Computing and Statistical Data Analysis
Stat 5: Multivariate Methods

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

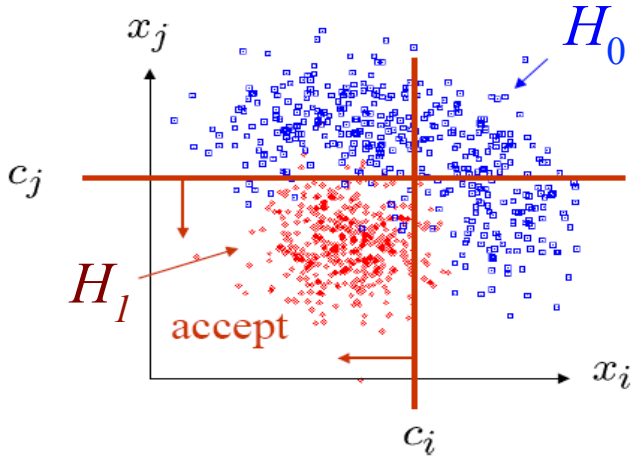
Course web page:

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

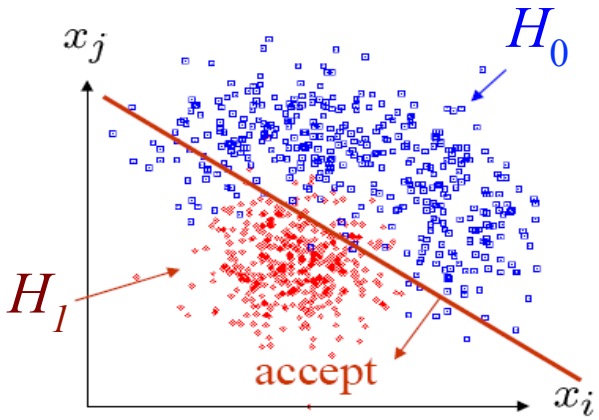
Finding an optimal decision boundary

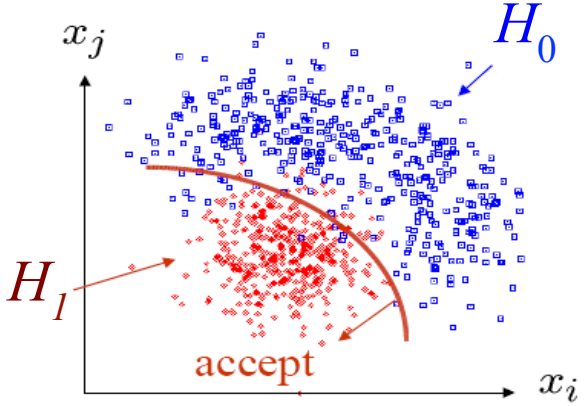
In particle physics usually start by making simple "cuts":

$$x_i < c_i$$
$$x_j < c_j$$



Maybe later try some other type of decision boundary:





Multivariate methods

Many new (and some old) methods:

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

New software for HEP, e.g.,

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

`StatPatternRecognition`, I. Narsky, physics/0507143

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

Materials from some recent meetings:

PHYSTAT conference series (2002, 2003, 2005, 2007,...) see
www.phystat.org

Caltech workshop on multivariate analysis, 11 February, 2008
indico.cern.ch/conferenceDisplay.py?confId=27385

SLAC Lectures on Machine Learning by Ilya Narsky (2006)
www-group.slac.stanford.edu/sluo/Lectures/Stat2006_Lectures.html

Software

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

From **tmva.sourceforge.net**, also distributed with ROOT
Variety of classifiers
Good manual

StatPatternRecognition, I. Narsky, physics/0507143

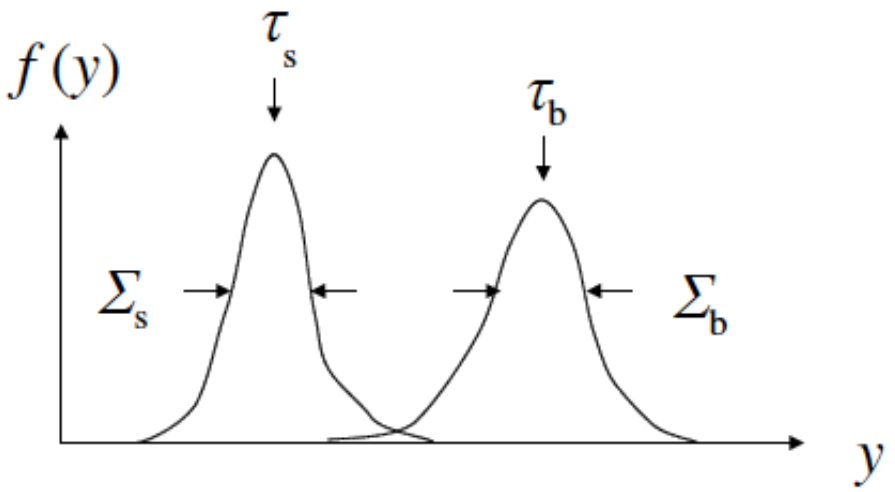
Further info from **www.hep.caltech.edu/~narsky/spr.html**
Also wide variety of methods, many complementary to **TMVA**
Currently appears project no longer to be supported

Linear test statistic

Ansatz: $y(\vec{x}) = \sum_{i=1}^{n} w_i x_i = \vec{w}^T \vec{x}$

Choose the parameters w_1, ..., w_n so that the pdfs $f(y|\mathrm{s}), f(y|\mathrm{b})$ have maximum 'separation'. We want:

large distance between mean values, small widths



→ Fisher: maximize $J(\vec{w}) = \dfrac{(\tau_s - \tau_b)^2}{\Sigma_s^2 + \Sigma_b^2}$

Coefficients for maximum separation

We have $(\mu_k)_i = \int x_i \, p(\vec{x}|H_k)\, d\vec{x}$ ← mean, covariance of $\boldsymbol{x}$

$$(V_k)_{ij} = \int (x-\mu_k)_i (x-\mu_k)_j \, p(\vec{x}|H_k)\, d\vec{x}$$

where $k = 0, 1$ (hypothesis)

and $i, j = 1, \ldots, n$ (component of $\boldsymbol{x}$)

For the mean and variance of $y(\vec{x})$ we find

$$\tau_k = \int y(\vec{x})\, p(\vec{x}|H_k)\, d\vec{x} = \vec{w}^T \vec{\mu}_k$$

$$\Sigma_k^2 = \int (y(\vec{x}) - \tau_k)^2 \, p(\vec{x}|H_k)\, d\vec{x} = \vec{w}^T V_k \vec{w}$$

Determining the coefficients $\boldsymbol{w}$

The numerator of $J(\boldsymbol{w})$ is

$$(\tau_0-\tau_1)^2=\sum_{i,\,j=1}^{n} w_i w_j (\mu_0-\mu_1)_i (\mu_0-\mu_1)_j$$

$$=\sum_{i,\,j=1}^{n} w_i w_j B_{ij}=\vec{w}^T B\,\vec{w}$$

'between' classes

and the denominator is

$$\Sigma_0^2+\Sigma_1^2=\sum_{i,\,j=1}^{n} w_i w_j (V_0+V_1)_{ij}=\vec{w}^T W\,\vec{w}$$

'within' classes

→ maximize $$J(\vec{w})=\frac{\vec{w}^T B\,\vec{w}}{\vec{w}^T W\,\vec{w}}=\frac{\text{separation between classes}}{\text{separation within classes}}$$

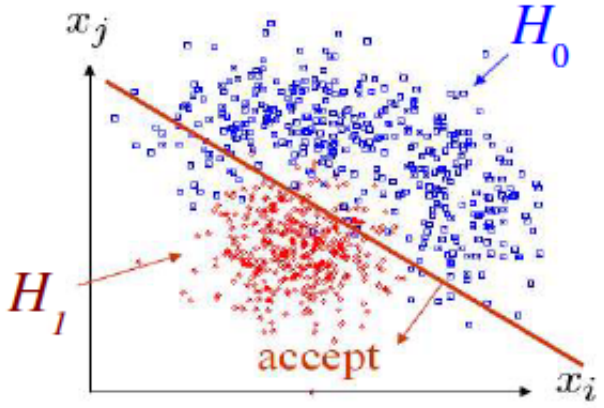
Fisher discriminant function

Setting $\dfrac{\partial J}{\partial w_i} = 0$ gives Fisher's linear discriminant function:

$$y(\vec{x}) = \vec{w}^T \vec{x} \qquad \text{with } \vec{w} \propto W^{-1}(\vec{\mu}_0 - \vec{\mu}_1)$$

Gives linear decision boundary.

Projection of points in direction of decision boundary gives maximum separation.



Fisher discriminant for Gaussian data

Suppose $f(\boldsymbol{x}|H_k)$ is a multivariate Gaussian with mean values

$$E_0[\vec{x}] = \vec{\mu}_0 \text{ for } H_0 \qquad E_1[\vec{x}] = \vec{\mu}_1 \text{ for } H_1$$

and covariance matrices $V_0 = V_1 = V$ for both. We can write the Fisher's discriminant function (with an offset) is

$$y(\vec{x}) = w_0 + (\vec{\mu}_0 - \vec{\mu}_1) V^{-1} \vec{x}$$

The likelihood ratio is thus

$$\frac{p(\vec{x}|H_0)}{p(\vec{x}|H_1)} = \exp\left[-\frac{1}{2}(\vec{x}-\vec{\mu}_0)^T V^{-1} + \frac{1}{2}(\vec{x}-\vec{\mu})_1^T V^{-1}(\vec{x}-\vec{\mu}_1)\right]$$

$$= e^y$$

Fisher for Gaussian data (2)

That is, $y(\boldsymbol{x})$ is a monotonic function of the likelihood ratio, so for this case the Fisher discriminant is equivalent to using the likelihood ratio, and is therefore optimal.

For non-Gaussian data this no longer holds, but linear discriminant function may be simplest practical solution.

Often try to transform data so as to better approximate Gaussian before constructing Fisher discrimimant.

Fisher and Gaussian data (3)

Multivariate Gaussian data with equal covariance matrices also gives a simple expression for posterior probabilities, e.g.,

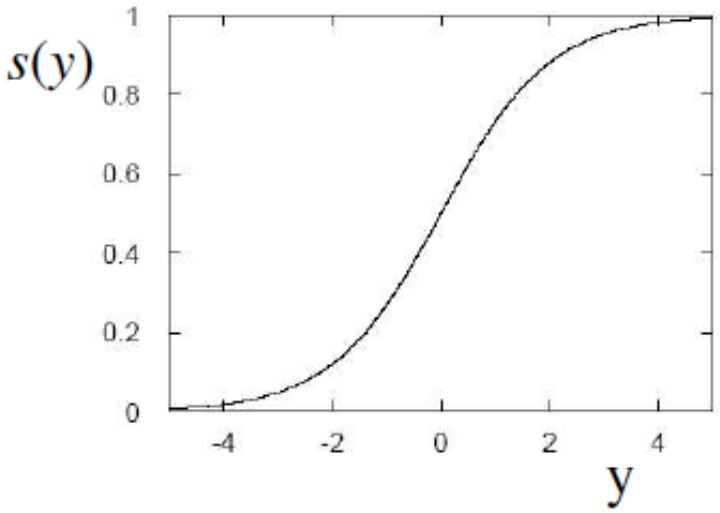
$$P(H_0|\vec{x}) = \frac{p(\vec{x}|H_0)\,P(H_0)}{p(\vec{x}|H_0)\,P(H_0) + p(\vec{x}|H_1)\,P(H_1)}$$

For Gaussian $\boldsymbol{x}$ and a particular choice of the offset w_0 this becomes:

$$P(H_0|\vec{x}) = \frac{1}{1+e^{-y(\vec{x})}} \equiv s(y(\vec{x}))$$

which is the logistic sigmoid function:

(We will use this later in connection with Neural Networks.)



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).

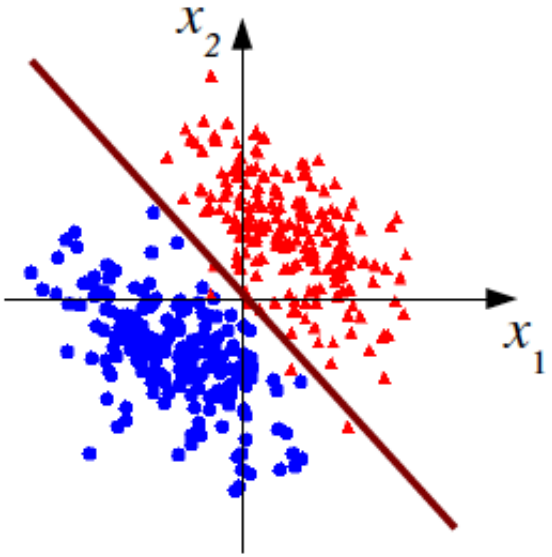
In other cases we will see that the basis functions only enter as dot products

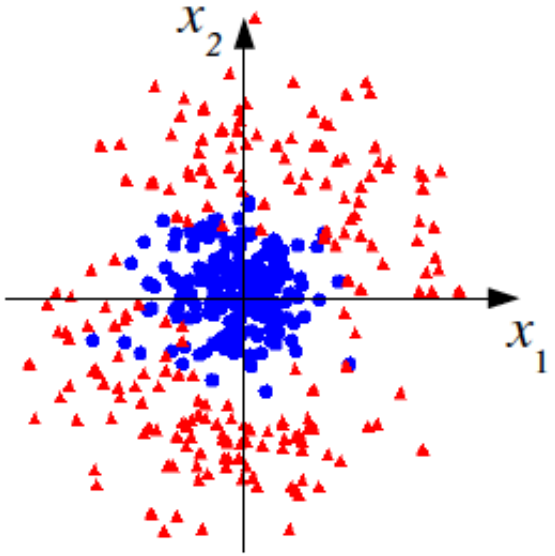
$$\vec{\varphi}(\vec{x}_i)\cdot\vec{\varphi}(\vec{x}_j) = K(\vec{x}_i, \vec{x}_j)$$

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.

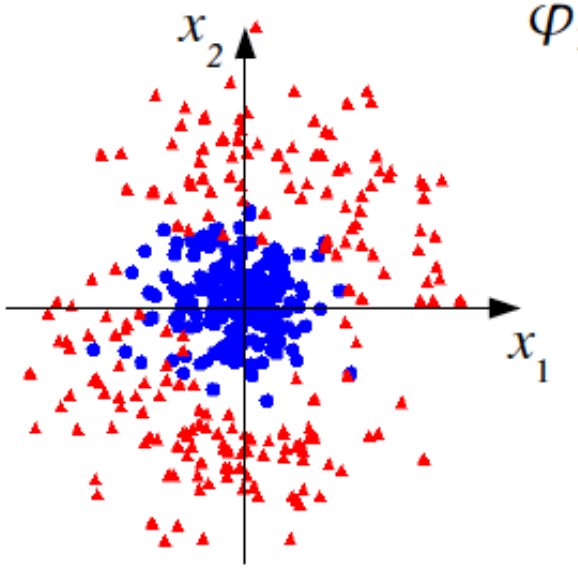
Nonlinear transformation of inputs

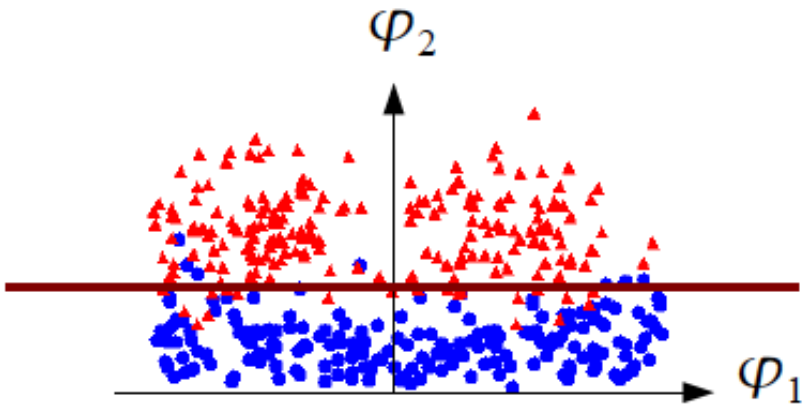
We can try to find a transformation, $x_1, \ldots, x_n \to \varphi_1(\vec{x}), \ldots, \varphi_m(\vec{x})$ so that the transformed "feature space" variables can be separated better by a linear boundary:

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

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

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.

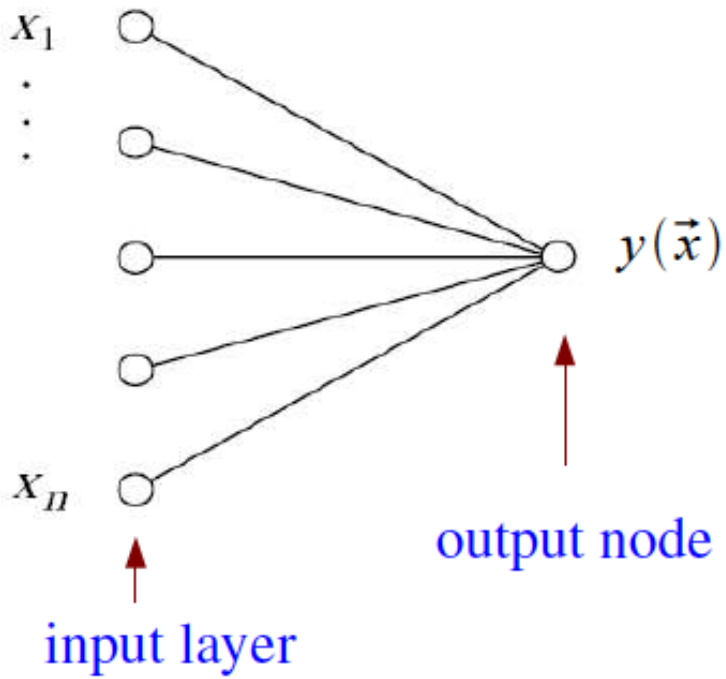
The training data are then used to adjust the parameters so that the resulting discriminant function has the best performance.

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.



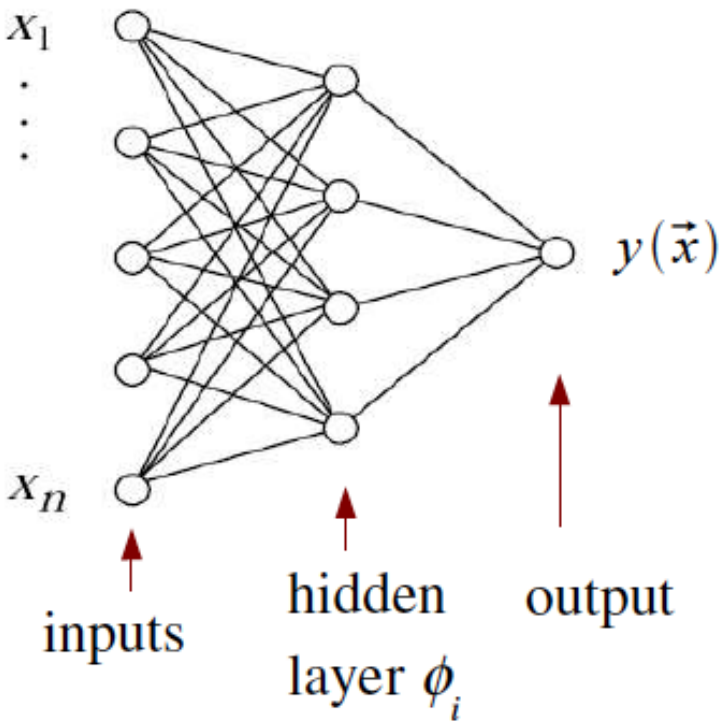
The multilayer perceptron

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: one hidden layer

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

In practice often choose a single hidden layer and try increasing the the number of nodes until no further improvement in performance is found.

More than one hidden layer

"Relatively little is known concerning the advantages and disadvantages of using a single hidden layer with many units (neurons) over many hidden layers with fewer units. The mathematics and approximation theory of the MLP model with more than one hidden layer is not well understood."

"Nonetheless there seems to be reason to conjecture that the two hidden layer model may be significantly more promising than the single hidden layer model, ..."

A. Pinkus, *Approximation theory of the MLP model in neural networks*, Acta Numerica (1999), pp. 143—195.

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

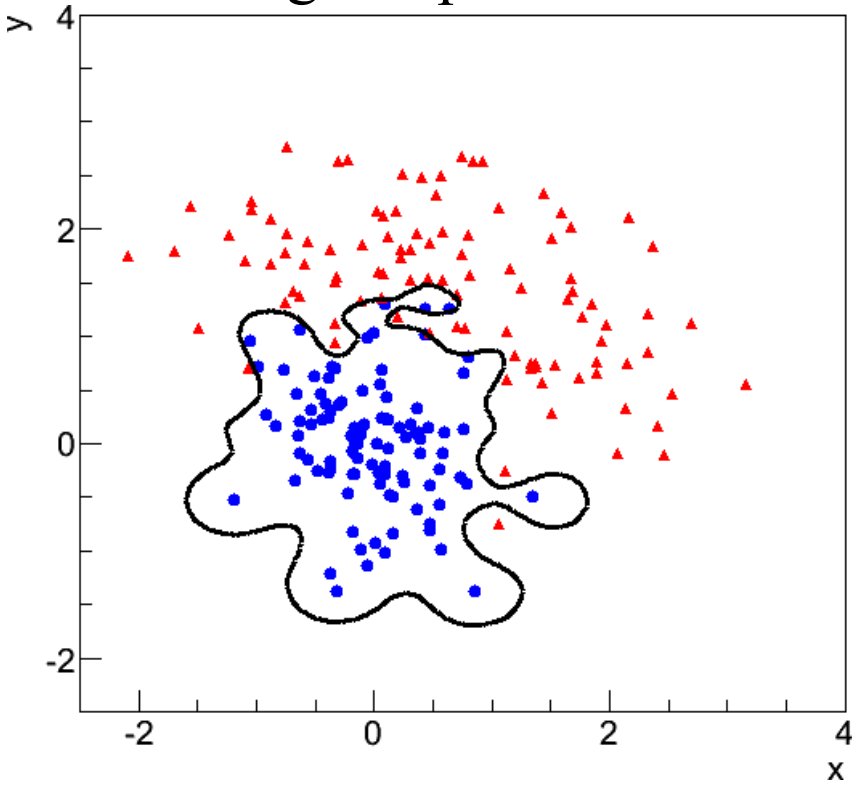
Chain rule gives all the needed derivatives.

Overtraining

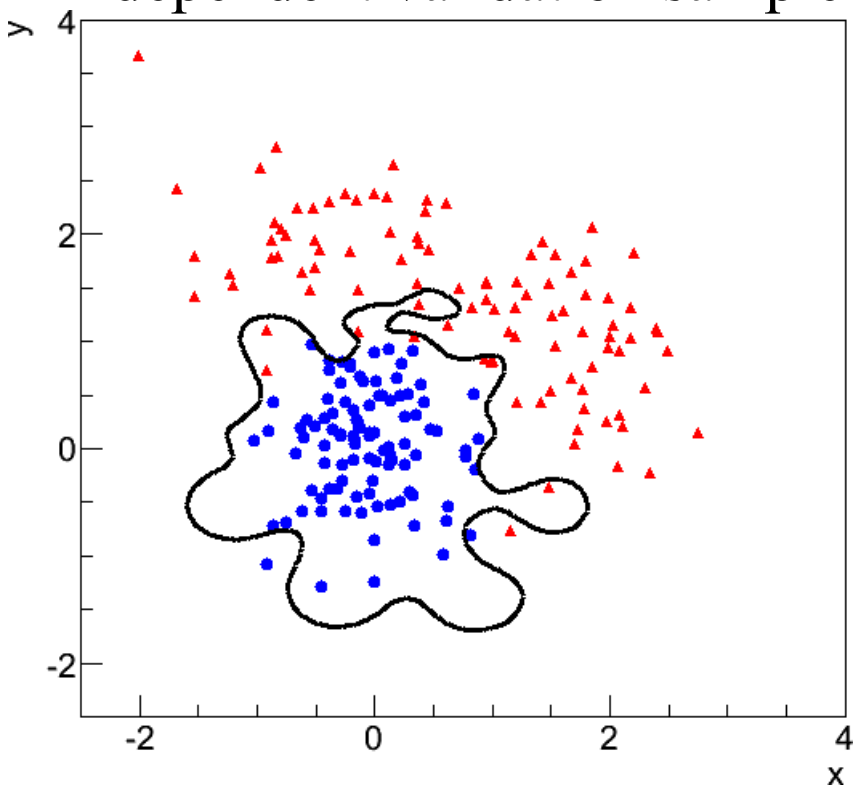
If decision boundary is too flexible it will conform too closely to the training points → overtraining.

Monitor by applying classifier to independent validation sample.

training sample

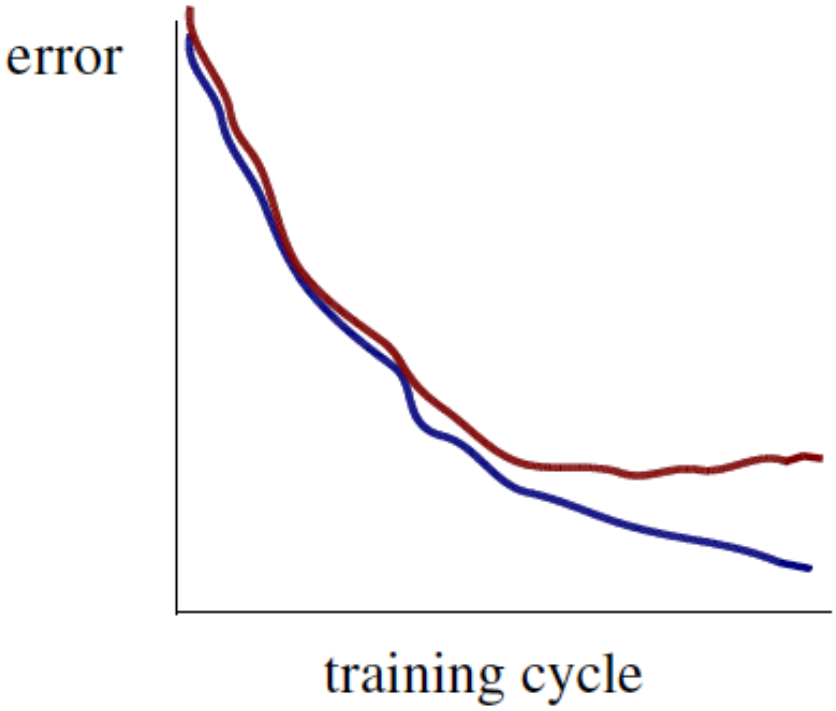


independent validation sample



Monitoring overtraining

If we monitor the value of the error function $E(\boldsymbol{w})$ at every cycle of the minimization, for the training sample it will continue to decrease.



But the validation sample it may initially decrease, and then at some point increase, indicating overtraining.

validation sample

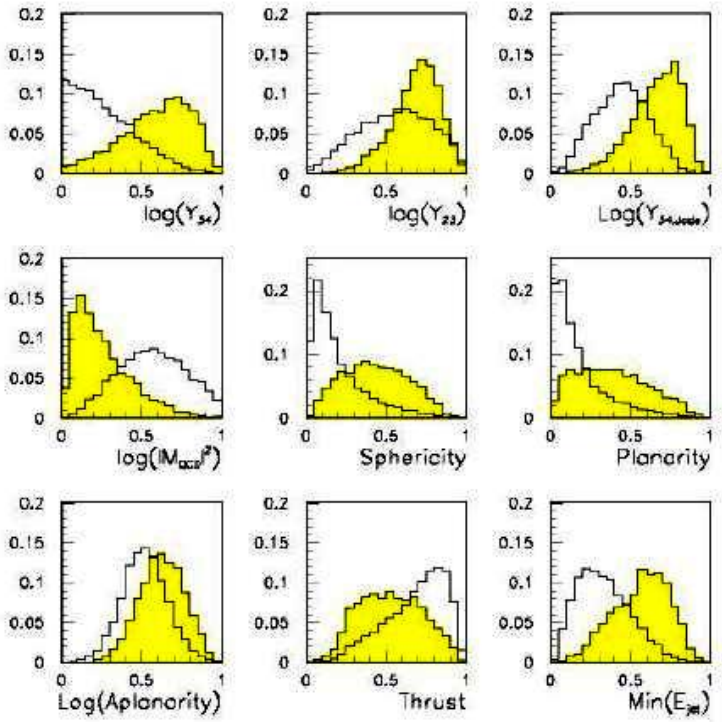
training sample

Choose classifier that minimizes error function for validation sample.

Neural network example from LEP II

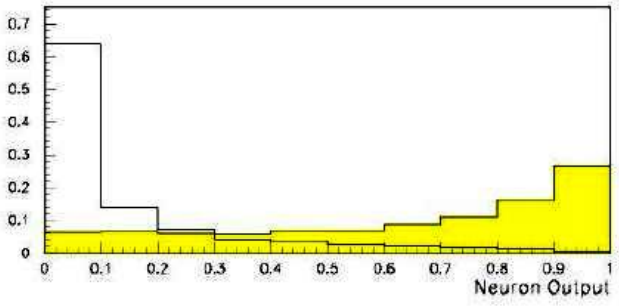
Signal: $e^+e^- \to W^+W^-$ (often 4 well separated hadron jets)

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:

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

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:

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

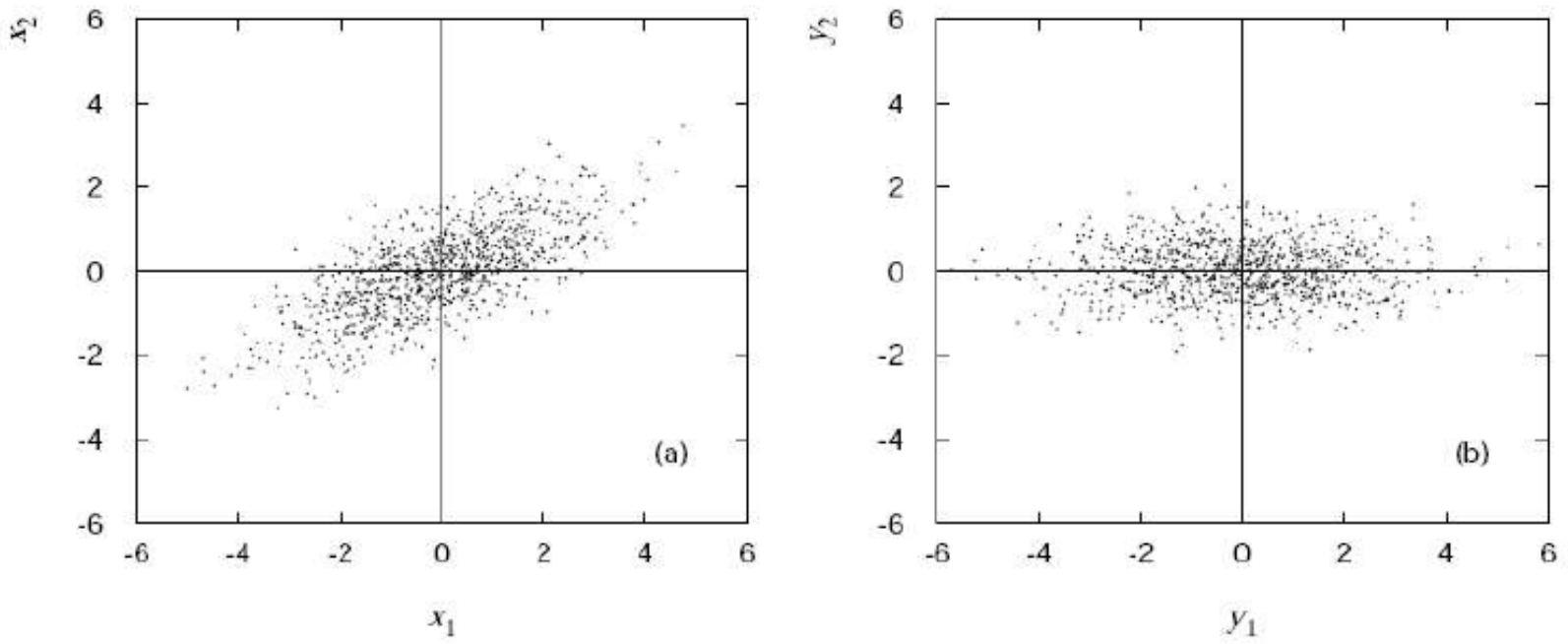
holds only if the components of $\boldsymbol{x}$ are independent

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

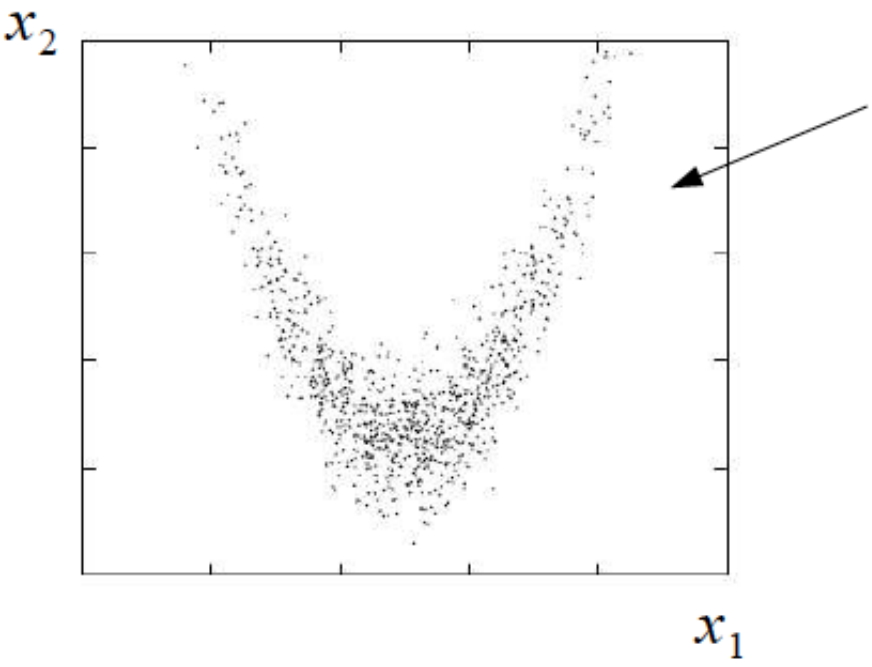
But we can define a set of uncorrelated input variables by a linear transformation, i.e., find the matrix A such that for $\vec{y} = A\vec{x}$ the covariances $\mathrm{cov}[y_i, y_j] = 0$:



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

But if the nonlinearities are not too great, it is reasonable to first decorrelate the inputs and take as our estimator for each pdf

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

So this at least reduces the problem to one of finding estimates of one-dimensional pdfs.

The resulting estimated likelihood ratio gives the Naive Bayes classifier (in HEP sometimes called the "likelihood method").

Kernel-based PDE (KDE, Parzen window)

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

$$\hat{f}(\vec{x}) = \frac{1}{Nh^d} \sum_{i=1}^{N} K\left(\frac{\vec{x} - \vec{x}_i}{h}\right)$$

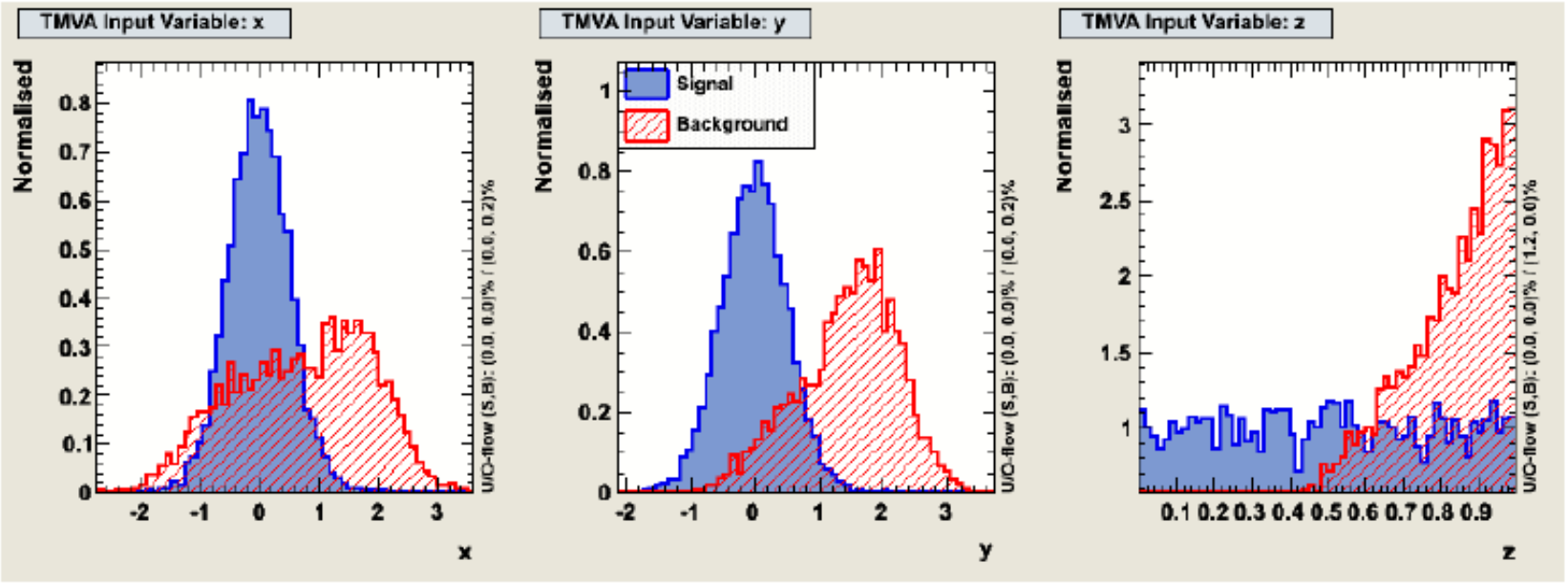
kernel

bandwidth (smoothing parameter)

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

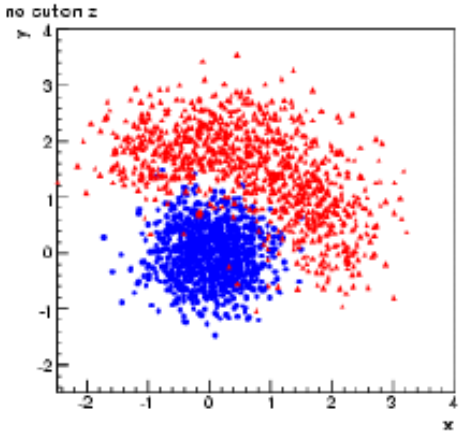
Need to sum N terms to evaluate function (slow); faster algorithms only count events in vicinity of $\boldsymbol{x}$ (k-nearest neighbor, range search).

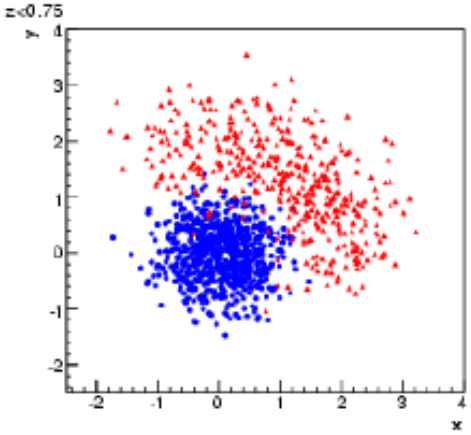
Test example with TMVA



Test example, x vs. y with cuts on z

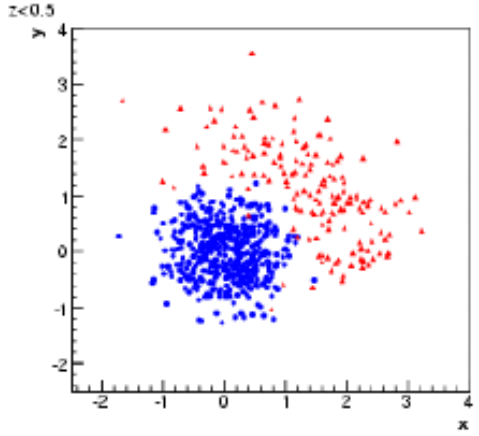
no cut on z

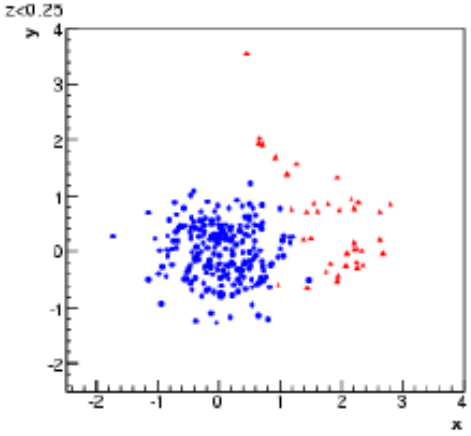




$z < 0.75$

$z < 0.5$





$z < 0.25$

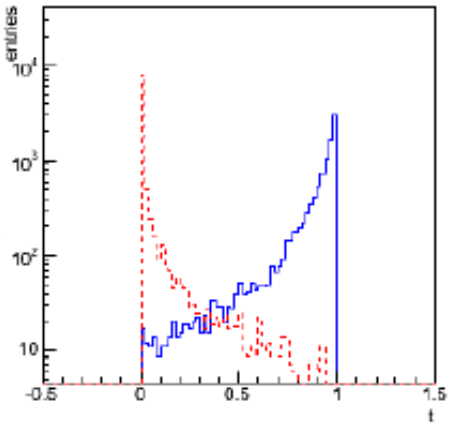
Test example results

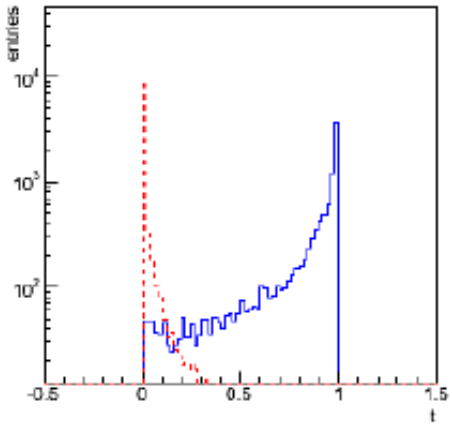
Fisher
discriminant

Find these on next homework assignment.

Multilayer
perceptron

Naive Bayes,
no decorrelation





Naive Bayes with
decorrelation