Data Analysis with ROOT

Lecture 4: Multivariate analysis with neural networks and TMVA package

Aatos Heikkinen [1]

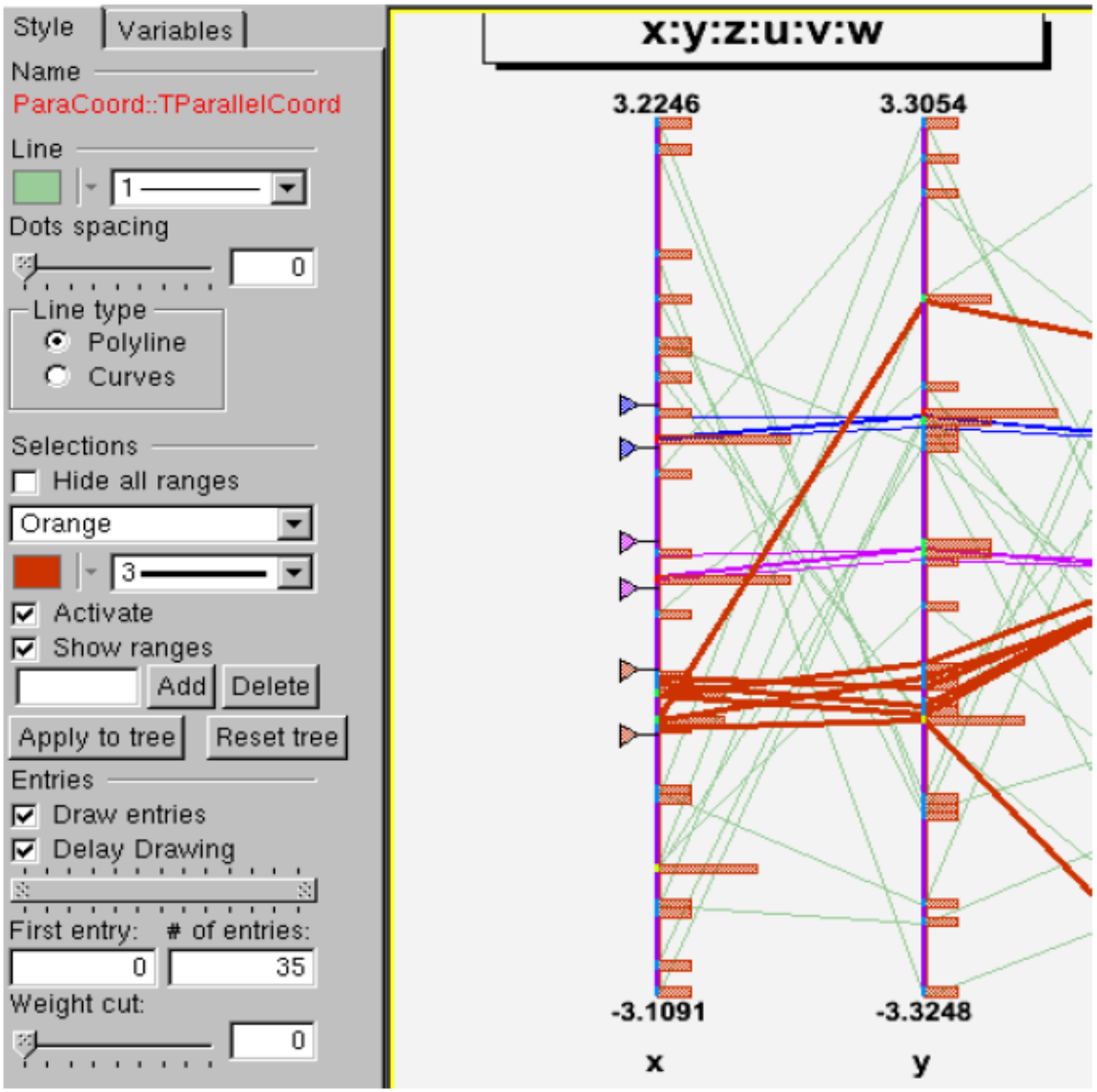
July 13, 2009

[1] Helsinki Institute of Physics, Helsinki, Finland. (aatos.heikkinen@cern.ch)

Outline

- Visual inspection of multidimensional data
- Seeking suitable coordinate system using principal component analysis
- A brief introduction to artificial neural networks (ANNs) with examples on
  - signal vs. background rejection, and
  - function estimation.
- TMVA – a system for multivariate data analysis
  - Introduction based on examples

Parallel coordinates



Parallel coordinates is a common way of visualising and analyzing multivariate data:

- **A point in n-dimensional space is represented as a polyline** with vertices on the parallel axes.

ROOT tutorial `parallelcoord.C` *demonstrating vertical histograms and the use of polylines.*

Interactive data analysis with parallelcoord.C (1/2)

```
#include "TParallelCoord.h"
//Author: Bastien Dallapiazza. Modified by A.
    Heikkinen for CSC'09.
Double_t r1 [...], r8;
Double_t dr = 3.5; TRandom *r;
void generate_random(Int_t i) {
   r1 = (2*dr*r->Rndm(i))-dr; [...]
   r8 = (2*dr*r->Rndm(i))-dr;
}
void parallelcoord() {
      TNtuple *nt = NULL; [...]
   nt = new TNtuple("nt","Demo ntuple","x:y:z:u:v:w");
   for (Int_t i=0; i<5; i++) {
      generate_random(i);
      nt->Fill(r1, r2, r3, r4, r5, r6);
   }
[...]
```

Interactive data analysis with parallelcoord.C (2/2)

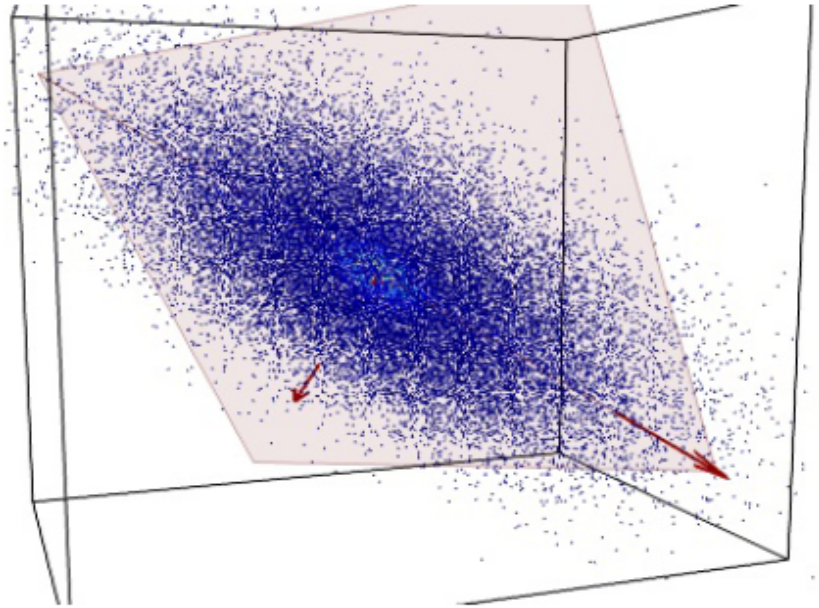
```
[...]
   nt->Draw("x:y:z:u:v:w","","para",5000);
   TParallelCoord* para = (TParallelCoord*)gPad->
    GetListOfPrimitives()->FindObject("ParaCoord");

   TParallelCoordVar* firstaxis =
         (TParallelCoordVar*)para->
          GetVarList()->FindObject("x");
   firstaxis->AddRange(new TParallelCoordRange(
      firstaxis,0.84,1.15));

   para->AddSelection("violet");
   para->GetCurrentSelection()->SetLineColor(kViolet);
        [...]
}
```

(More details in exercises.)

PCA

The Principal Component Method consists of applying a linear transformation to a sample of n-dimensional points:

- This transformation is described by an orthogonal matrix and is equivalent to a rotation of the original pattern space into a new set of coordinate vectors.
- A goal is to provide easier **feature identification**.
- The sample variances have minimum/maximum extremes along the coordinate axis.

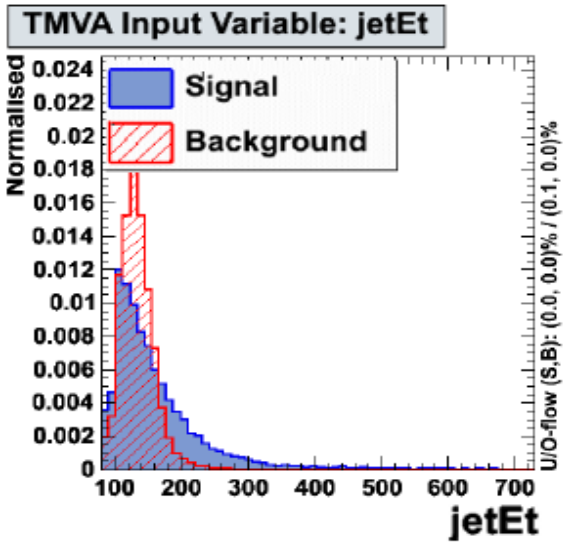


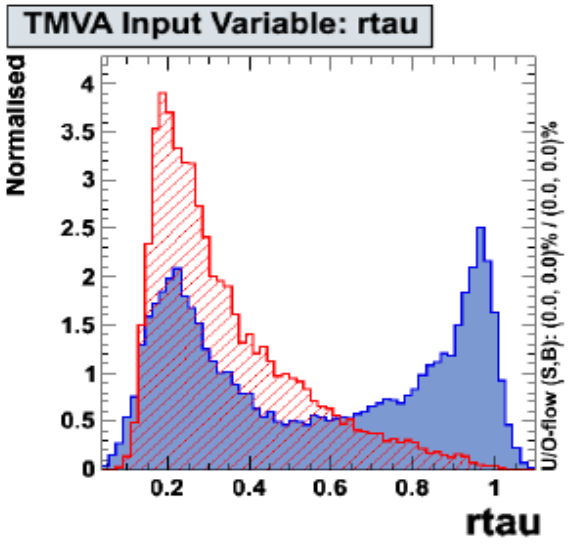
- The greatest variance by any projection of the data comes to lie on the first coordinate (called the **first principal component**),
- the second greatest variance on the second coordinate, etc.

In exploratory data analysis PCA is used to reduce the dimensionality of the data.

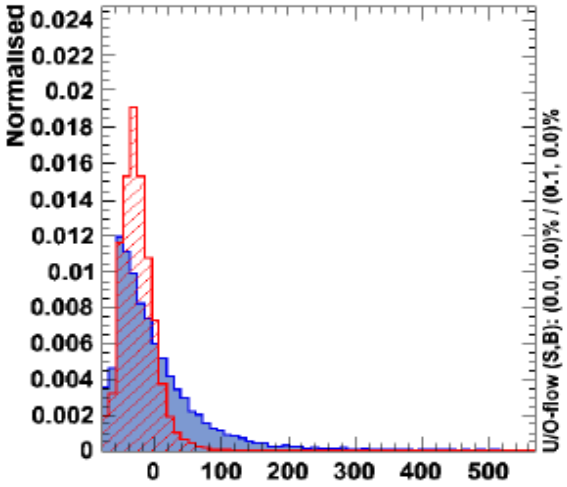
PCA of a 3D Gaussian distribution. The plane defined by the two largest eigenvectors (Image from Gael Varoquaux: *Dimension reduction with PCA*).

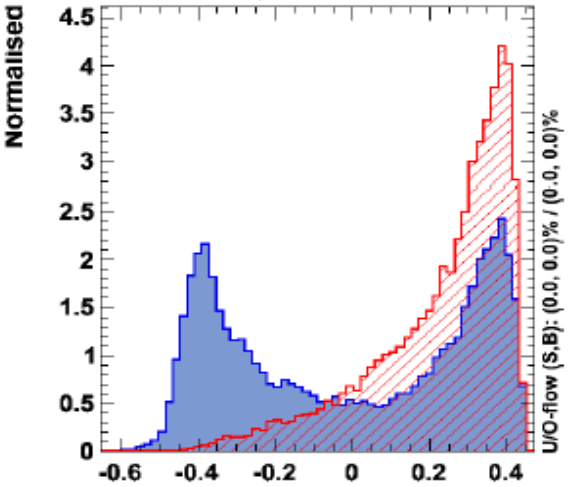
Example: PCA in the ROOT TMVA package






```
factory->BookMethod( [...]
                     VarTransform=PCA");
```





ROOT class `TPricipal` implements PCA.

Variables jetEt and rtau transformed according to principal component method.

(More details in exercise.)

Neural Networks: Basics

Artificial Neural Networks (ANNs) are inspired by the structure of biological neural networks and their way of encoding and solving problems[a]

The human brain contains $\sim 10^{12}$ neurons. The cell body receives electric signals to the dendrites by means of ions.

The accumulated effect of several simultaneous signals arriving is usually linearly additive, where as the resulting signal propagating the discharge is strongly non-linear.

[a] This section adapted from CSC'91 lectures *An Introduction to Artificial Neural Networks* by C. Peterson and T. Rögnvaldsson.

Neural Networks: Basics of feed-forward networks (1/2)

The neuron is defined as

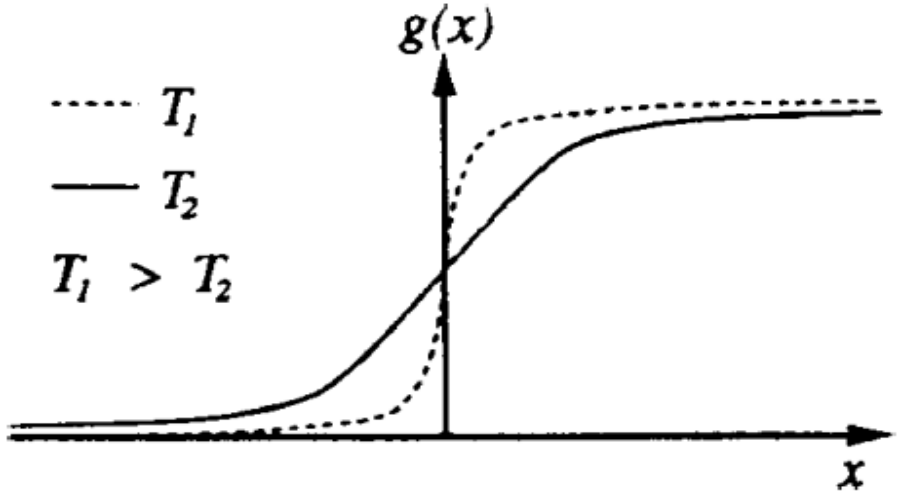
$$v_i = g(\sum_j \omega_{ij} v_j + \theta_i),$$

where v_j neurons feed the neuron v_i through synaptic weights ω_{ij}.

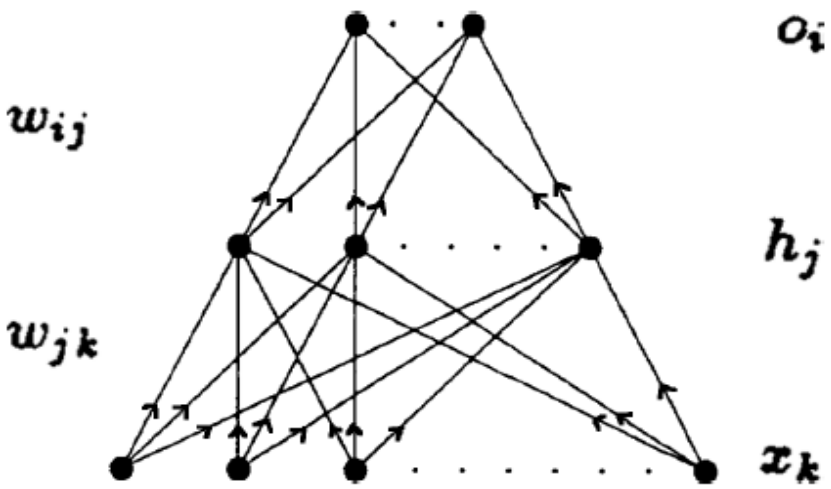
For neurons taking real values within the interval [-1,1]

- typical **transfer function** is $g(a_i) = \tanh a_i / T$,
- where $a_i = \sum_j \omega_{ij} v_j$.

'Temperature' T is used to define sigmoid shape:



Neural Networks: Basics of feed-forward networks (2/2)



In pattern recognition **input patterns** $\vec{x^p} = (x_1, x_2, ..., x_N)^p$ are categorized in terms of **different features** o_i.

$\vec{x^p}$ is fed into an input layer:

$$o_i(\vec{x^p}) = g(\sum_{j=0} \omega_{ij} g(\sum_{k=0} \omega_{jk} x_k^p)).$$

When generalised, this equation defines a feed-forward **multilayer perceptron** architecture.

Fitting ω to a given data set is called **supervised learning**. It **is done minimizing an error function** e.g. the summed squared error

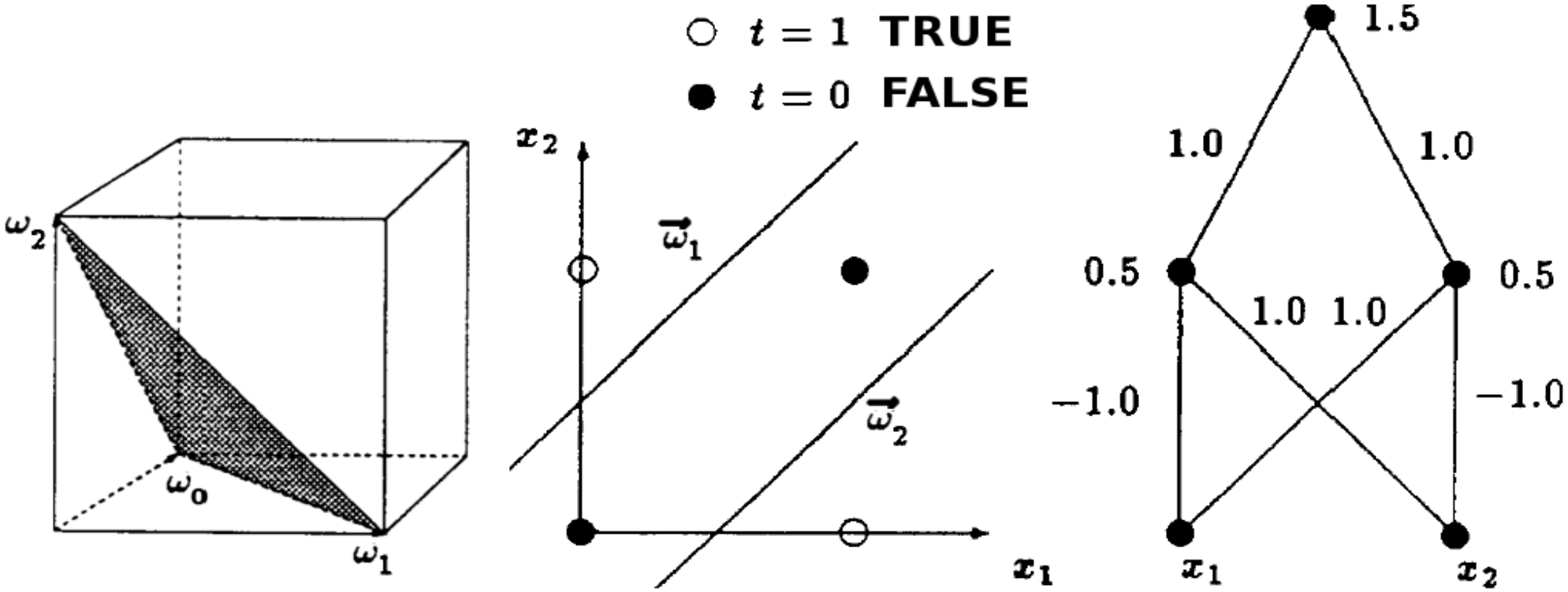
$$E = \frac{1}{2} \sum_p \sum_i (o_i^p - t_i^p)^2,$$

where t^p is a known **target pattern** (e.g. signal=1, background=0).

Weight updates $\Delta\omega$ are the calculated using suitable gradient descent method e.g. $\Delta\omega_{ij} = -\eta \partial E / \partial \omega_{ij}$, where η is the **learning rate** parameter.

Example: An explicit solution to the XOR problem

A network with a layer of hidden units is able to perform **non-linear mappings**:



State space of the Boolean exclusive-or (XOR) problem. The hyperplane representation of the hidden units reveals an internal representation of the data.

This MLP gives an explicit solution to the XOR problem. Hidden units are defined as $\vec{\omega_1} = (\omega_0, \omega_1, \omega_2) = (0.5, -1, 1)$ and $\vec{\omega_2} = (0.5, 1, -1)$.

Neural networks – practicalities

ANN can be seen as **a function that minimizes the error**, so the user should identify:

- how fast this function converges, and
- how well it can avoid local minima's.

A true motivation to use NNs is found **when traditional tools are failing**:

- in many dimensions, for complex problems, and
- for limited data samples.

User should pay attention to teaching data.

- Variable selection (e.g. PCA), normalisation, and **pre-processing** (e.g. suitable transformation) **is critical**[2].
- If possible, large problems should be divided into sub-problems (e.g. variable grouping).
- All information available (e.g. symmetry) should be used.
- A large amount of data is needed for many-dimensional problem.

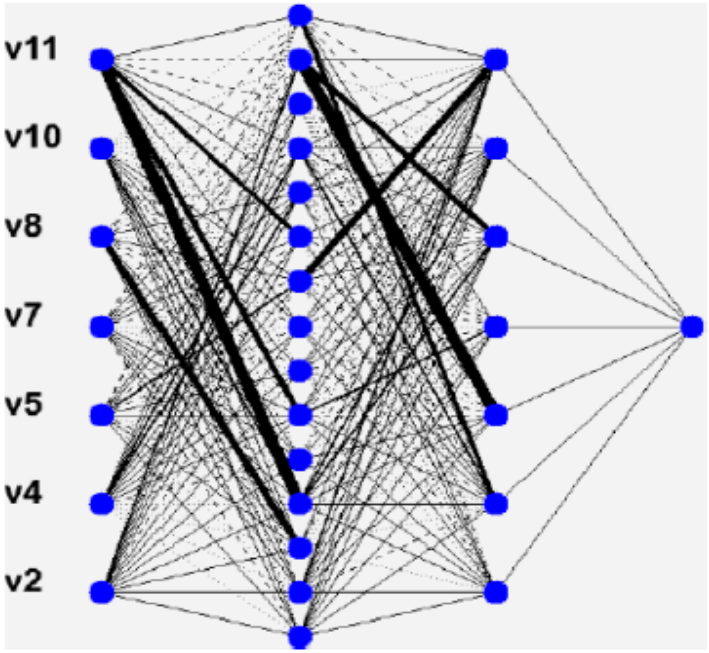
Any continuous function can be expressed with **one hidden layer** MLP.

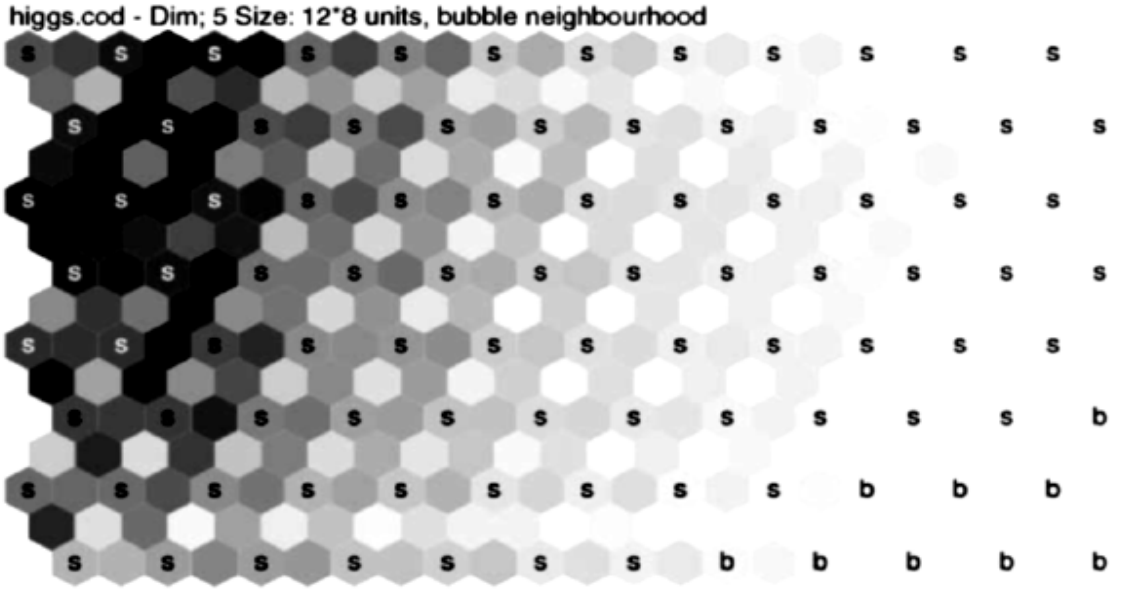
- Function with discontinuities requires **two hidden layers**.

[2] C. .M. Bishop *Neural Networks for Pattern Recognition*, Oxford University Press, 1995.

A real life application[3]

Realistic MLP architecture used in jet analysis:





Example graphics from SOM_PAK. Semantic map after unsupervised learning phase is divided into two regions representing signal (s) and background (b).

A limitation of supervised MLP: the features o_i must be known beforehand.

Unsupervised learning:

- The learner is given only unlabeled examples.
- Example: Self Organizing Maps SOMs.

[3] A. Heikkinen and S. Lehti , *Tagging b jets associated with heavy neutral MSSM Higgs bosons*, Nucl. Instr. and Meth., A559, 2006, 195-198

Example of a multilayer perceptron mlpHiggs.C(1/3)

```
void mlpHiggs(Int_t ntrain=100) {
// A Higgs boson search by C.Delaere and
// modified by A.Heikkinen for CSC'09
   if (!gROOT->GetClass("TMultiLayerPerceptron")) {
      gSystem->Load("libMLP");
   }
   // Prepare inputs [...]
   TTree *signal=(TTree*)input->Get("sig_filtered");
   TTree *background=(TTree*)input->Get("bg_filtered")
      ;
   TTree *simu = new TTree("MonteCarlo", "Filtered
      Monte Carlo Events");
   Float_t ptsumf, qelep, [...]; Int_t type;
   signal->SetBranchAddress("ptsumf",&ptsumf);
   signal->SetBranchAddress("qelep", &qelep); //[...]
   background->SetBranchAddress("ptsumf", &ptsumf);
   background->SetBranchAddress("qelep",  &qelep);
[...]
```

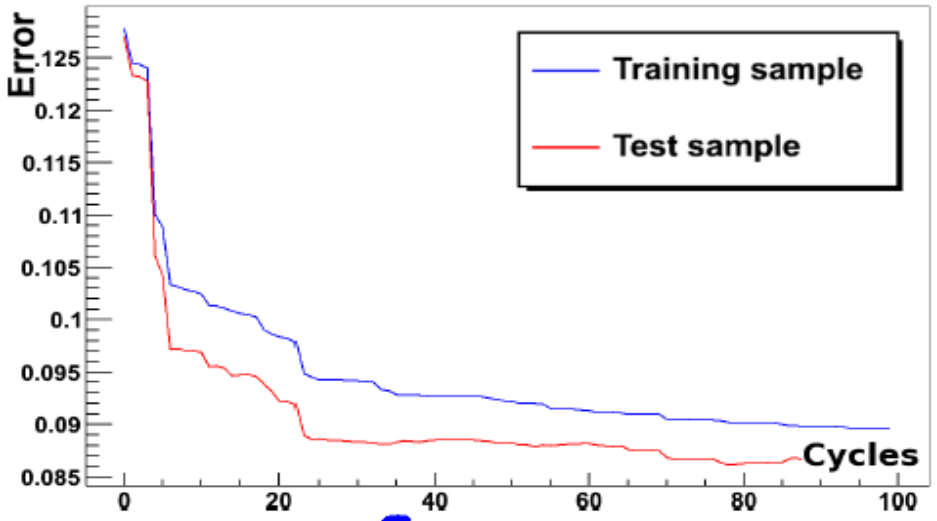
Example of a multilayer perceptron `mlpHiggs.C` (2/3)

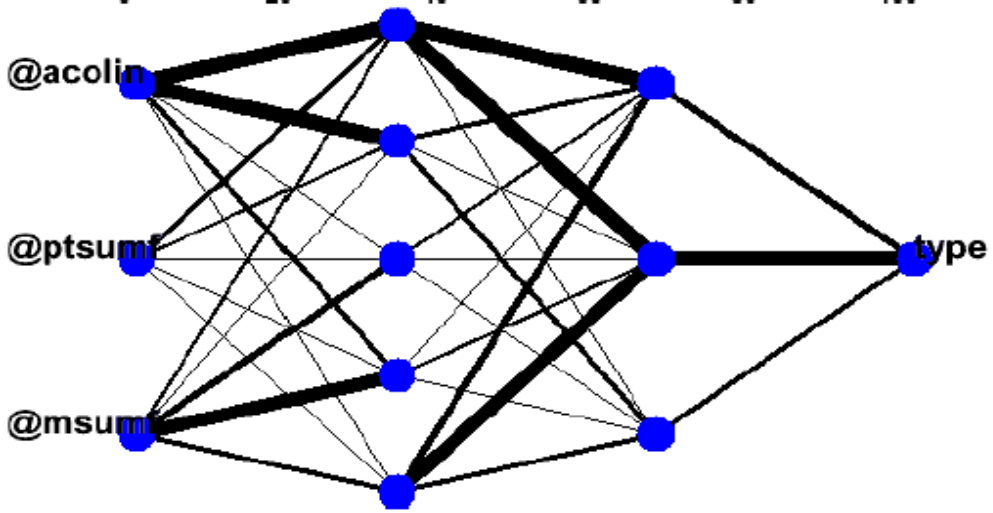
```
[...]
   simu->Branch("ptsumf", &ptsumf, "ptsumf/F");
   simu->Branch("qelep",  &qelep,  "qelep/F");

   for (i = 0; i < signal->GetEntries(); i++) {
      signal->GetEntry(i);
      simu->Fill();
   }

   for (i = 0; i < background->GetEntries(); i++) {
      background->GetEntry(i);
      simu->Fill();
   }

[...]
```

Example of a multilayer perceptron mlpHiggs.C (3/3)

```
[...]
   TMultiLayerPerceptron *mlp =
    new TMultiLayerPerceptron("@msumf,@ptsumf,@acolin
        :5:3:type",
        "ptsumf",simu,"Entry$%2","(Entry$+1)%2");

   mlp->Train(ntrain, "text,graph,update=10");
   mlp->Export("test","python");
   mlp->Draw();

   TMLPAnalyzer ana(mlp);
   ana.GatherInformations(); // Initialisation
   ana.CheckNetwork();       // output to the console
   ana.DrawDInputs(); // shows variable influences
   ana.DrawNetwork(0,"type==1","type==0");
}
```

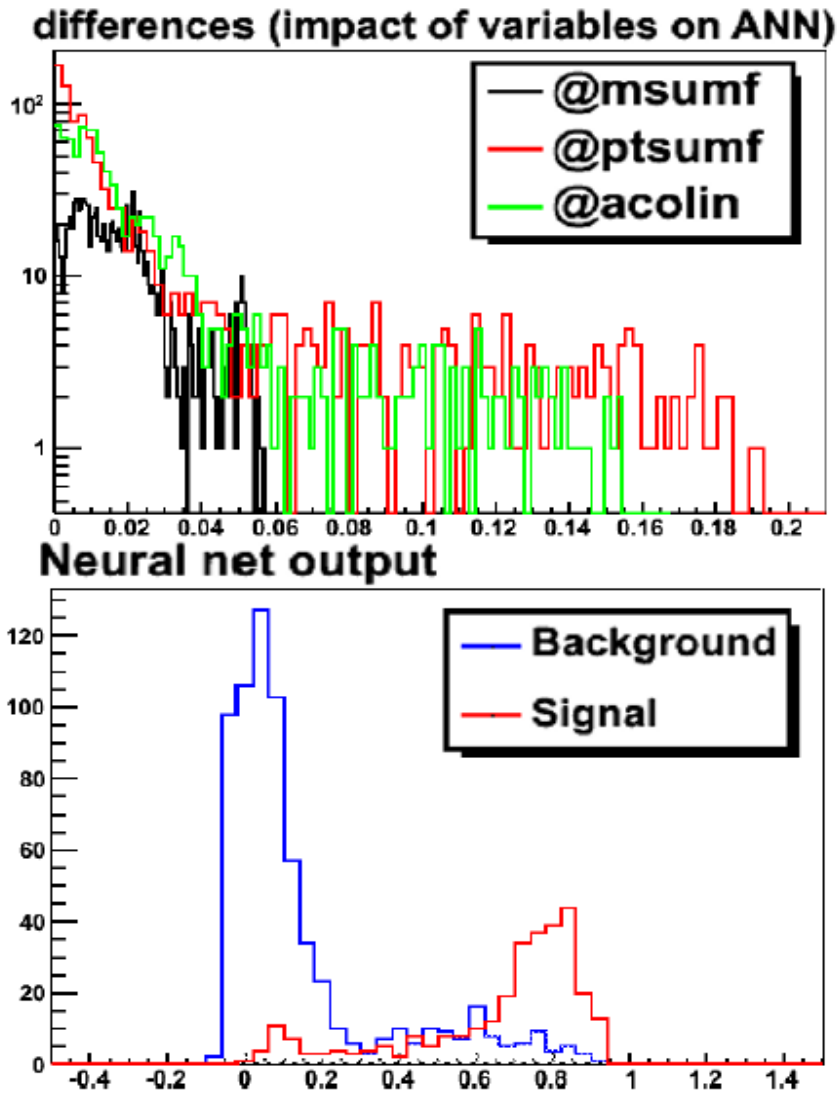
Example of a multilayer perceptron mlpHiggs.C






```
> root mlpHiggs.C
Processing mlpHiggs.C...
Info in
<TMultiLayerPerceptron::Train>:
Using 979 train and
979 test entries.
Training the Neural Network
Epoch: 0 learn=0.1283 test=0.1273
Epoch:10 learn=0.1054 test=0.0994
[...]
Epoch:90 learn=0.0892 test=0.0847
Epoch:99 learn=0.0891 test=0.0849
Training done.
test.py created.
Network with structure:
  @msumf,@ptsumf,@acolin:5:3:type

[...]
```

Example of a multilayer perceptron `mlpHiggs.C`




```
[...]
inputs with low values in the
differences plot may not be needed
@ptsumf -> 0.0359 +/- 0.0342
@acolin -> 0.0305 +/- 0.0287
@msumf  -> 0.0201 +/- 0.0228
root [1]
```

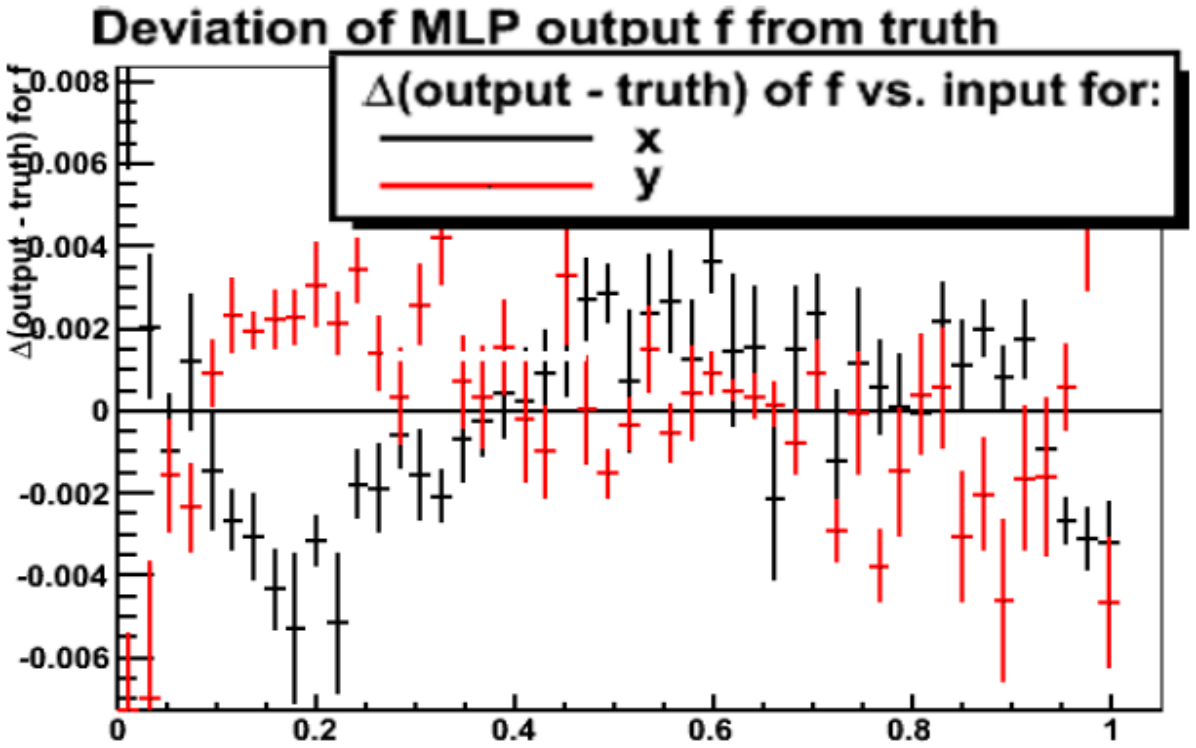
Variable ptsumf has best predictive power, while msumf is weakest variable.

Network tested with previously unseen data.

(More details in exercises.)

Regression analysis with mlpRegression.C (1/4)

```
/* Given a set {i} of input vectors i and
a set {o} of output vectors o,
we estimate using MLP the unknown function f(i)=o.
Axel Naumann, modified by and A.Heikkinen */
Double_t theUnknownFunction(Double_t x, Double_t y) {
   return sin((1.7+x) * (x-0.3)-2.3 * (y+0.7));
}
```



Regression analysis with mlpRegression.C (2/4)

```
void mlpRegression() {
   TNtuple* t=new TNtuple("tree","tree","x:y:f");
   TRandom r;
   for (Int_t i=0; i<1000; i++) {
      Float_t x=r.Rndm(); Float_t y=r.Rndm();
      t->Fill(x,y,theUnknownFunction(x,y));
   }
   TMultiLayerPerceptron* mlp=
       new TMultiLayerPerceptron("x,y:10:8:f",
           t,"Entry$%2","(Entry$%2)==0");
   mlp->Train(150,"graph update=10");
   TMLPAnalyzer* mlpa=new TMLPAnalyzer(mlp);
   mlpa->GatherInformations();
   mlpa->CheckNetwork();
   mlpa->DrawDInputs();

[...]
```

Regression analysis with mlpRegression.C (3/4)

```
[...]
   TCanvas* cIO=new TCanvas("TruthDeviation",
                "TruthDeviation");
   // draw the difference between the ANN's output
   // for (x,y) and the true value
   // f(x,y), vs. f(x,y), as TProfiles
   // evaluate the quality of the approximation.
   mlpa->DrawTruthDeviations();
   mlpa->DrawTruthDeviationInsOut();
   // draw a box plot of the ANN's output
   // for (x,y) vs f(x,y)
   mlpa->GetIOTree()->Draw("Out.Out0-True.True0:
        True.True0>>hDelta","","goff");

[...]
```

Regression analysis with mlpRegression.C (4/4)

```
[...]
   TH2F* hDelta=(TH2F*)gDirectory->Get("hDelta");
   hDelta->SetTitle("Diff. between ANN output and
      truth vs. truth"); hDelta->Draw("BOX");
   // difference of ANN's output (x,y) vs f(x,y)
   Double_t vx[225], vy[225], delta[225], v[2];
   for (Int_t ix=0; ix<15; ix++) {
      v[0]=ix/5.-1.;
      for (Int_t iy=0; iy<15; iy++) {
         Int_t idx=ix*15+iy;
         v[1]=iy/5.-1.; vx[idx]=v[0]; vy[idx]=v[1];
         delta[idx]=mlp->Evaluate(0, v)-
            theUnknownFunction(v[0],v[1]);
      }
   }
   TGraph2D* g2Extrapolate=new TGraph2D("Extrapolate",
  "ANN out-truth", 225, vx, vy, delta); g2Extrapolate
     ->Draw();
}
```

TMVA - Toolkit for Parallel Multivariate Data Analysis

Statistical classification is a procedure in which variables are arranged into groups (typically signal vs. background) based on a training set of previously labeled items.

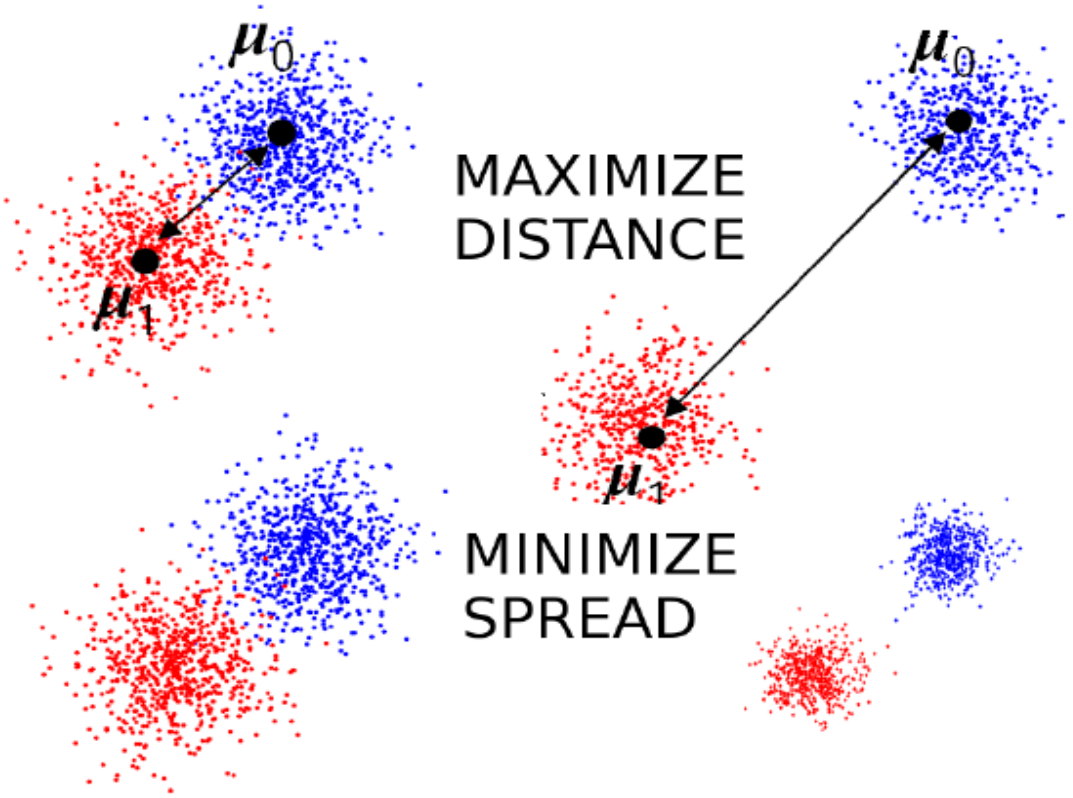
ROOT-integrated TMVA is a framework the training, testing and performance evaluation of multivariate classification techniques.

TMVA works in **transparent factory mode to guarantee an unbiased performance comparison between impressive set of classifiers**, such as:

- Rectangular cut optimisation
- Projective likelihood estimation (PDE approach)
- Multidimensional k-nearest neighbour classifier
- Boosted/Bagged decision trees (BDT)
- Support Vector Machine (SVM)
- Artificial neural networks (ANN)
- Linear discriminant analysis (H-Matrix and **Fisher discriminants**)
- For details of these methods consult http://tmva.sourceforge.net

Example: TMVA Fisher discriminant

Fisher's linear discriminant is defined as a measure of separation:

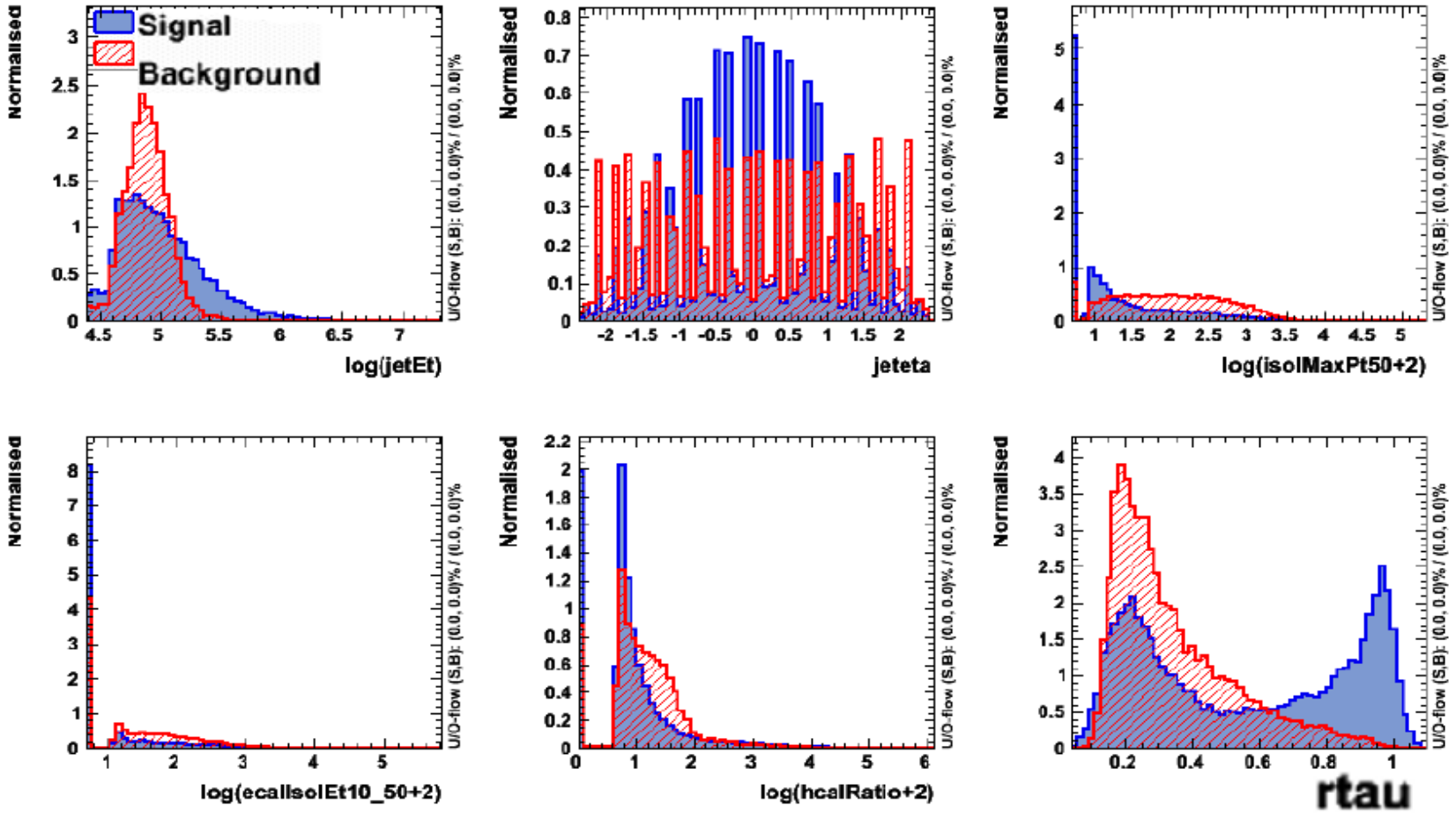


$$J = \frac{\text{separation between classes}}{\text{variance within classes}}$$

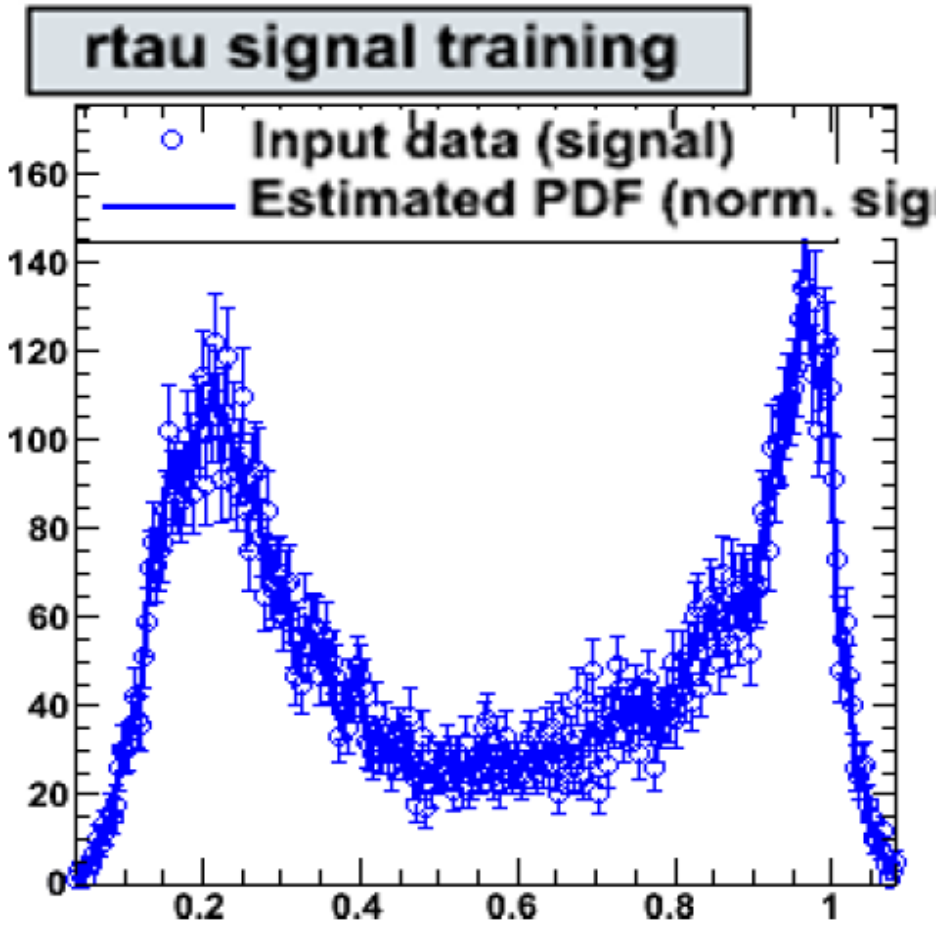
```
factory->BookMethod(
TMVA::Types::kFisher,
"Fisher", [...]);
```

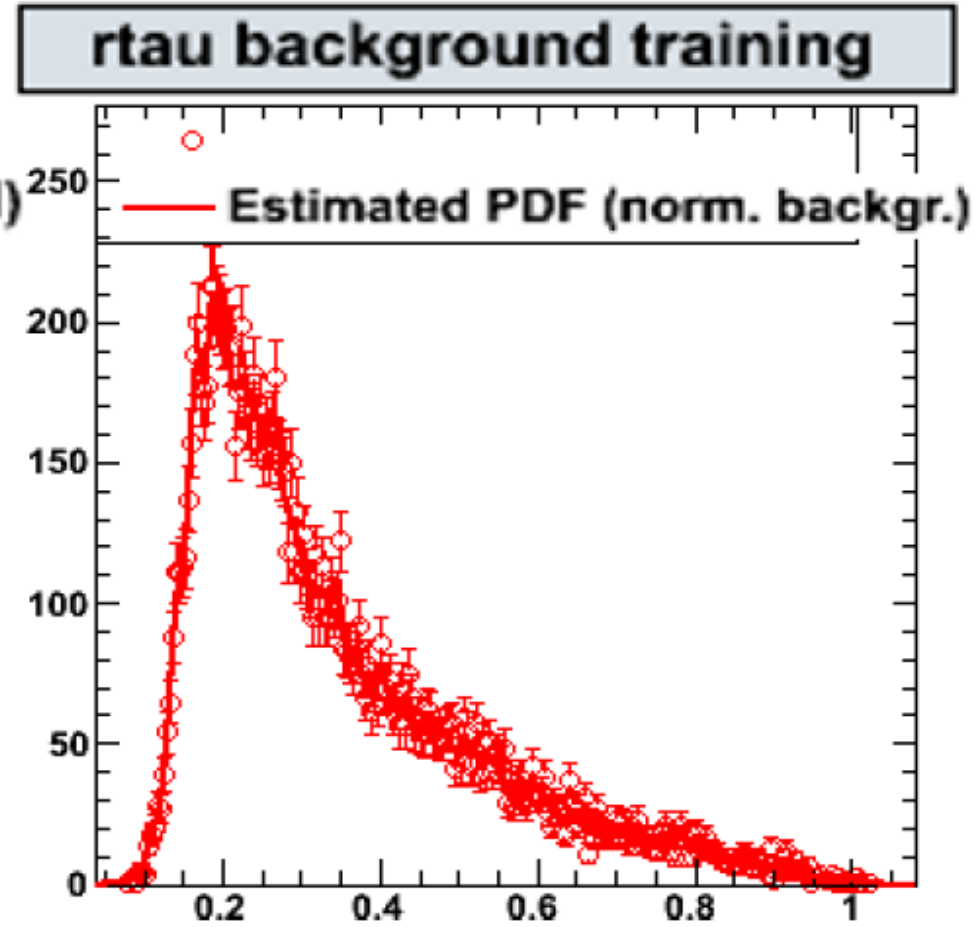
Example: separating a Higgs signal from the dominating QCD jet background

In addition to useful parallel coordinate plot, TMVA produces a **large amount of diagnostic plots**. In our example we have six variables:

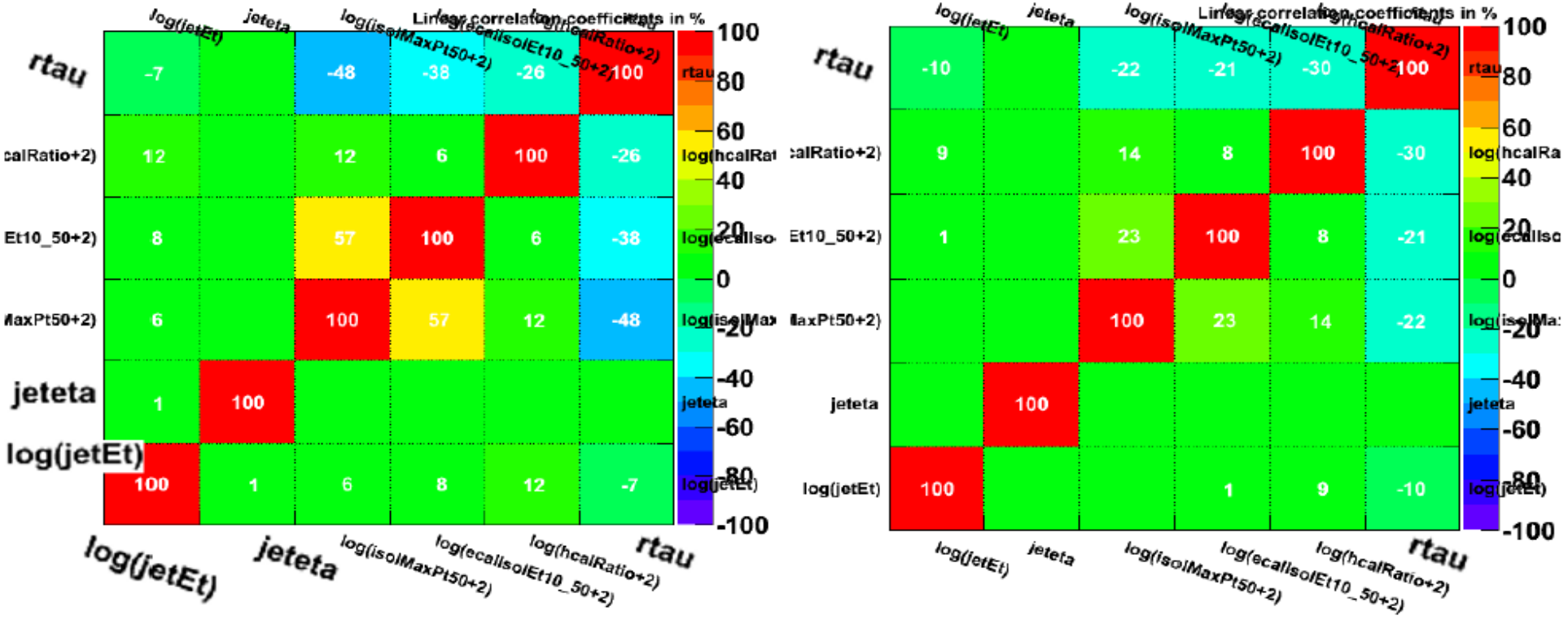


Example: PDF estimation of *rtau* variable

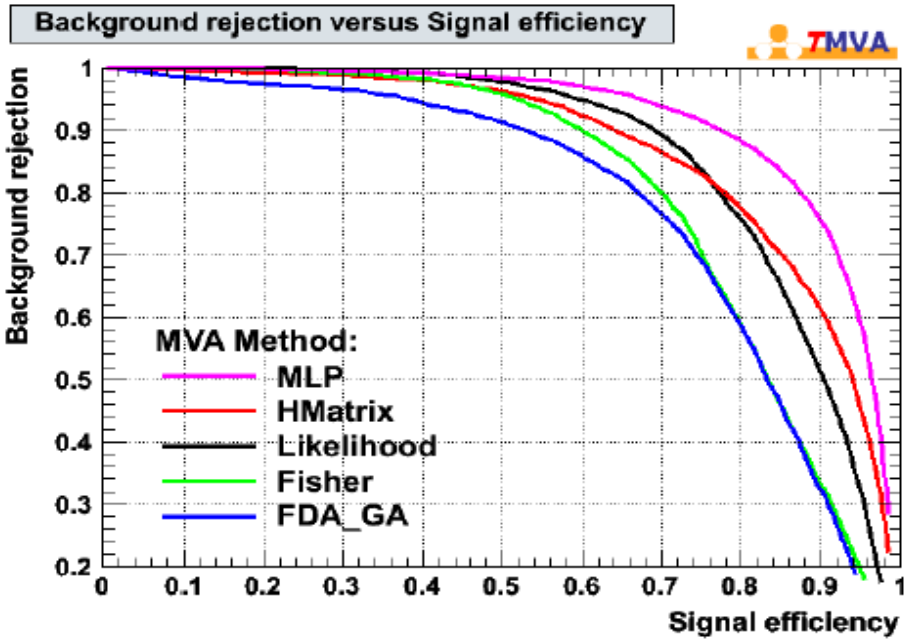


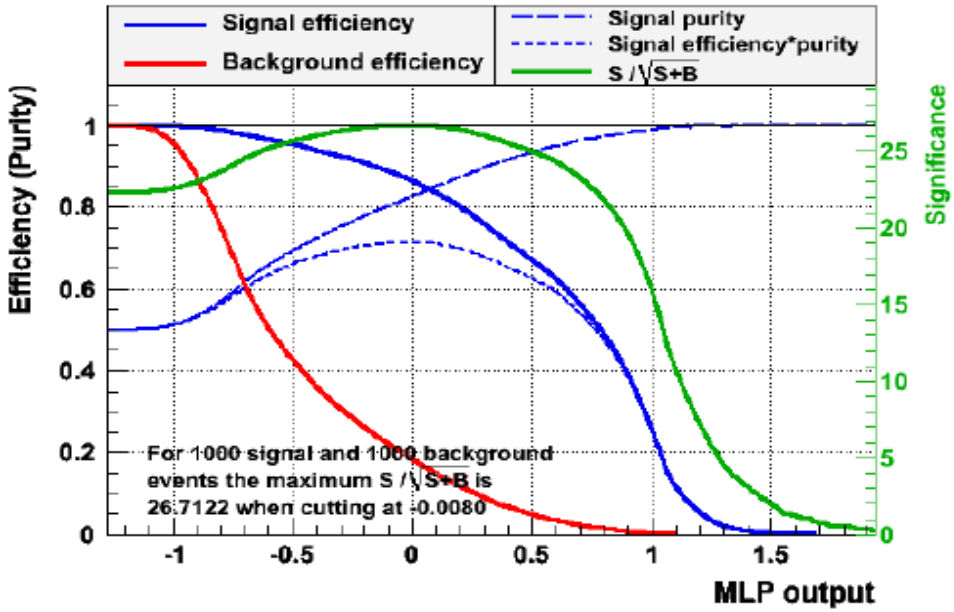


Example: correlations between variables

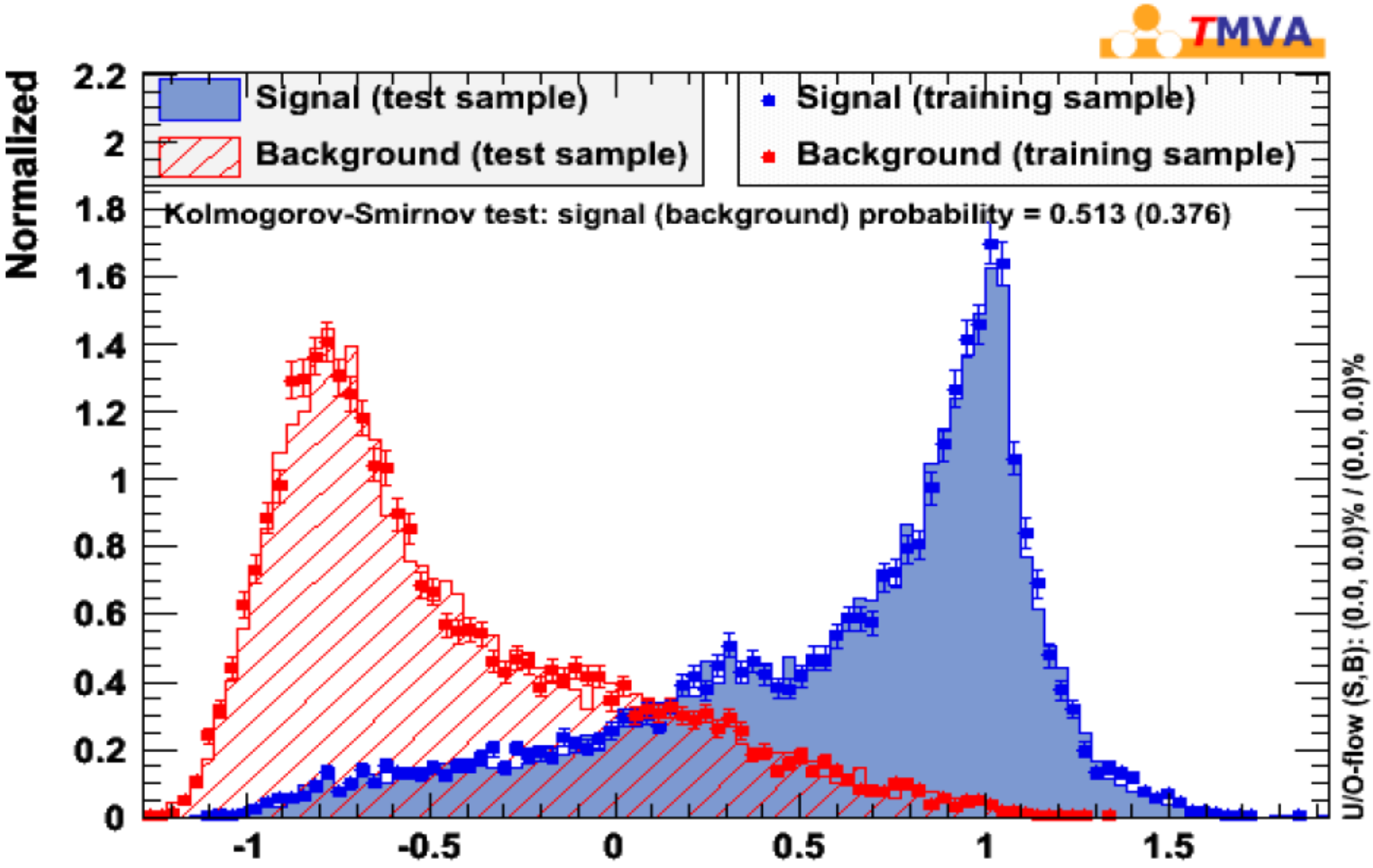


Example: background rejection vs. signal efficiency





Example: MLP classifier response



Example: TMVA script (1/3)

```
// TMVA - a ROOT toolkit for MVA
// Example of training and testing the classifiers.
// CSC'09 / Aatos Heikkinen
#include "TMVAGui.C"
#include "TMVA/Tools.h"
#include "TMVA/Factory.h"
void cscExample() {
   Bool_t Use_CutsGA    = 0; // Not used
   Bool_t Use_HMatrix   = 1; [...]
   TFile* outFile=TFile::Open("TMVA.root","RECREATE");
   TMVA::Factory *factory = new TMVA::Factory("
      TMVAnalysis", outFile,
            Form("!V:!Silent:%sColor",
                 gROOT->IsBatch()?"!":"") );
   factory->AddVariable("jeteta", 'F');
   // tracker isolation:
   //         pT threshold for tracks to be ignored
   factory->AddVariable("log(isolMaxPt50+2)", 'F');
[...]
```

Example: TMVA script (2/3)

```
[...]
  TString fs = "pythiaH200.root";// S: Higgs 200GeV/c
  TString fb = "pythiaGCD.root"; // B: QCD jets
  std::cout << "Accessing " << fname << std::endl;
  inputS = TFile::Open(fs); inputB = TFile::Open(fb);

  TTree *s = (TTree*)inputS->Get("TauID_h200_Tauola");
  TTree *b = (TTree*)inputB->Get("TauID_QCD_120_170");

  // Register trees
  Double_t signalWeight     = 1.0;
  Double_t backgroundWeight = 1.0;
  factory->AddSignalTree(     s, signalWeight);
  factory->AddBackgroundTree(b, backgroundWeight);

  // Apply additional cuts on the S and B samples
  TCut mycutS = "signalTracks == 1 && jetEt > 80 &&
                 abs(jeteta) < 2.4 && ldgPt > 20";
[...]
```

Example: TMVA script (3/3)

```
[...]
   // Use all remaining events for testing
   factory->PrepareTrainingAndTestTree( mycutS, mycutB,
      "SplitMode=Random:NormMode=NumEvents:!V" );

 if (Use_MLP)    // TMVA ANN: MLP
   factory->BookMethod( TMVA::Types::kMLP, "MLP",
      "H:!V:!Normalise:NeuronType=tanh:NCycles=100:
       HiddenLayers=N+5:TestRate=5:VarTransform=PCA");

   factory->TrainAllMethods();
   factory->TestAllMethods();
   factory->EvaluateAllMethods();

   outFile->Close();
   if (!gROOT->IsBatch()) TMVAGui("TMVA.root"); // GUI
}
```

Summary

- We have shown how to inspect data using parallel coordinate plot
- The usefulness of principal component method was discussed
  - In exploratory data analysis PCA is used to reduce the dimensionality of problems
- Most common ANN multilayer perceptron was introduced
- An TMVA environment for multivariate data analysis was discussed
  - It was demonstrated how the performance of different classifiers can be compared

(In exercises these topics are studied further using ROOT data analysis software.)