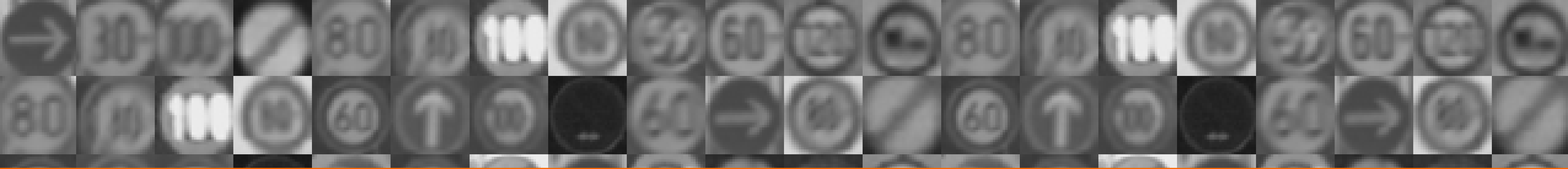
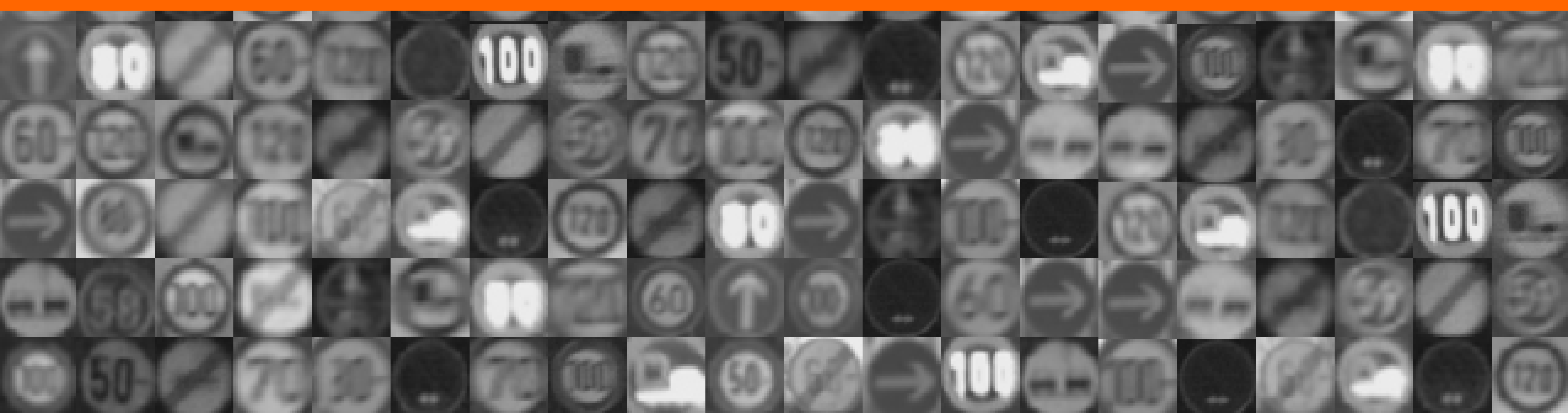


Multivariate Data Analysis and Machine Learning in High Energy Physics (II)

Helge Voss (MPI–K, Heidelberg)

Graduierten-Kolleg , Freiburg, 11.5-15.5, 2009



Outline

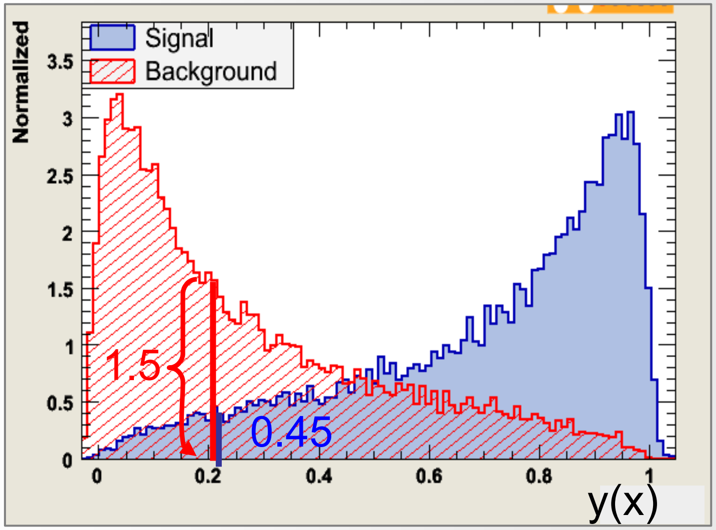
- Summary of last lecture
- A few more words on regression

- Classifiers
  - Bayes Optimal Analysis -- Kernel Methods and Likelihood Estimators
  - Linear Fisher Discriminant
- Introduction to TMVA

What we've heard so far:

- Traditional "cuts" on individual variables → certainly not the most powerful selection
- Multivariate Analysis is typically more powerful because:
  - → combination of variables
  - → effectively acting in the "D"-dimensional input variable (feature) space which also allows to include/exploit variable correlations
  - y(x), our discriminating function projects the "D"-dimensional feature space onto a 1-dimensional axis
    - the distributions of y(x|S) and y(x|B) represent $PDF_S(y)$ and $PDF_B(y)$
    - from those PDFs one can calculate the posteriori probability of an event with variable y being either signal or background, once the overall S/B ratio is known in the data sample:

$$\frac{f_S PDF_S(y)}{f_S PDF_S(y) + f_B PDF_B(y)} = P(C = S \mid y)$$

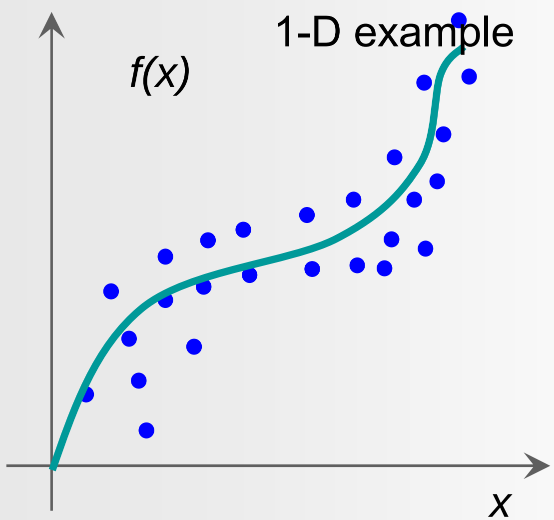


    - placing a CUT then on this 1-dimensional variable y and accepting all events "right of the cut" as signal defines signal selection efficiency as well as background efficiency (rejection)
    - this simple cut corresponds to a possibly complicated shaped separation boundary in the original D-dimensional feature space

- So far we have not yet seen any "explicit" discriminating function y(x) apart from the "Fisher Discriminant" in the exercises

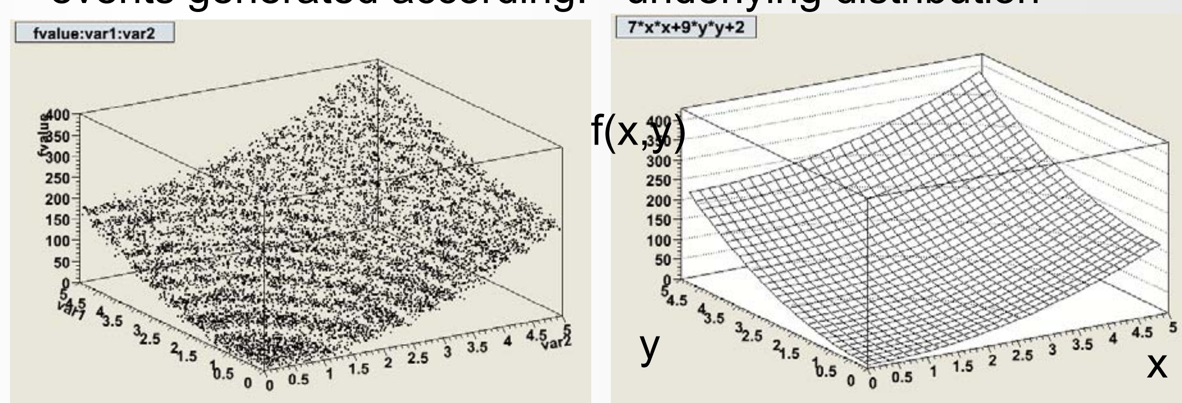
Regression

- how to estimate a “functional behaviour” from a given set of ‘known measurements” ?
- Assume for example “D”-variables that somehow characterize the shower in your calorimeter.
  - from Monte Carlo or testbeam measurements, you have a data sample with events that measure all D-Variables and the corresponding particle energy
  - the particle energy as a function of the D observables is a surface in the D+1 dimensional space given by D-observables + energy



2-D example

events generated according: underlying distribution

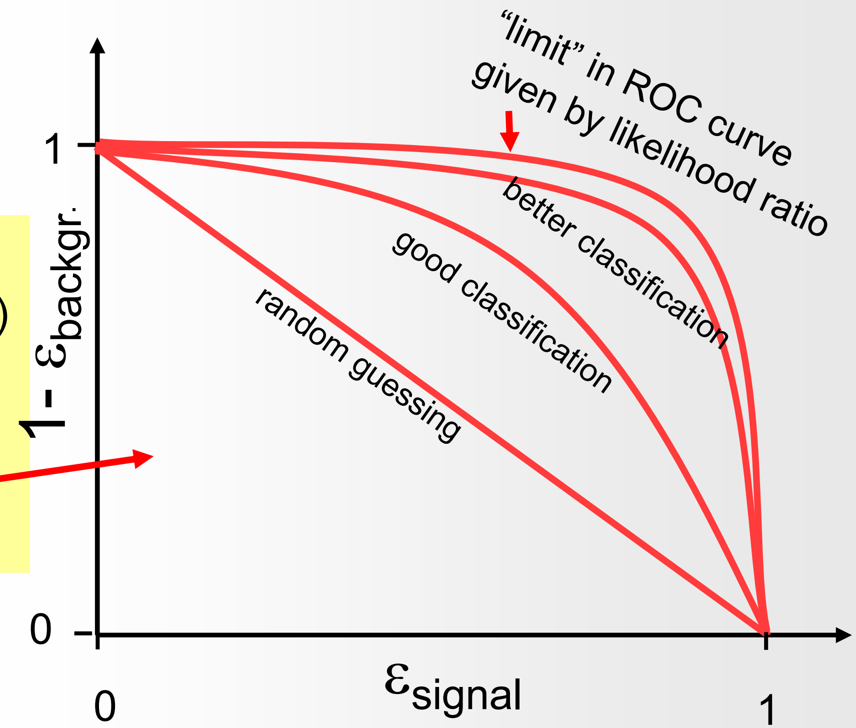


- while in the typical regression you fit a known anlytic function (i.e. in the above 2-D example, you’d fit the function ax^2+by^2+c to the data. In the regression I’m talking about, the regression function might be so complicated, or the model unknown, that you cannot give a reasonable analytic fit function in closed form. Hence you want something more general, i.e. piecewise defined splines, kernel estimators, decision trees to approximate the function f(x)

Neyman-Pearson Lemma

differential cross section of signal, bkg process

$$\text{Likelihood Ratio}: y(x) = \frac{P(x\,|\,S)}{P(x\,|\,B)}$$



Neyman-Peason:
The Likelihood ratio used as "selection criterium" y(x) gives for each selection efficiency the best possible background rejection.
i.e. it maximises the area under the "Receiver Operation Characteristics" (ROC) curve

- Unfortunately, the true probability densities functions are typically unknown:

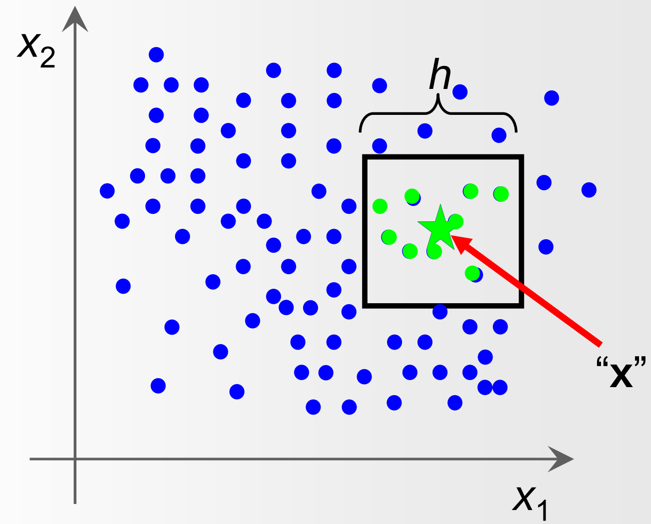
→ Neyman-Pearsons lemma doesn't really help us directly

- try to estimate the directly functional form of p(x|C): (i.e. the differential cross section folded with the detector influences) from training events from which the likelihood ratio can be obtained
  - → e.g. D-dimensional histogram, Kernel densitiy estimators, …
- find a "discrimination function" y(x) and corresponding decision boundary (i.e. hypersurface in the "feature space": y(x) = const) that optimially separates signal from background
  - → e.g. Linear Discriminator, Neural Networks, …

Nearest Neighbour and Kernel Density Estimator

- Trying to estimate the probability density p(x) in D-dimensional space:
- The only thing at our disposal is our "training data"
- Say we want to know p(x) at "this" point "x"
- One expects to find in a volume V around point "**x**" $N^*\int_V p(x)dx$ events from a dataset with N events
- Say we choose as volume the square drawn then we find in our dataset of N events, one finds K-events:

"events" distributed according to p(x)



$$K = \sum_{n=1}^{N} k\left(\frac{\mathrm{x}-\mathrm{x}_n}{h}\right), \quad \text{with} \quad k(\mathrm{u}) = \begin{cases} 1, & |u_i| \le \frac{1}{2}, i = 1 \ldots D \\ 0, & \text{otherwise} \end{cases}$$

k(u): is called a Kernel function: rectangular → Parzen-Window

- K determined from the "training data" hence now gives an estimate of the mean of the p(x) over the volume V: $\int_V p(x)dx = K/N$
- <u>Classification:</u> Determine PDF$_S$(x) and PDF$_B$(x)

→ likelihood ratio as classifier!

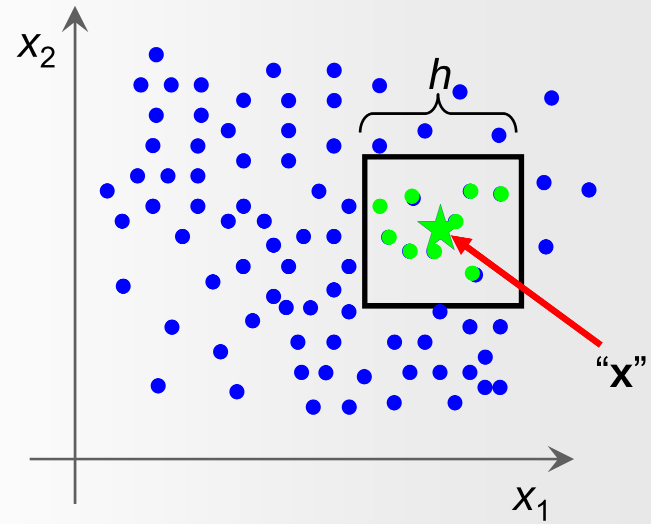
$$p(\mathrm{x}) = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{h^D}k\left(\frac{\mathrm{x}-\mathrm{x}_n}{h}\right)$$

→ Kernel Density estimator of the probability density

Nearest Neighbour and Kernel Density Estimator

- Trying to estimate the probability density p(x) in D-dimensional space:
- The only thing at our disposal is our "training data"
- Say we want to know p(x) at "this" point "x"
- One expects to find in a volume V around point "**x**" $N^*\int_V p(x)dx$ events from a dataset with N events
- Say we choose as volume the square drawn then we find in our dataset of N events, one finds K-events:

"events" distributed according to p(x)



$$K = \sum_{n=1}^{N} k\left(\frac{x - x_n}{h}\right), \quad \text{with} \quad k(u) = \begin{cases} 1, & |u_i| \le \frac{1}{2}, i = 1 \ldots D \\ 0, & \text{otherwise} \end{cases}$$

k(u): is called a Kernel function: rectangular → Parzen-Window

- K determined from the "training data" hence now gives an estimate of the mean of the p(x) over the volume V: $\int_V p(x)dx = K/N$

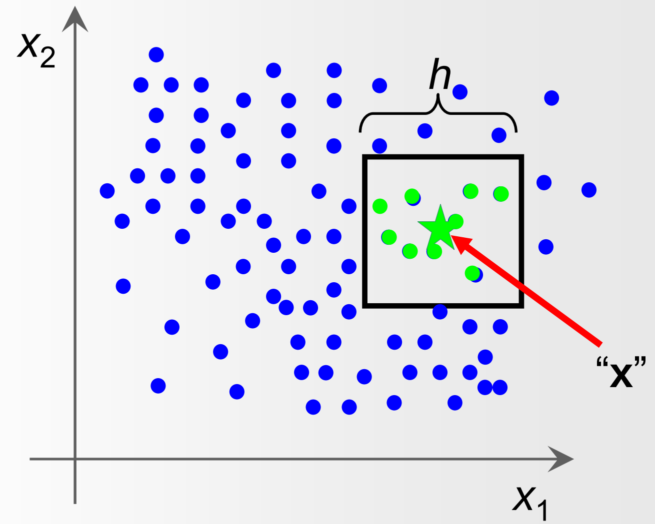
- Regression: If each events with (x_1,x_2) carries a "function value" $f(x_1,x_2)$ (e.g. energy of incident particle) →

$$\frac{1}{N}\sum_{i}^{N} f(x_1^i, x_2^i) = \int_V \hat{f}(x_1,x_2)p(\vec{x})d\vec{x}$$

Nearest Neighbour and Kernel Density Estimator

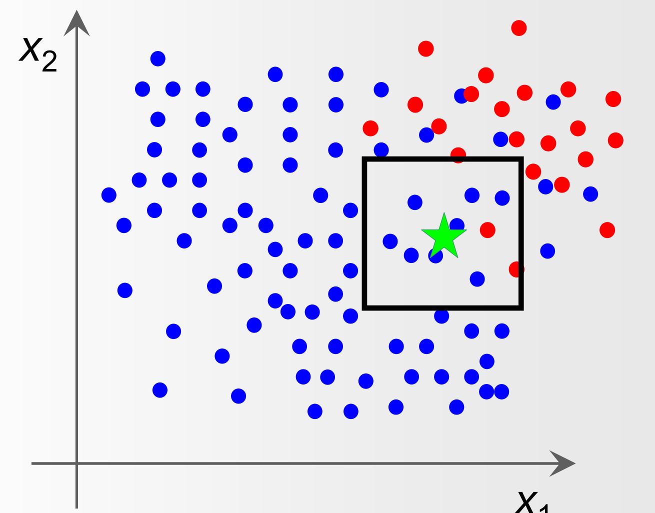
- Trying to estimate the probability density p(x) in D-dimensional space:
- The only thing at our disposal is our "training data"
- Say we want to know p(x) at "this" point "x"
- One expects to find in a volume V around point "**x**" $N^*\int_V p(x)dx$ events from a dataset with N events
- Say we choose as volume the square drawn then we find in our dataset of N events, one finds K-events:

"events" distributed according to p(x)



$$K = \sum_{n=1}^{N} k\left(\frac{x - x_n}{h}\right), \quad \text{with} \quad k(u) = \begin{cases} 1, & |u_i| \le \frac{1}{2}, i = 1 \dots D \\ 0, & \text{otherwise} \end{cases}$$

- determine K from the "training data" with signal and background mixed together



→ kNN : k-Nearest Neighbours
relative number events of the various classes amongst the k-nearest neighbours

$$y(x) = \frac{n_S}{K}$$

Kernel Density Estimator

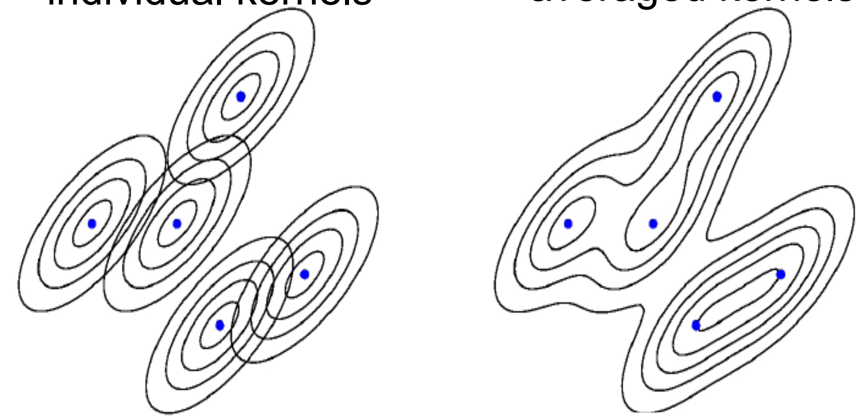
- Parzen Window: "rectangular Kernel" → discontinuities at window edges

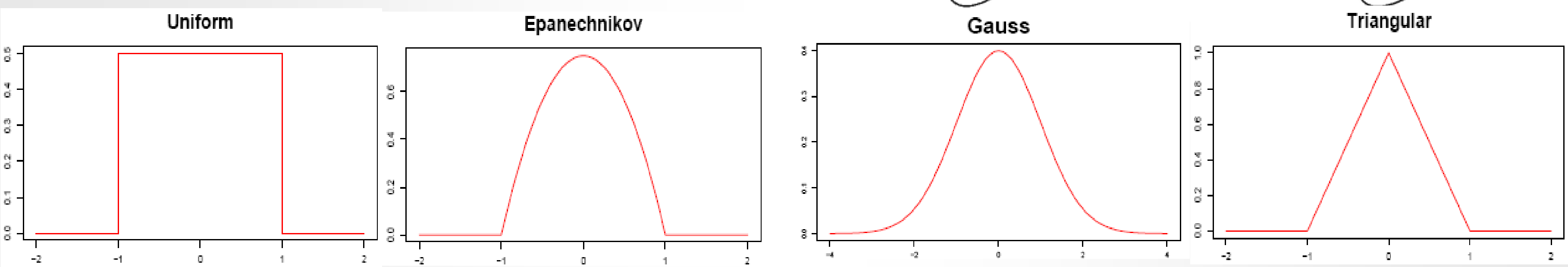
→ smoother model for p(x) when using smooth Kernel Fuctions: e.g. Gaussian

$$p(\mathrm{x}) = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{(2\pi h^2)^{1/2}}\exp\left(\frac{\left\|\mathrm{x}-\mathrm{x}_n\right\|^2}{2h^2}\right)$$ ↔ probability density estimator

- place a "Gaussian" around each "training data point" and sum up their contributions at arbitrary points "**x**" → p(**x**)
- h: "size" of the Kernel → "smoothing parameter"
- there is a large variety of possible Kernel functions

individual kernels averaged kernels

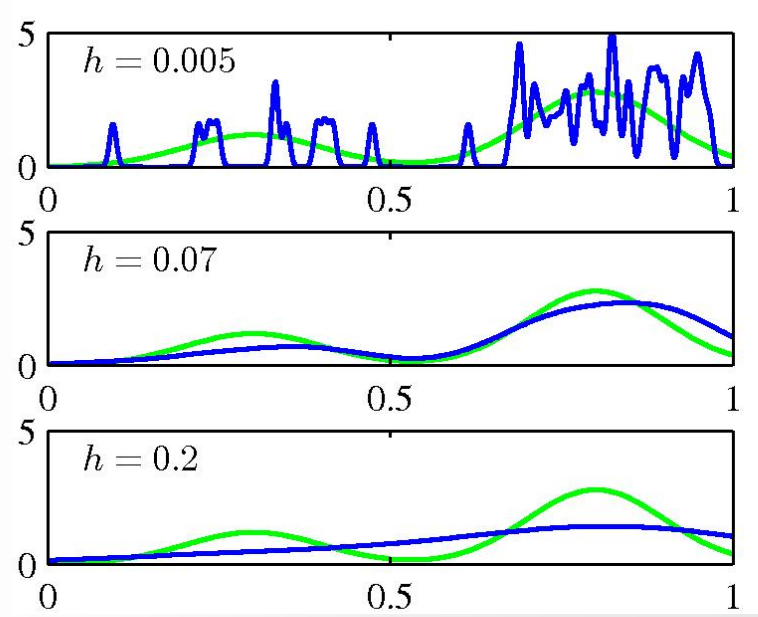




Kernel Density Estimator

$$p(\mathbf{x}) = \frac{1}{N}\sum_{n=1}^{N} K_h(\mathbf{x} - \mathbf{x}_n)$$: a general probability density estimator using kernel K

- h: "size" of the Kernel → "smoothing parameter"
- chosen size of the "smoothing-parameter" → more important than kernel function
- h too small: overtraining
- h too large: not sensitive to features in p(x)



(Christopher M.Bishop)

for Gaussian kernels, the optimum in terms of MISE (mean integrated squared error) is given by: $h_{xi}=(4/(3N))^{1/5}\,\sigma_{xi}$ with σ_{xi}=RMS in variable x_i

- a drawback of Kernel density estimators:

Evaluation for any test events involves ALL TRAINING DATA → typically very time consuming

- binary search trees (i.e. Kd-trees) are typically used in kNN methods to speed up searching

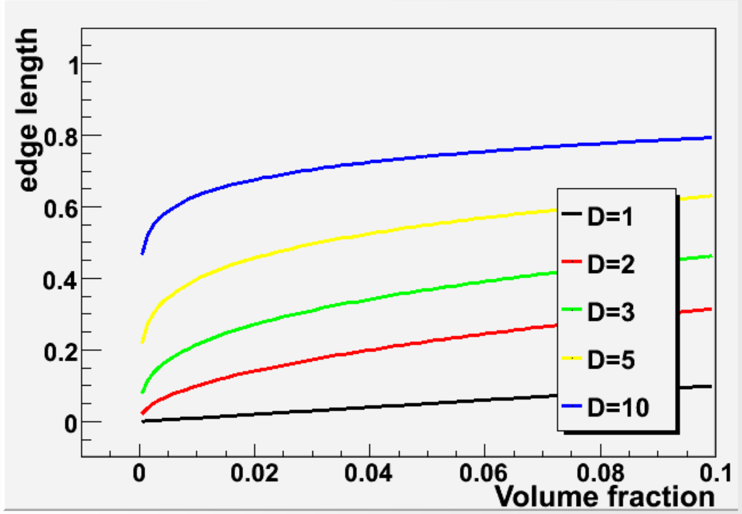
"Curse of Dimensionality"

Bellman, R. (1961), Adaptive Control Processes: A Guided Tour, Princeton University Press.

We all know:

Filling a D-dimensional histogram to get a mapping of the PDF is typically unfeasable due to lack of Monte Carlo events.

Shortcoming of nearest-neighbour strategies:



- in higher dimensional classification/regression cases the idea of looking at "training events" in a reasonably small "vicinity" of the space point to be classified becomes difficult:

consider: total phase space volume $V=1^D$

for a cube of a particular fraction of the volume:

$$\text{edge length} = (\text{fraction of volume})^{1/D}$$

- In 10 dimensions: in order to capture 1% of the phase space

→ 63% of range in each variable necessary → that's not "local" anymore..☹

→Therefore we still need to develop all the alternative classification techniques

Naïve Bayesian Classifier
"often called: (projective) Likelihood"

while Kernel methods or Nearest Neighbour classifiers try to estimate the full D-dimensional joint probability distributions

If correlations between variables are weak: → $p(\mathbf{x}) \cong \prod_{i=0}^{D} p_i(\mathbf{x})$

$$y(x_{\text{PDE},k_{\text{event}}}) = \frac{\prod\limits_{i \in \{\text{variables}\}} p_i^{\text{signal}}(x_{i,k_{\text{event}}})}{\sum\limits_{C \in \{classes\}} \left(\prod\limits_{i \in \{\text{variables}\}} p_i^{C}(x_{i,k_{\text{event}}}) \right)}$$

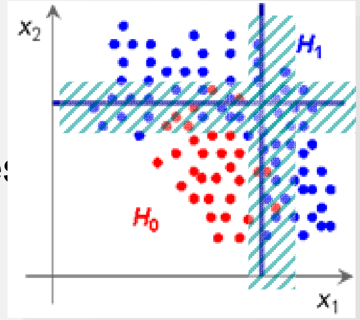
Likelihood ratio for event *event*

PDFs

discriminating variables

Classes: signal, background types

- One of the first and still very popular MVA-algorithm in HEP
  - rather than making hard cuts on individual variables, allow for some "fuzzyness": one very signal like variable may counterweigh another les[s] signal like variable
- This is about the optimal method in the absence of correlations



PDE introduces fuzzy logic

Naïve Bayesian Classifier
“often called: (projective) Likelihood”

How parameterise the 1-dim PDFs ??

event counting (histogramming)

parametric (function) fitting

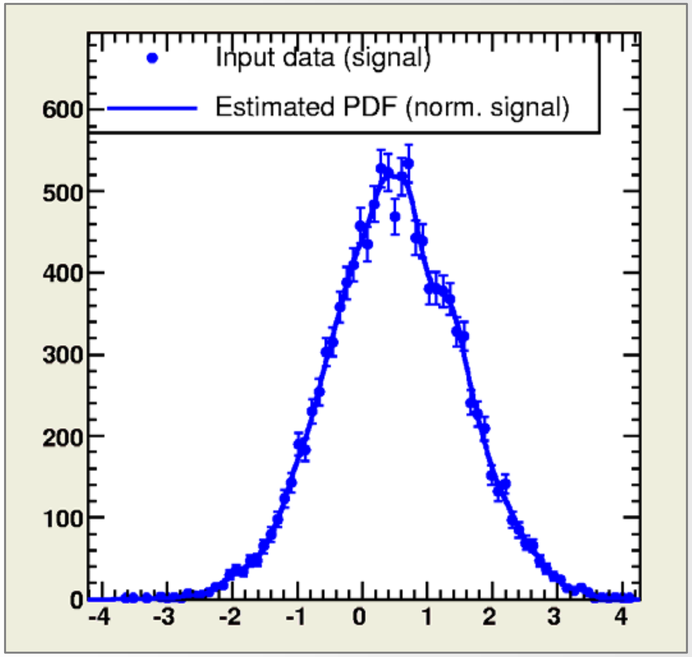
nonparametric fitting (i.e. splines,kernel)

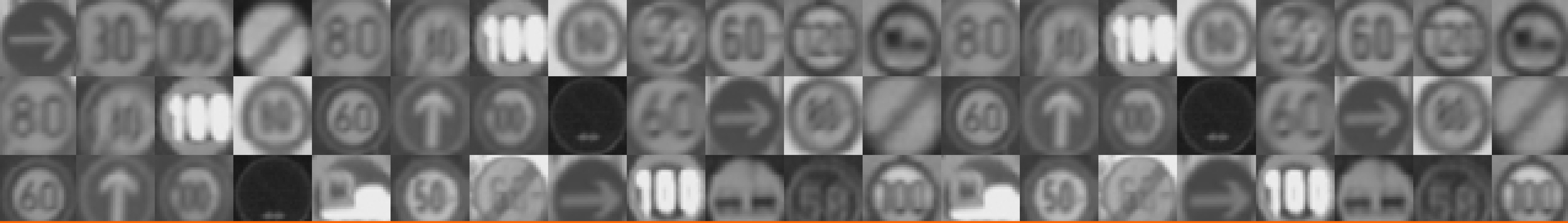
Automatic, unbiased, but suboptimal

Difficult to automate for arbitrary PDFs

Easy to automate, can create artefacts/suppress information

original distribution is Gaussian





*T*MVA-Toolkit for Multivariate Data Analysis

General Introduction

What is *T*MVA

- One framework for "all" MVA-techniques, available in ROOT
  - Have a common platform/interface for all MVA classifiers:
  - Have common data pre-processing capabilities
  - Train and test all classifiers on same data sample and evaluate consistently
  - Provide common analysis (ROOT scripts) and application framework
  - Provide access with and without ROOT, through macros, C++ executables or python
- *T*MVA is a sourceforge (SF) package for world-wide access
  - Home page ………………..http://tmva.sf.net/
  - SF project page …………. http://sf.net/projects/tmva
  - View CVS …………………http://tmva.cvs.sf.net/tmva/TMVA/
  - Mailing list ………………..http://sf.net/mail/?group_id=152074
  - Tutorial TWiki …………….https://twiki.cern.ch/twiki/bin/view/TMVA/WebHome
- Integrated and distributed with ROOT since ROOT v5.11/03

*T*MVA Content

- Currently implemented classifiers
  - Rectangular cut optimisation
  - Projective and multidimensional likelihood estimator
  - k-Nearest Neighbor algorithm
  - Fisher and H-Matrix discriminants
  - Function discriminant
  - Artificial neural networks (3 *multilayer perceptron* implementations)
  - Boosted/bagged decision trees
  - Rule Fitting
  - Support Vector Machine

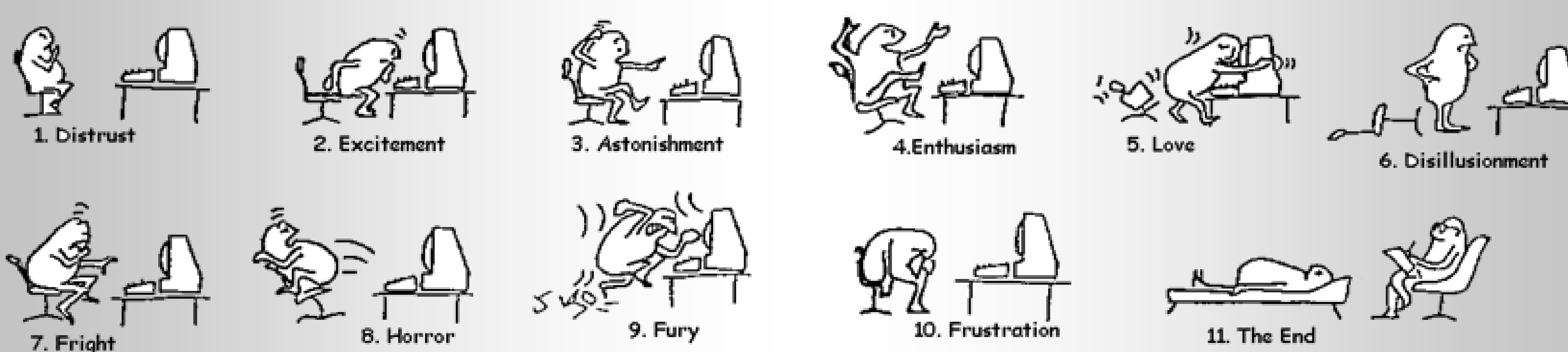
- Currently implemented data preprocessing stages:
  - Decorrelation
  - Principal Value Decomposition
  - Transformation to uniform and Gaussian distributions (*in preparation*)

Using *T*MVA

A typical *T*MVA analysis consists of two main steps:

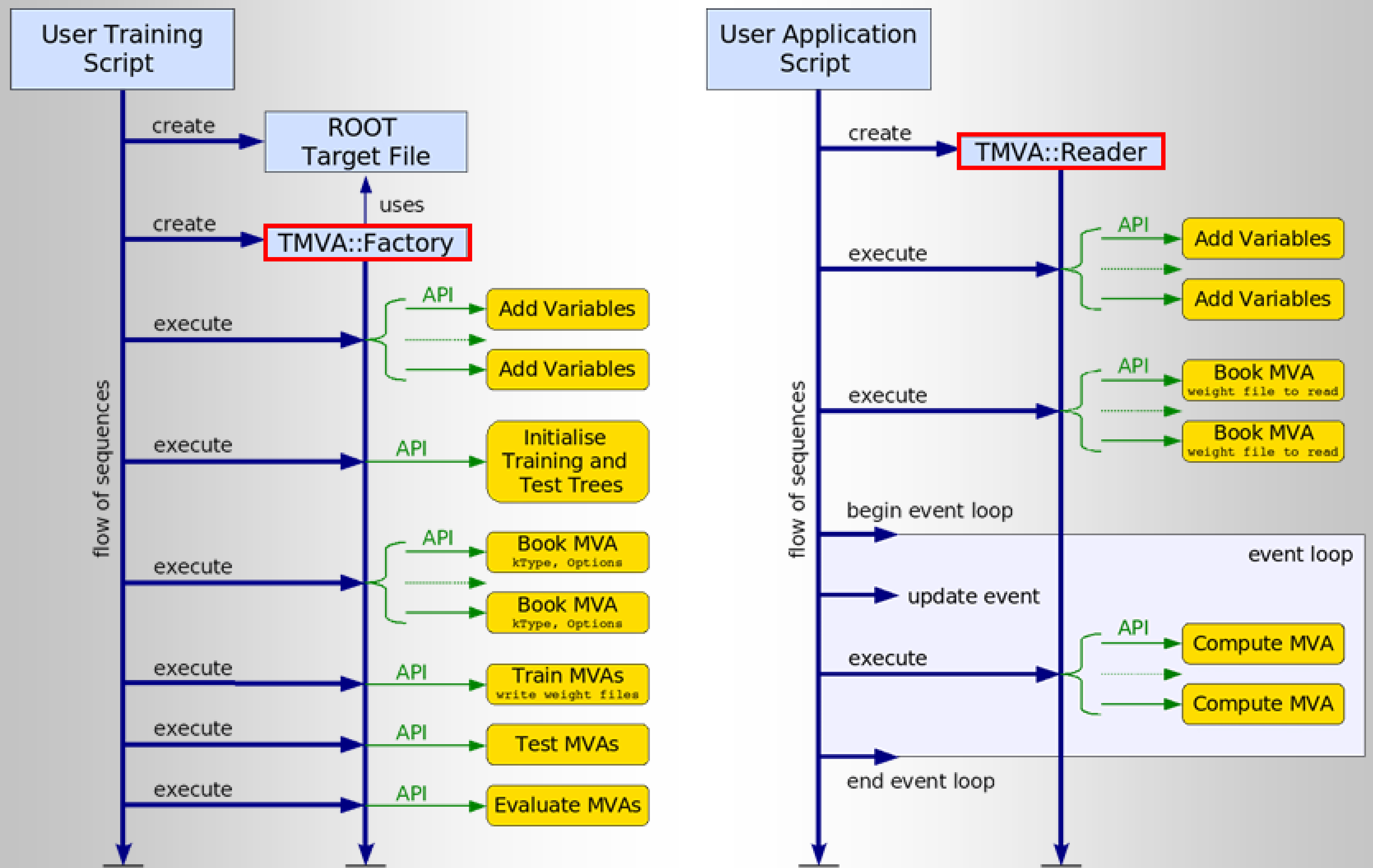
1. *Training phase*: training, testing and evaluation of classifiers using data samples with known signal and background composition
2. *Application phase*: using selected trained classifiers to classify unknown data samples

➧ Illustration of these steps with toy data samples



→ *T*MVA tutorial

Code Flow for *Training* and *Application* Phases



→ TMVA tutorial

A Simple Example for *Training*

```
void TMVAnalysis( )
{
   TFile* outputFile = TFile::Open( "TMVA.root", "RECREATE" );

   TMVA::Factory *factory = new TMVA::Factory( "MVAnalysis", outputFile,"!V");

   TFile *input = TFile::Open("tmva_example.root");

   factory->AddSignalTree         ( (TTree*)input->Get("TreeS"), 1.0 );
   factory->AddBackgroundTree ( (TTree*)input->Get("TreeB"), 1.0 );

   factory->AddVariable("var1+var2", 'F');
   factory->AddVariable("var1-var2", 'F');
   factory->AddVariable("var3", 'F');
   factory->AddVariable("var4", 'F');

   factory->PrepareTrainingAndTestTree("", "NSigTrain=3000:NBkgTrain=3000:SplitMode=Random:!V" );

   factory->BookMethod( TMVA::Types::kLikelihood, "Likelihood",
                        "!V:!TransformOutput:Spline=2:NSmooth=5:NAvEvtPerBin=50" );

   factory->BookMethod( TMVA::Types::kMLP, "MLP", "!V:NCycles=200:HiddenLayers=N+1,N:TestRate=5" );

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

   outputFile->Close();
   delete factory;
}
```

← create *Factory*

← give training/test trees

← register input variables

← select MVA methods

← train, test and evaluate

→ *TMVA tutorial*

A Simple Example for an *Application*

```
void TMVApplication( )
{
  TMVA::Reader *reader = new TMVA::Reader("!Color");

  Float_t var1, var2, var3, var4;
  reader->AddVariable( "var1+var2", &var1 );
  reader->AddVariable( "var1-var2",  &var2 );
  reader->AddVariable( "var3", &var3 );
  reader->AddVariable( "var4", &var4 );

  reader->BookMVA( "MLP classifier",  "weights/MVAnalysis_MLP.weights.txt" );

  TFile *input = TFile::Open("tmva_example.root");
  TTree* theTree = (TTree*)input->Get("TreeS");

  // … set branch addresses for user TTree
  for (Long64_t ievt=3000; ievt<theTree->GetEntries();ievt++) {
    theTree->GetEntry(ievt);

    var1 = userVar1 + userVar2;
    var2 = userVar1 - userVar2;
    var3 = userVar3;
    var4 = userVar4;

    Double_t out =  reader->EvaluateMVA(  "MLP classifier"  );

    // do something with it …
  }
  delete reader;
}
```

← create *Reader*

← register the variables

← book classifier(s)

← prepare event loop

← compute input variables

← calculate classifier output

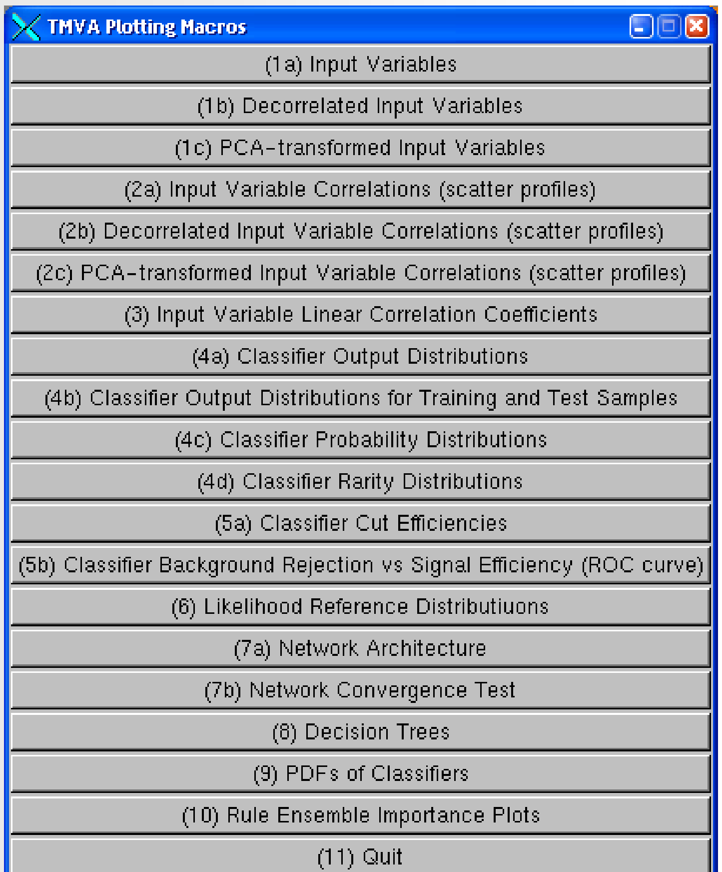
→ TMVA tutorial

Data Preparation

- Data input format: ROOT TTree or ASCII
- Selection any subset or combination or function of available variables
- Apply pre-selection cuts (possibly independent for signal and bkg)
- Define global event weights for signal or background input files
- Define individual event weight (use of any input variable present in training data)
- Choose on out of various methods for splitting into training and test samples:
  - Block wise
  - Randomly
  - Periodically (*i.e*. periodically 3 testing ev., 2 training ev., 3 testing ev, 2 training ev. ….)
  - User defined training and test trees
- Choose preprocessing of input variables (*e.g.*, decorrelation)

MVA Evaluation Framework

- TMVA is not only a collection of classifiers, but an MVA framework
- After training, TMVA provides ROOT evaluation scripts (through GUI)



Plot all signal (S) and background (B) input variables with and without pre-processing

Correlation scatters and linear coefficients for S & B

Classifier outputs (S & B) for test and training samples (spot overtraining)

Classifier *Rarity* distribution

Classifier significance with optimal cuts

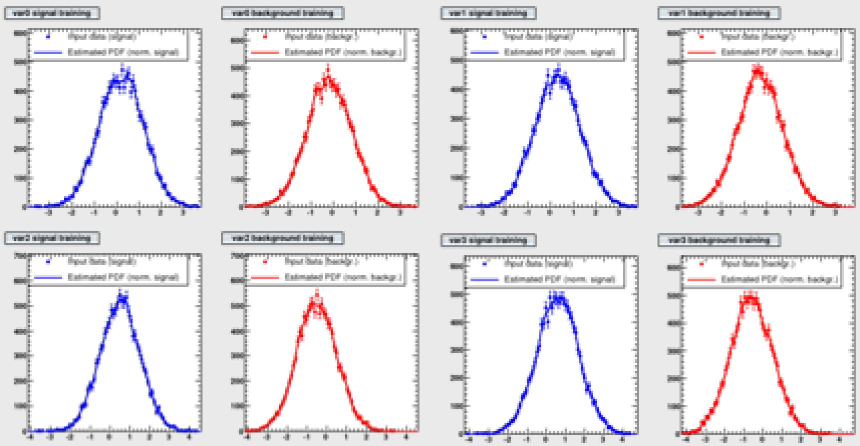
B rejection versus S efficiency

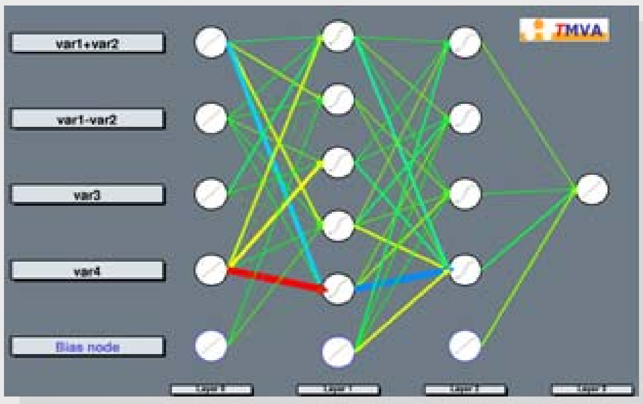
Classifier-specific plots:
- Likelihood reference distributions
- Classifier PDFs (for probability output and Rarity)
- Network architecture, weights and convergence
- Rule Fitting analysis plots

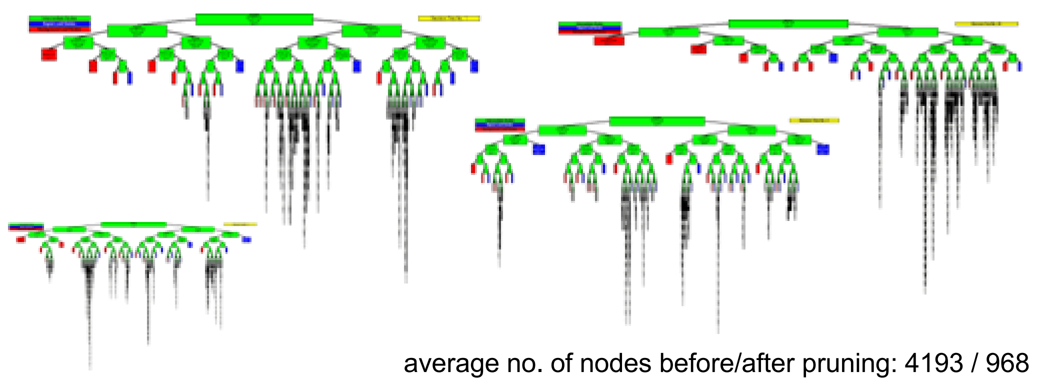
- Visualise decision trees

Evaluating the Classifier Training (I)

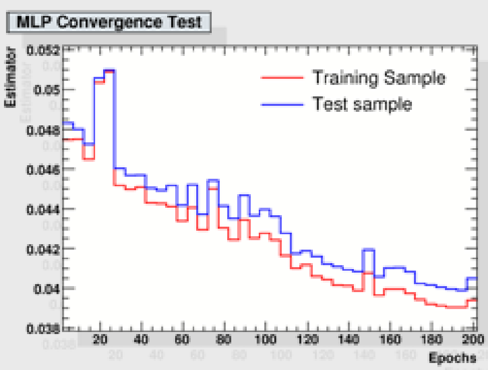
- Projective likelihood PDFs, MLP training, BDTs, …





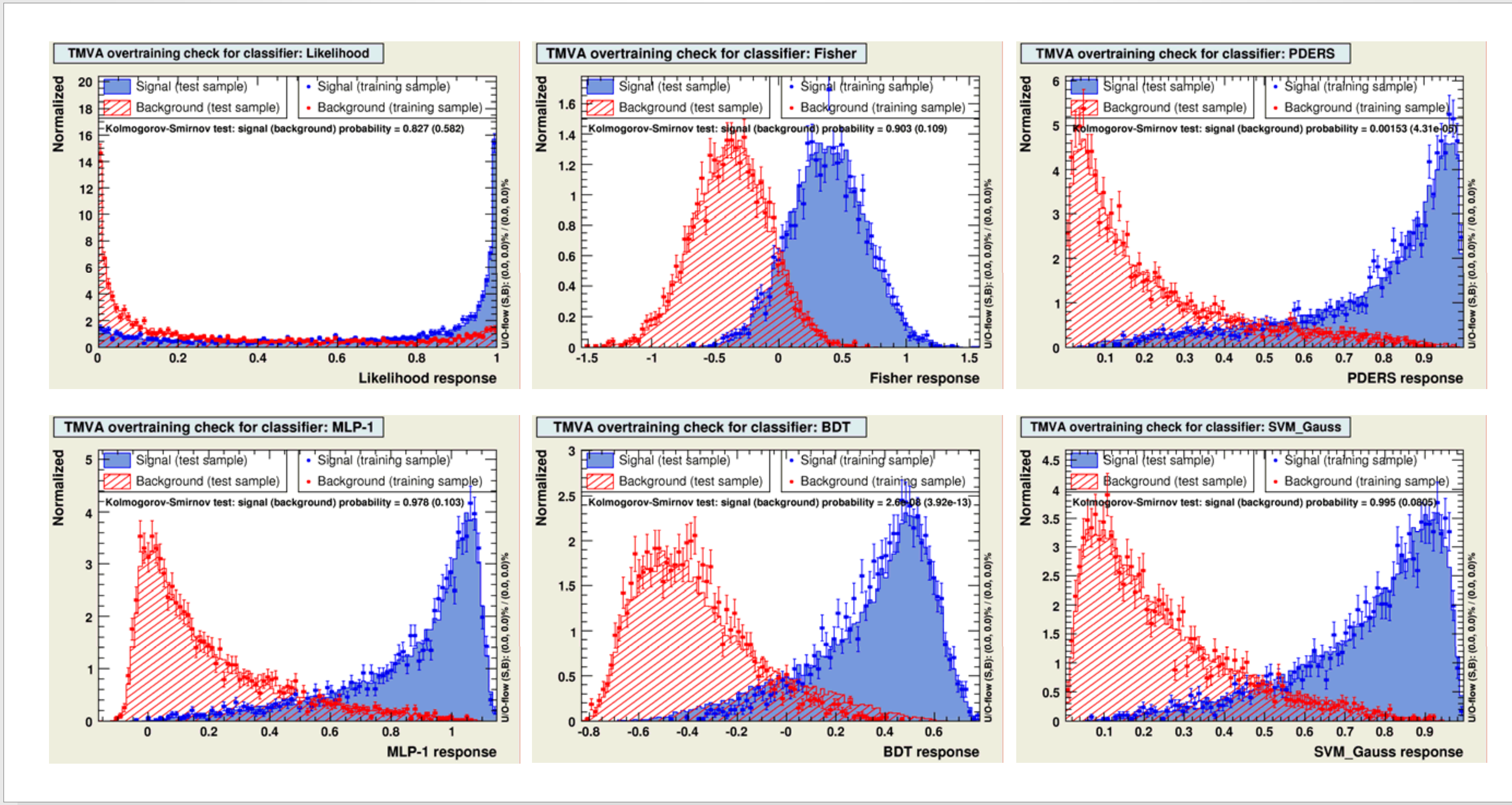


average no. of nodes before/after pruning: 4193 / 968



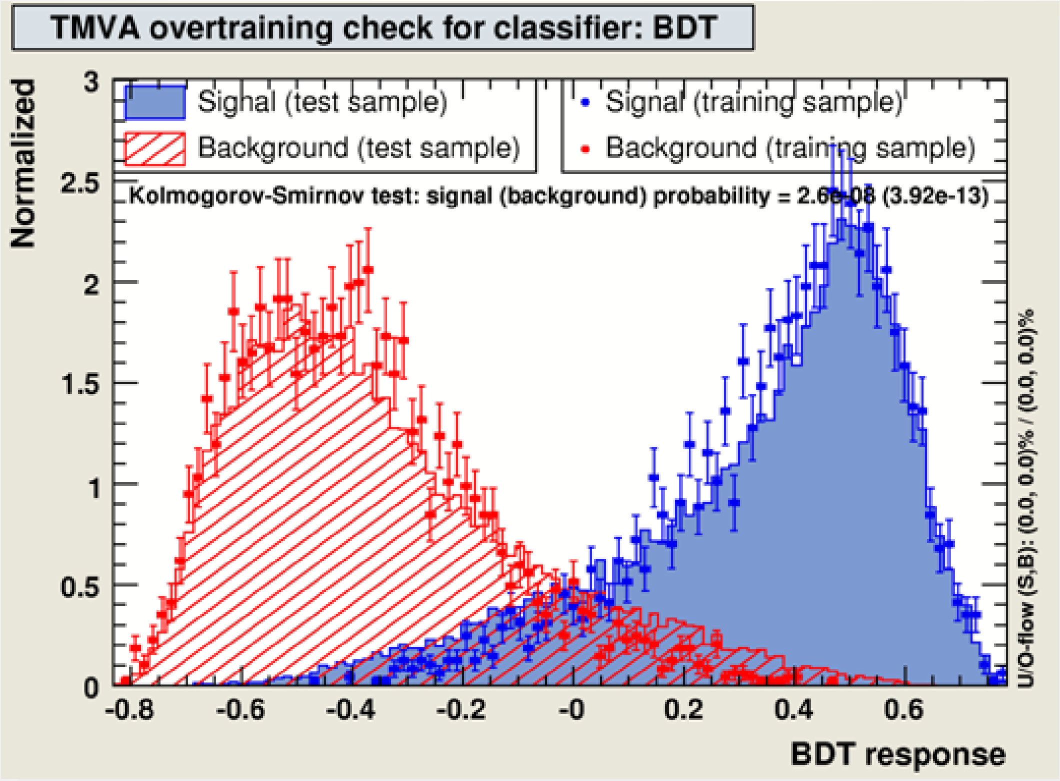
Testing the Classifiers

- Classifier output distributions for independent test sample:



Evaluating the Classifier Training

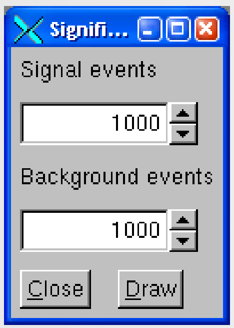
- Check for overtraining: classifier output for test *and* training samples …

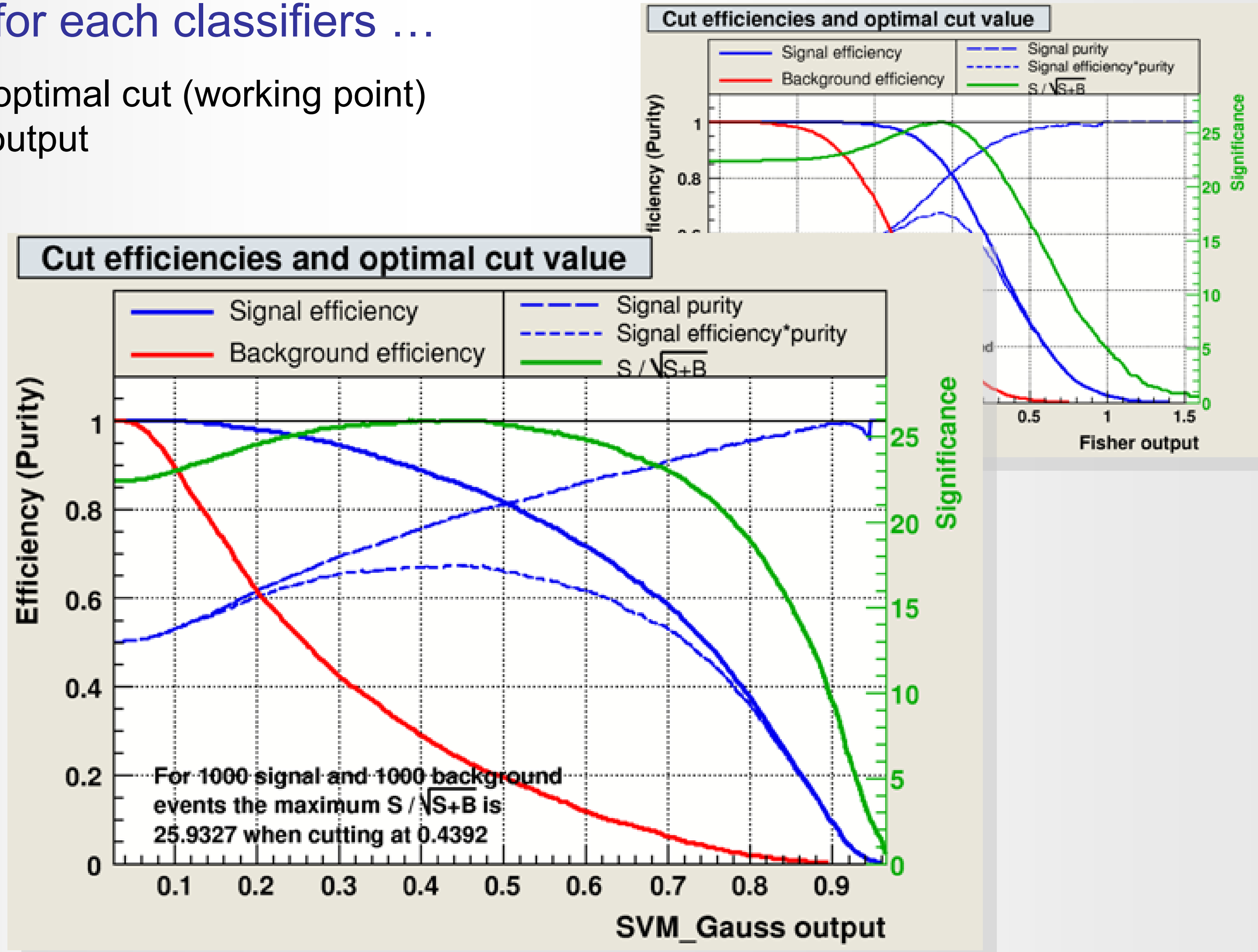


Evaluating the Classifier Training

- Optimal cut for each classifiers …

  Determine the optimal cut (working point) on a classifier output





Evaluating the Classifiers Training (taken from TMVA output…)

Input Variable Ranking

```
--- Fisher       : Ranking result (top variable is best ranked)
--- Fisher       : ---------------------------------------------
--- Fisher       : Rank : Variable  : Discr. power
--- Fisher       : ---------------------------------------------
--- Fisher       :    1 : var4      : 2.175e-01
--- Fisher       :    2 : var3      : 1.718e-01
--- Fisher       :    3 : var1      : 9.549e-02
--- Fisher       :    4 : var2      : 2.841e-02
--- Fisher       : ---------------------------------------------
```

- How discriminating is a

Classifier correlation and overlap

```
--- Factory      : Inter-MVA overlap matrix (signal):
--- Factory      : ------------------------------
--- Factory      :             Likelihood  Fisher
--- Factory      : Likelihood:     +1.000  +0.667
--- Factory      :     Fisher:     +0.667  +1.000
--- Factory      : ------------------------------
```

- Do classifiers select the same … signal and background ?
  If not, there is something

Evaluating the Classifiers Training (VII) (taken from TMVA output…)

Evaluation results ranked by best signal efficiency and purity (area)

Better classifier

MVA Methods:	Signal efficiency at bkg eff. (error): @B=0.01	@B=0.10	@B=0.30	Area	Separation:	Significance:
Fisher	0.268(03)	0.653(03)	0.873(02)	0.882	0.444	1.189
MLP	0.266(03)	0.656(03)	0.873(02)	0.882	0.444	1.260
LikelihoodD	0.259(03)	0.649(03)	0.871(02)	0.880	0.441	1.251
PDERS	0.223(03)	0.628(03)	0.861(02)	0.870	0.417	1.192
RuleFit	0.196(03)	0.607(03)	0.845(02)	0.859	0.390	1.092
HMatrix	0.058(01)	0.622(03)	0.868(02)	0.855	0.410	1.093
BDT	0.154(02)	0.594(04)	0.838(03)	0.852	0.380	1.099
CutsGA	0.109(02)	1.000(00)	0.717(03)	0.784	0.000	0.000
Likelihood	0.086(02)	0.387(03)	0.677(03)	0.757	0.199	0.682

Testing efficiency compared to training efficiency (overtraining check)

Check for over-training

MVA Methods:	Signal efficiency: from test sample (from traing sample) @B=0.01	@B=0.10	@B=0.30
Fisher	0.268 (0.275)	0.653 (0.658)	0.873 (0.873)
MLP	0.266 (0.278)	0.656 (0.658)	0.873 (0.873)
LikelihoodD	0.259 (0.273)	0.649 (0.657)	0.871 (0.872)
PDERS	0.223 (0.389)	0.628 (0.691)	0.861 (0.881)
RuleFit	0.196 (0.198)	0.607 (0.616)	0.845 (0.848)
HMatrix	0.058 (0.060)	0.622 (0.623)	0.868 (0.868)
BDT	0.154 (0.268)	0.594 (0.736)	0.838 (0.911)
CutsGA	0.109 (0.123)	1.000 (0.424)	0.717 (0.715)
Likelihood	0.086 (0.092)	0.387 (0.379)	0.677 (0.677)

Receiver Operating Characteristics (ROC) Curve

- Smooth background rejection versus signal efficiency curve:
  (from cut on classifier output)