision boundary:

linear



or nonlinear



How can we do this in an 'optimal' way?

Test statistics

The boundary of the critical region for an n-dimensional data space $\boldsymbol{x} = (x_1,..., x_n)$ can be defined by an equation of the form

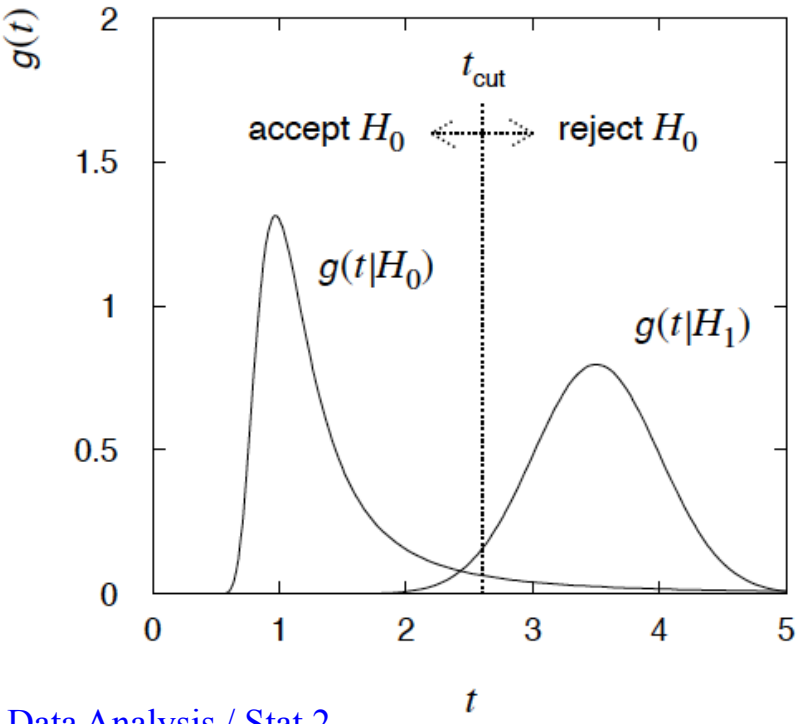
$$t(x_1, \ldots, x_n) = t_{\text{cut}}$$

where $t(x_1,\ldots, x_n)$ is a scalar test statistic.

We can work out the pdfs $g(t|H_0),\ g(t|H_1),\ \ldots$

Decision boundary is now a single 'cut' on t, defining the critical region.

So for an n-dimensional problem we have a corresponding 1-d problem.



Test statistic based on likelihood ratio

How can we choose a test's critical region in an 'optimal way'?

Neyman-Pearson lemma states:

To get the highest power for a given significance level in a test of H_0, (background) versus H_1, (signal) the critical region should have

$$\frac{f(\mathbf{x}|H_1)}{f(\mathbf{x}|H_0)} > c$$

inside the region, and $\leq c$ outside, where c is a constant chosen to give a test of the desired size.

Equivalently, optimal scalar test statistic is

$$t(\mathbf{x}) = \frac{f(\mathbf{x}|H_1)}{f(\mathbf{x}|H_0)}$$

N.B. any monotonic function of this is leads to the same test.

Classification viewed as a statistical test

Probability to reject H_0 if true (type I error): $\alpha = \int_W f(\mathbf{x}|H_0)d\mathbf{x}$

α = size of test, significance level, false discovery rate

Probability to accept H_0 if H_1 true (type II error) $\beta = \int_{\overline{W}} f(\mathbf{x}|H_1)d\mathbf{x}$

$1 - \beta$ = power of test with respect to H_1

Equivalently if e.g. H_0 = background event, H_1 = signal event, use efficiencies:

$$\varepsilon_{\mathrm{b}} = \int_W f(\mathbf{x}|H_0)\, d\mathbf{x} = \alpha$$

$$\varepsilon_{\mathrm{s}} = \int_W f(\mathbf{x}|H_1)\, d\mathbf{x} = 1 - \beta = \mathrm{power}$$

Purity / misclassification rate

Consider the probability that an event of signal (s) type classified correctly (i.e., the event selection purity),

Use Bayes' theorem:

Here W is signal region

ε_s

prior probability

$$P(\mathrm{s}|\mathbf{x} \in W) = \frac{P(\mathbf{x} \in W|\mathrm{s})P(\mathrm{s})}{P(\mathbf{x} \in W|\mathrm{s})P(\mathrm{s}) + P(\mathbf{x} \in W|\mathrm{b})P(\mathrm{b})}$$

ε_b

posterior probability = signal purity

= 1 – signal misclassification rate

Note purity depends on the prior probability for an event to be signal or background as well as on s/b efficiencies.

Neyman-Pearson doesn't usually help

We usually don't have explicit formulae for the pdfs $f(\boldsymbol{x}|\mathrm{s})$, $f(\boldsymbol{x}|\mathrm{b})$, so for a given $\boldsymbol{x}$ we can't evaluate the likelihood ratio

$$t(\mathbf{x}) = \frac{f(\mathbf{x}|s)}{f(\mathbf{x}|b)}$$

Instead we may have Monte Carlo models for signal and background processes, so we can produce simulated data:

$$\text{generate } \boldsymbol{x} \sim f(\boldsymbol{x}|\mathrm{s}) \quad \rightarrow \quad \boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$$

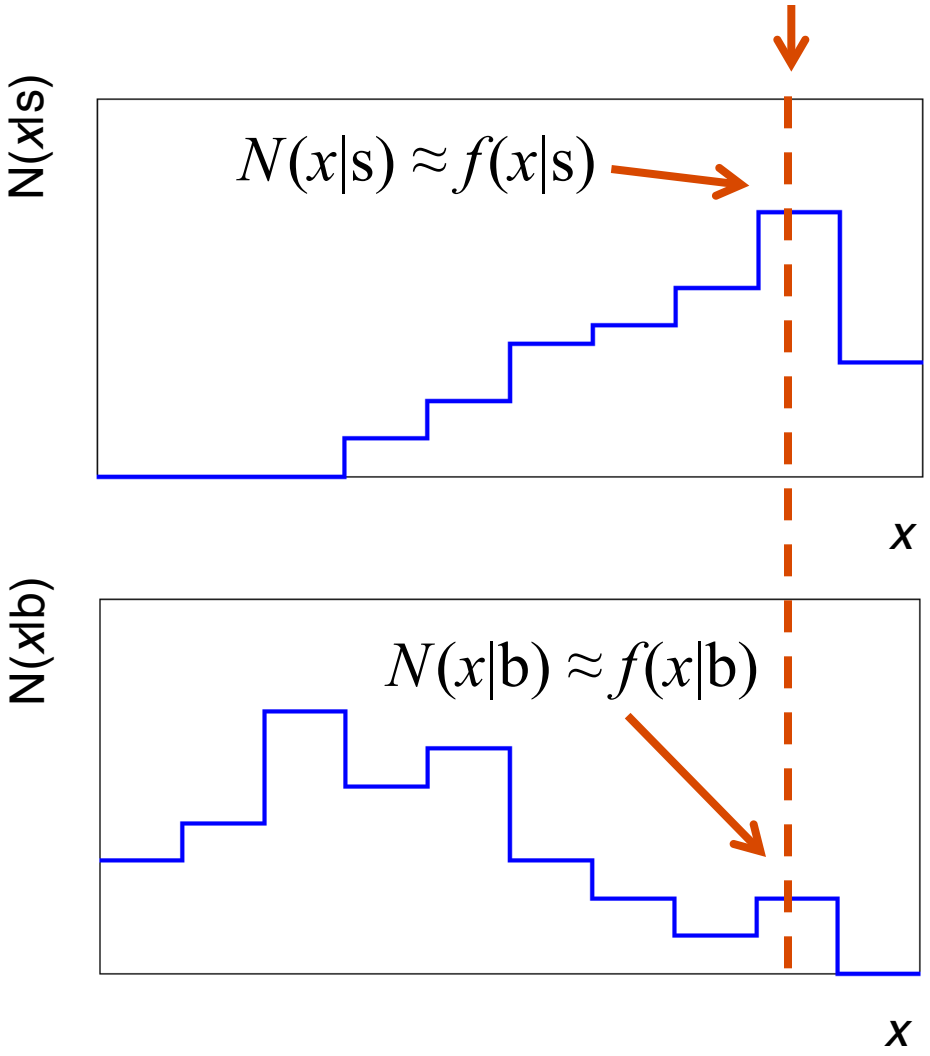
$$\text{generate } \boldsymbol{x} \sim f(\boldsymbol{x}|\mathrm{b}) \quad \rightarrow \quad \boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$$

This gives samples of "training data" with events of known type.

Can be expensive (1 fully simulated LHC event ~ 1 CPU minute).

Approximate LR from histograms

Want $t(x) = f(x|\mathrm{s})/f(x|\mathrm{b})$ for x here



One possibility is to generate MC data and construct histograms for both signal and background.

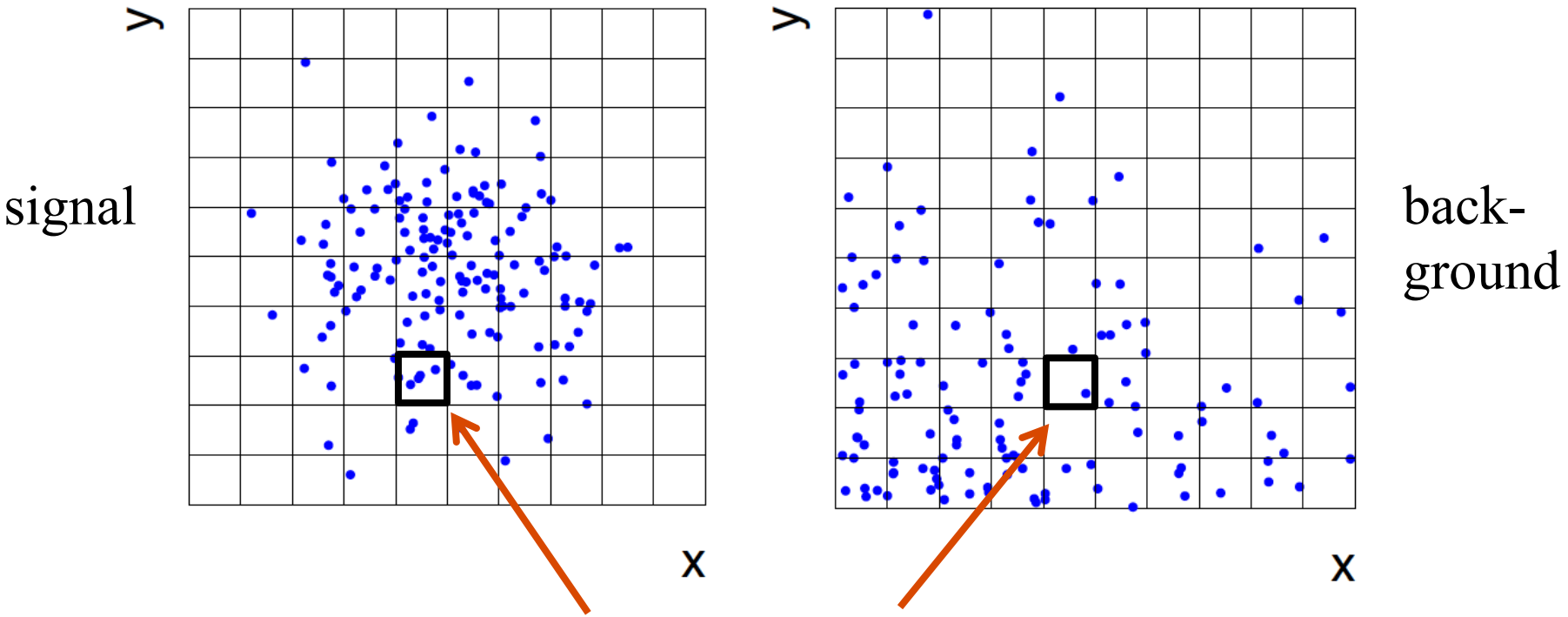
Use (normalized) histogram values to approximate LR:

$$t(x) \approx \frac{N(x|s)}{N(x|b)}$$

Can work well for single variable.

Approximate LR from 2D-histograms

Suppose problem has 2 variables. Try using 2-D histograms:



Approximate pdfs using $N(x,y|\mathrm{s})$, $N(x,y|\mathrm{b})$ in corresponding cells.

But if we want M bins for each variable, then in n-dimensions we have M^n cells; can't generate enough training data to populate.

→ Histogram method usually not usable for $n > 1$ dimension.

Strategies for multivariate analysis

Neyman-Pearson lemma gives optimal answer, but cannot be used directly, because we usually don't have $f(\boldsymbol{x}|\mathrm{s})$, $f(\boldsymbol{x}|\mathrm{b})$.

Histogram method with M bins for n variables requires that we estimate M^n parameters (the values of the pdfs in each cell), so this is rarely practical.

A compromise solution is to assume a certain functional form for the test statistic $t(\boldsymbol{x})$ with fewer parameters; determine them (using MC) to give best separation between signal and background.

Alternatively, try to estimate the probability densities $f(\boldsymbol{x}|\mathrm{s})$ and $f(\boldsymbol{x}|\mathrm{b})$ (with something better than histograms) and use the estimated pdfs to construct an approximate likelihood ratio.

Multivariate methods

Many new (and some old) methods:

- Fisher discriminant
- Neural networks
- Kernel density methods
- Support Vector Machines
- Decision trees
  - Boosting
  - Bagging

Resources on multivariate methods

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

T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning, 2nd ed., Springer, 2009

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

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

Ilya Narsky and Frank C. Porter, *Statistical Analysis Techniques in Particle Physics*, Wiley, 2014.

朱永生（编著），实验数据多元统计分析，科学出版社，北京，2009。

Software

Rapidly growing area of development – two important resources:

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

- From **`tmva.sourceforge.net`**, also distributed with ROOT
- Variety of classifiers
- Good manual, widely used in HEP

scikit-learn

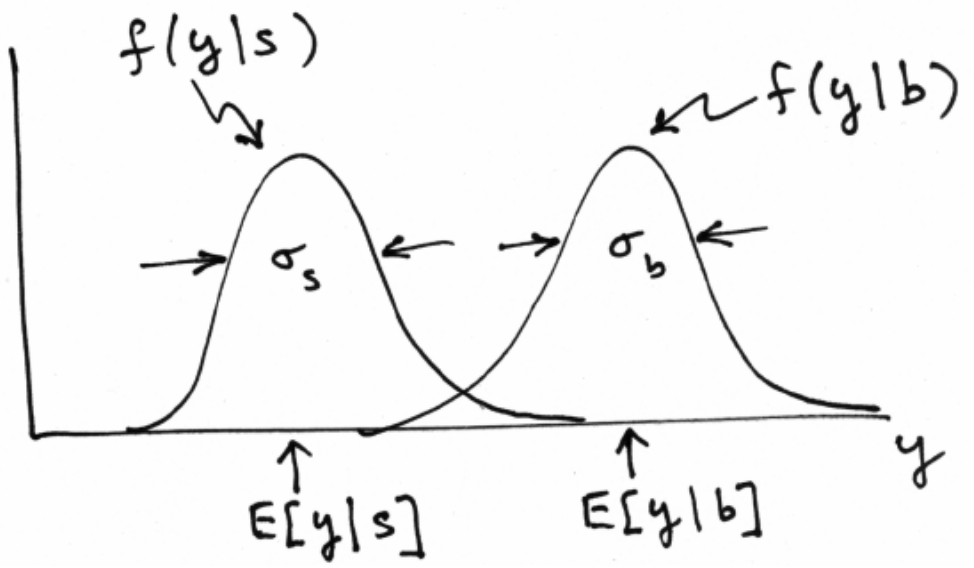
- Python-based tools for Machine Learning
- **`scikit-learn.org`**
- Large user community

Linear test statistic

Suppose there are n input variables: $\boldsymbol{x} = (x_1, \ldots, x_n)$.

Consider a linear function: $$y(\mathbf{x}) = \sum_{i=1}^{n} w_i x_i$$

For a given choice of the coefficients $\boldsymbol{w} = (w_1, \ldots, w_n)$ we will get pdfs $f(y|\mathrm{s})$ and $f(y|\mathrm{b})$:



Linear test statistic

Fisher: to get large difference between means and small widths for $f(y|\mathrm{s})$ and $f(y|\mathrm{b})$, maximize the difference squared of the expectation values divided by the sum of the variances:

$$J(\mathbf{w}) = \frac{(E[y|s] - E[y|b])^2}{V[y|s] + V[y|b]}$$

Setting $\partial J / \partial w_i = 0$ gives:

$$\mathbf{w} \propto W^{-1}(\boldsymbol{\mu}_\mathrm{b} - \boldsymbol{\mu}_\mathrm{s})$$

$$W_{ij} = \mathrm{cov}[x_i, x_j|\mathrm{s}] + \mathrm{cov}[x_i, x_j|\mathrm{b}]$$

$$\mu_{i,\mathrm{s}} = E[x_i|s], \qquad \mu_{i,\mathrm{b}} = E[x_i|b]$$

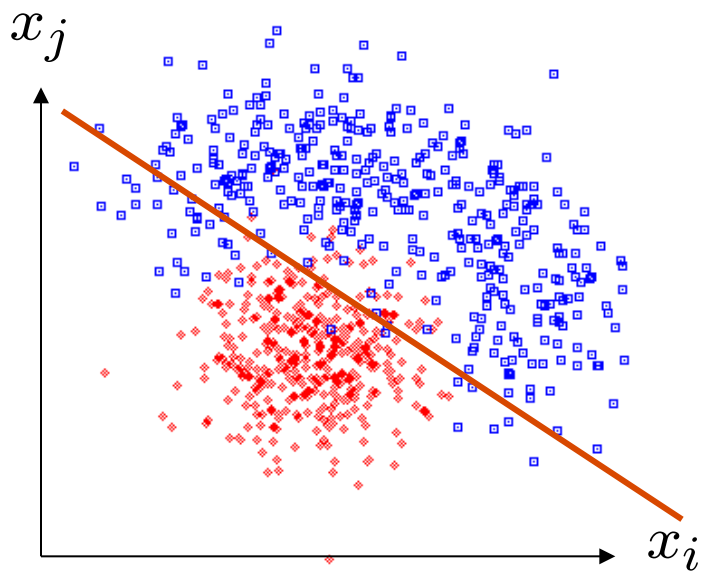
The Fisher discriminant

The resulting coefficients w_i define a Fisher discriminant.

Coefficients defined up to multiplicative constant; can also add arbitrary offset, i.e., usually define test statistic as

$$y(\mathbf{x}) = w_0 + \sum_{i=1}^{n} w_i x_i$$

Boundaries of the test's critical region are surfaces of constant $y(\boldsymbol{x})$, here linear (hyperplanes):



Fisher discriminant for Gaussian data

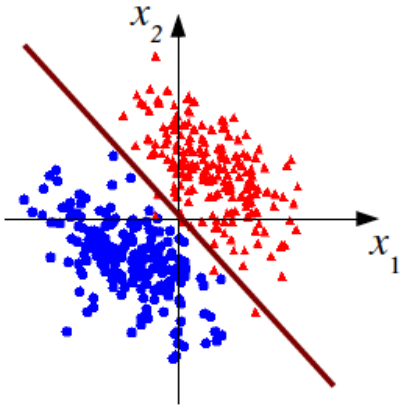
Suppose the pdfs of the input variables, $f(\boldsymbol{x}|\mathrm{s})$ and $f(\boldsymbol{x}|\mathrm{b})$, are both multivariate Gaussians with same covariance but different means:

$$f(\boldsymbol{x}|\mathrm{s}) = \mathrm{Gauss}(\boldsymbol{\mu}_\mathrm{s}, V)$$

$$f(\boldsymbol{x}|\mathrm{b}) = \mathrm{Gauss}(\boldsymbol{\mu}_\mathrm{b}, V)$$

Same covariance

$V_{ij} = \mathrm{cov}[x_i, x_j]$



In this case it can be shown that the Fisher discriminant is

$$y(\mathbf{x}) \sim \ln \frac{f(\mathbf{x}|\mathrm{s})}{f(\mathbf{x}|\mathrm{b})}$$

i.e., it is a monotonic function of the likelihood ratio and thus leads to the same critical region. So in this case the Fisher discriminant provides an optimal statistical test.

Transformation of inputs

If the data are not Gaussian with equal covariance, a linear decision boundary is not optimal. But we can try to subject the data to a transformation

$$\varphi_1(\vec{x}), \ldots, \varphi_m(\vec{x})$$

and then treat the ϕ_i as the new input variables. This is often called "feature space" and the ϕ_i are "basis functions". The basis functions can be fixed or can contain adjustable parameters which we optimize with training data (cf. neural networks).

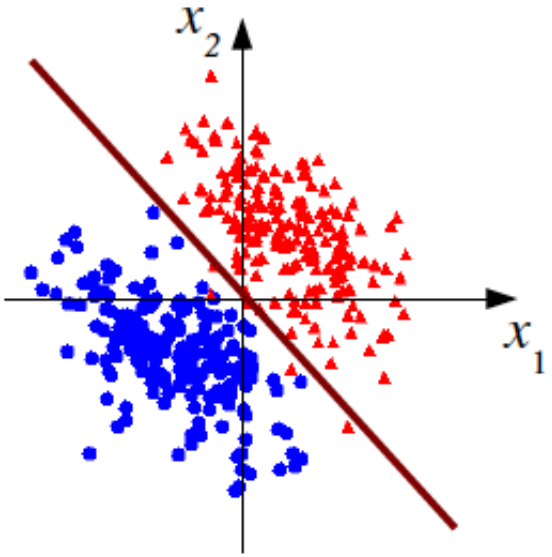
In other cases we will see that the basis functions only enter as dot products

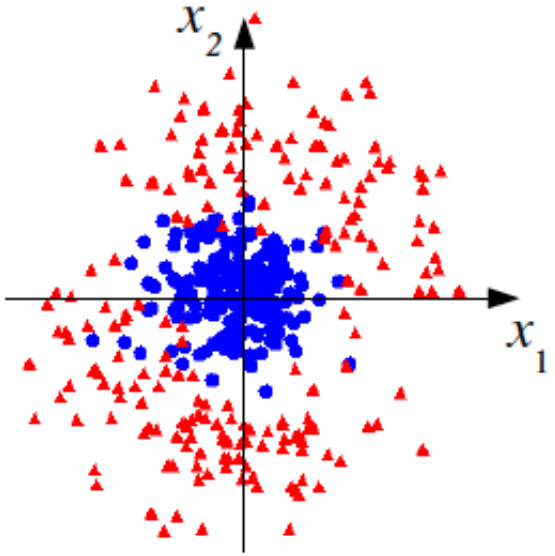
$$\vec{\varphi}(\vec{x}_i)\cdot\vec{\varphi}(\vec{x}_j) = K(\vec{x}_i, \vec{x}_j)$$

and thus we will only need the "kernel function" $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$

Linear decision boundaries

A linear decision boundary is only optimal when both classes follow multivariate Gaussians with equal covariances and different means.





For some other cases a linear boundary is almost useless.

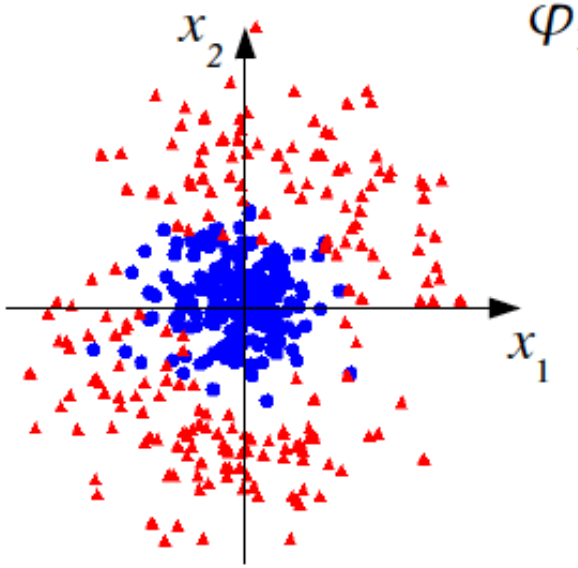
Nonlinear transformation of inputs

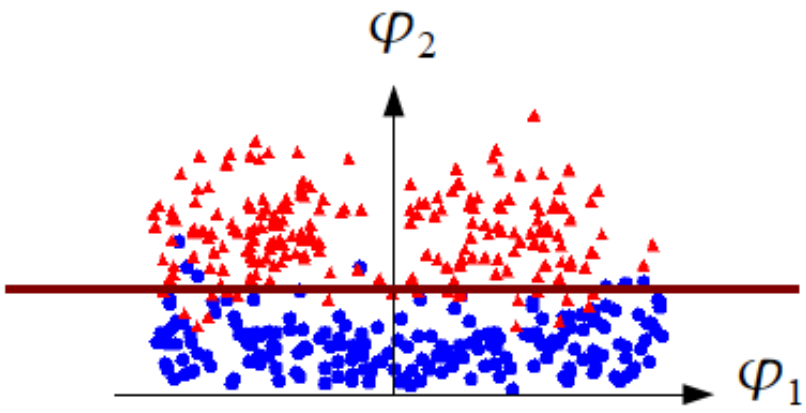
We can try to find a transformation, $x_1, \ldots, x_n \to \varphi_1(\vec{x}), \ldots, \varphi_m(\vec{x})$ so that the transformed "feature space" variables can be separated better by a linear boundary:

$$\varphi_1 = \tan^{-1}(x_2/x_1)$$

$$\varphi_2 = \sqrt{x_1^2 + x_2^2}$$

Here, guess fixed basis functions (no free parameters)





Neural networks

Neural networks originate from attempts to model neural processes (McCulloch and Pitts, 1943; Rosenblatt, 1962).

Widely used in many fields, and for many years the only "advanced" multivariate method popular in HEP.

We can view a neural network as a specific way of parametrizing the basis functions used to define the feature space transformation.

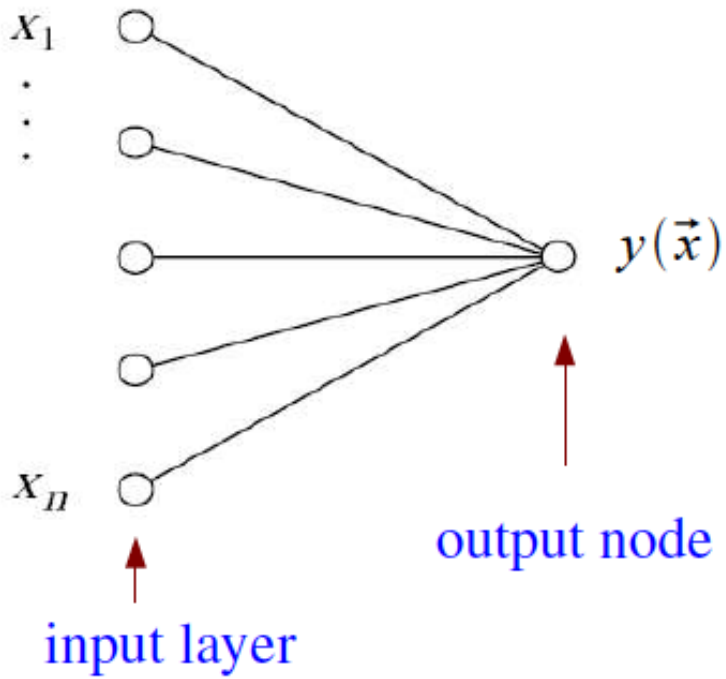
The training data are then used to adjust the parameters so that the resulting discriminant function has the best performance.

The single layer perceptron

Define the discriminant using $y(\vec{x}) = h\left(w_0 + \sum_{i=1}^{n} w_i x_i\right)$

where h is a nonlinear, monotonic activation function; we can use e.g. the logistic sigmoid $h(x) = (1 + e^{-x})^{-1}$.

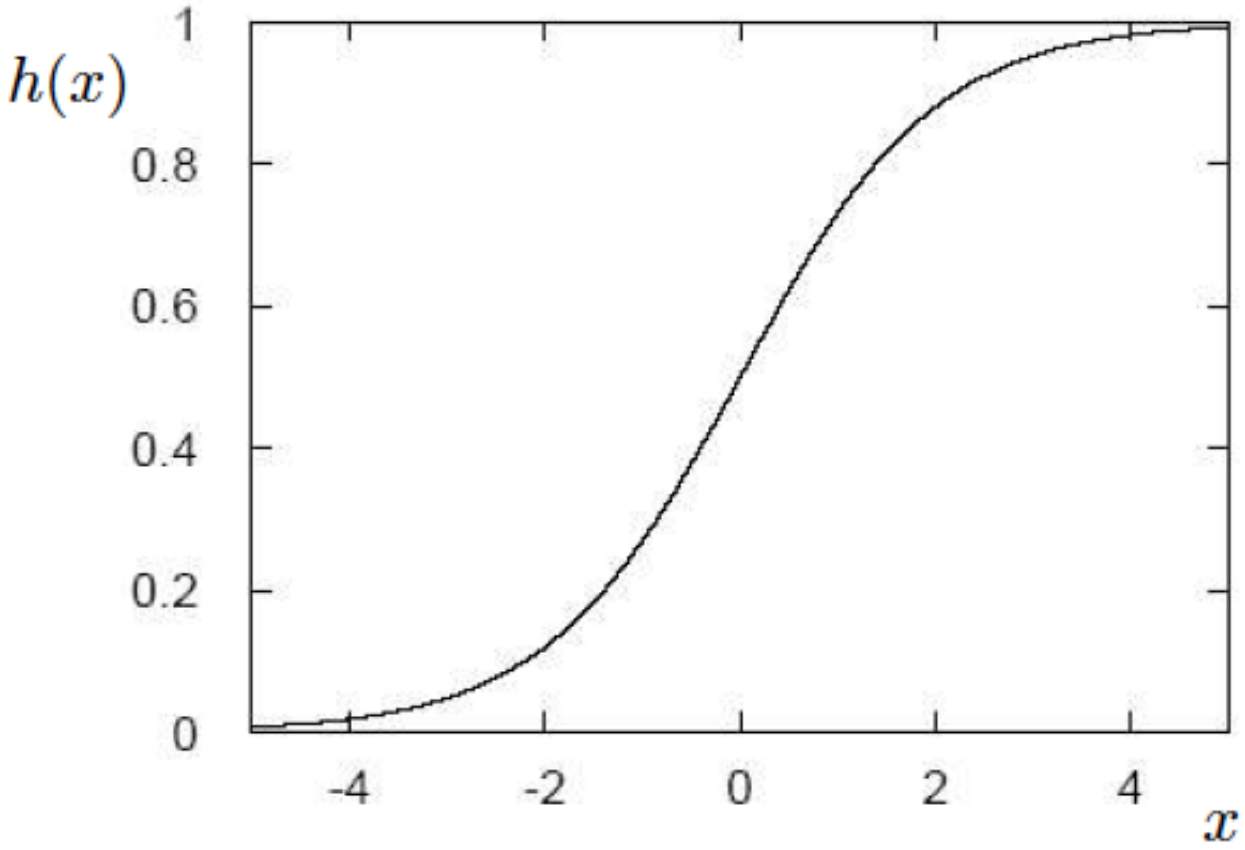
If the activation function is monotonic, the resulting $y(\boldsymbol{x})$ is equivalent to the original linear discriminant. This is an example of a "generalized linear model" called the single layer perceptron.



The activation function

For activation function $h(\cdot)$ often use logistic sigmoid:

$$h(x) = \frac{1}{1 + e^{-x}}$$



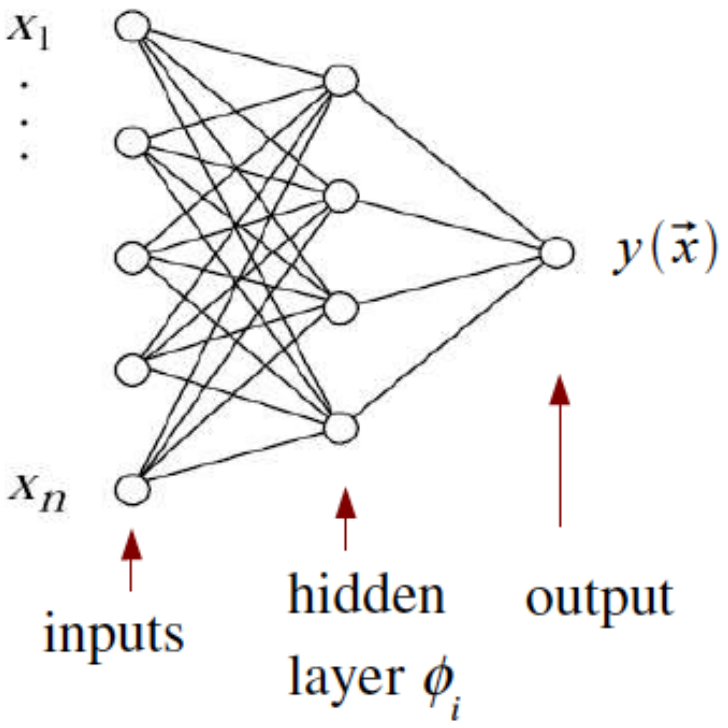
The multilayer perceptron

Now use this idea to define not only the output $y(\boldsymbol{x})$, but also the set of transformed inputs $\varphi_1(\vec{x}), \ldots, \varphi_m(\vec{x})$ that form a "hidden layer":

Superscript for weights indicates layer number

$$\varphi_i(\vec{x}) = h\left(w_{i0}^{(1)} + \sum_{j=1}^{n} w_{ij}^{(1)} x_j\right)$$

$$y(\vec{x}) = h\left(w_{10}^{(2)} + \sum_{j=1}^{n} w_{1j}^{(2)} \varphi_j(\vec{x})\right)$$



This is the multilayer perceptron, our basic neural network model; straightforward to generalize to multiple hidden layers.

Network architecture

Theorem: An MLP with a single hidden layer having a sufficiently large number of nodes can approximate arbitrarily well the optimal decision boundary.

Holds for any continuous non-polynomial activation function
Leshno, Lin, Pinkus and Schocken (1993) Neural Networks 6, 861-867

However, the number of required nodes may be very large; cannot train well with finite samples of training data.

Recent advances in *Deep Neural Networks* have shown important advantages in having multiple hidden layers.

For a particle physics application of Deep Learning, see e.g.
Baldi, Sadowski and Whiteson, *Nature Communications* 5 (2014); arXiv:1402.4735.

Network training

The type of each training event is known, i.e., for event a we have:

$\vec{x}_a = (x_1, \ldots, x_n)$ the input variables, and

$t_a = 0, 1$ a numerical label for event type ("target value")

Let $\boldsymbol{w}$ denote the set of all of the weights of the network. We can determine their optimal values by minimizing a sum-of-squares "error function"

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{a=1}^{N} |y(\vec{x}_a, \boldsymbol{w}) - t_a|^2 = \sum_{a=1}^{N} E_a(\boldsymbol{w})$$

Contribution to error function from each event

Numerical minimization of $E(\boldsymbol{w})$

Consider gradient descent method: from an initial guess in weight space $\boldsymbol{w}^{(1)}$ take a small step in the direction of maximum decrease. I.e. for the step τ to $\tau+1$,

$$\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} - \eta \nabla E(\boldsymbol{w}^{(\tau)})$$

learning rate ($\eta>0$)

If we do this with the full error function $E(\boldsymbol{w})$, gradient descent does surprisingly poorly; better to use "conjugate gradients".

But gradient descent turns out to be useful with an online (sequential) method, i.e., where we update $\boldsymbol{w}$ for each training event a, (cycle through all training events):

$$\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} - \eta \nabla E_a(\boldsymbol{w}^{(\tau)})$$

Error backpropagation

Error backpropagation ("backprop") is an algorithm for finding the derivatives required for gradient descent minimization.

The network output can be written $y(\boldsymbol{x}) = h(u(\boldsymbol{x}))$ where

$$u(\vec{x}) = \sum_{j=0} w_{1j}^{(2)} \varphi_j(\vec{x}), \qquad \varphi_j(\vec{x}) = h\left(\sum_{k=0} w_{jk}^{(1)} x_k \right)$$

where we defined $\phi_0 = x_0 = 1$ and wrote the sums over the nodes in the preceding layers starting from 0 to include the offsets.

So e.g. for event a we have $\dfrac{\partial E_a}{\partial w_{1j}^{(2)}} = (y_a - t_a)\, h'(u(\vec{x}))\, \varphi_j(\vec{x})$

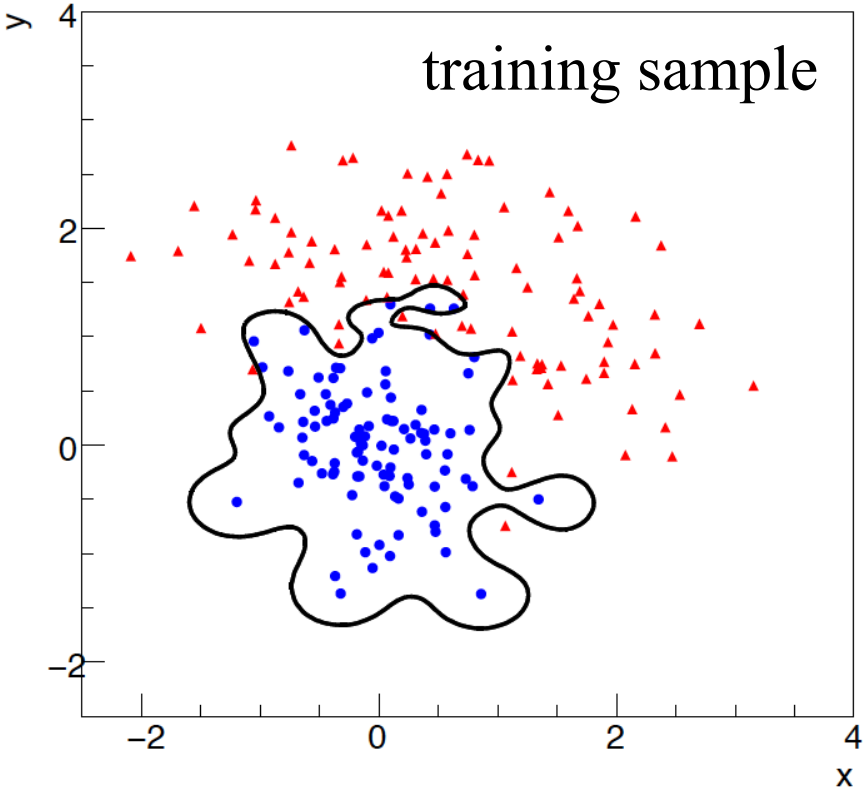
derivative of activation function

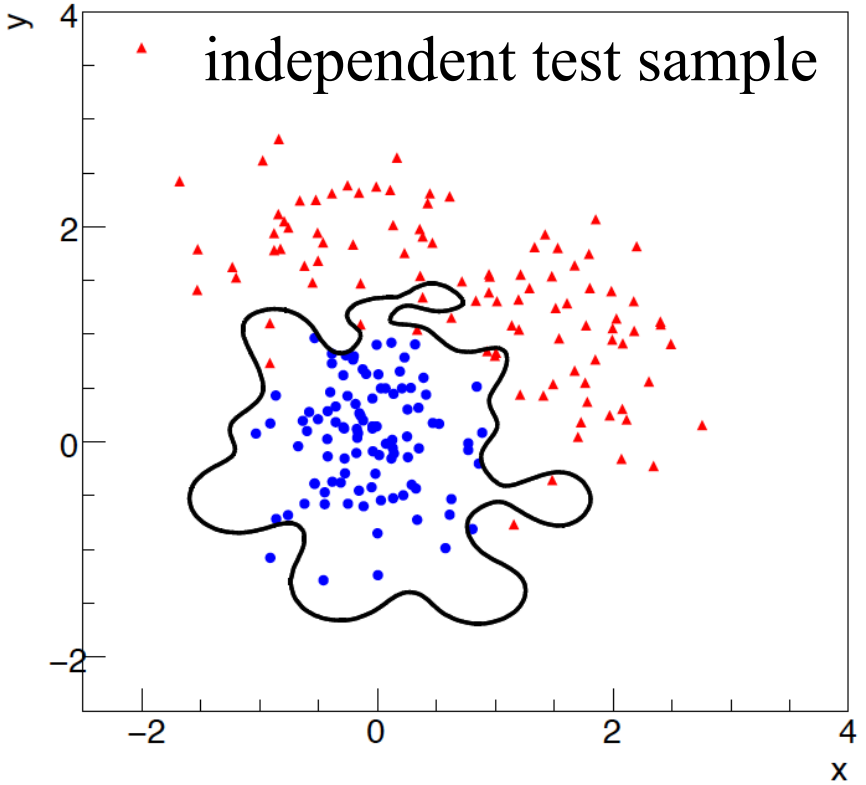
Chain rule gives all the needed derivatives.

Overtraining

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

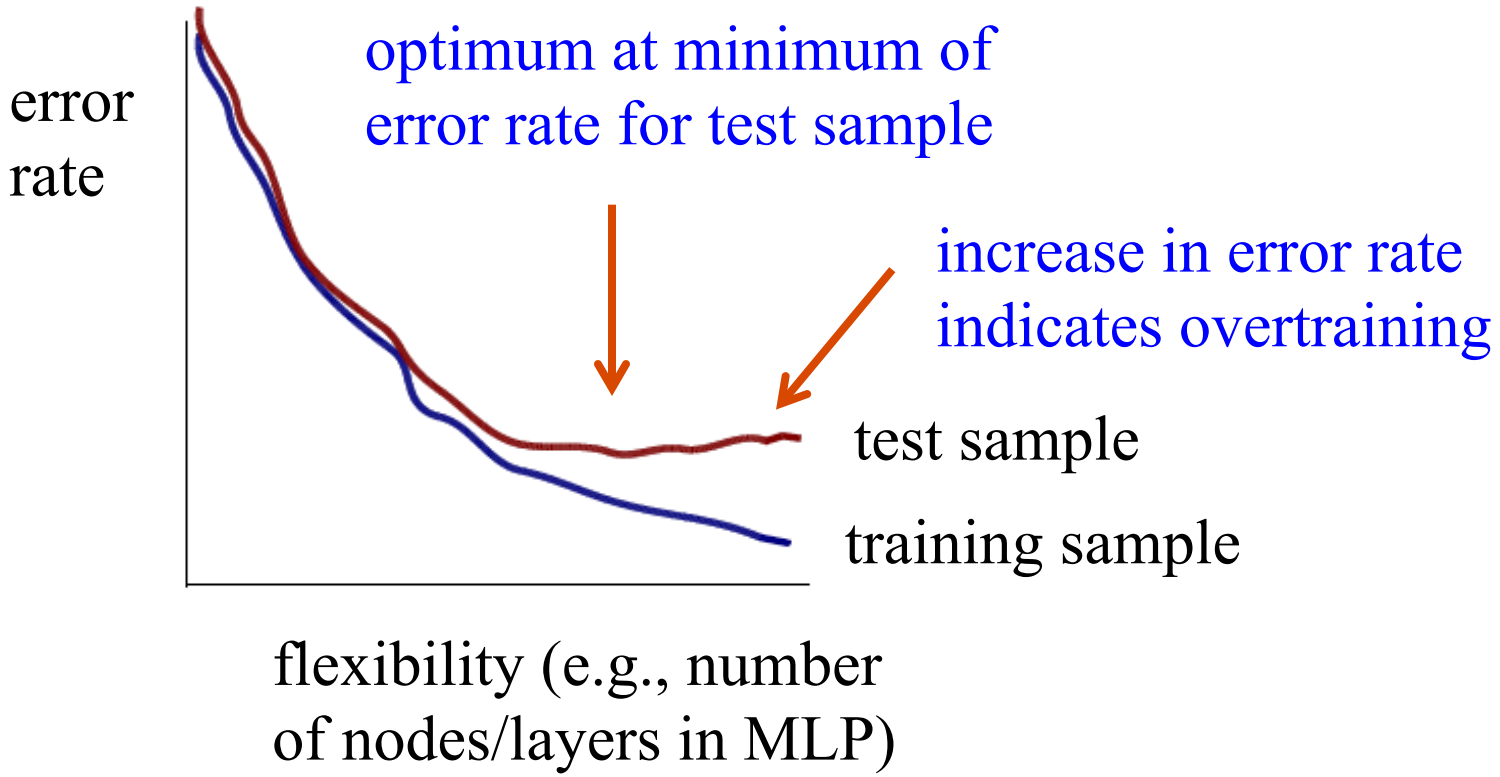
A "flexible" classifier may conform too closely to the training points; the same boundary will not perform well on an independent test data sample (→ "overtraining").





Monitoring overtraining

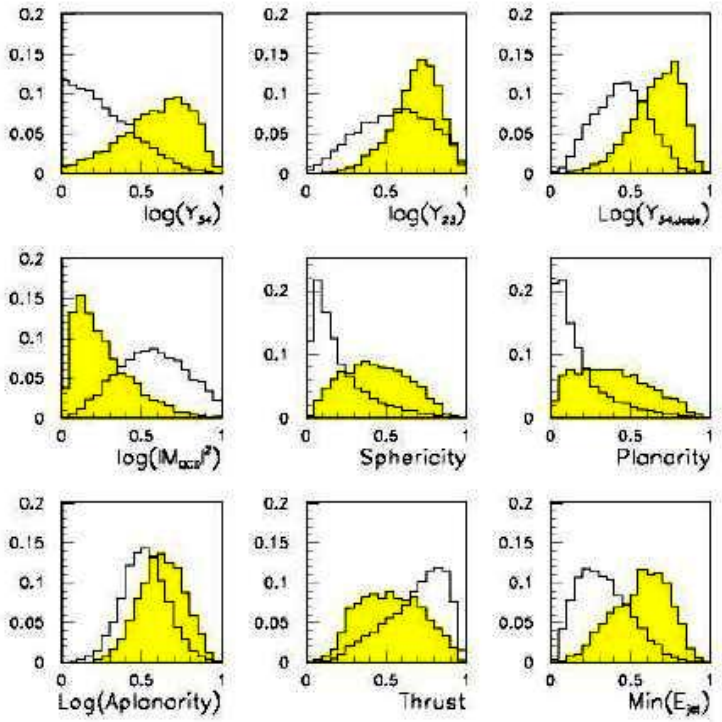
If we monitor the fraction of misclassified events (or similar, e.g., error function $E(\boldsymbol{w})$) for test and training samples, it will usually decrease for both as the boundary is made more flexible:



Neural network example from LEP II

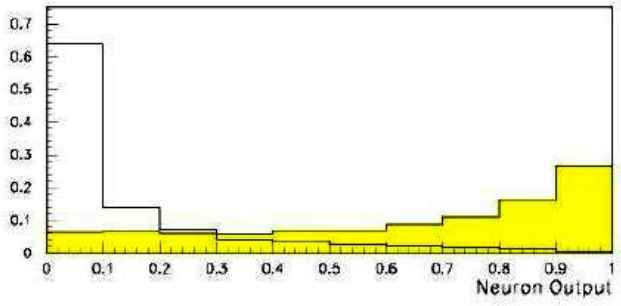
Signal: $e^+e^- \to W^+W^-$ (often 4 well separated hadron jets)

Background: $e^+e^- \to qqgg$ (4 less well separated hadron jets)



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

Neural network output:



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

Probability Density Estimation (PDE)

Construct non-parametric estimators for the pdfs of the data $\boldsymbol{x}$ for the two event classes, $p(\boldsymbol{x}|H_0)$, $p(\boldsymbol{x}|H_1)$ and use these to construct the likelihood ratio, which we use for the discriminant function:

$$y(\mathbf{x}) = \frac{\hat{p}(\mathbf{x}|H_1)}{\hat{p}(\mathbf{x}|H_0)}$$

n-dimensional histogram is a brute force example of this; we will see a number of ways that are much better.

Correlation vs. independence

In a general a multivariate distribution $p(\boldsymbol{x})$ does not factorize into a product of the marginal distributions for the individual variables:

$$p(\vec{x}) = \prod_{i=1}^{n} p_i(x_i)$$

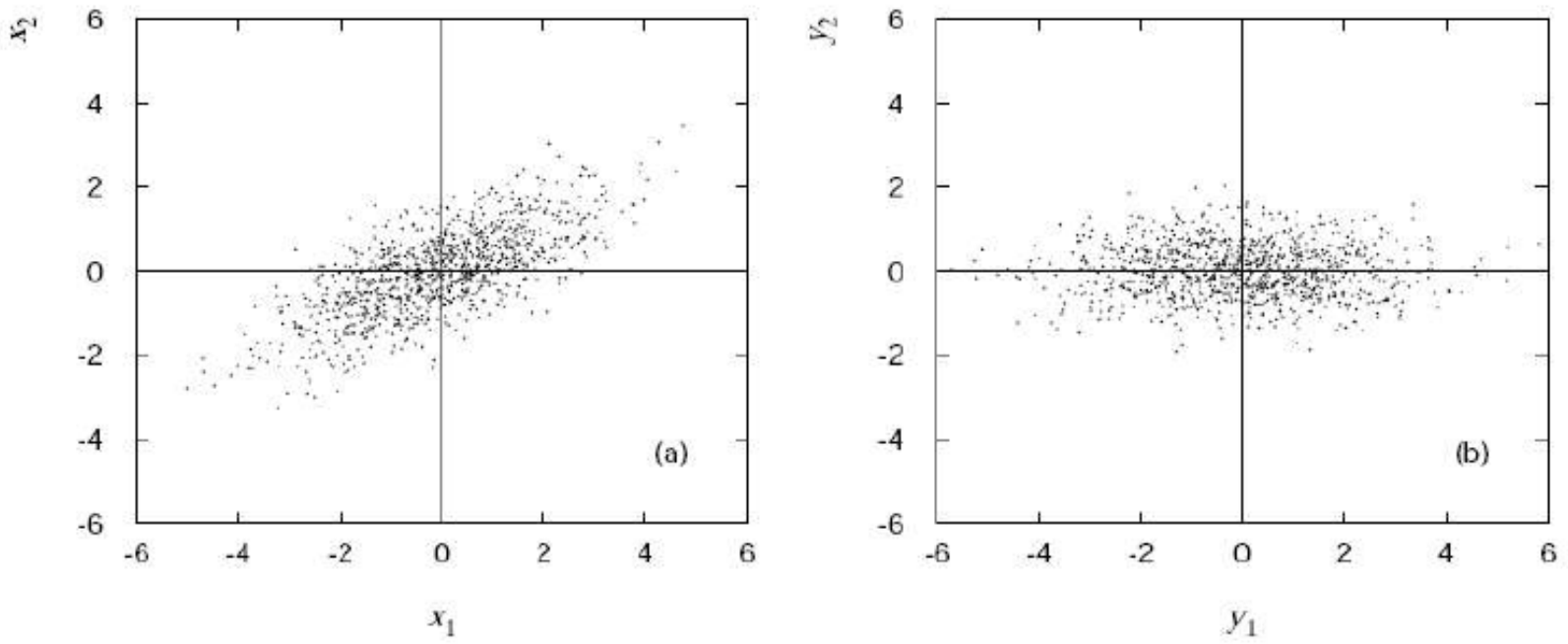
holds only if the components of $\boldsymbol{x}$ are independent

Most importantly, the components of $\boldsymbol{x}$ will generally have nonzero covariances (i.e. they are correlated):

$$V_{ij} = \mathrm{cov}[x_i, x_j] = E[x_i x_j] - E[x_i]E[x_j] \neq 0$$

Decorrelation of input variables

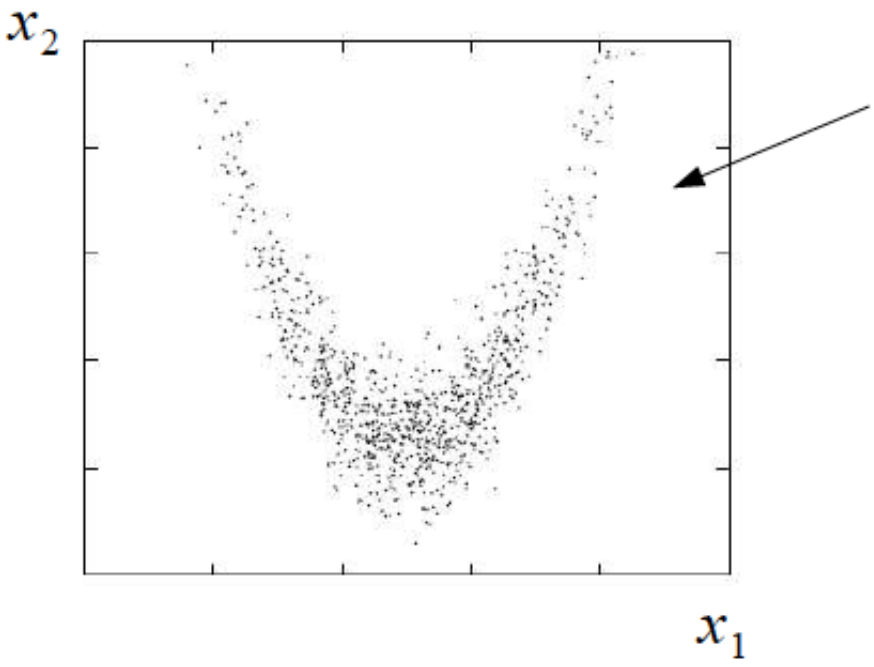
But we can define a set of uncorrelated input variables by a linear transformation, i.e., find the matrix A such that for $\vec{y} = A\vec{x}$ the covariances $\mathrm{cov}[y_i, y_j] = 0$:



For the following suppose that the variables are "decorrelated" in this way for each of $p(\mathbf{x}|H_0)$ and $p(\mathbf{x}|H_1)$ separately (since in general their correlations are different).

Decorrelation is not enough

But even with zero correlation, a multivariate pdf $p(\boldsymbol{x})$ will in general have nonlinearities and thus the decorrelated variables are still not independent.



pdf with zero covariance but components still not independent, since clearly

$$p(x_2|x_1) \equiv \frac{p(x_1, x_2)}{p_1(x_1)} \neq p_2(x_2)$$

and therefore

$$p(x_1, x_2) \neq p_1(x_1)\, p_2(x_2)$$

Naive Bayes method

First decorrelate $\boldsymbol{x}$, i.e., find $\boldsymbol{y} = A\boldsymbol{x}$, with $\mathrm{cov}[y_i, y_j] = V[y_i]\,\delta_{ij}$.

Pdfs of $\boldsymbol{x}$ and $\boldsymbol{y}$ are then related by

$$f(\mathbf{x}) = |J| g(\mathbf{y}(\mathbf{x})) \qquad \text{where} \qquad J = \det(A)$$

If nonlinear features of g($\boldsymbol{y}$) not too important, estimate using product of marginal pdfs:

$$\hat{f}(\mathbf{x}) = |J| \prod_{i=1}^{n} g_i(y_i(\mathbf{x})) = |\det(A)| \prod_{i=1}^{n} g_i((A\mathbf{x})_i)$$

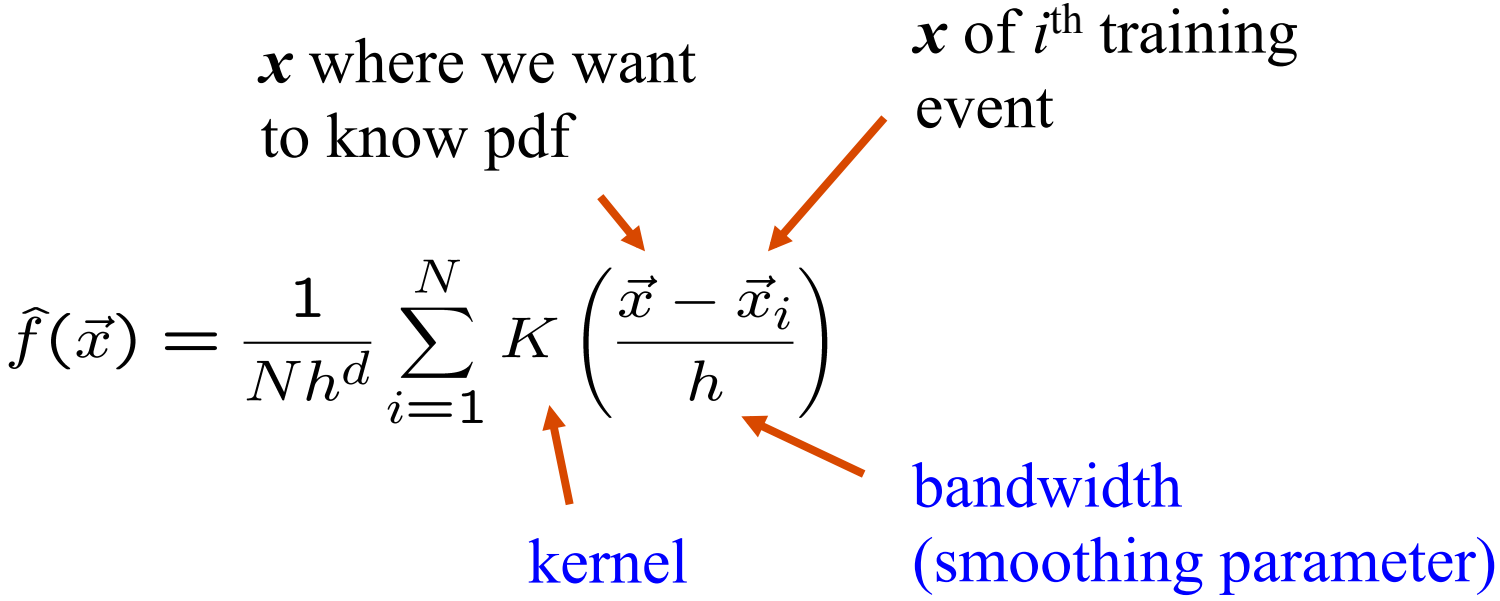
Do separately for the two hypotheses s and b (separate matrices A_s and A_b and marginal pdfs $g_{\mathrm{s},i}$, $g_{\mathrm{b},i}$). Then define test statistic as

$$y(\mathbf{x}) = \frac{\hat{f}_\mathrm{s}(\mathbf{x})}{\hat{f}_\mathrm{b}(\mathbf{x})}$$

Called Naive Bayes classifier. Reduces problem of estimating an n-dimensional pdf to finding n one-dimensional marginal pdfs.

Kernel-based PDE (KDE)

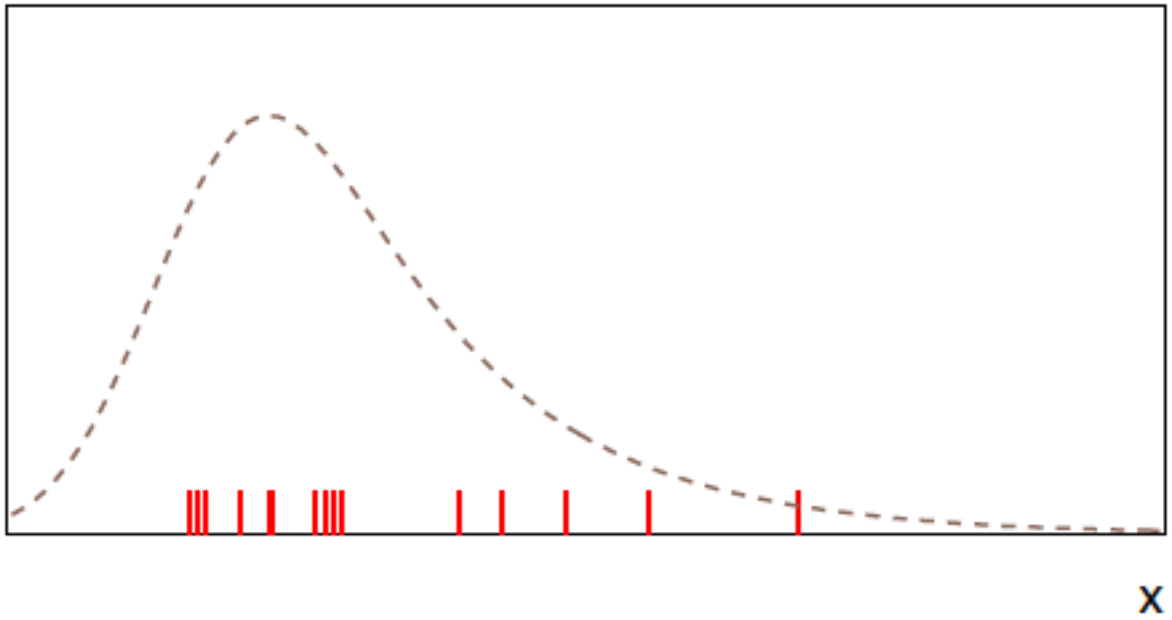
Consider d dimensions, N training events, $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, estimate $f(\boldsymbol{x})$ with



Use e.g. Gaussian kernel: $K(\vec{x}) = \frac{1}{(2\pi)^{d/2}} e^{-|\vec{x}|^2/2}$

Gaussian KDE in 1-dimension

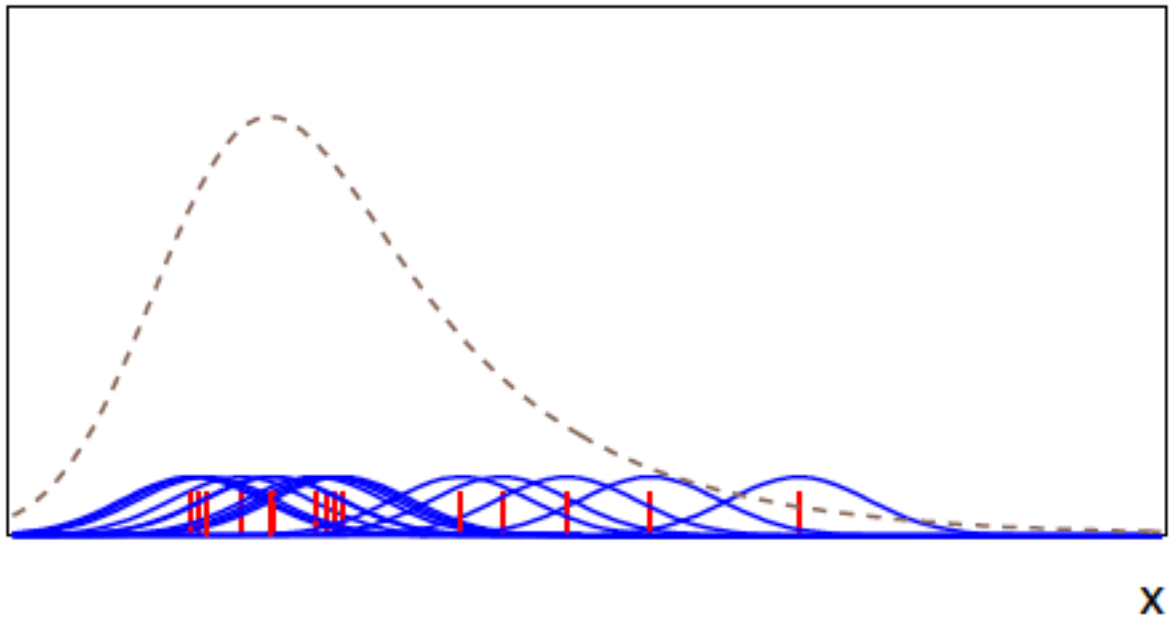
Suppose the pdf (dashed line) below is not known in closed form, but we can generate events that follow it (the red tick marks):



Goal is to find an approximation to the pdf using the generated date values.

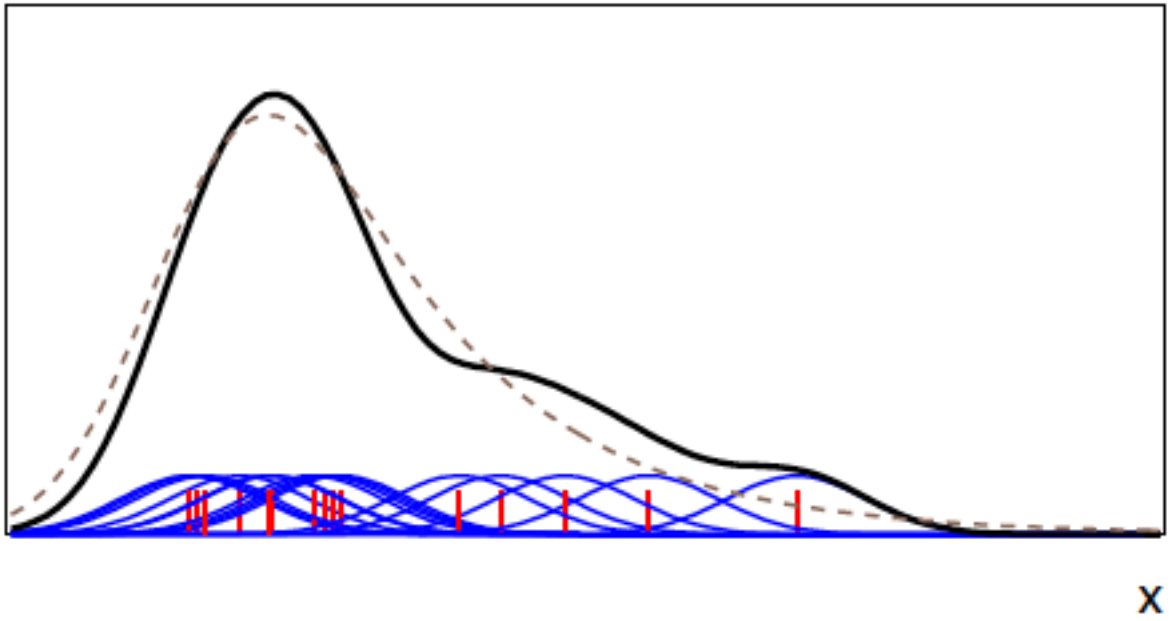
Gaussian KDE in 1-dimension (cont.)

Place a kernel pdf (here a Gaussian) centred around each generated event weighted by $1/N_{\text{event}}$:



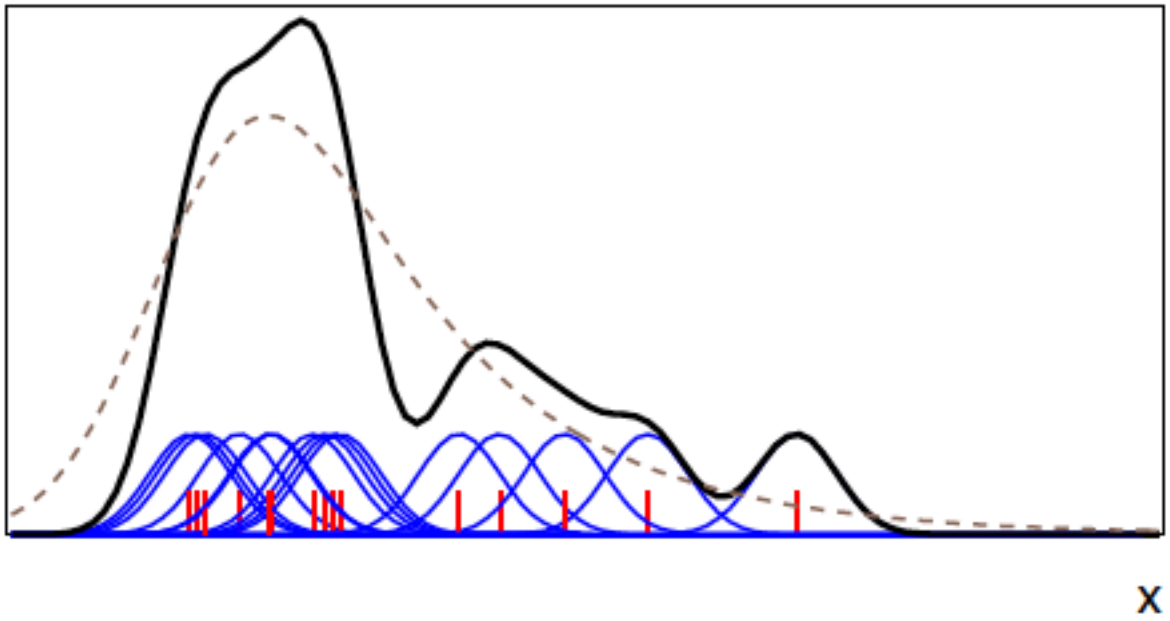
Gaussian KDE in 1-dimension (cont.)

The KDE estimate the pdf is given by the sum of all of the Gaussians:



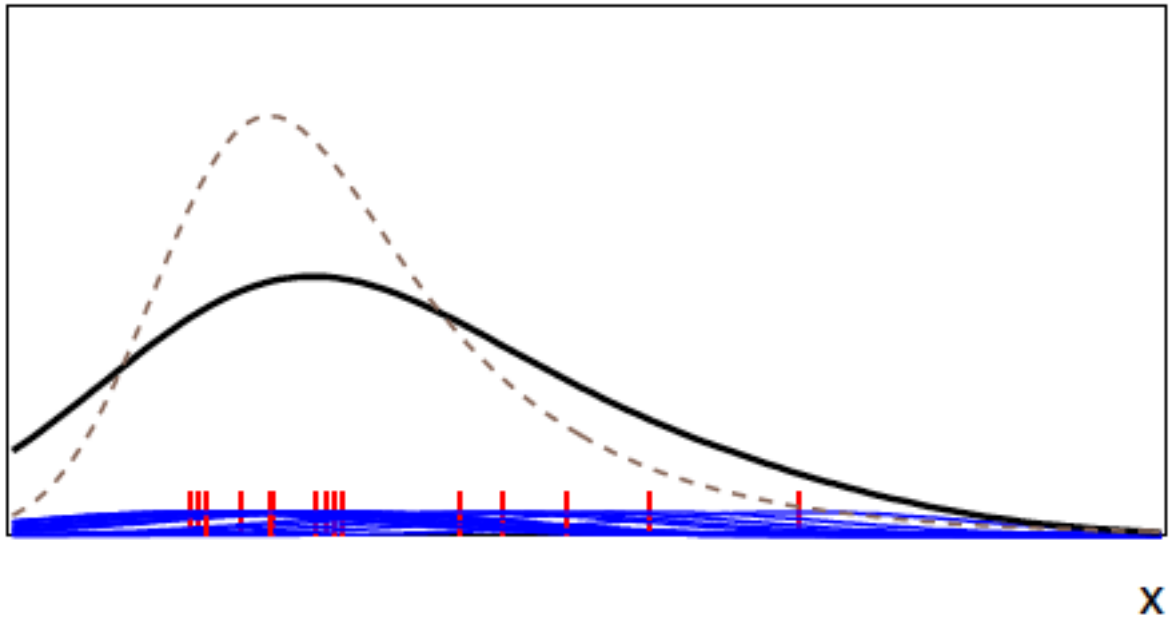
Choice of kernel width

The width h of the Gaussians is analogous to the bin width of a histogram. If it is too small, the estimator has noise:



Choice of kernel width (cont.)

If width of Gaussian kernels too large, structure is washed out:



KDE discussion

Various strategies can be applied to choose width h of kernel based trade-off between bias and variance (noise).

Adaptive KDE allows width of kernel to vary, e.g., wide where target pdf is low (few events); narrow where pdf is high.

Advantage of KDE: no training!

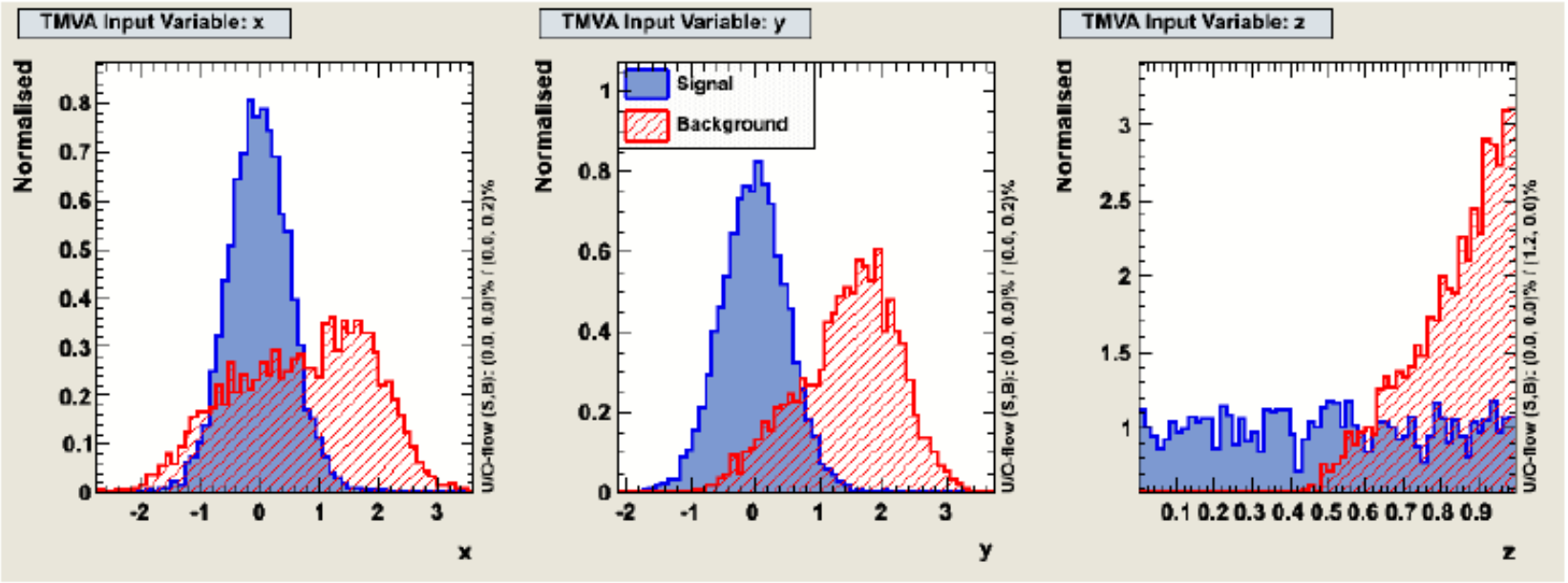
Disadvantage of KDE: to evaluate we need to sum N_{event} terms, so if we have many events this can be slow.

Special treatment required if kernel extends beyond range where pdf defined. Can e.g., renormalize the kernels to unity inside the allowed range; alternatively "mirror" the events about the boundary (contribution from the mirrored events exactly compensates the amount lost outside the boundary).

Software in ROOT: RooKeysPdf (K. Cranmer, CPC 136:198,2001)

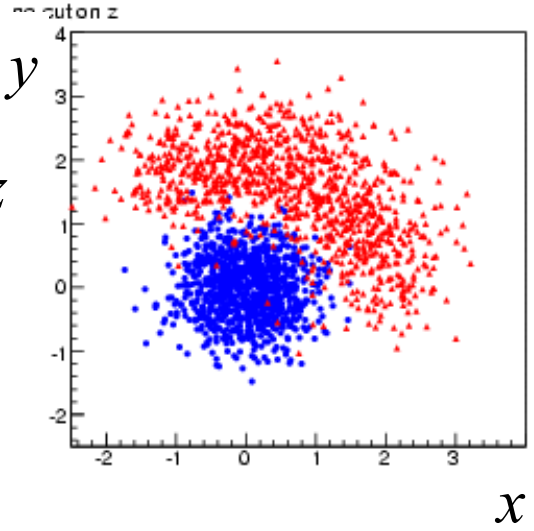
Test example with TMVA

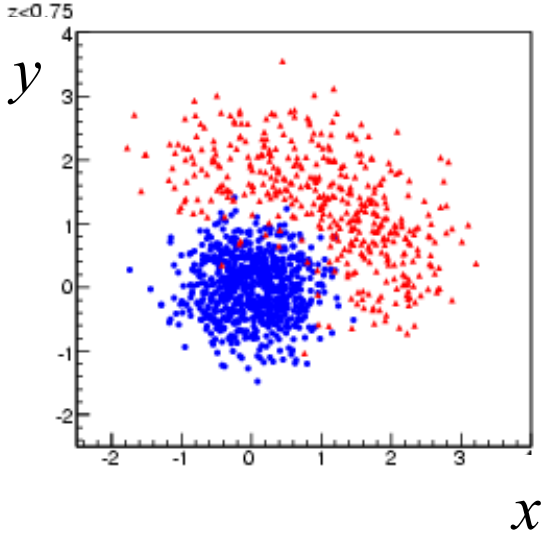
Each event characterized by 3 variables, x, y, z:



Test example (x, y, z)

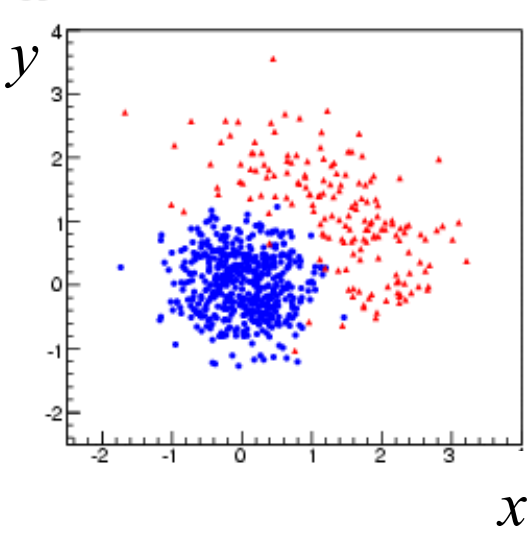
no cut on z

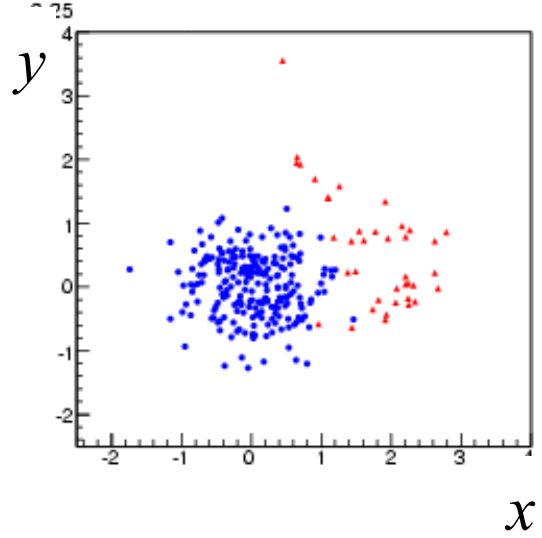




$z < 0.75$

$z < 0.5$





$z < 0.25$

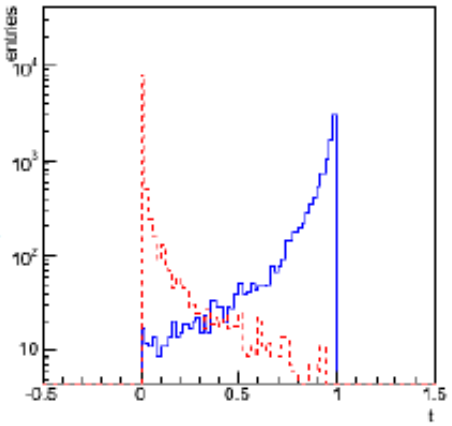
Test example results

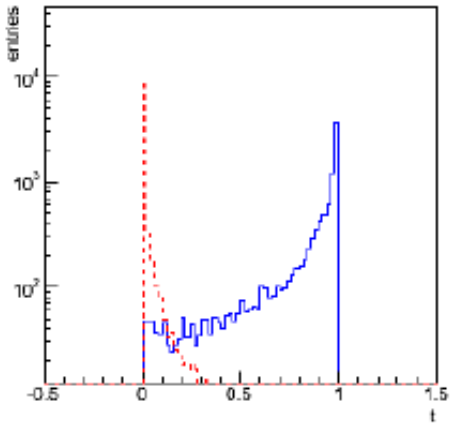
Fisher discriminant

Multilayer perceptron

Find these on next homework assignment.

Naive Bayes, no decorrelation

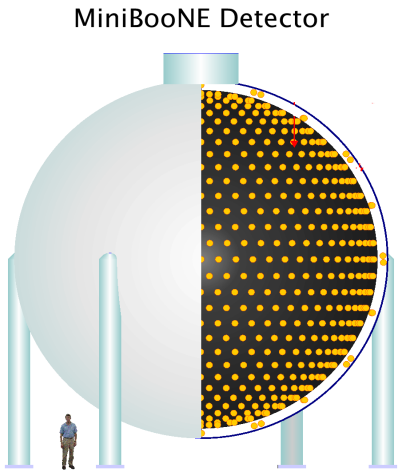




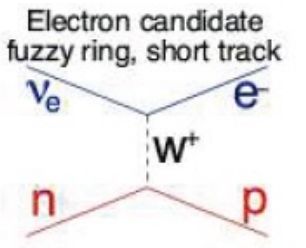
Naive Bayes with decorrelation

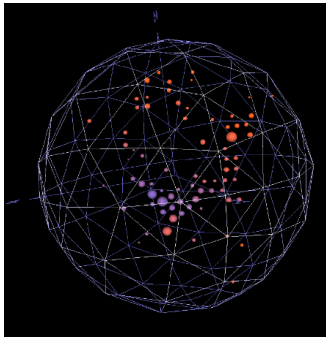
Particle i.d. in MiniBooNE

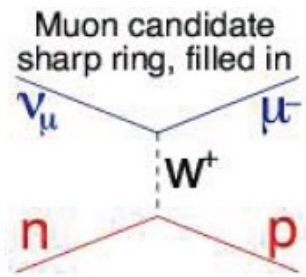
Detector is a 12-m diameter tank of mineral oil exposed to a beam of neutrinos and viewed by 1520 photomultiplier tubes:

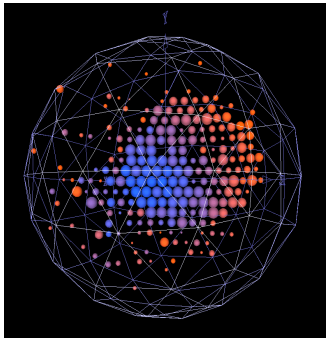


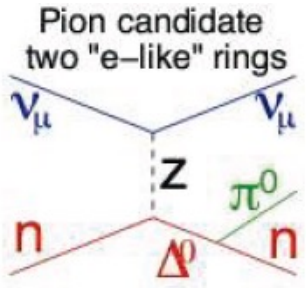
Search for ν_μ to ν_e oscillations required particle i.d. using information from the PMTs.

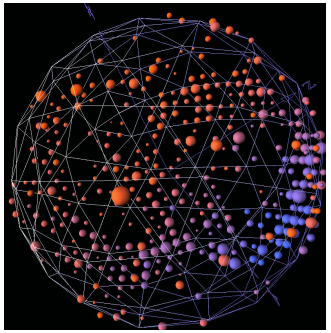












H.J. Yang, MiniBooNE PID, DNP06

Decision trees

Out of all the input variables, find the one for which with a single cut gives best improvement in signal purity:

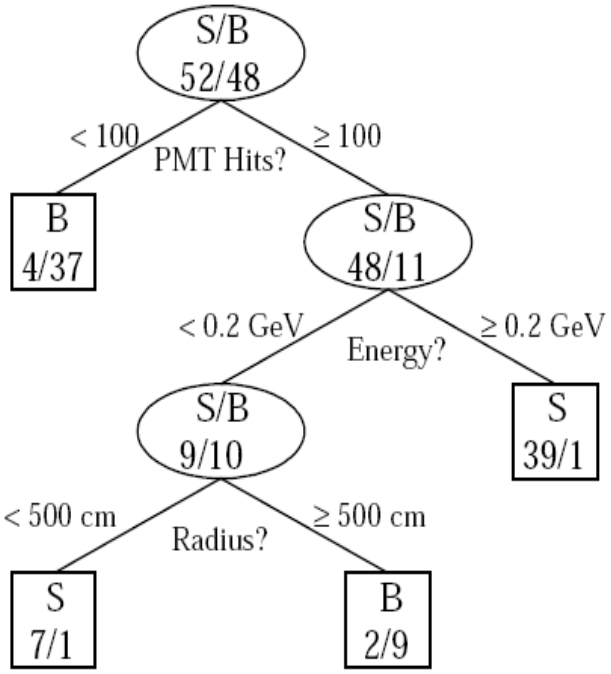
$$P = \frac{\sum_{\text{signal}} w_i}{\sum_{\text{signal}} w_i + \sum_{\text{background}} w_i}$$

where w_i. is the weight of the ith event.

Resulting nodes classified as either signal/background.

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

The set of cuts defines the decision boundary.



Example by MiniBooNE experiment, B. Roe et al., NIM 543 (2005) 577

Finding the best single cut

The level of separation within a node can, e.g., be quantified by the *Gini coefficient*, calculated from the (s or b) purity as:

$$G = p(1-p)$$

For a cut that splits a set of events a into subsets b and c, one can quantify the improvement in separation by the change in weighted Gini coefficients:

$$\Delta = W_a G_a - W_b G_b - W_c G_c \quad \text{where, e.g.,} \quad W_a = \sum_{i \in a} w_i$$

Choose e.g. the cut to the maximize Δ; a variant of this scheme can use instead of Gini e.g. the misclassification rate:

$$\varepsilon = 1 - \max(p, 1-p)$$

Decision trees (2)

The terminal nodes (leaves) are classified a signal or background depending on majority vote (or e.g. signal fraction greater than a specified threshold).

This classifies every point in input-variable space as either signal or background, a decision tree classifier, with discriminant function

$$f(\boldsymbol{x}) = 1 \text{ if } \boldsymbol{x} \text{ in signal region, } -1 \text{ otherwise}$$

Decision trees tend to be very sensitive to statistical fluctuations in the training sample.

Methods such as boosting can be used to stabilize the tree.

Boosting

Boosting is a general method of creating a set of classifiers which can be combined to achieve a new classifier that is more stable and has a smaller error than any individual one.

Often applied to decision trees but, can be applied to any classifier.

Suppose we have a training sample T consisting of N events with

$\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ event data vectors (each $\boldsymbol{x}$ multivariate)

$y_1, \ldots, y_N$ true class labels, +1 for signal, −1 for background

$w_1, \ldots, w_N$ event weights

Now define a rule to create from this an ensemble of training samples $T_1, T_2, \ldots$, derive a classifier from each and average them.

Trick is to create modifications in the training sample that give classifiers with smaller error rates than those of the preceding ones.

A successful example is AdaBoost (Freund and Schapire, 1997).

AdaBoost

First initialize the training sample T_1 using the original

$\boldsymbol{x}_1,\ldots,\boldsymbol{x}_N$	event data vectors
$\boldsymbol{y}_1,\ldots,\boldsymbol{y}_N$	true class labels (+1 or −1)
$\boldsymbol{w}_1^{(1)},\ldots,\boldsymbol{w}_N^{(1)}$	event weights

with the weights equal and normalized such that

$$\sum_{i=1}^{N} w_i^{(1)} = 1$$

Then train the classifier $f_1(\boldsymbol{x})$ (e.g., a decision tree) with a method that uses the event weights. Recall for an event at point $\boldsymbol{x}$,

$f_1(\boldsymbol{x}) = +1$ for $\boldsymbol{x}$ in signal region, −1 in background region

We will define an iterative procedure that gives a series of classifiers $f_1(\boldsymbol{x})$, $f_2(\boldsymbol{x})$,...

Error rate of the *k*th classifier

At the *k*th iteration the classifier $f_k(\boldsymbol{x})$ has an error rate

$$\varepsilon_k = \sum_{i=1}^{N} w_i^{(k)} I(y_i f_k(\mathbf{x}_i) \leq 0)$$

where $I(X) = 1$ if X is true and is zero otherwise.

Next assign a score to the *k*th classifier based on its error rate,

$$\alpha_k = \frac{1}{2} \ln \frac{1 - \varepsilon_k}{\varepsilon_k}$$

Updating the event weights

The classifier at each iterative step is found from an updated training sample, in which the weight of event *i* is modified from step *k* to step *k*+1 according to

$$w_i^{(k+1)} = w_i^{(k)} \frac{e^{-\alpha_k f_k(\mathbf{x}_i) y_i}}{Z_k}$$

Here Z_k is a normalization factor defined such that the sum of the weights over all events is equal to one.

That is, the weight for event *i* is increased in the *k*+1 training sample if it was classified incorrectly in step *k*.

Idea is that next time around the classifier should pay more attention to this event and try to get it right.

Defining the classifier

After K boosting iterations, the final classifier is defined as a weighted linear combination of the $f_k(\boldsymbol{x})$,

$$t(\mathbf{x}) = \sum_{k=1}^{K} \alpha_k f_k(\mathbf{x})$$

One can show that the error rate on the training data of the final classifier satisfies the bound

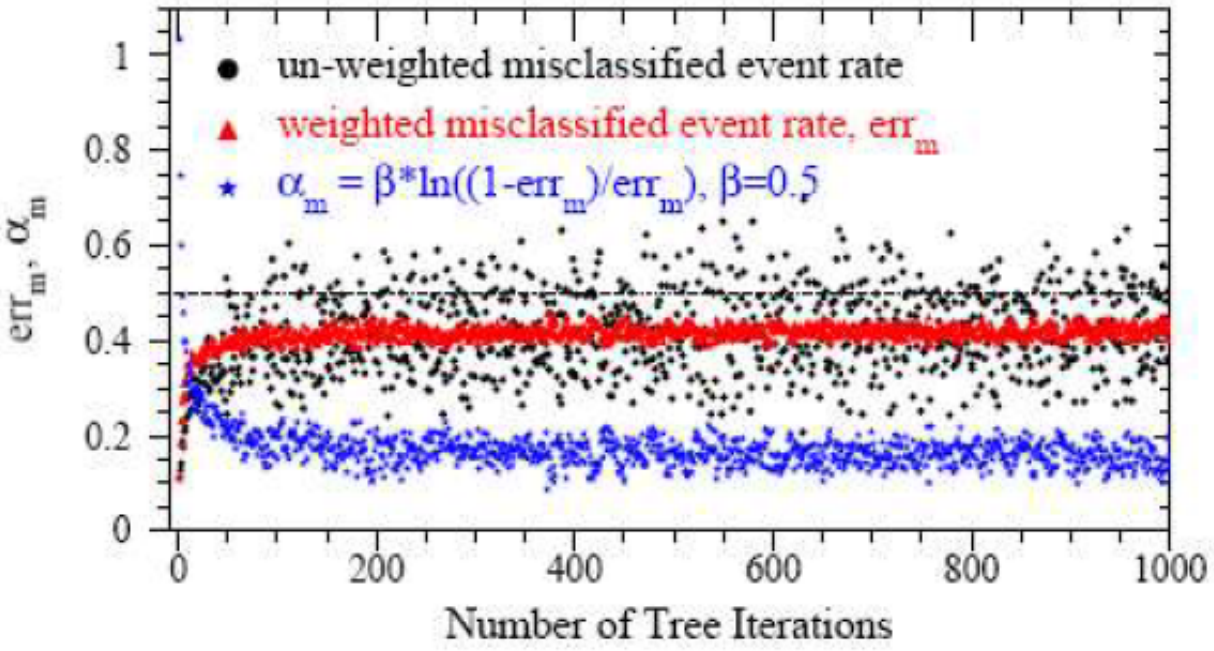
$$\varepsilon \leq \prod_{k=1}^{K} 2\sqrt{\varepsilon_k(1-\varepsilon_k)}$$

i.e. as long as the $\varepsilon_k < ½$ (better than random guessing), with enough boosting iterations every event in the training sample will be classified correctly.

BDT example from MiniBooNE

~200 input variables for each event (ν interaction producing e, μ or π).

Each individual tree is relatively weak, with a misclassification error rate ~ 0.4 – 0.45

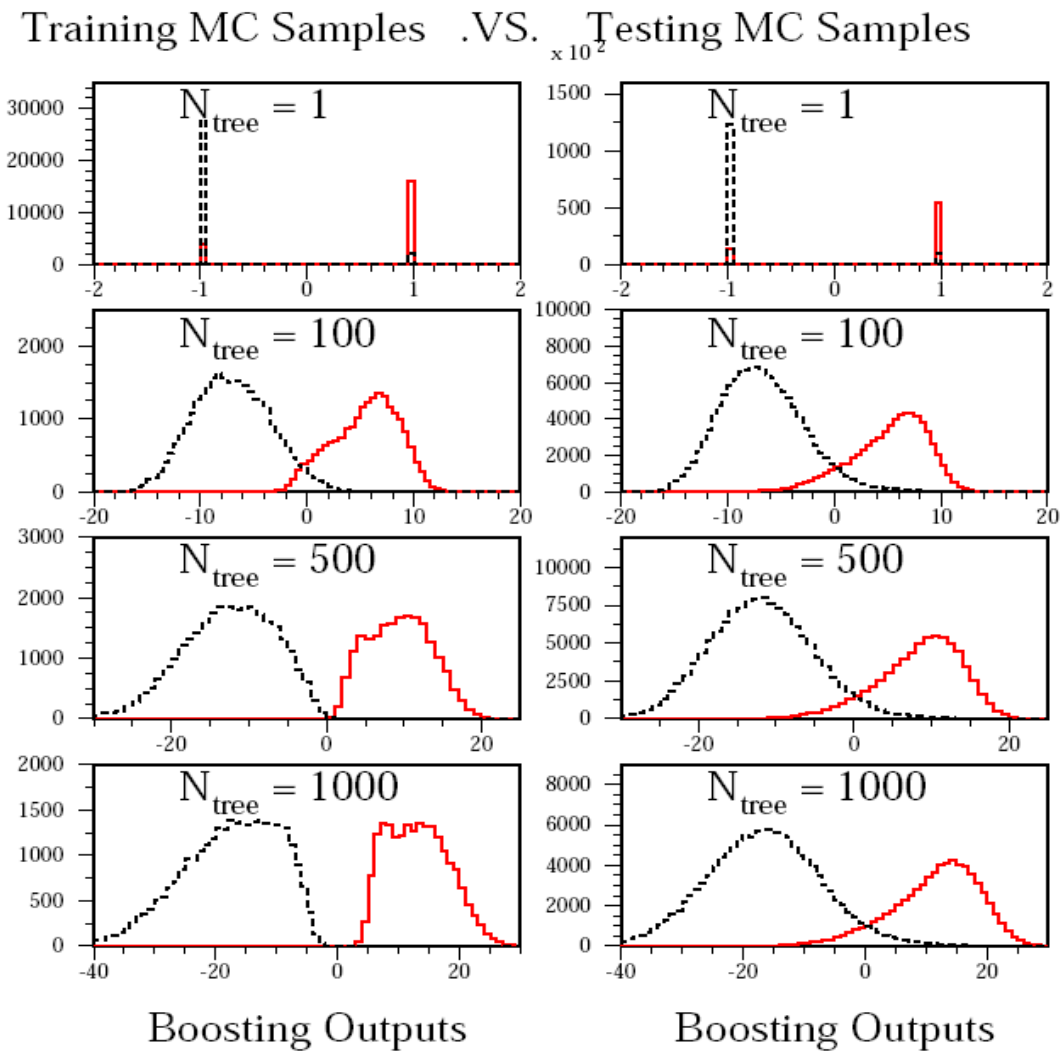


B. Roe et al., NIM 543 (2005) 577

Monitoring overtraining

From MiniBooNE example:

Performance stable after a few hundred trees.



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).

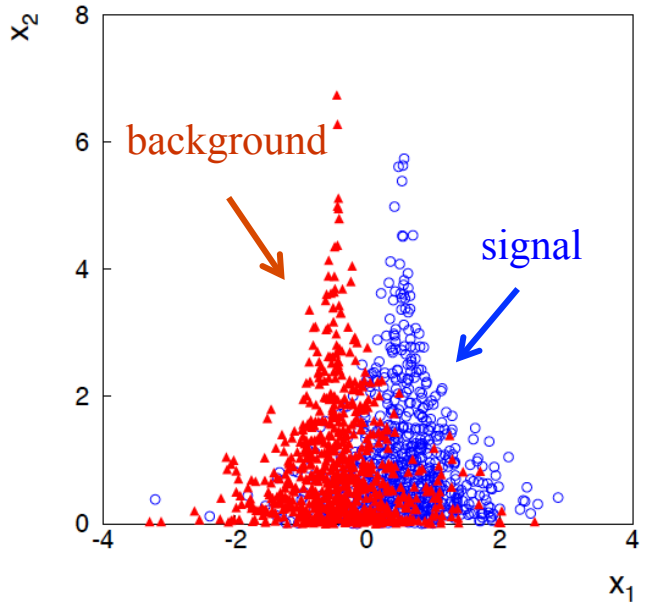
$f(x_1|x_2)$ ~ Gaussian, different means for s/b,
Gaussians have same σ, which depends on x_2,
$f(x_2)$ ~ exponential, same for both s and b,
$f(x_1, x_2) = f(x_1|x_2)\, f(x_2)$:

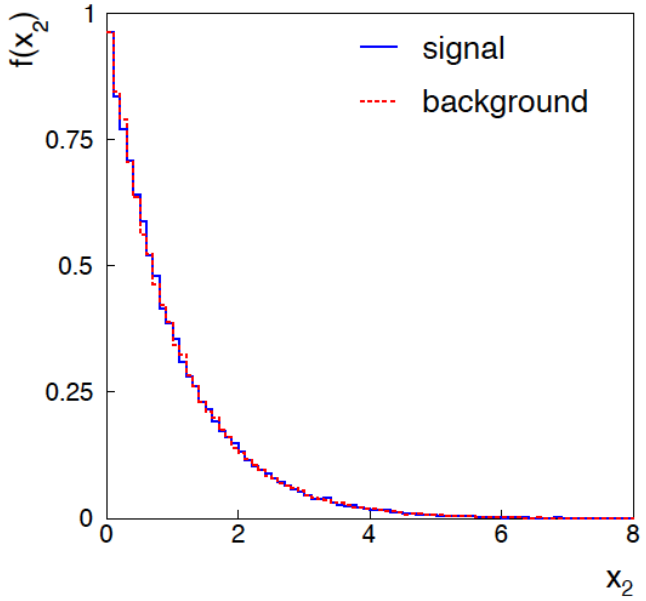
$$f(x_1, x_2|\mathrm{s}) = \frac{1}{\sqrt{2\pi}\sigma(x_2)} e^{-(x_1-\mu_\mathrm{s})^2/2\sigma^2(x_2)} \frac{1}{\lambda} e^{-x_2/\lambda}$$

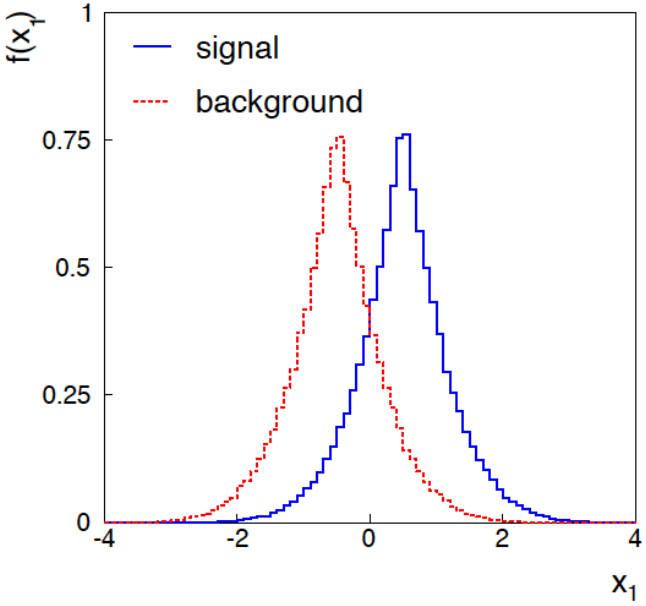
$$f(x_1, x_2|\mathrm{b}) = \frac{1}{\sqrt{2\pi}\sigma(x_2)} e^{-(x_1-\mu_\mathrm{b})^2/2\sigma^2(x_2)} \frac{1}{\lambda} e^{-x_2/\lambda}$$

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

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:

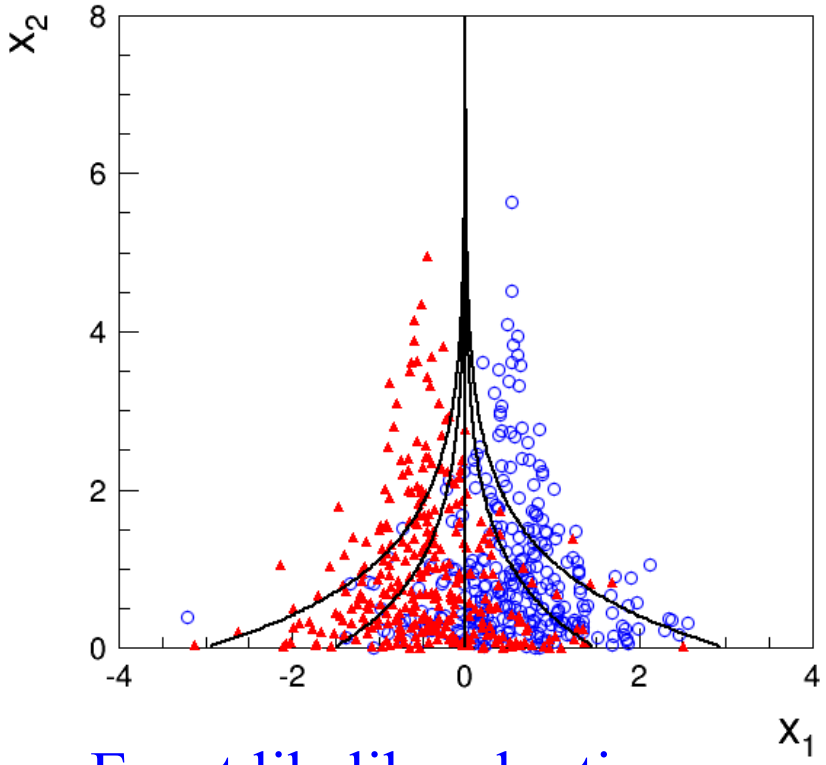
$$t(x_1, x_2) = \frac{f(x_1, x_2|\mathrm{s})}{f(x_1, x_2|\mathrm{b})}$$

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

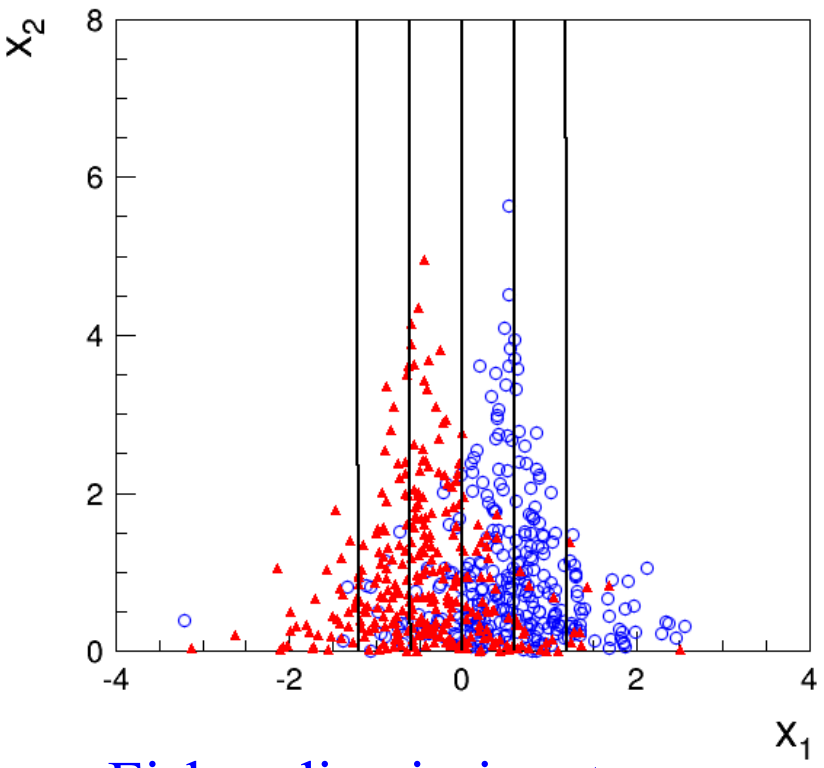
$$\ln t = \frac{\frac{1}{2}(\mu_\mathrm{b}^2 - \mu_\mathrm{s}^2) + (\mu_\mathrm{s} - \mu_\mathrm{b})x_1}{\sigma_0^2 e^{-2x_2/\xi}}$$

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

Contours of constant MVA output

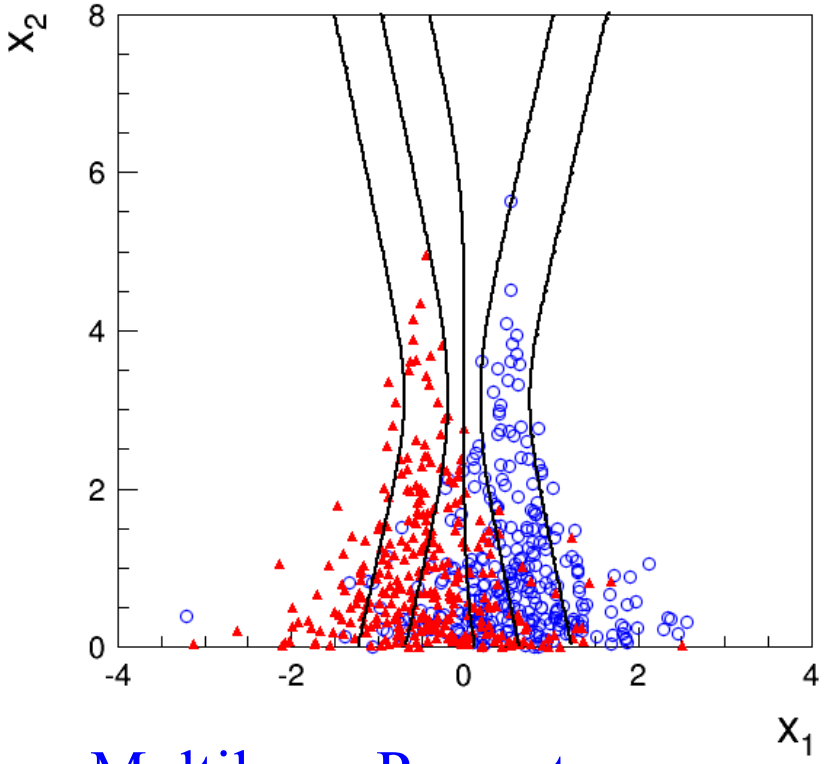


Exact likelihood ratio

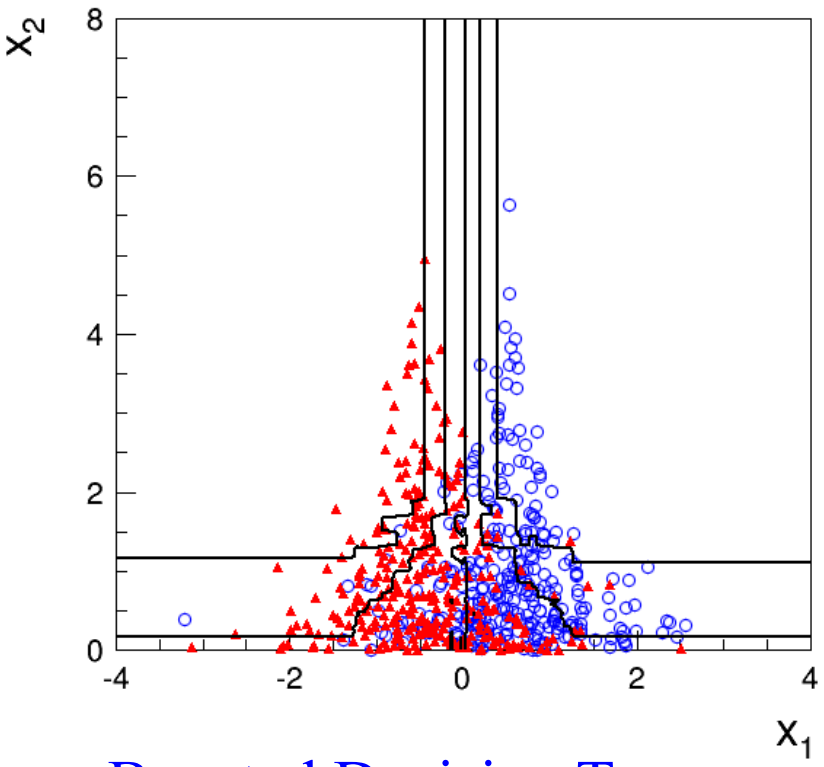


Fisher discriminant

Contours of constant MVA output



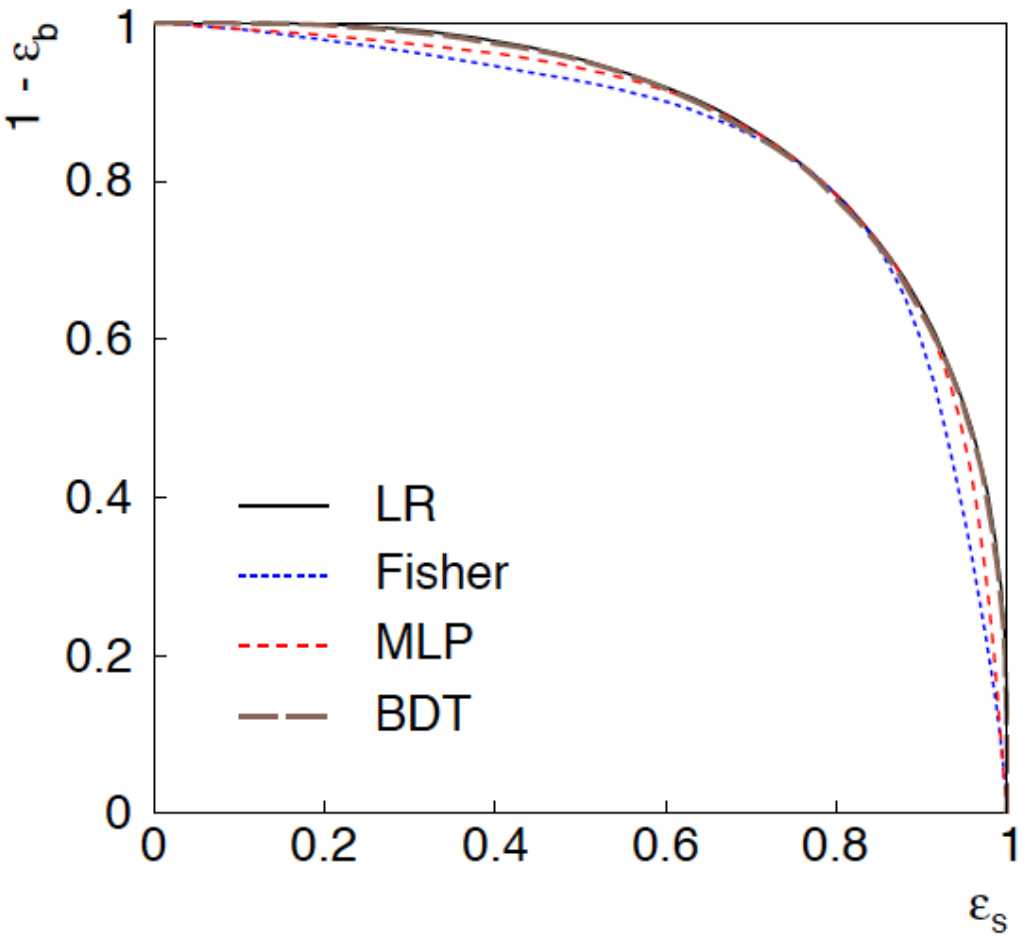
Multilayer Perceptron
1 hidden layer with 2 nodes



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 ($1-\varepsilon_b$) versus signal efficiency ε_s.

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

2D Example: discussion

Even though the distribution of x_2 is same for signal and background, x_1 and x_2 are not independent, so using x_2 as an input variable helps.

Here we can understand why: high values of x_2 correspond to a smaller σ for the Gaussian of x_1. So high x_2 means that the value of x_1 was well measured.

If we don't consider x_2, then all of the x_1 measurements are lumped together. Those with large σ (low x_2) "pollute" the well measured events with low σ (high x_2).

Often in HEP there may be variables that are characteristic of how well measured an event is (region of detector, number of pile-up vertices,...). Including these variables in a multivariate analysis preserves the information carried by the well-measured events, leading to improved performance.

Summary on multivariate methods

Particle physics has used several multivariate methods for many years:

linear (Fisher) discriminant
neural networks
naive Bayes

and has in recent years started to use a few more:

boosted decision trees
support vector machines
kernel density estimation
k-nearest neighbour

The emphasis is often on controlling systematic uncertainties between the modeled training data and Nature to avoid false discovery.

Although many classifier outputs are "black boxes", a discovery at 5σ significance with a sophisticated (opaque) method will win the competition if backed up by, say, 4σ evidence from a cut-based method.