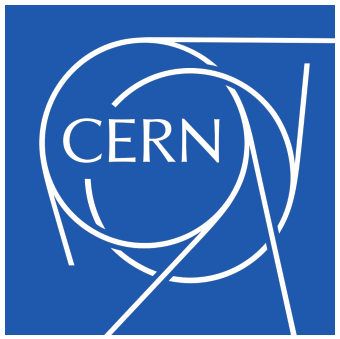
Statistics for Particle Physicists
Lecture 4



Summer Student Lectures
CERN / Zoom
29 June – 2 July 2021

https://indico.cern.ch/event/1051046/

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

Outline

Lecture 1: Introduction, probability,

Lecture 2: Parameter estimation

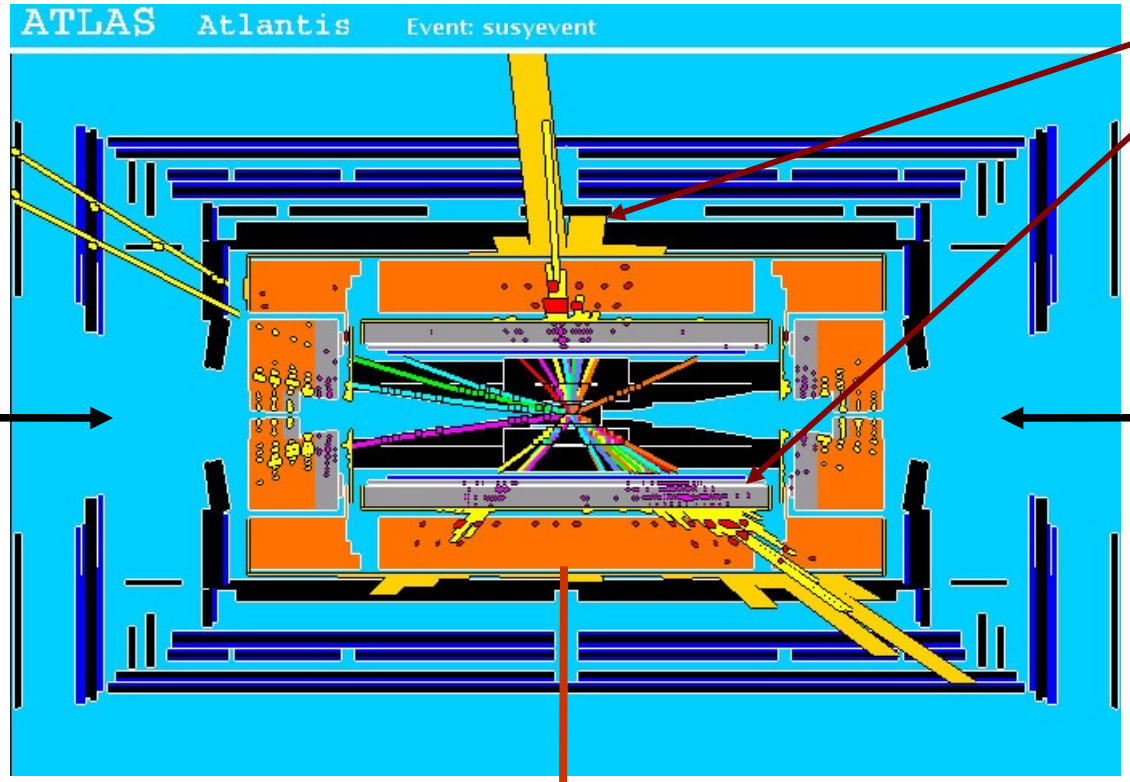
Lecture 3: Hypothesis tests

→ Lecture 4: Further methods:

From hypothesis tests to Machine Learning

Particle Physics context for a hypothesis test

A simulated SUSY event ("signal"):



high p_T muons

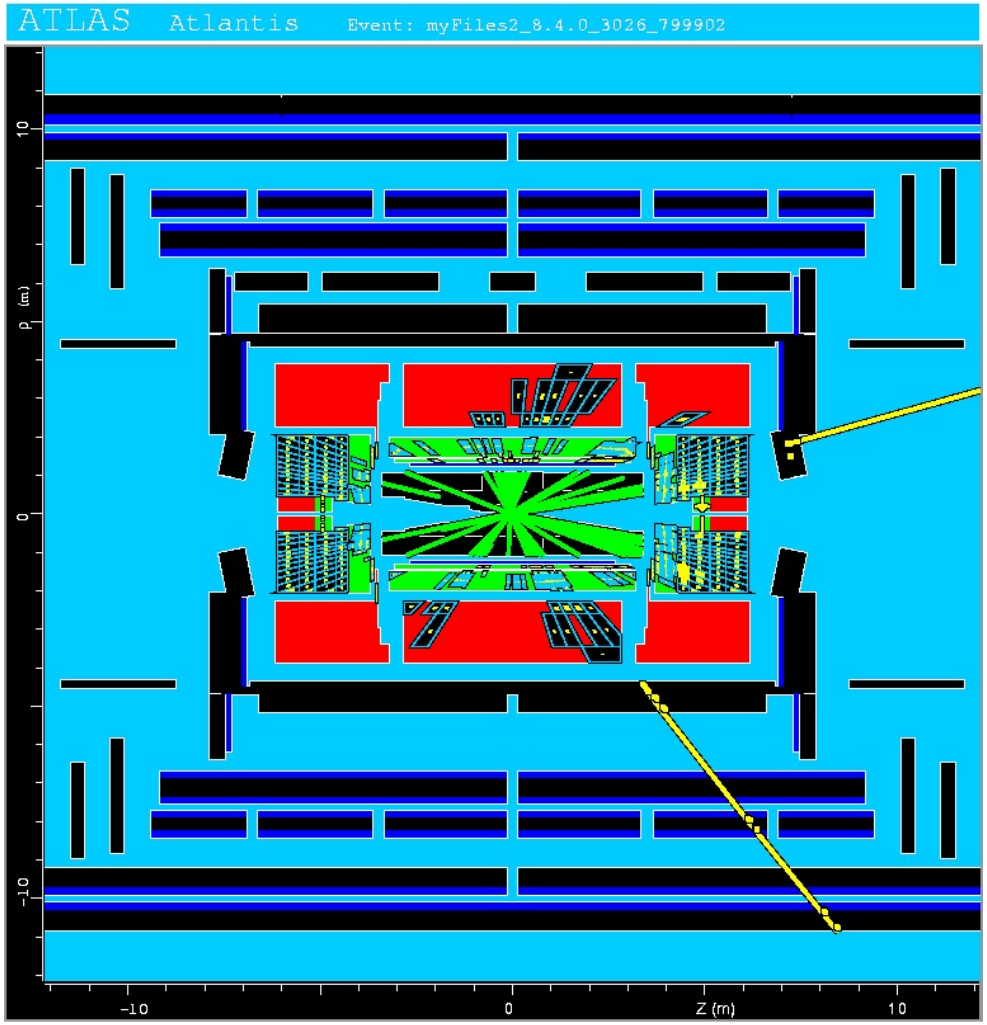
high p_T jets of hadrons

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 signal event.

Classification of proton-proton collisions

Proton-proton collisions can be considered to come in two classes:

signal (the kind of event we're looking for, $y = 1$)
background (the kind that mimics signal, $y = 0$)

For each collision (event), we measure a collection of features:

x_1 = energy of muon
x_2 = angle between jets
x_3 = total jet energy
x_4 = missing transverse energy
x_5 = invariant mass of muon pair
x_6 = ...

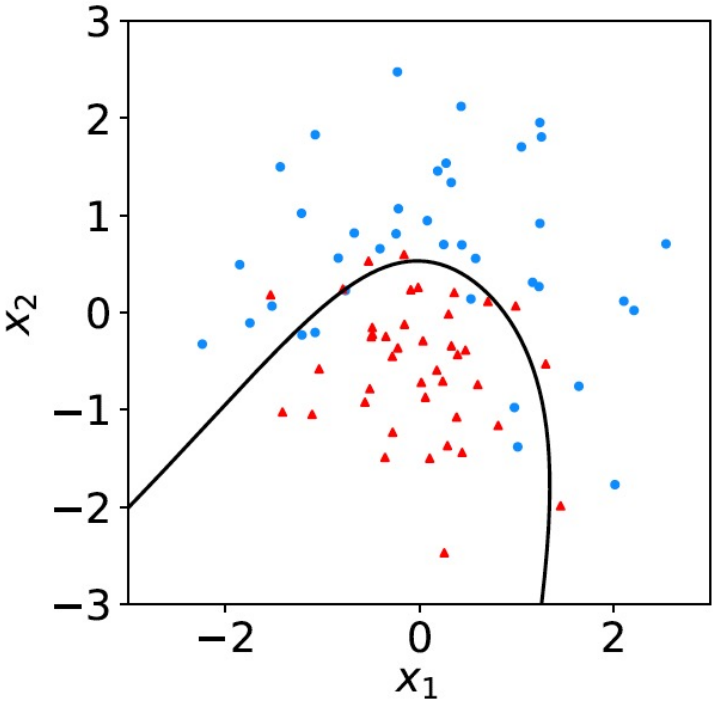
The real events don't come with true class labels, but computer-simulated events do. So we can have a set of simulated events that consist of a feature vector $\boldsymbol{x}$ and true class label y (0 for background, 1 for signal):

$$(\boldsymbol{x}, y)_1, (\boldsymbol{x}, y)_2, \ldots, (\boldsymbol{x}, y)_N$$

The simulated events are called "training data".

Distributions of the features

If we consider only two features $\boldsymbol{x} = (x_1, x_2)$, we can display the results in a scatter plot (red: $y = 0$, blue: $y = 1$).



For real events, the dots are black (true type is not known).

For each real event test the hypothesis that it is background.

(Related to this: test that a sample of events is *all* background.)

The test's critical region is defined by a "decision boundary" – without knowing the event type, we can classify them by seeing where their measured features lie relative to the boundary.

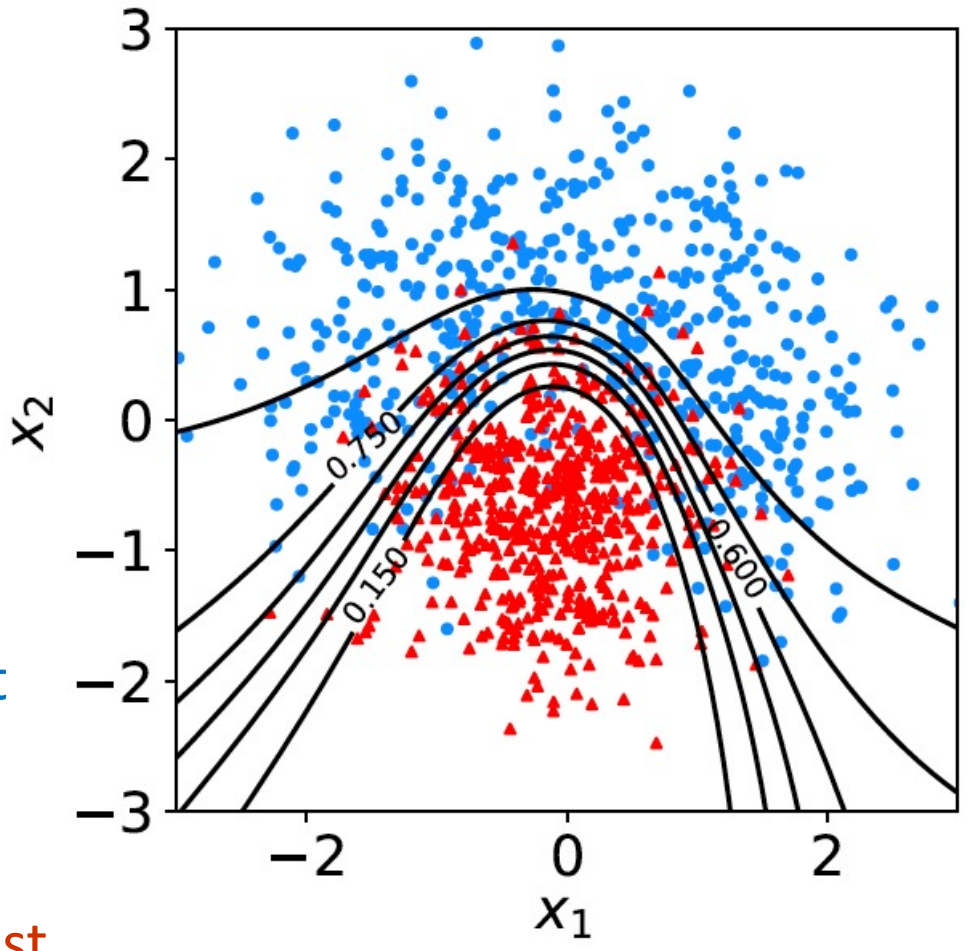
Decision function, test statistic

A surface in an n-dimensional space can be described by

$$t(x_1, \ldots, x_n) = t_c$$

scalar function

constant

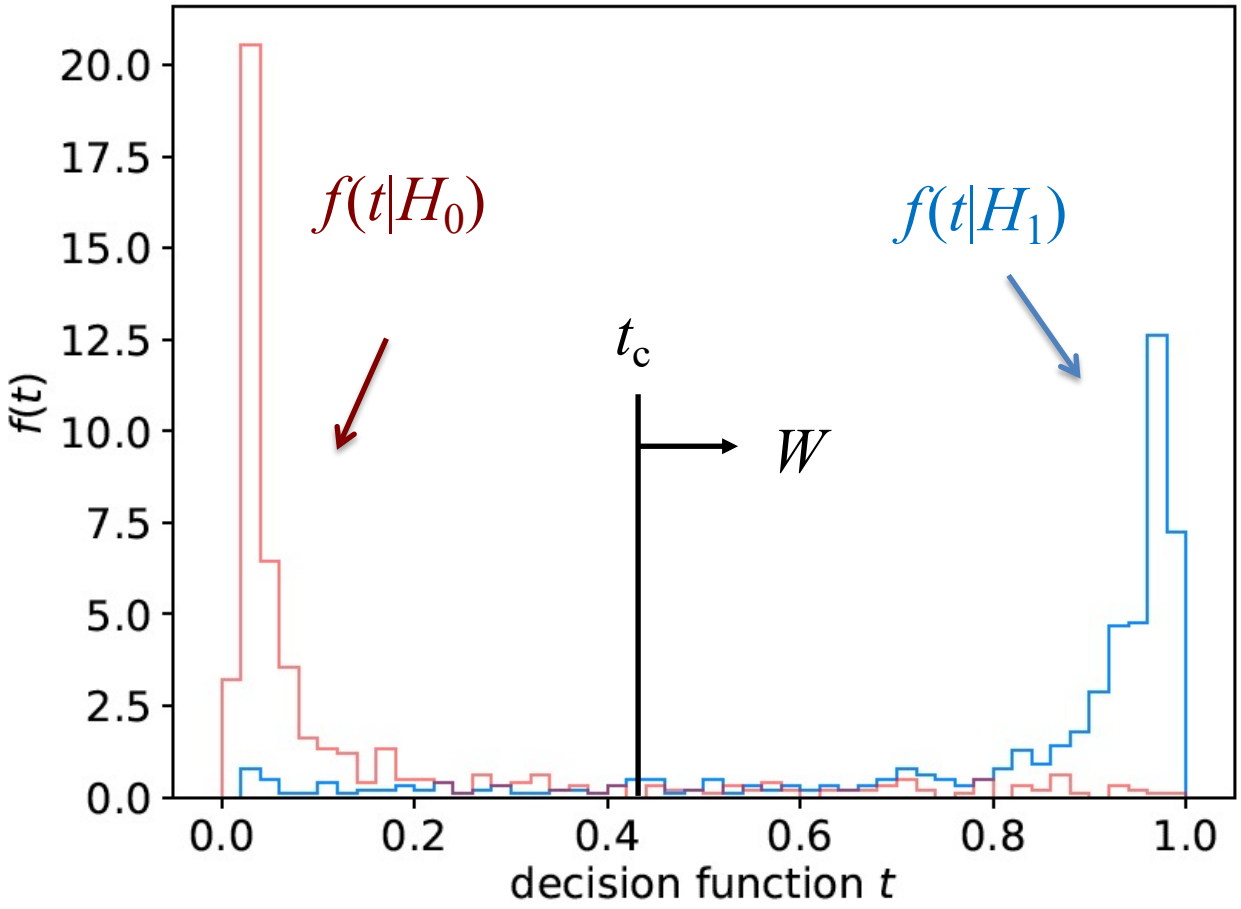


Different values of the constant t_c result in a family of surfaces.

Problem is reduced to finding the best decision function or test statistic $t(\boldsymbol{x})$.

Distribution of $t(\boldsymbol{x})$

By forming a test statistic $t(\boldsymbol{x})$, the boundary of the critical region in the n-dimensional $\boldsymbol{x}$-space is determined by a single single value t_c.

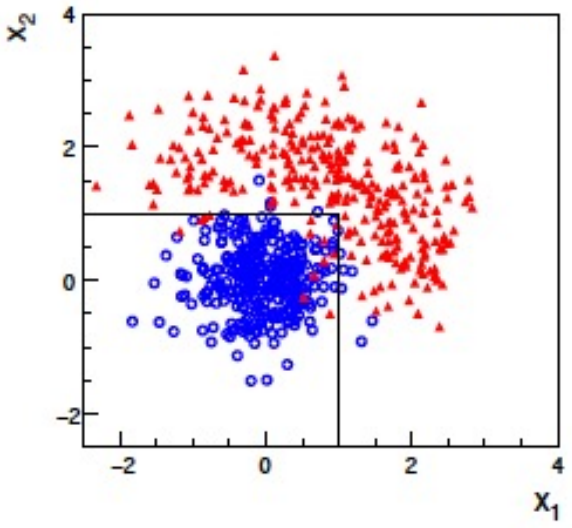


Types of decision boundaries

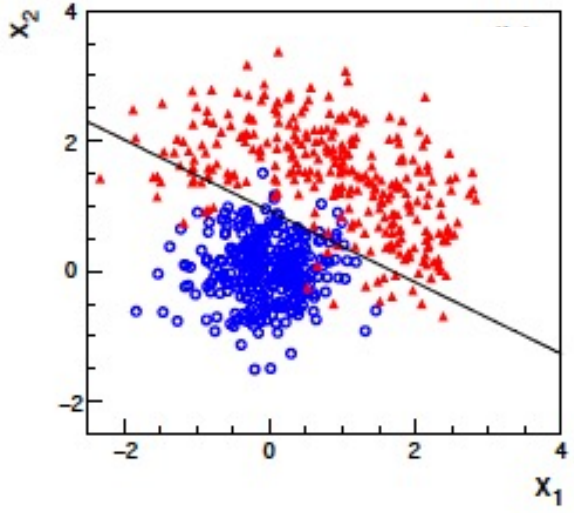
So what is the optimal boundary for the critical region, i.e., what is the optimal test statistic $t(\boldsymbol{x})$?

First find best $t(\boldsymbol{x})$, later address issue of optimal size of test.

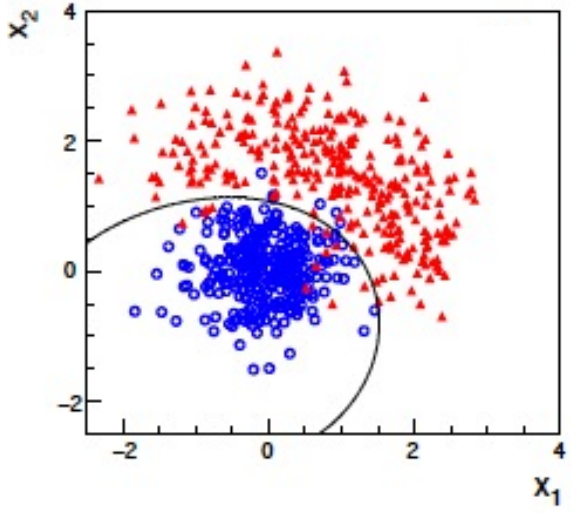
Remember $\boldsymbol{x}$-space can have many dimensions.



"cuts"



linear



non-linear

Test statistic based on likelihood ratio

How can we choose a test's critical region in an 'optimal way', in particular if the data space is multidimensional?

Neyman-Pearson lemma states:

For a test of H_0 of size α, to get the highest power with respect to the alternative H_1 we need for all $\boldsymbol{x}$ in the critical region W

"likelihood ratio (LR)" →

$$\frac{P(\mathbf{x}|H_1)}{P(\mathbf{x}|H_0)} \geq c_\alpha$$

inside W and $\leq c_\alpha$ outside, where c_α is a constant chosen to give a test of the desired size.

Equivalently, optimal scalar test statistic is

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

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

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:

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

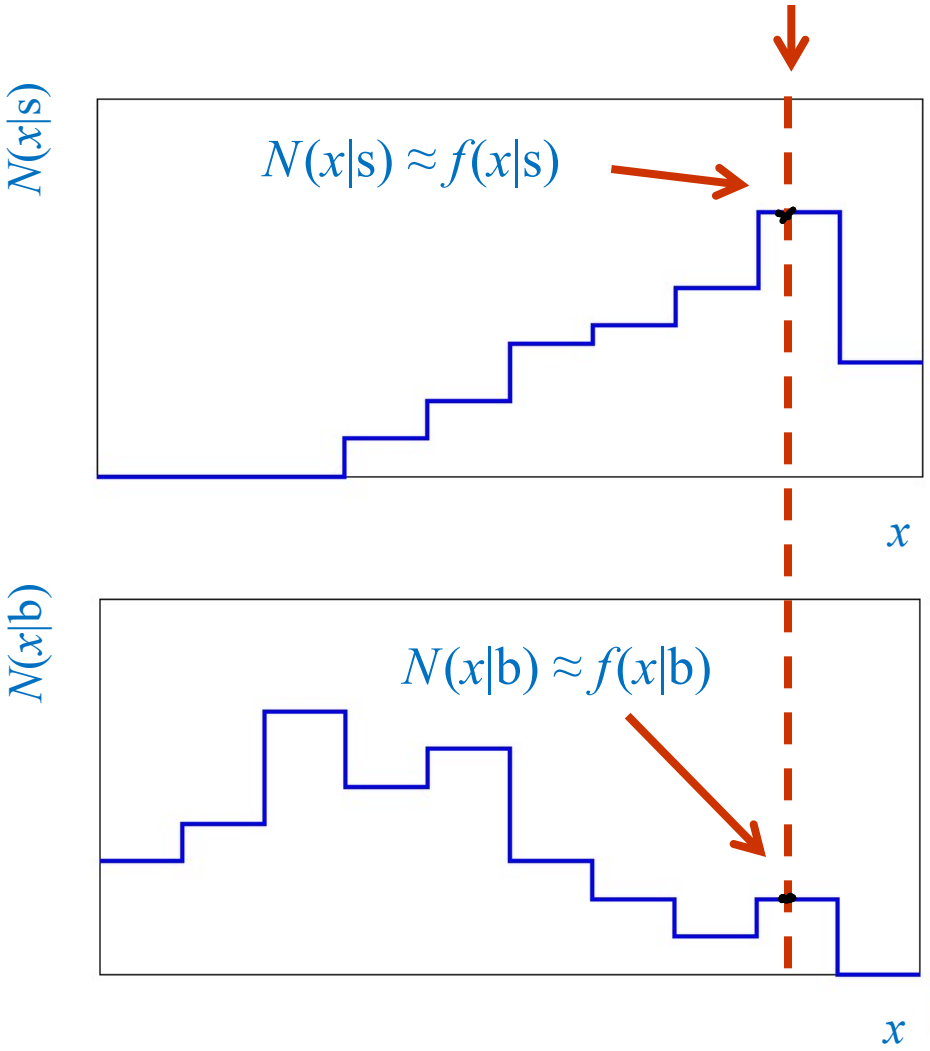
generate $\boldsymbol{x} \sim f(\boldsymbol{x}|\mathrm{b}) \quad \rightarrow \quad \boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$

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

- Use these to construct a statistic that is as close as possible to the optimal likelihood ratio (→ Machine Learning).

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:

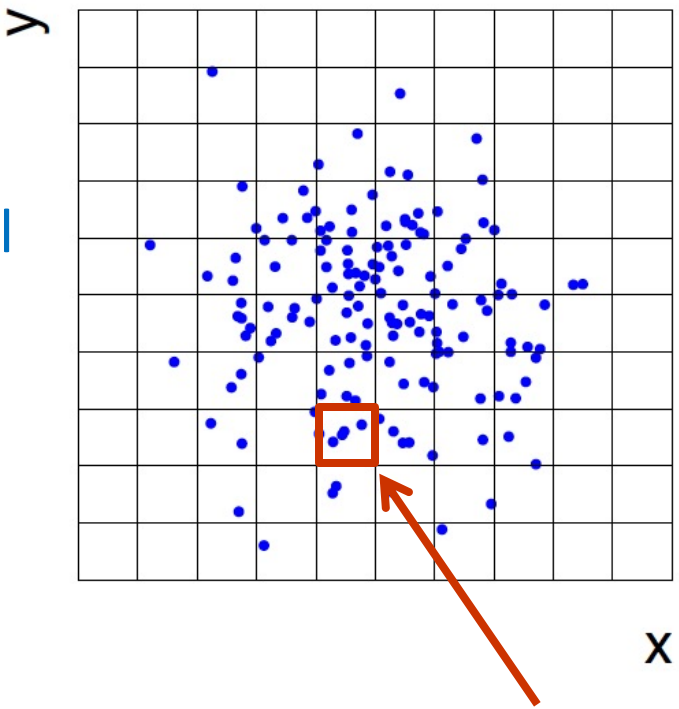
$$t(x) \approx \frac{N(x|s)}{N(x|b)}$$

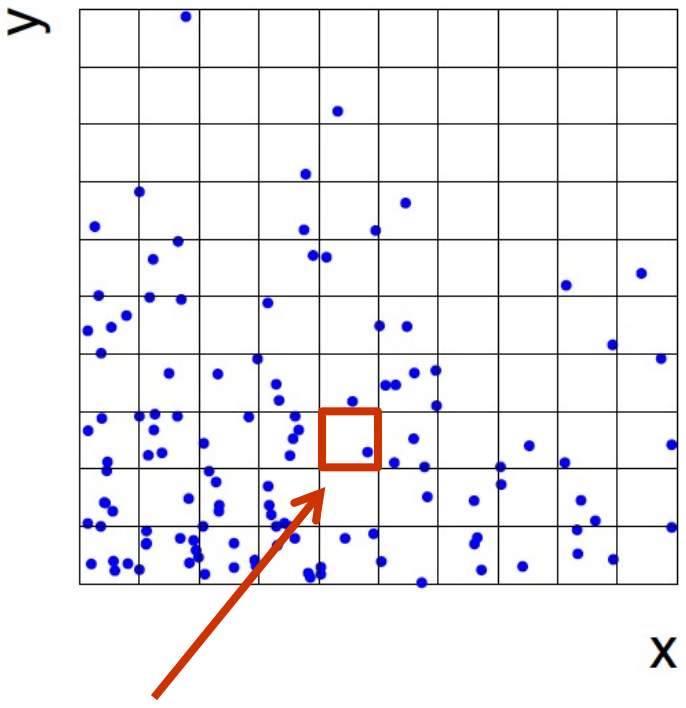
Can work well for single variable.

Approximate LR from 2D-histograms

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

signal





back-ground

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 (Machine Learning)

Many new (and some old) methods:

- Fisher discriminant
- (Deep) Neural Networks
- Kernel density methods
- Support Vector Machines
- Decision trees
  - Boosting
  - Bagging

Software

Rapidly growing area of development – two resources:

scikit-learn

Python-based tools for Machine Learning

scikit-learn.org

Large user community

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

Distributed with ROOT (root.cern.ch)

Variety of classifiers

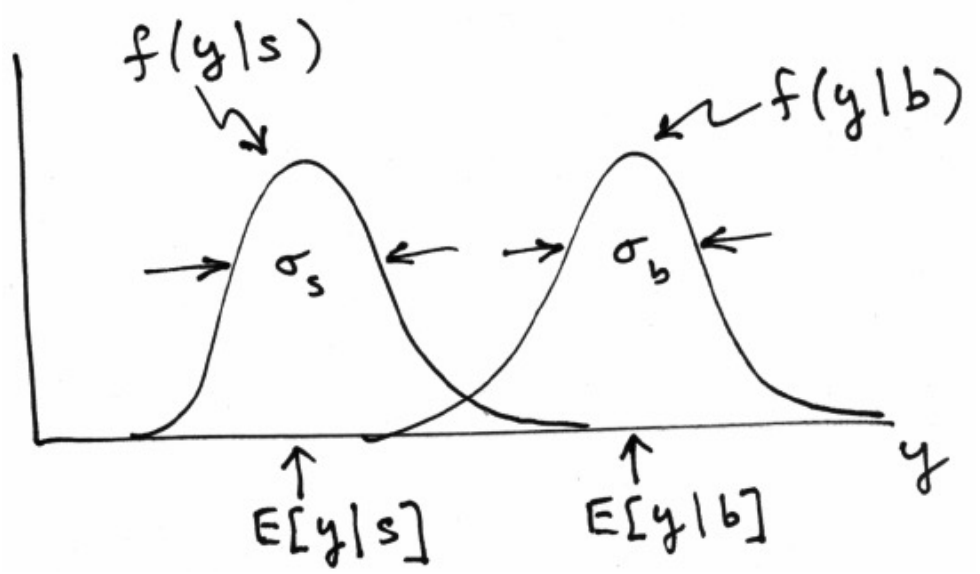
Good manual, widely used in HEP

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:

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

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

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

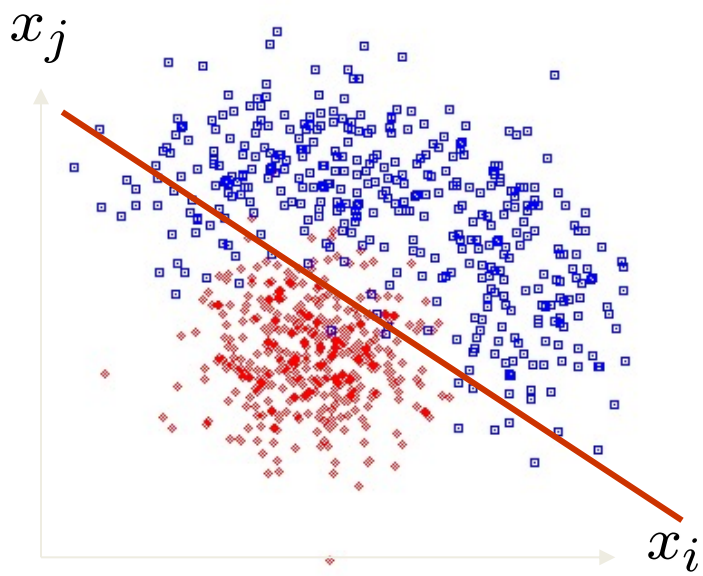
The Fisher discriminant

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

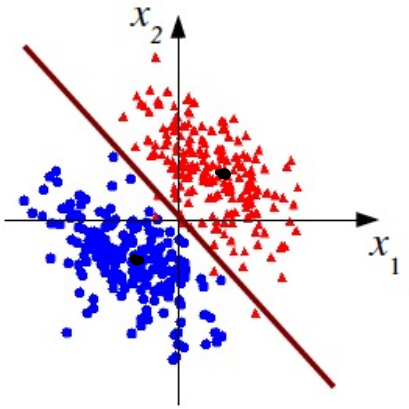
Suppose the pdfs of the input variables, $f(\boldsymbol{x}|\mathrm{s})$ and $f(\boldsymbol{x}|\mathrm{b})$, are both multivariate Gaussians with same covariance but different means:

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

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

Same covariance

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



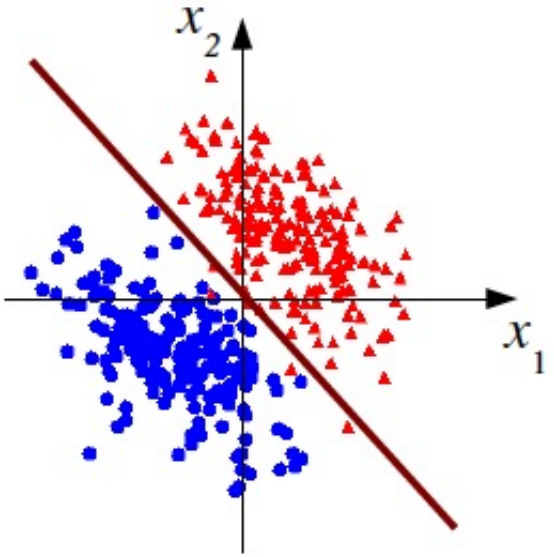
In this case it can be shown that the Fisher discriminant is

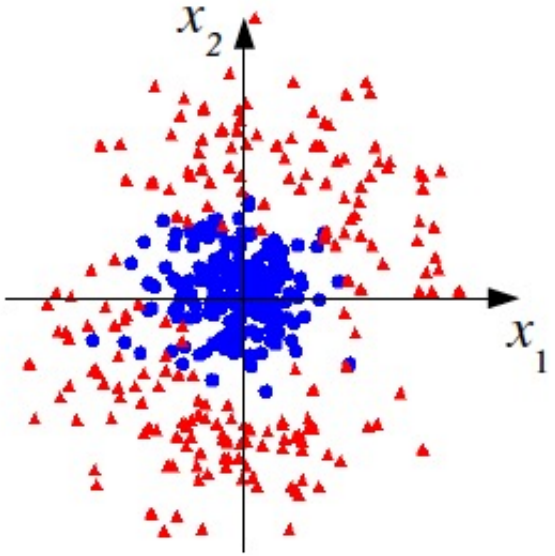
$$y(\mathbf{x}) \sim \ln \frac{f(\mathbf{x}|\mathrm{s})}{f(\mathbf{x}|\mathrm{b})}$$

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.

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.

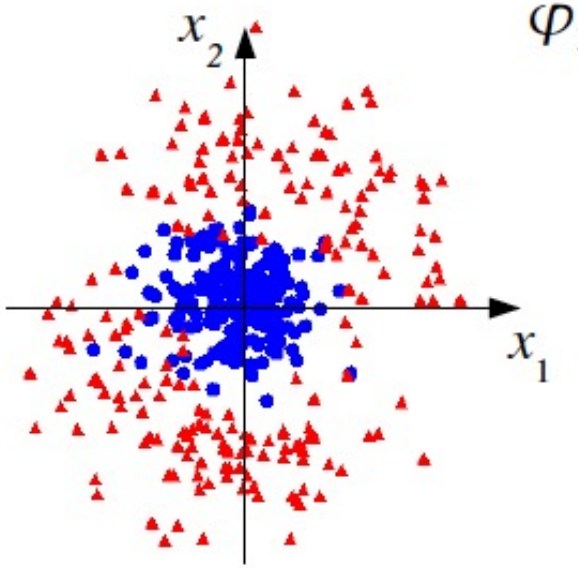
Nonlinear transformation of inputs

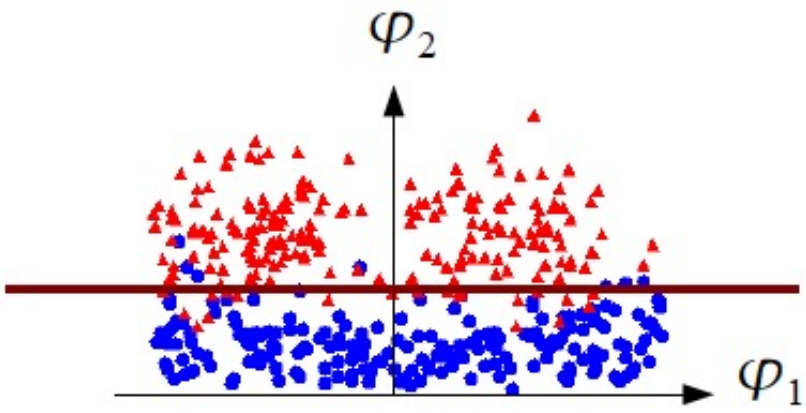
We can try to find a transformation, $x_1, \ldots, x_n \to \varphi_1(\vec{x}), \ldots, \varphi_m(\vec{x})$ so that the transformed "feature space" variables can be separated better by a linear boundary:

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

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

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.

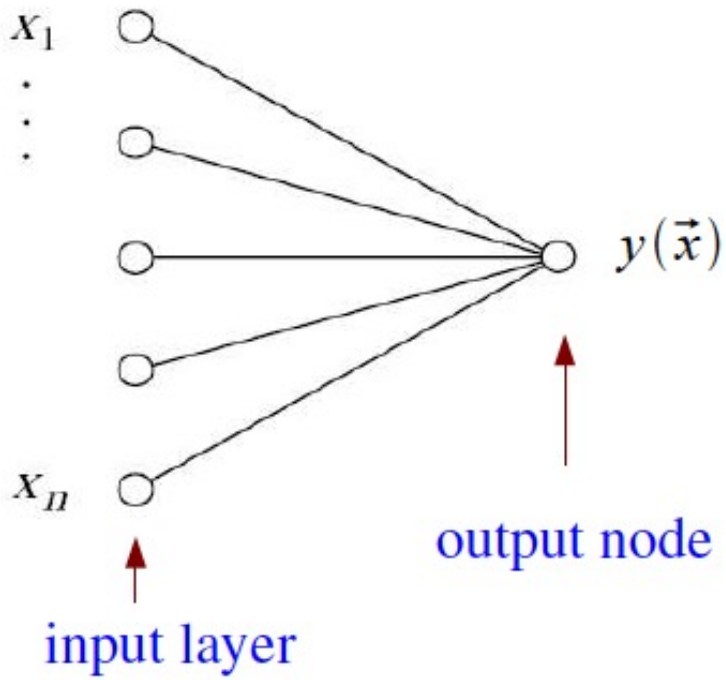
The single layer perceptron

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.

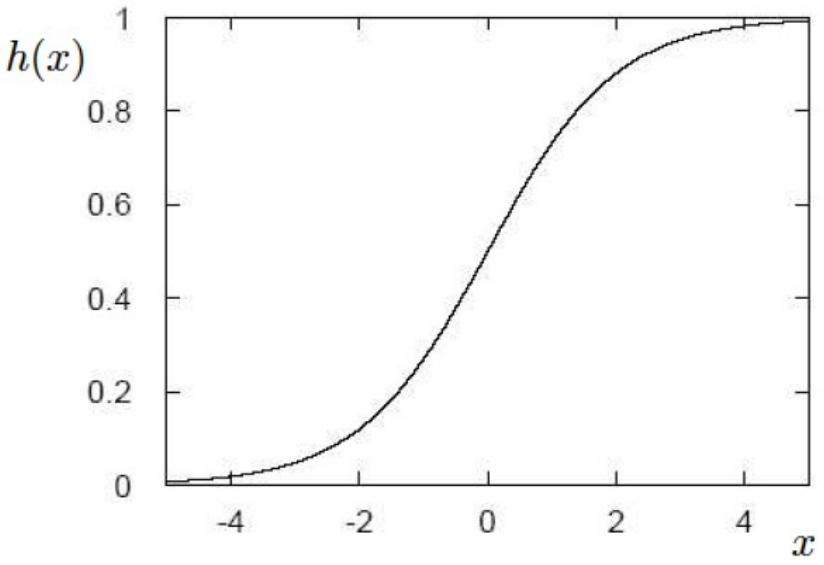
This is called the single layer perceptron:



The activation function

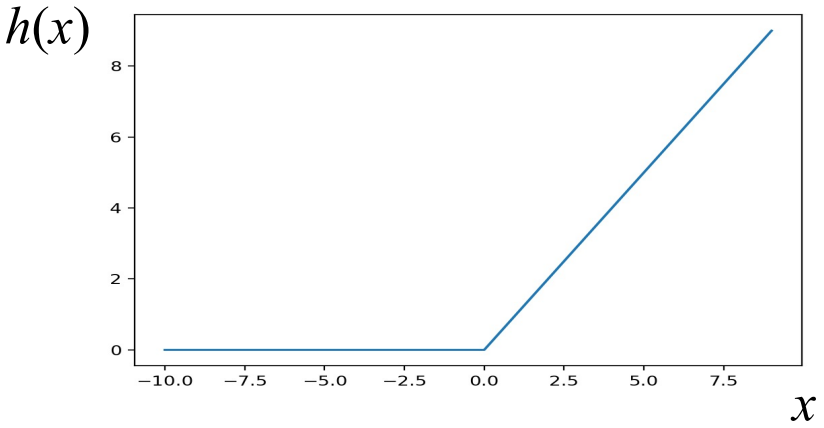
Can use e.g. the ”logistic sigmoid”:

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



or (esp. with deep neural networks) the “Rectified Linear Unit” (ReLU) function:

$$h(x) = \begin{cases} 0 & x \leq 0, \\ x & x > 0. \end{cases}$$



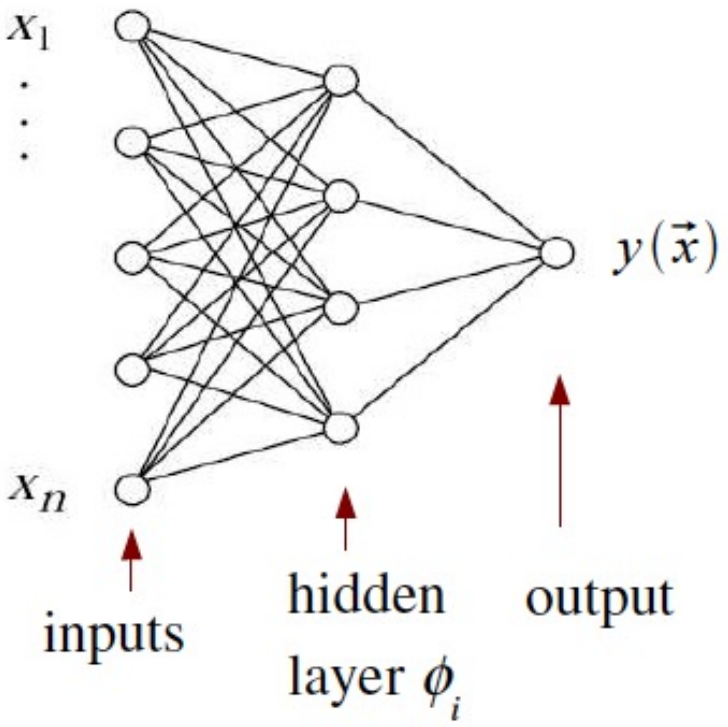
The multilayer perceptron

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 training

For each of the training events we have the feature vector and true event type (class label):

$$x_a = (x_1,..., x_n), \;\; y_a \;\; = 0,1, \;\; a = 0,...,N$$

We have a functional form for the decision function $t(\boldsymbol{x}; \boldsymbol{w})$ that depends on a vector of weights $\boldsymbol{w}$.

Use the training data to determine the weights by minimizing a "loss function". Various possibilities, e.g.,

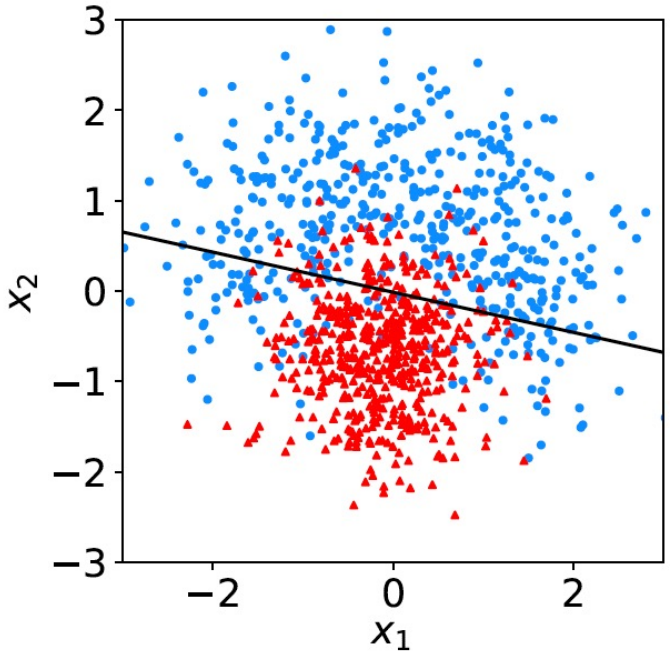
$$E(\mathbf{w}) = \frac{1}{2}\sum_{a=1}^{N} |t(\mathbf{x}_a, \mathbf{w}) - y_a|^2$$

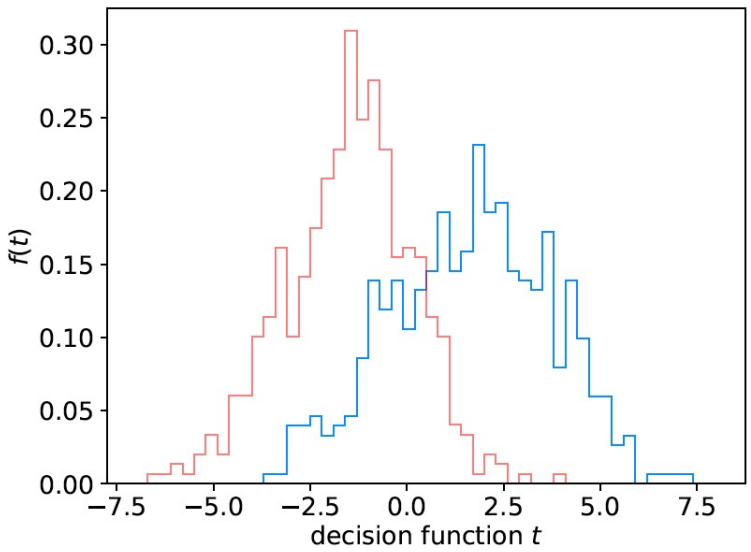
quadratic loss function

$$L_{\mathrm{CE}}(\mathbf{w}) = -\sum_{a=1}^{N} [y_a \log t(\mathbf{x}_a; \mathbf{w}) + (1 - y_a) \log(1 - t(\mathbf{x}_a; \mathbf{w}))]$$

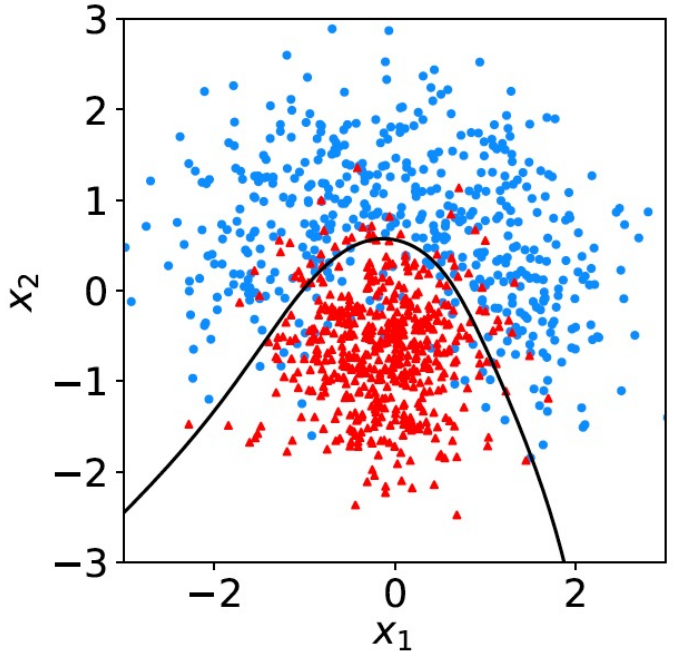
cross entropy

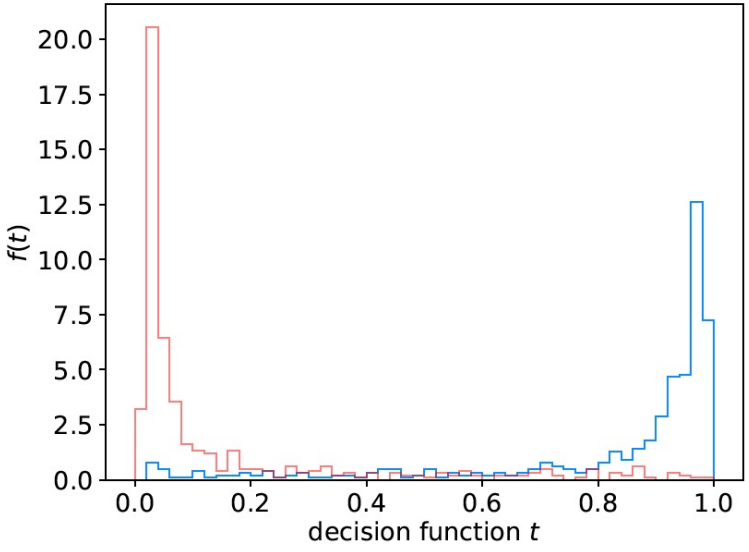
Fisher (linear):





Neural network:

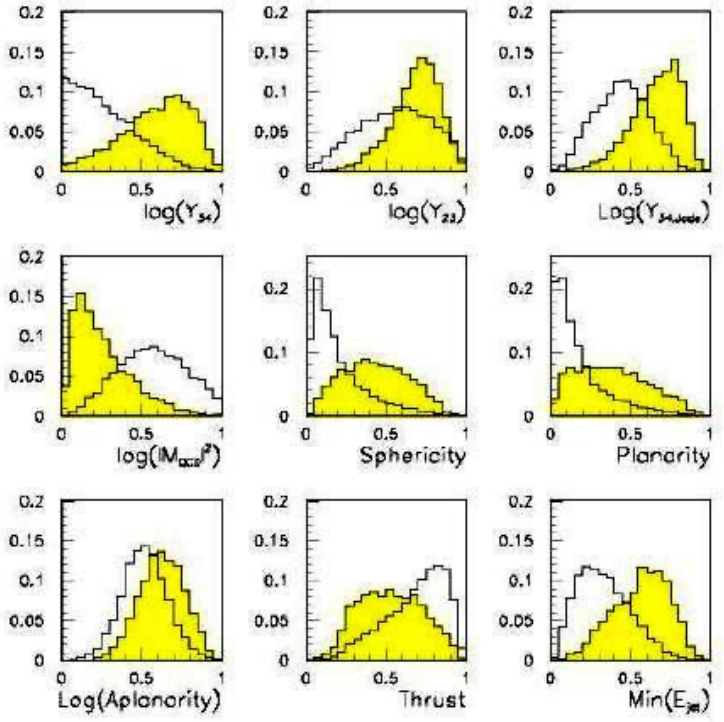




Neural network example from LEP II

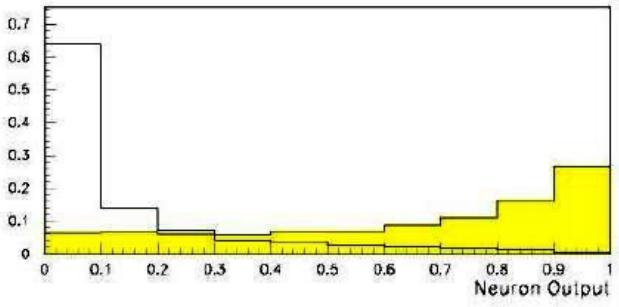
Signal: $e^+e^- \to W^+W^-$ (often 4 well separated hadron jets)

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)

Finally

Four lectures only enough for a brief introduction to:

- Probability, frequentist & Bayesian approaches
- Parameter estimation, maximum likelihood
- Hypothesis tests, *p*-values, limits
- Intro to Machine Learning

Many other important areas:

- Treatment of systematic uncertainties (nuisance parameters)
- Profile likelihood ratio tests, asymptotics,...

Final thought: once the basic formalism is fixed, most of the work focuses on writing down the likelihood, e.g., $P(\boldsymbol{x}|\boldsymbol{\theta})$, and including in it enough parameters to adequately describe the data (true for both Bayesian and frequentist approaches).

Extra slides

How is it we don't have $f(\boldsymbol{x}|H)$?

In a Monte Carlo simulation of a complex process, the fundamental hypothesis does not predict the pdf for the finally measured variables $\boldsymbol{x}$ but rather for some intermediate set of "latent" variables, say, $\boldsymbol{z}_1$.

So in step 1 we sample $\boldsymbol{z}_1 \sim f(\boldsymbol{z}_1|H)$, followed by many further intermediate steps:

$$\boldsymbol{z}_2 \sim f(\boldsymbol{z}_2|\boldsymbol{z}_1)$$
$$\boldsymbol{z}_3 \sim f(\boldsymbol{z}_3|\boldsymbol{z}_2)$$
$$\vdots$$
$$\boldsymbol{x} \sim f(\boldsymbol{x}|\boldsymbol{z}_n)$$

See, e.g., Kyle Cranmer, Johann Brehmer, Gilles Louppe, *The frontier of simulation-based inference*, arXiv:1911.01429 [stat.ML], PNAS doi.org/10.1073/pnas.1912789117

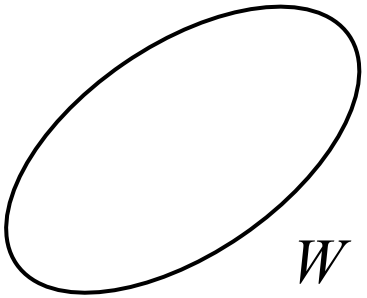
So even though H is fully defined and we can generate $\boldsymbol{x}$ according to it, the formula for $f(\boldsymbol{x}|H)$ is an enormous integral that we cannot compute:

$$f(\mathbf{x}|H) = \int \cdots \int d\mathbf{z}_1 \cdots d\mathbf{z}_n f(\mathbf{x}|\mathbf{z}_n)\, f(\mathbf{z}_n|\mathbf{z}_{n-1}) \cdots f(\mathbf{z}_1|\mathbf{z}_1)\, f(\mathbf{z}_1|H)$$

Proof of Neyman-Pearson Lemma

Consider a critical region W and suppose the LR satisfies the criterion of the Neyman-Pearson lemma:

$$P(\boldsymbol{x}|H_1)/P(\boldsymbol{x}|H_0) \geq c_\alpha \quad \text{for all } \boldsymbol{x} \text{ in } W,$$
$$P(\boldsymbol{x}|H_1)/P(\boldsymbol{x}|H_0) \leq c_\alpha \quad \text{for all } \boldsymbol{x} \text{ not in } W.$$



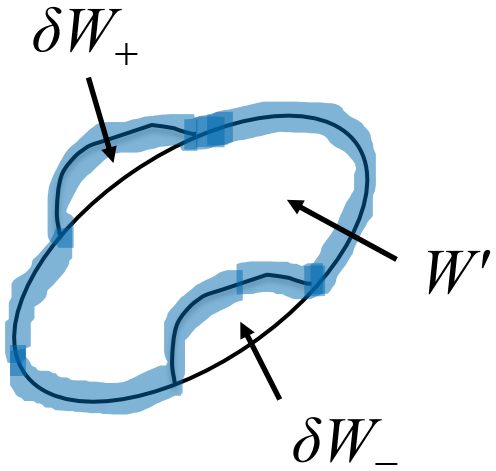
Try to change this into a different critical region W' retaining the same size α, i.e.,

$$P(\mathbf{x} \in W'|H_0) = P(\mathbf{x} \in W|H_0) = \alpha$$

To do so add a part δW_+, but to keep the size α, we need to remove a part δW_-, i.e.,

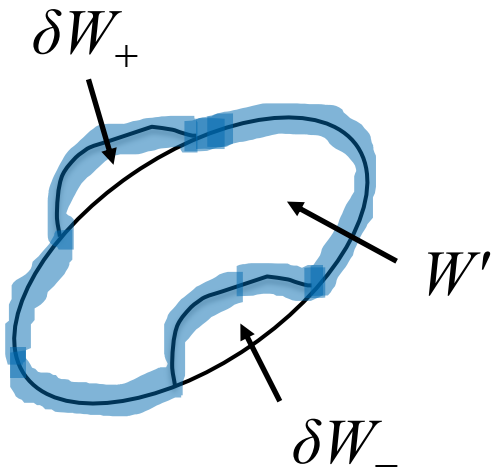
$$W \to W' = W + \delta W_+ - \delta W_-$$

$$P(\mathbf{x} \in \delta W_+|H_0) = P(\mathbf{x} \in \delta W_-|H_0)$$



Proof of Neyman-Pearson Lemma (2)

But we are supposing the LR is higher for all $\boldsymbol{x}$ in δW_- removed than for the $\boldsymbol{x}$ in δW_+ added, and therefore



$$P(\mathbf{x} \in \delta W_+ | H_1) \leq P(\mathbf{x} \in \delta W_+ | H_0) c_\alpha$$

$$P(\mathbf{x} \in \delta W_- | H_1) \geq P(\mathbf{x} \in \delta W_- | H_0) c_\alpha$$

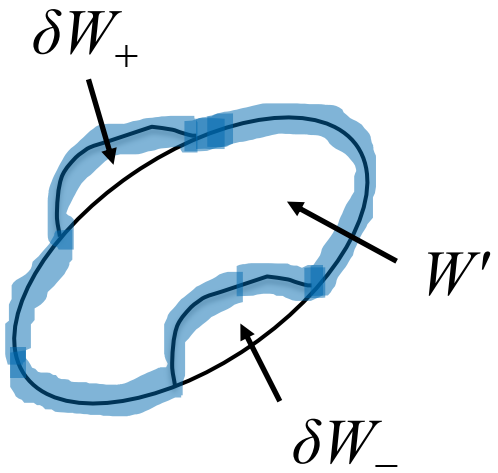
The right-hand sides are equal and therefore

$$P(\mathbf{x} \in \delta W_+ | H_1) \leq P(\mathbf{x} \in \delta W_- | H_1)$$

Proof of Neyman-Pearson Lemma (3)

We have

$$W \cup W' = W \cup \delta W_+ = W' \cup \delta W_-$$



Note W and δW_+ are disjoint, and W' and δW_- are disjoint, so by Kolmogorov's 3rd axiom,

$$P(\mathbf{x} \in W') + P(\mathbf{x} \in \delta W_-) = P(\mathbf{x} \in W) + P(\mathbf{x} \in \delta W_+)$$

Therefore

$$P(\mathbf{x} \in W'|H_1) = P(\mathbf{x} \in W|H_1) + P(\mathbf{x} \in \delta W_+|H_1) - P(\mathbf{x} \in \delta W_-|H_1)$$

≤ 0

Proof of Neyman-Pearson Lemma (4)

And therefore

$$P(\mathbf{x} \in W' | H_1) \leq P(\mathbf{x} \in W | H_1)$$

i.e. the deformed critical region W' cannot have higher power than the original one that satisfied the LR criterion of the Neyman-Pearson lemma.