Advanced statistical methods for data analysis – Lecture 2

Glen Cowan

RHUL Physics

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

Universität Mainz

Klausurtagung des GK

"Eichtheorien – exp. Tests..."

Bullay/Mosel

15–17 September, 2008

Outline

Multivariate methods in particle physics

Some general considerations

Brief review of statistical formalism

Multivariate classifiers:

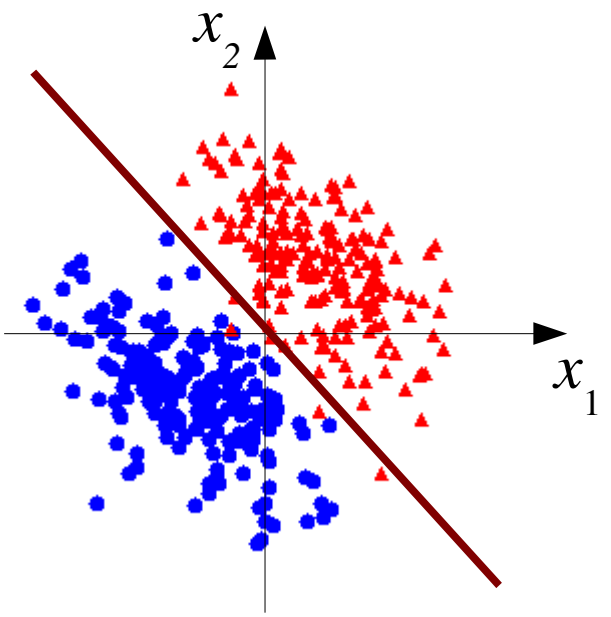
- Linear discriminant function

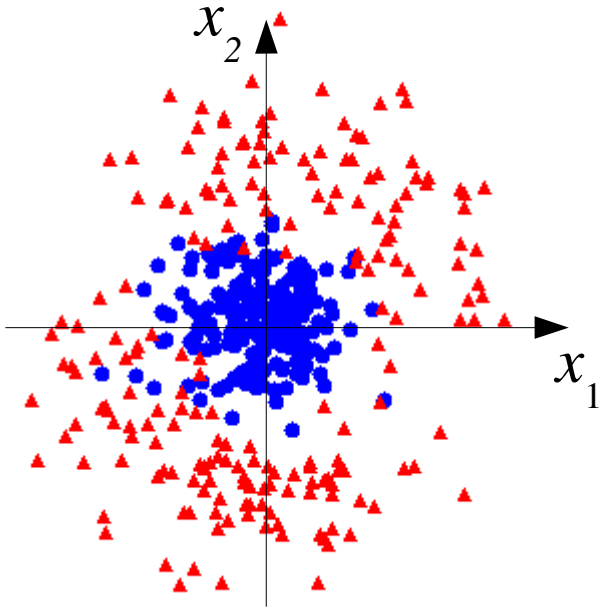
Lecture 2 start

- Neural networks
- Naive Bayes classifier
- *k*-Nearest-Neighbour method
- Decision trees
- Support Vector Machines

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

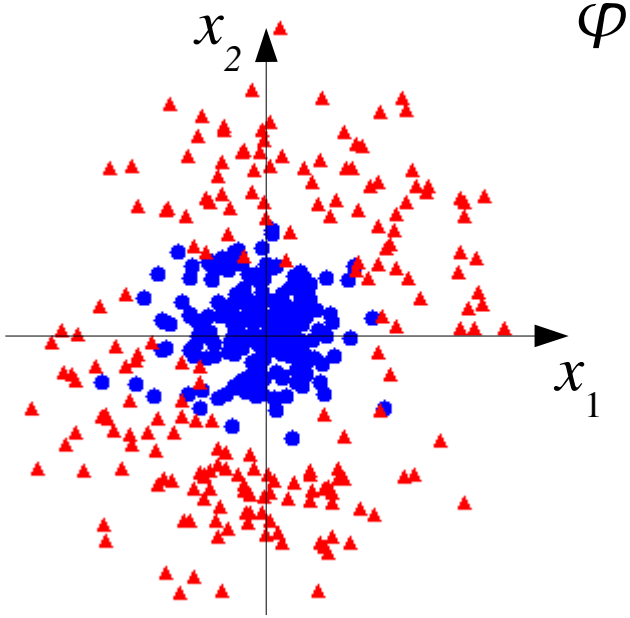
We can try to find a transformation, $x_1, \ldots, x_n \to \varphi_1(\vec{x}), \ldots, \varphi_m(\vec{x})$

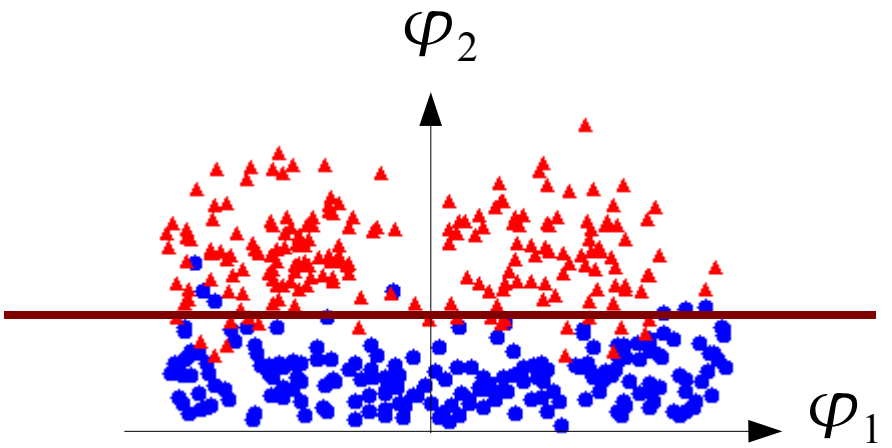
so that the transformed "feature space" variables can be separated better by a linear boundary:

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

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

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.

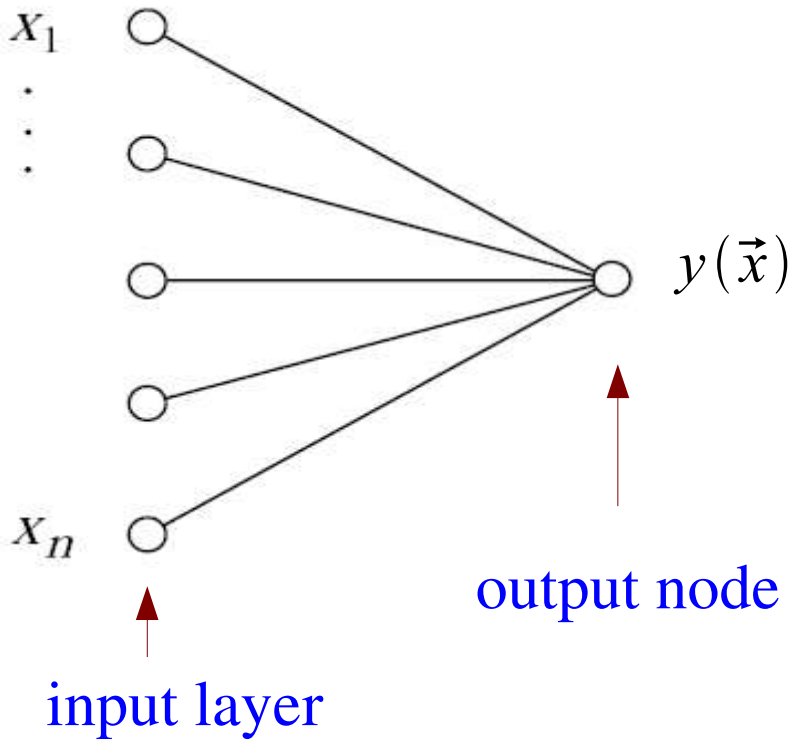
The training data are then used to adjust the parameters so that the resulting discriminant function has the best performance.

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.



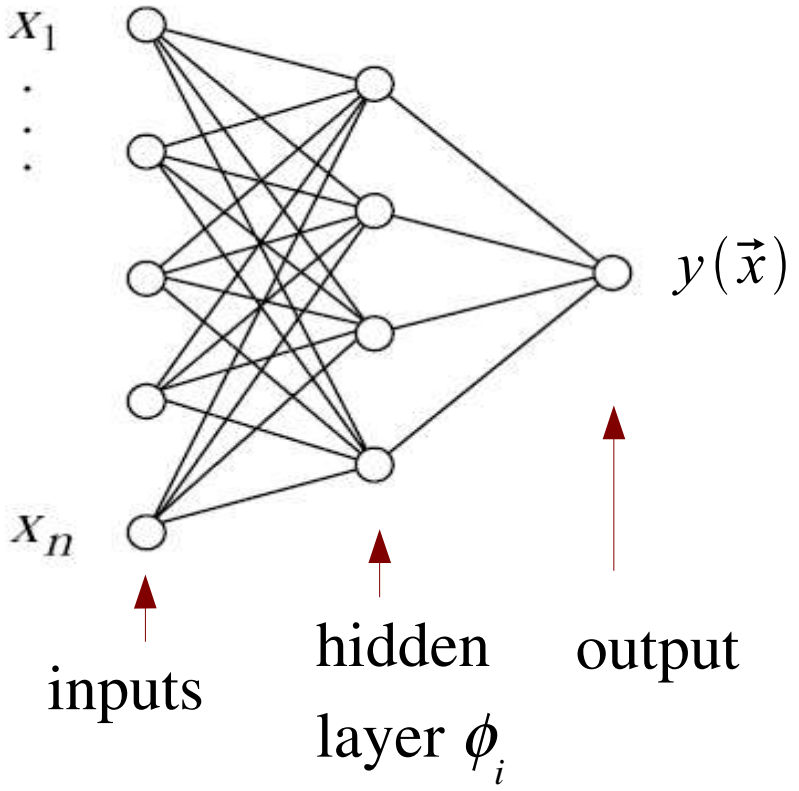
The multilayer perceptron

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 Bayes 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 ***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.

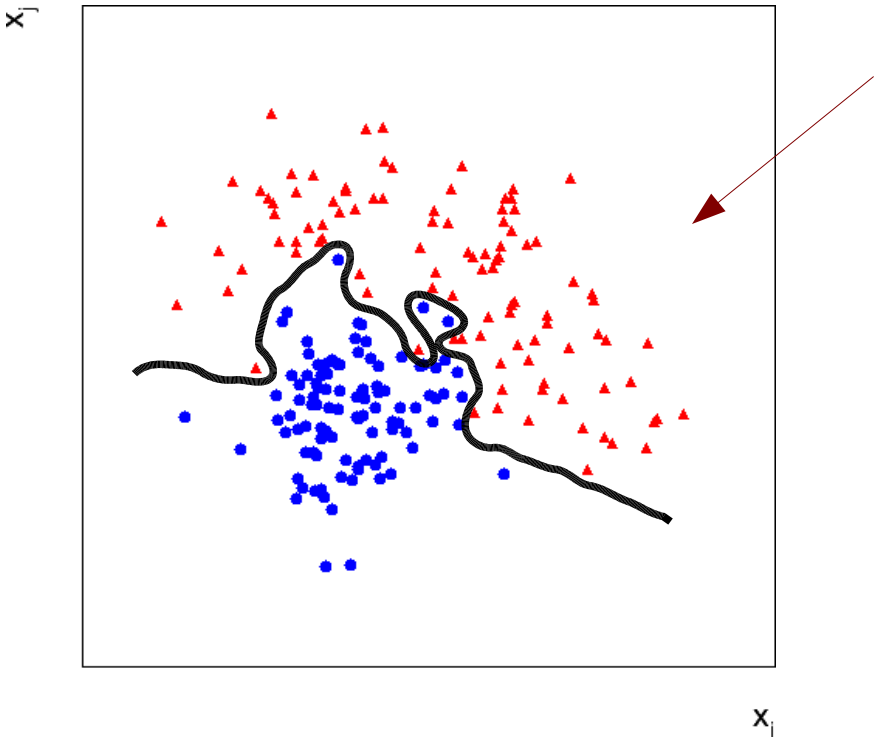
So e.g. for event a we have $\dfrac{\partial E_a}{\partial w_{1j}^{(2)}} = (y_a - t_a)\, h'(u(\vec{x}))\, \varphi_j(\vec{x})$

derivative of activation function

Chain rule gives all the needed derivatives.

Overtraining

If the network has too many nodes, after training it will tend to conform too closely to the training data:



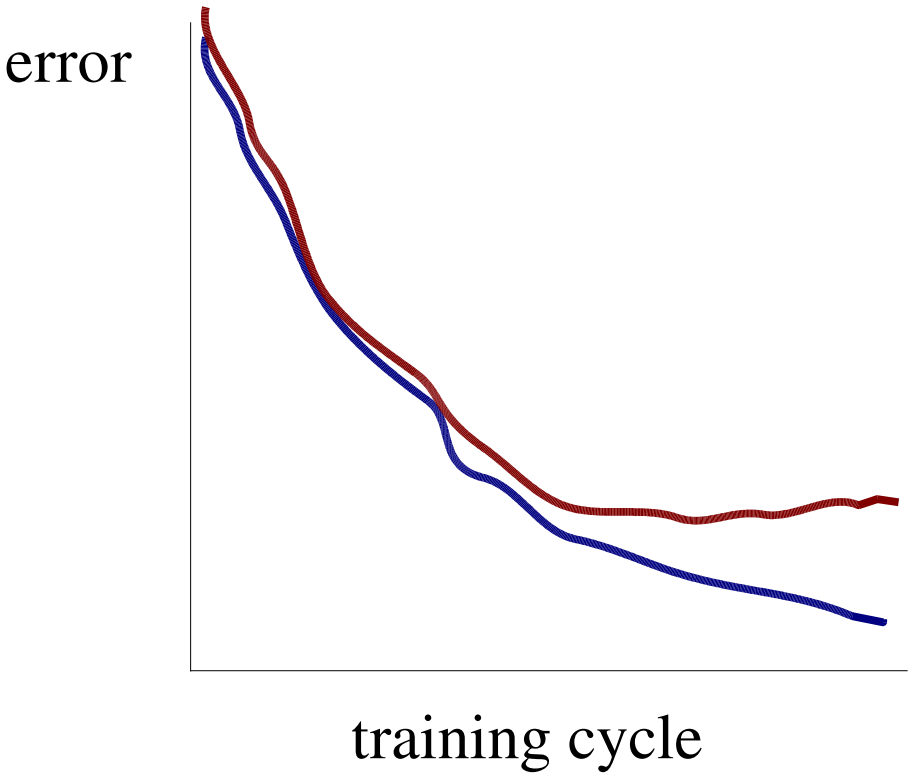
Overtraining

The classification error rate on the training sample may be very low, but it would be much higher on an independent data sample.

Therefore it is important to evaluate the error rate with a statistically 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

Validation and testing

The validation sample can be used to make various choices about the network architecture, e.g., adjust the number of hidden nodes so as to obtain good "generalization performance" (ability to correctly classify unseen data).

If the validation stage is iterated may times, the estimated error rate based on the validation sample has a bias, so strictly speaking one should finally estimate the error rate with an independent test sample.

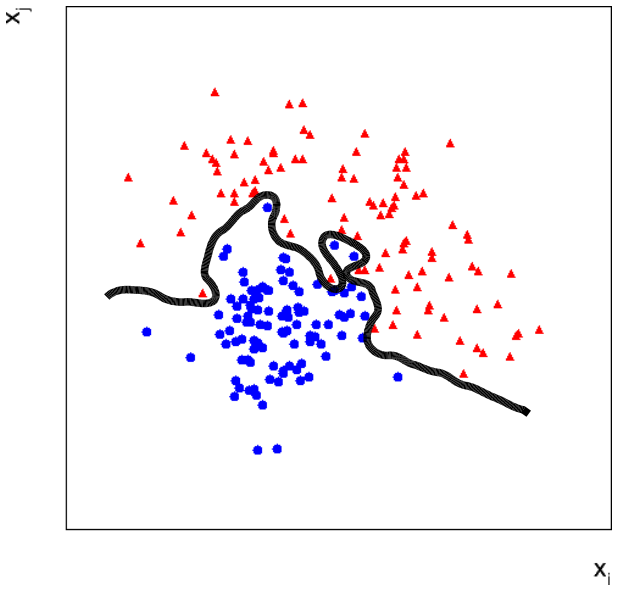
	train : validate : test
Rule of thumb if data not too expensive (Narsky):	50 : 25 : 25

But this depends on the type of classifier. Often the bias in the error rate from the validation sample is small and one can omit the test step.

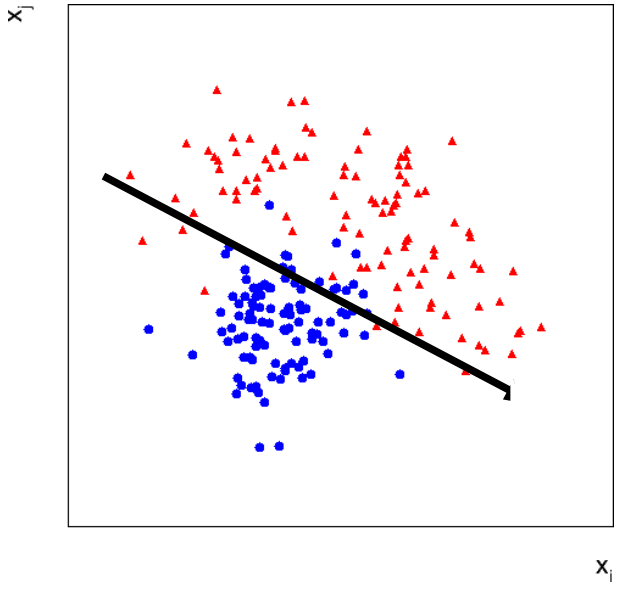
Bias – variance trade-off

For a finite amount of training data, an increasing number of network parameters (layers, nodes) means that the estimates of these parameters have increasingly large statistical errors (variance, overtraining).

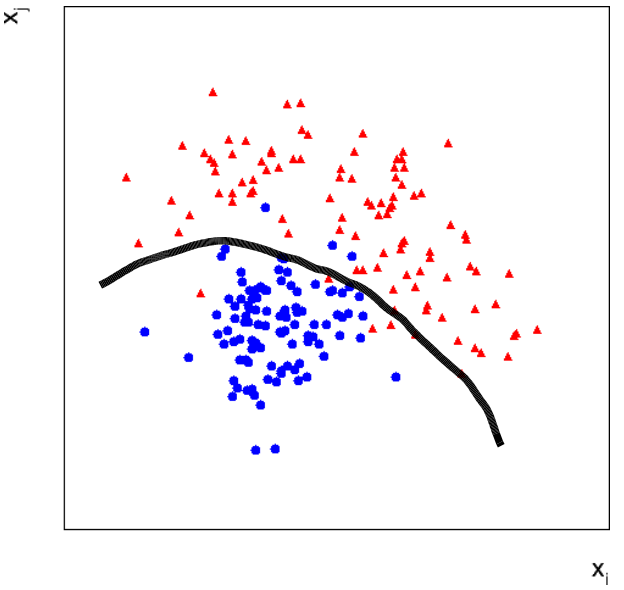
Having too few parameters doesn't allow the network to exploit the existing nonlinearities, i.e., it has a bias.



high variance



high bias



good trade-off

Regularized neural networks

Often one uses the test sample to optimize the number of hidden nodes.

Alternatively one may use a relatively large number of hidden nodes but include in the error function a regularization term that penalizes overfitting, e.g.,

regularization parameter

$$\tilde{E}(\boldsymbol{w}) = E(\boldsymbol{w}) + \frac{\lambda}{2}\boldsymbol{w}^T\boldsymbol{w}$$

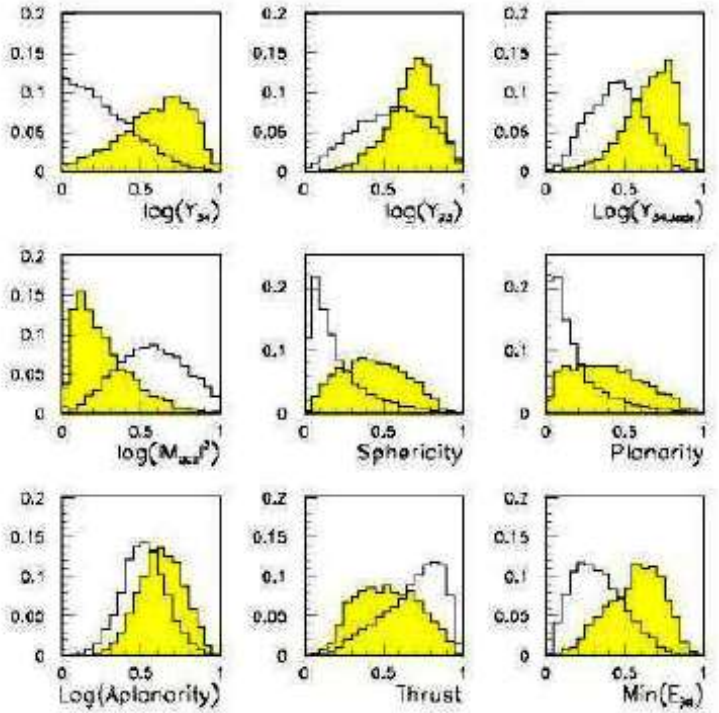
Increasing λ gives a smoother boundary (higher bias, lower variance)

Known as "weight decay", since the weights are driven to zero unless supported by the data (an example of "parameter shrinkage").

Neural network example from LEP II

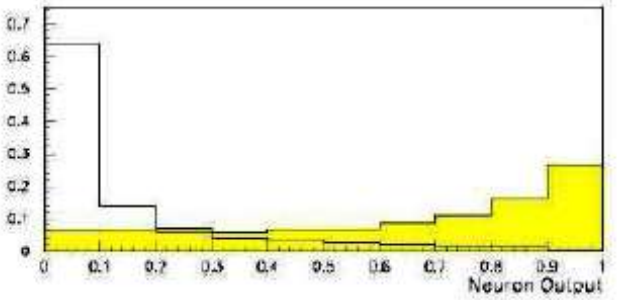
Signal: $e^+e^- \to W^+W^-$ (often 4 well separated hadron jets)

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 does better...



(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:

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

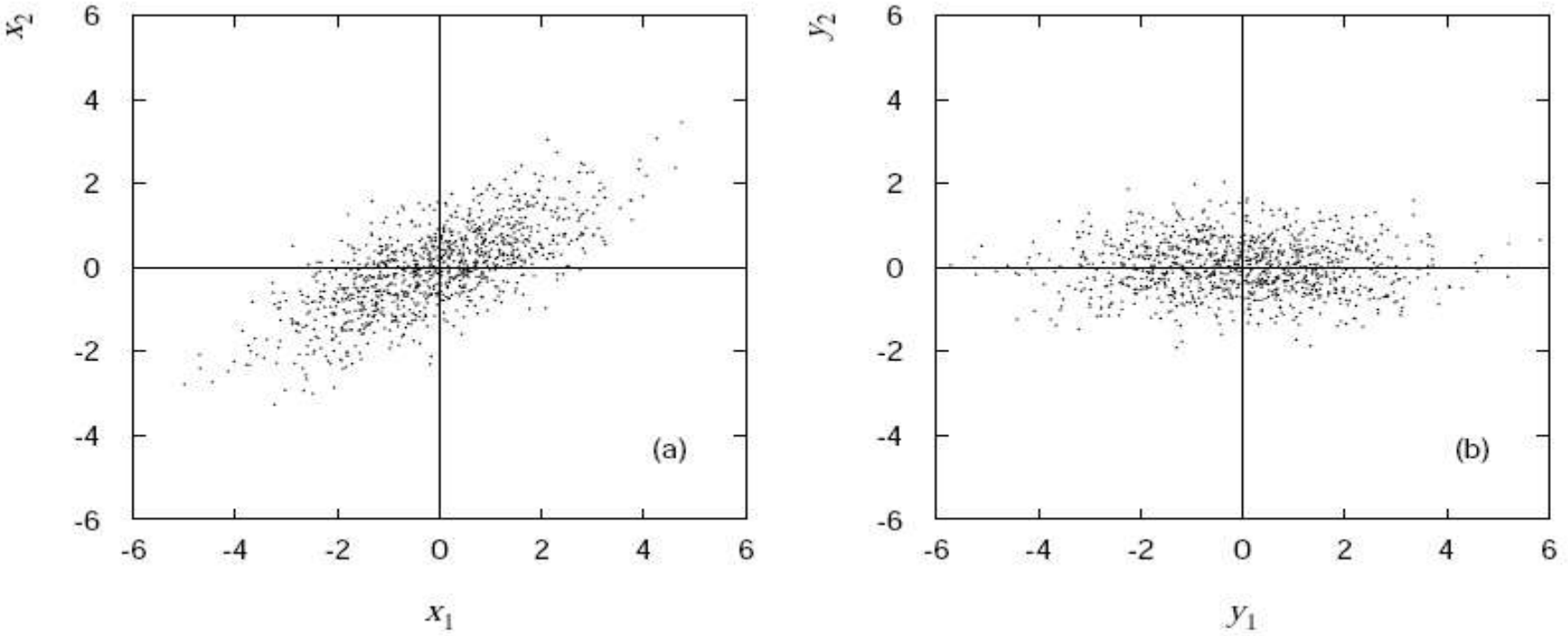
holds only if the components of $\boldsymbol{x}$ are independent

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

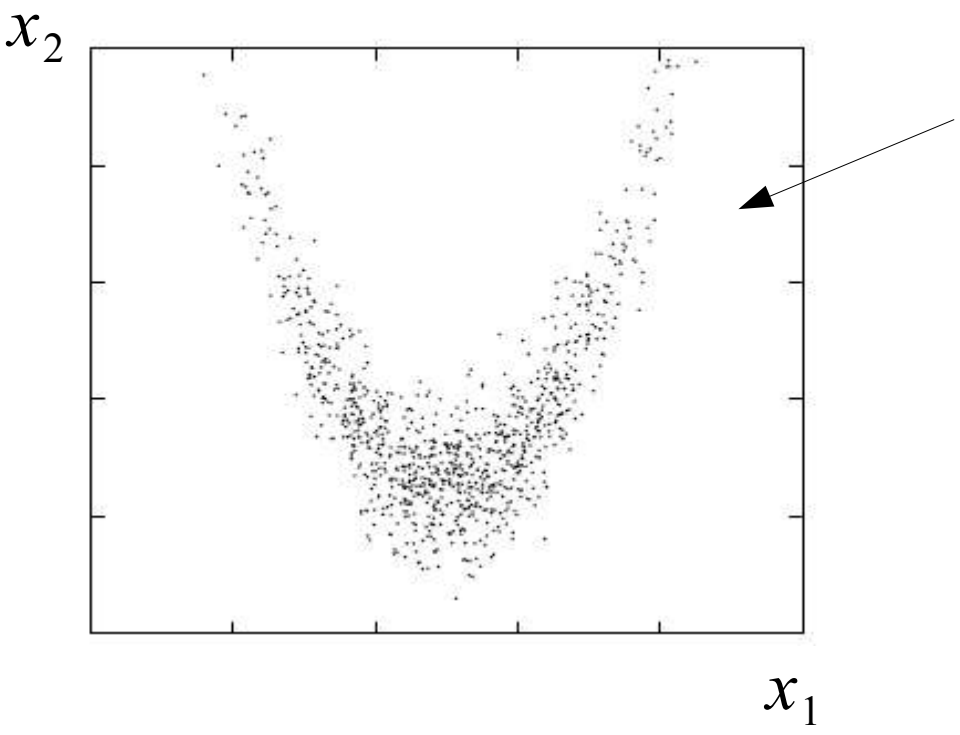
But we can define a set of uncorrelated input variables by a linear transformation, i.e., find the matrix A such that for $\vec{y} = A\vec{x}$ the covariances cov[y_i, y_j] = 0:



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

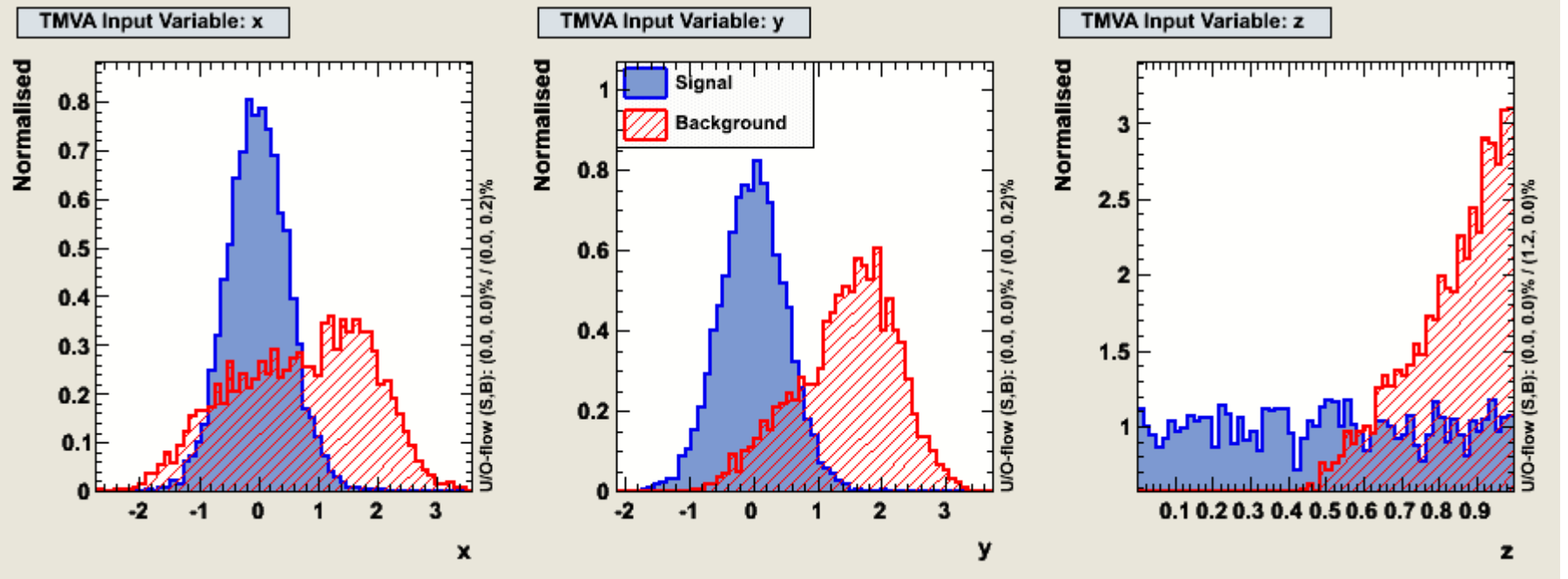
But if the nonlinearities are not too great, it is reasonable to first decorrelate the inputs and take as our estimator for each pdf

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

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

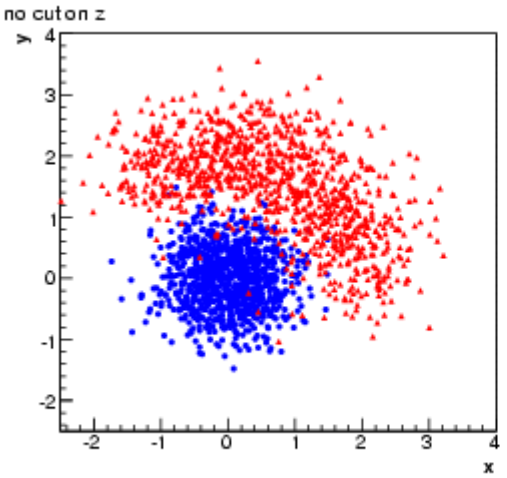
The resulting estimated likelihood ratio gives the Naive Bayes classifier (in HEP sometimes called the "likelihood method").

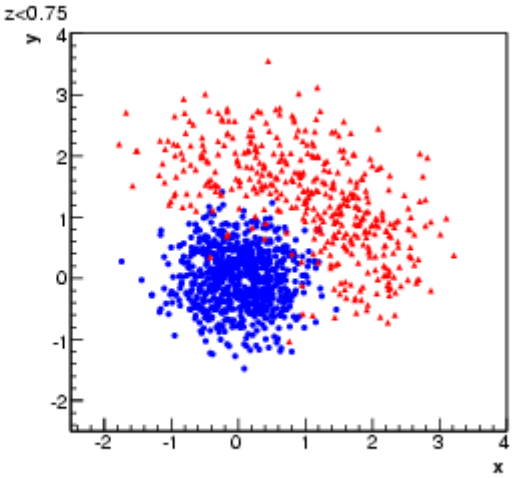
Test example with TMVA



Test example, x vs. y with cuts on z

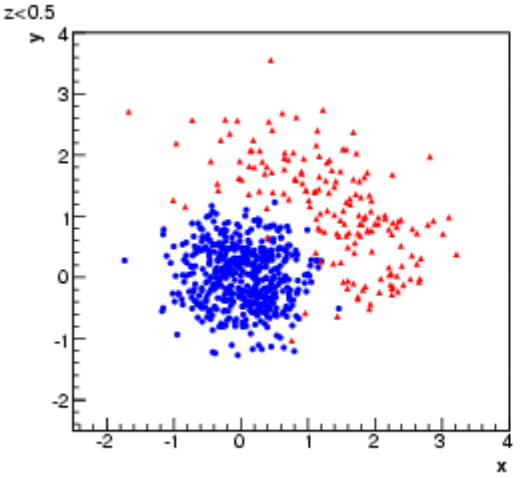
no cut on z

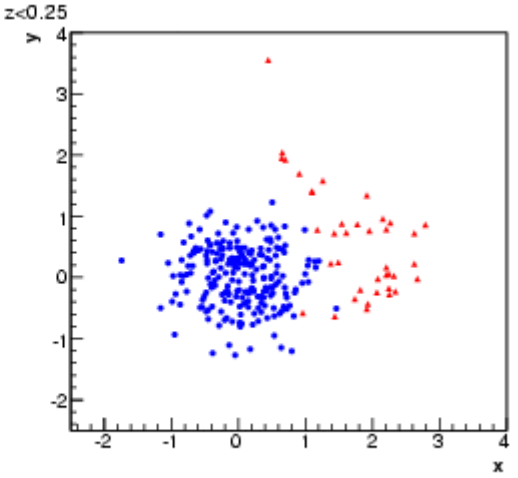




$z < 0.75$

$z < 0.5$

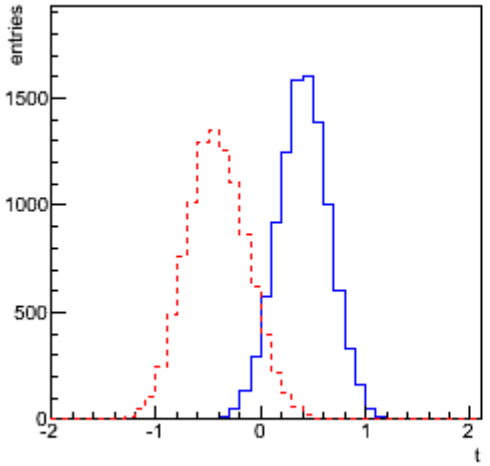


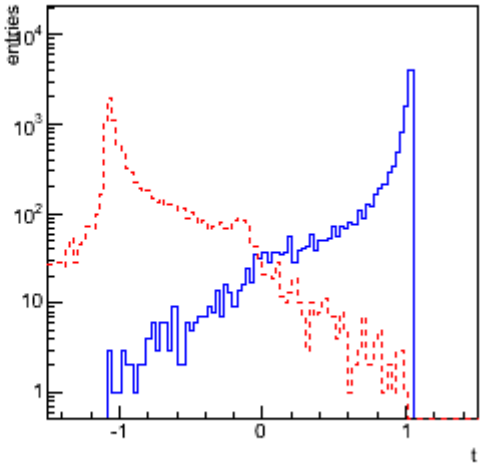


$z < 0.25$

Test example results

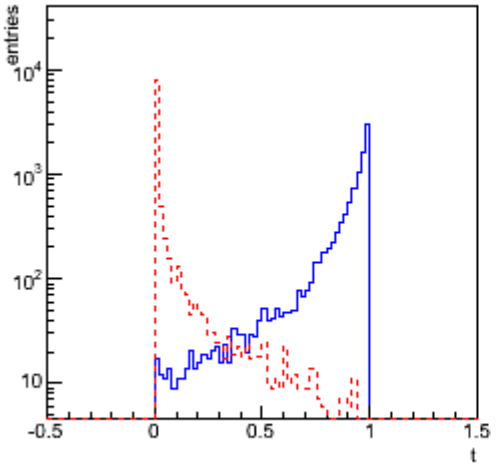
Fisher discriminant

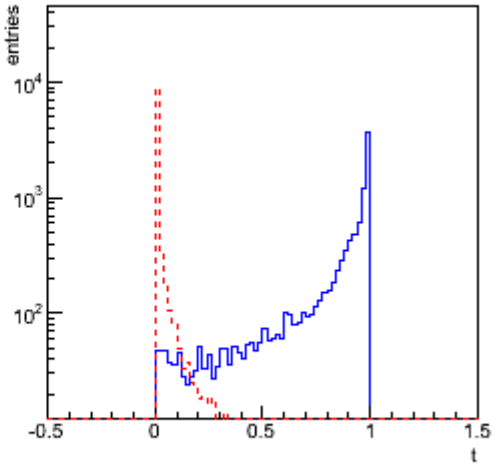




Multilayer perceptron

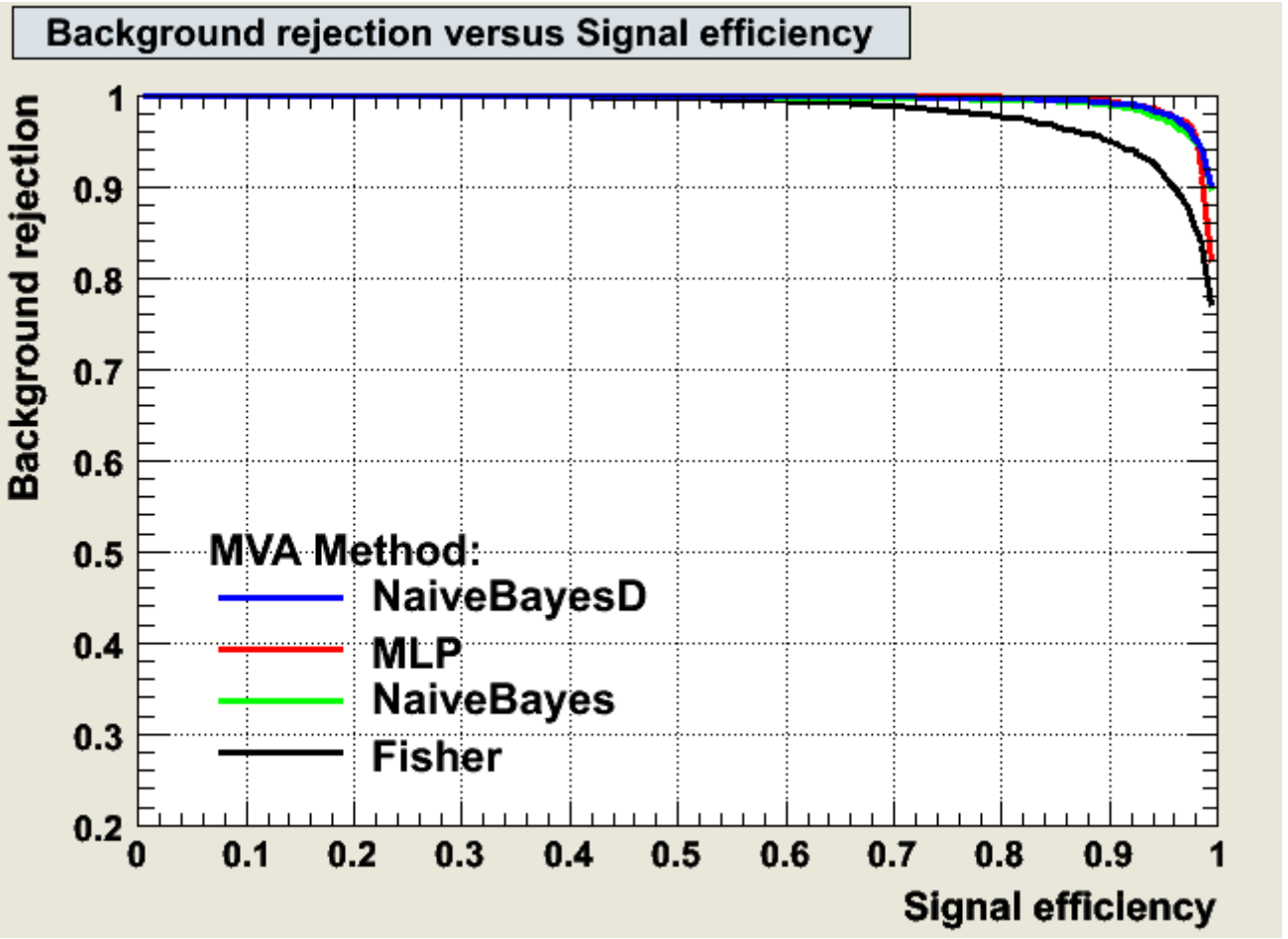
Naive Bayes, no decorrelation





Naive Bayes with decorrelation

Test example ROC curves

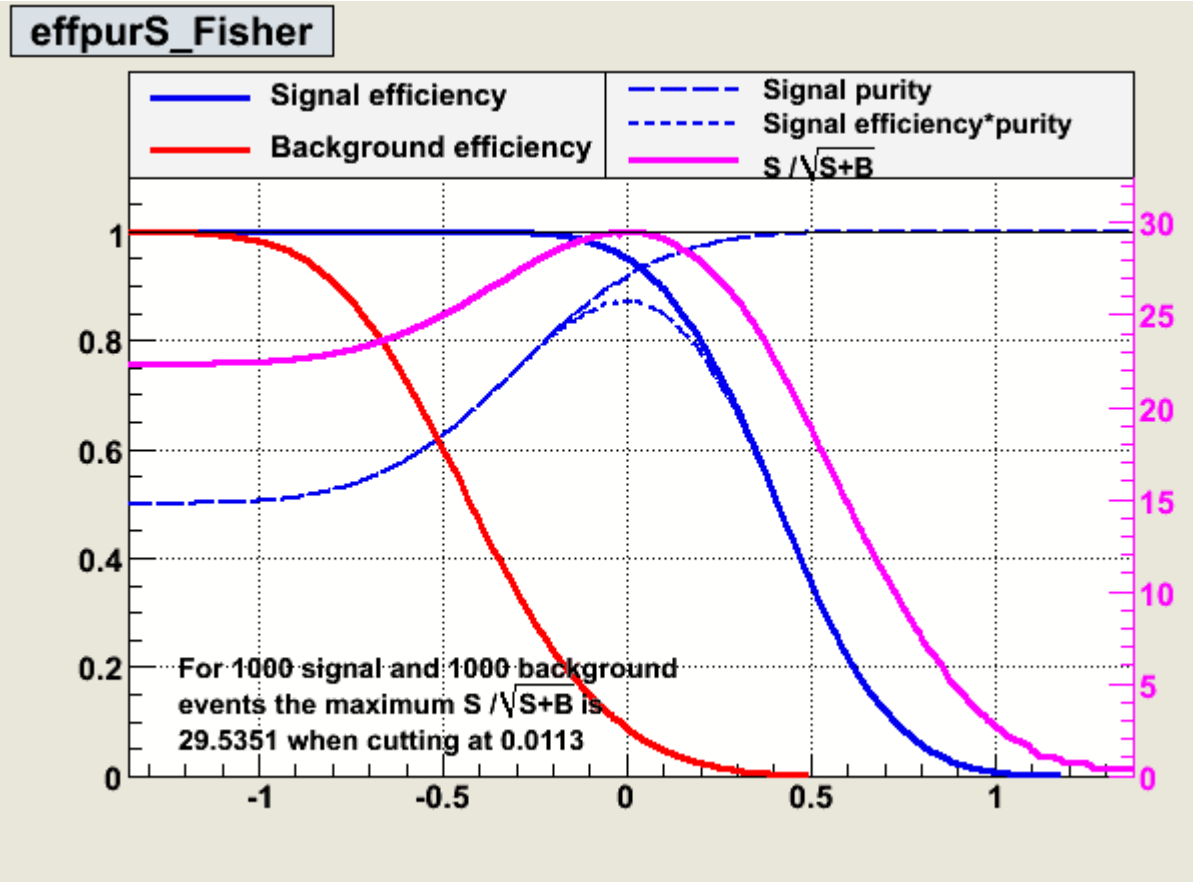


TMVA macro **`efficiencies.C`**

Efficiencies versus cut value

Select signal by cutting on output: $y > y_{\text{cut}}$

Fisher discriminant

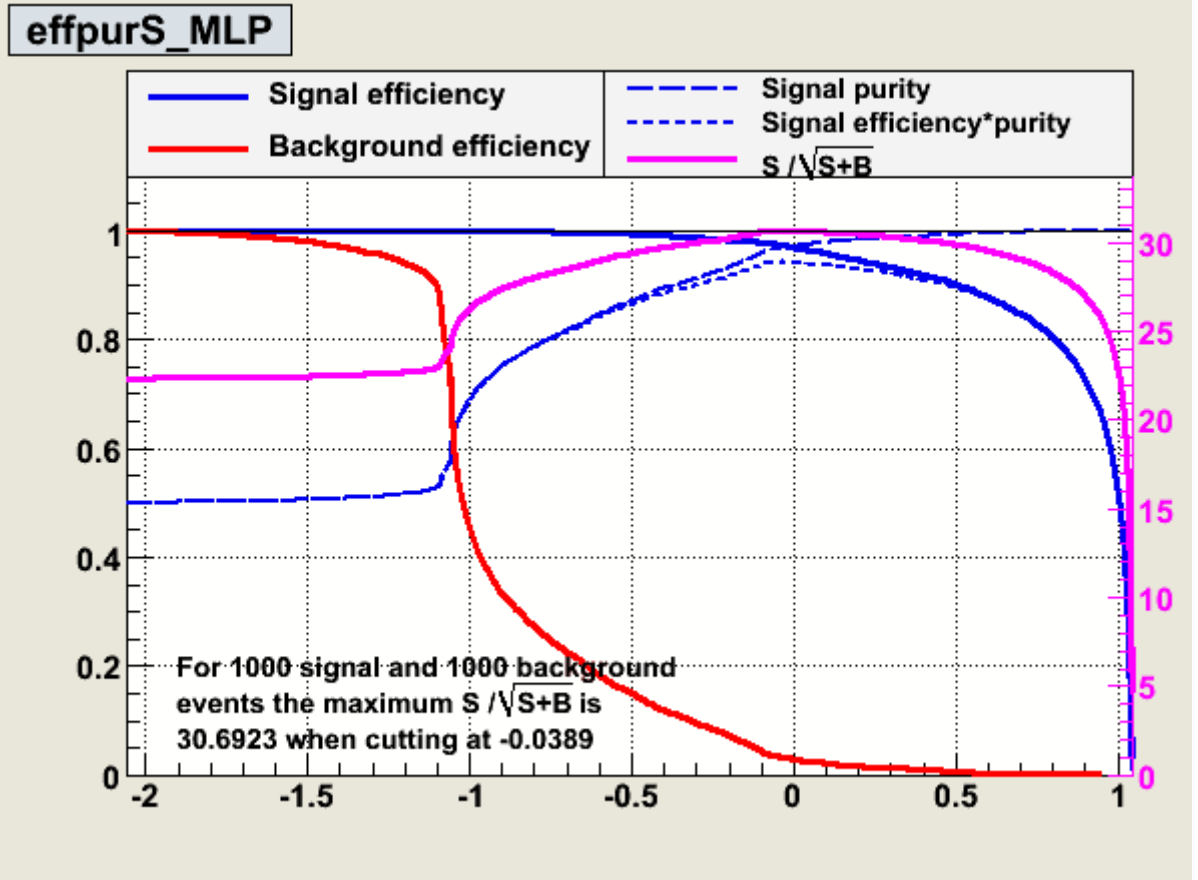


TMVA macro **`mvaeffs.C`**

Efficiencies versus cut value

Select signal by cutting on output: $y > y_{\text{cut}}$

Multilayer perceptron

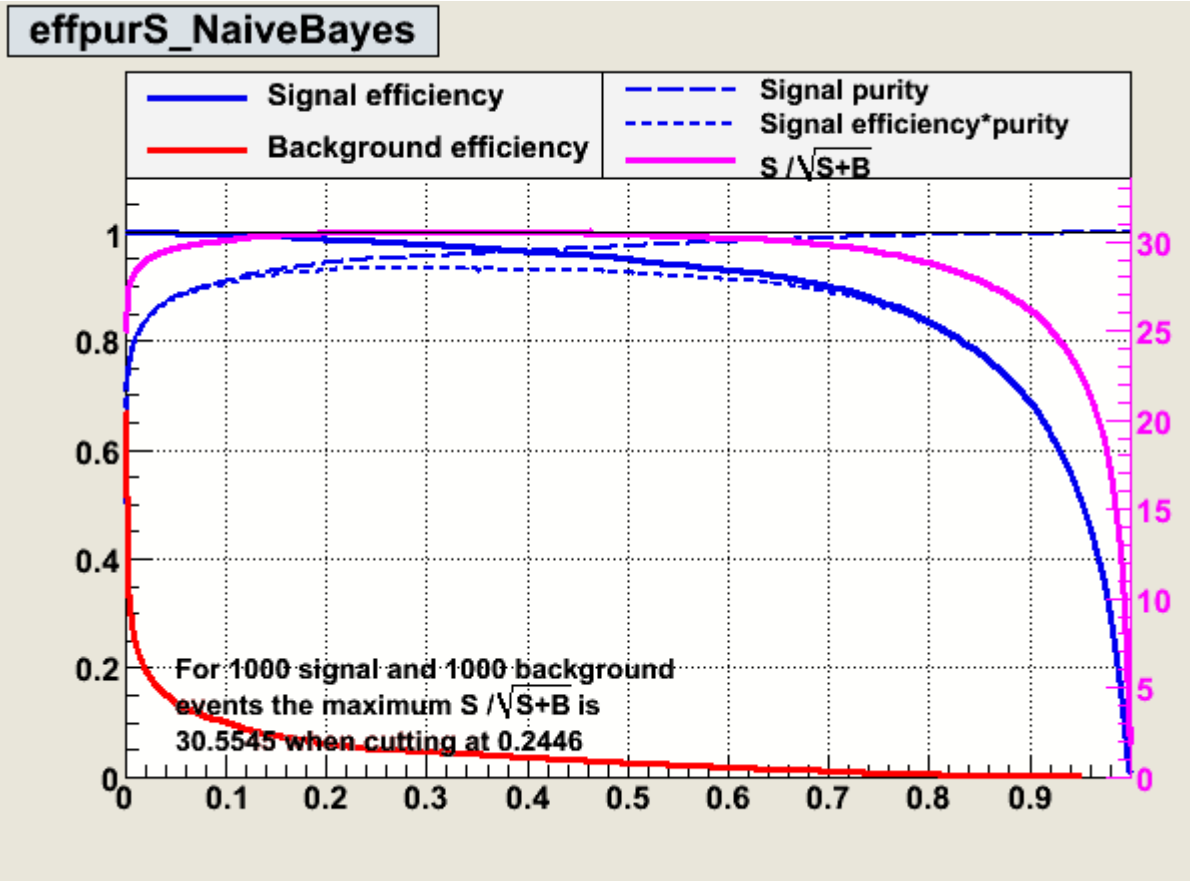


TMVA macro **`mvaeffs.C`**

Efficiencies versus cut value

Select signal by cutting on output: $y > y_{\text{cut}}$

Naive Bayes,
no decorrelation

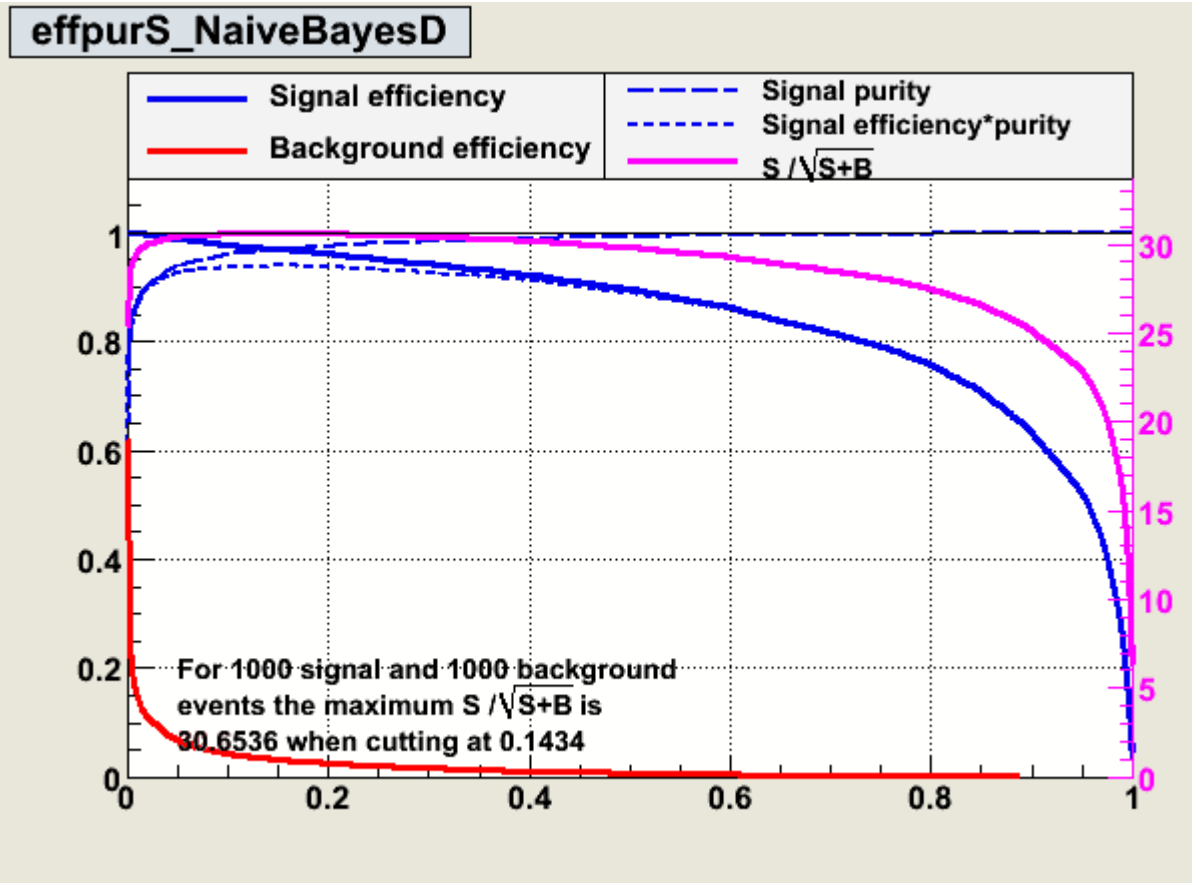


TMVA macro **`mvaeffs.C`**

Efficiencies versus cut value

Select signal by cutting on output: $y > y_{\text{cut}}$

Naive Bayes with decorrelation



TMVA macro **`mvaeffs.C`**

Lecture 2 summary

We have generalized the classifiers to allow nonlinear decision boundaries.

In neural networks, the user chooses a certain number of hidden layers and nodes; having more allows for an increasingly accurate approximation to the optimal decision boundary.

But having more parameters means that their estimates given a finite amount of training data are increasingly subject to statistical fluctuations, which shows up as overtraining.

The "naive Bayes" method seeks to approximate the joint pdfs of the classes as the product of 1-dimensional marginal pdfs (after decorrelation). To pursue this further we should therefore refine our approximations of 1-d pdfs.