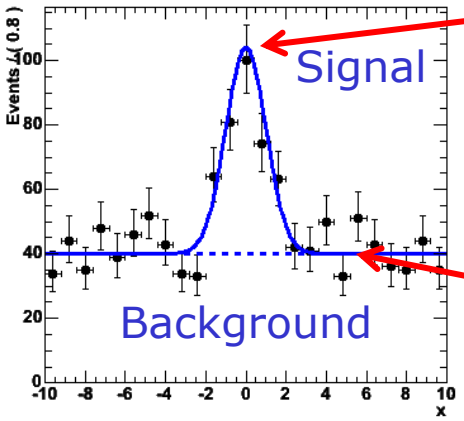
Event classification

— Comparing discriminating variables
— Choosing the optimal cut
— Working in more than one dimension
— Approximating the optimal discriminant
— Techniques: Principal component analysis,
Fisher Discriminant, Neural Network,
Probability Density Estimate, Empirical Modeling

Introduction to event classification

- Most HEP analysis involve classification of events in 'signal' and 'background'
  - Statistics connection : *Hypothesis testing*
  - Determine consistency of each event with a signal and background hypothesis
  - No certain answer on signal-vs-background classification for each event, but rather a *probability*



P(sig)=50%

Signal hypothesis
Gaussian distribution in x

P(sig)=0%

Background hypothesis
Flat distribution in x

  - In simple cases like above example no need to make cut on signal probability
  - Go straight to parameter estimation: what is # of signal events?

Multivariate event classification

- Many HEP problem are much more difficult than preceding example
  - Many observables, overwhelming background.
  - Various approaches possible
- Analyses perform various combination of two techniques
  - Selection of events (throw out events that are not very much signal-like)
  - Parameter estimation (assign signal probability to each event → determine total number of signal events in sample)
- "Cut and count" (or fit)
  - First make preselection of events that are most 'signal-like'
  - Then do parameter estimation or simple counting on those events
- "Compactification"
  - Compactify information in multiple observables into one observable (e.g. output of neural network) → 'test statistic t(x)'
  - Do parameter estimation on compactify representation
- "Big fit"
  - Make explicit model of signal and background hypothesis in all observables and fit all data to estimate model parameters (#signal events, Higgs mass etc…)

Merits of various approaches

- Statistical sensitivity
  - Cutting usually throws away some information.
  - Compactification can loose some information, but not necessarily
  - Big fit keeps all information
- Feasibility
  - Cutting usually easiest
  - Compactification. Recent developments in machine learning make this a lot easier
  - Big fit clearly most ambitious
- What is the best you can do? Best (this context):

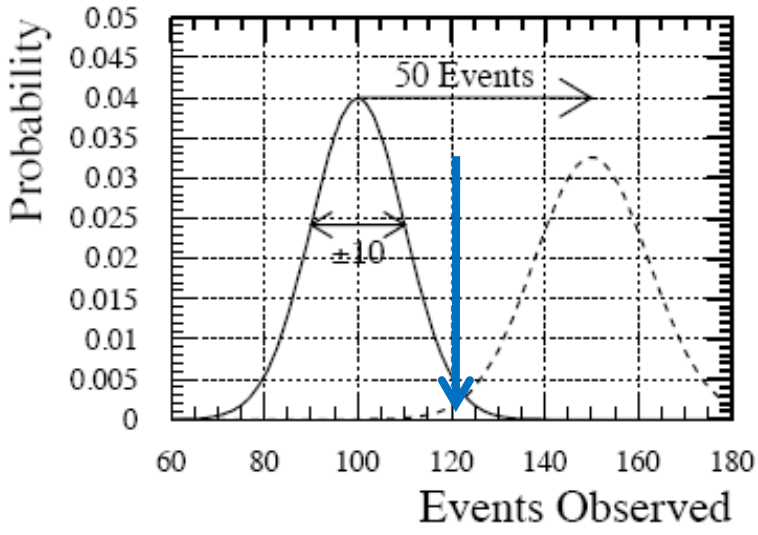
  Smallest stat. error on measurement

  - Equivalent statement for a selection cut:

    lowest possible bkg. efficiency at a given sig. efficiency

Hypothesis testing

- Introduce some terminology of hypothesis testing
- Assumed one has a model for data under two hypotheses
  - Null hypothesis (H_0) = Background only
  - Alternate hypotheses (H_1) = e.g. Signal + Background
- One makes a measurement and then need to decide to accept or reject H_0



Hypothesis testing

- Definition of terms
  - Rate of type-I error = α
  - Rate of type-II error = β
  - Power of test is 1-β

		Actual condition	
		Guilty	**Not guilty**
Decision	**Verdict of 'guilty'**	True Positive	False Positive (i.e. guilt reported unfairly) **Type I error**
	Verdict of 'not guilty'	False Negative (i.e. guilt not detected) **Type II error**	True Negative

- Treat hypotheses asymmetrically
  - Null hypo is special → Fix rate of type-I error
- Now can define a well stated goal
  - Maximize the power of test (minimized rate of type-II error) for given α

The Neyman-Pearson lemma

- In 1932-1938 Neyman and Pearson developed in which one must consider competing hypotheses
  - Null hypothesis (H_0) = Background only
  - Alternate hypotheses (H_1) = e.g. Signal + Background
- The region W that minimizes the rate of the type-II error (not reporting true discovery) is a contour of the Likelihood Ratio

$$\frac{P(x|H_1)}{P(x|H_0)} > k_\alpha$$

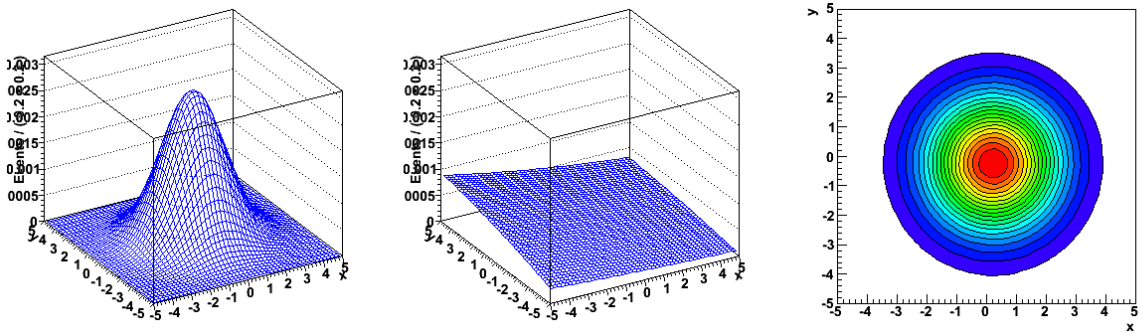
- Any other region of the same size will have less power

What is the best you can do?

- Neyman-Pearson lemma:
  - For a problem described by a continuous signal distribution f(x|s) and a continuous bkg distribution f(x|b) the optimal acceptance region is defined by

$$\frac{f(\vec{x}|\mathrm{s})}{f(\vec{x}|\mathrm{b})} > c$$

$f(\vec{x}|\mathrm{s})$ $f(\vec{x}|\mathrm{b})$ $\frac{f(\vec{x}|\mathrm{s})}{f(\vec{x}|\mathrm{b})} > c$



  - Note that for continuous distributions you *cannot always achieve perfect separation*

- Translation for 3 approaches
  - Cut and count: Optimal cut is defined by surface S(x)/B(x)>α
  - Compactification: Optimal 1-D test statistics t(x) = S(x)/B(x)
  - Big fit: Optimal fit is when model M(x)=$N_S \cdot S(x) + N_B \cdot B(x)$

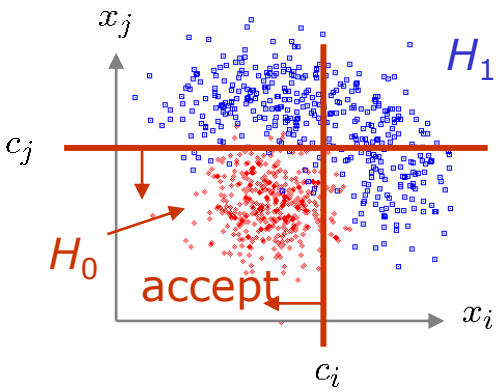
Why Neyman-Pearson doesn't always help

- The problem is that we usually don't have explicit formulae for the pdfs $f(\vec{x}|\mathrm{s}),\ f(\vec{x}|\mathrm{b})$.
  - Instead we may have Monte Carlo models for signal and background processes, so we can produce simulated data, and enter each event into an *n*-dimensional histogram. Use e.g. *M* bins for each of the *n* dimensions, total of M^n cells.
- But *n* is potentially large → prohibitively large number of cells to populate with Monte Carlo data.
  - Compromise: make *Ansatz* for form of **test statistic** $t(\vec{x})$ with fewer parameters; determine them (e.g. using MC) to give best discrimination between signal and background.

Using test statistics to describe the selection

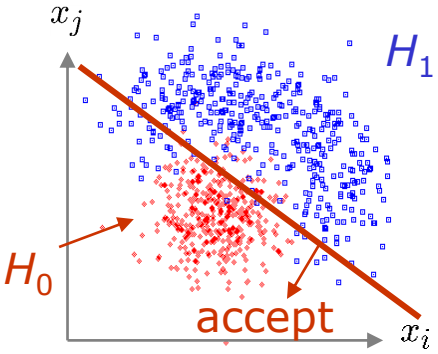
- All decision boundaries ('cuts') can be expressed as $t(x) < \alpha$ where *t(x)* is a *test statistic* and α is a parameter
- *Goal* is a test statistic as close as possible to t(x)=S(x)/D(x)

Rectangular cut



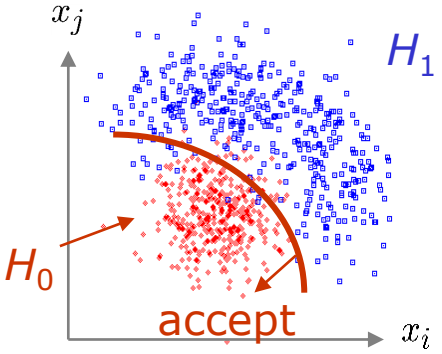
$$t(x) = \theta(x_j - c_j)\theta(x_i - c_i)$$

Linear cut



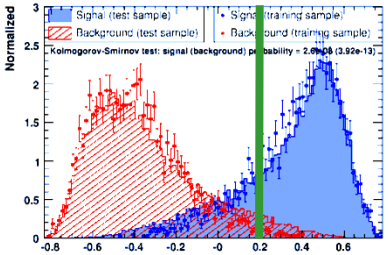
$$t(x) = a_j \cdot x_j + a_i \cdot x_i$$

Non-linear cut



$$t(x) = \vec{a} \cdot \vec{x} + \vec{x} A \vec{x} + \ldots$$

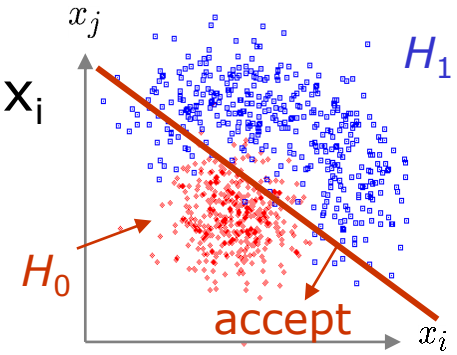
- Two separate questions
  - What is the optimal form of t(x) – *Independent of ratio N_{sig}/N_{bkg}*
  - What is the best value of α – *Depends on ratio N_{sig}/N_{bkg}*



Constructing test statistics – Linear discriminants

- A linear discriminant constructs t(x) from a linear combination of the variables x_i

$$t(\vec{x}) = \sum_{i=1}^{N} a_i x_i = \vec{a} \cdot \vec{x}$$



  - Optimize discriminant by chosing a_i to maximize separation between signal and background

- Most common form of the linear discriminant is the Fisher discriminant

$$F(\vec{x}) = \overbrace{\left(\vec{\mu}_S - \vec{\mu}_B\right)^T V^{-1}}^{\vec{a}} \vec{x}$$

Mean values in x_i for sig,bkg

Inverse of variance matrix of signal/background (assumed to be the same)

R.A. Fisher
Ann. Eugen. 7(1936) 179.

Ansatz test statistics – The Fisher discriminant

$$F(\vec{x}) = \underbrace{\left(\vec{\mu}_S - \vec{\mu}_B\right)^T V^{-1}}_{\vec{a}} \vec{x}$$

R.A. Fisher
Ann. Eugen. 7(1936) 179.

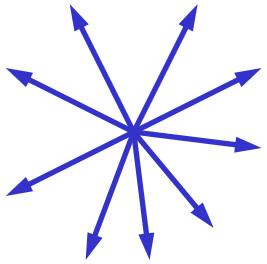
Mean values in x_i for sig,bkg

Inverse of variance matrix of signal/background (assumed to be the same)

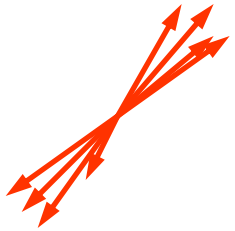
- Advantage of Fisher Discriminant:
  - Ingredients $\mu_s, \mu_b, \mathbf{V}$ can all be calculated directly from data or simulation samples. No 'training' or 'tuning'
- Disadvantages of Fisher Discriminant
  - Fisher discriminant only exploits difference in means.
  - If signal and background have different variance, this information is not used.

Example of Fisher discriminant

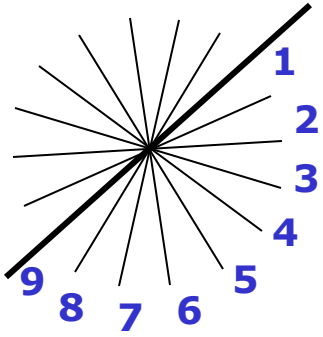
- The "CLEO" Fisher discriminant
  - Goal: distinguish between e+e- → Y4s → bb and uu,dd,ss,cc
  - Method: Measure energy flow in 9 concentric cones around direction of B candidate



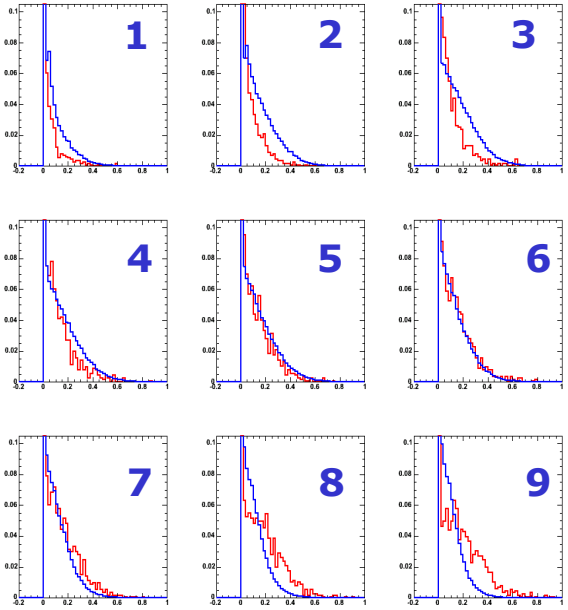
Energy flow in bb



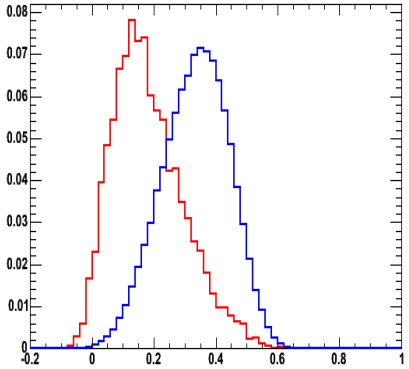
Energy flow in u,d,s,c



Cone Energy flows



F(x)



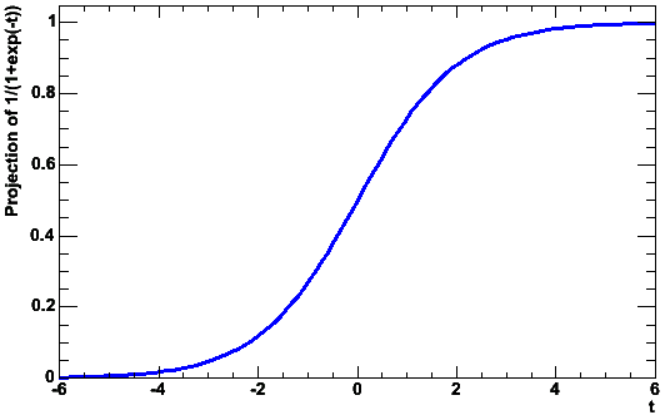
When is Fisher discriminant is the optimal discriminant?

- A very simple dataset

$$S = \prod_i Gauss(x_i; \mu_i^S, \sigma_i)$$
$$B = \prod_i Gauss(x_i; \mu_i^B, \sigma_i)$$

Multivariate Gaussian distributions with **different means** but **same width** for signal and background

- Fisher is optimal discriminant for this case
  - In this case we can also directly correlate F(x) to **absolute signal probability**

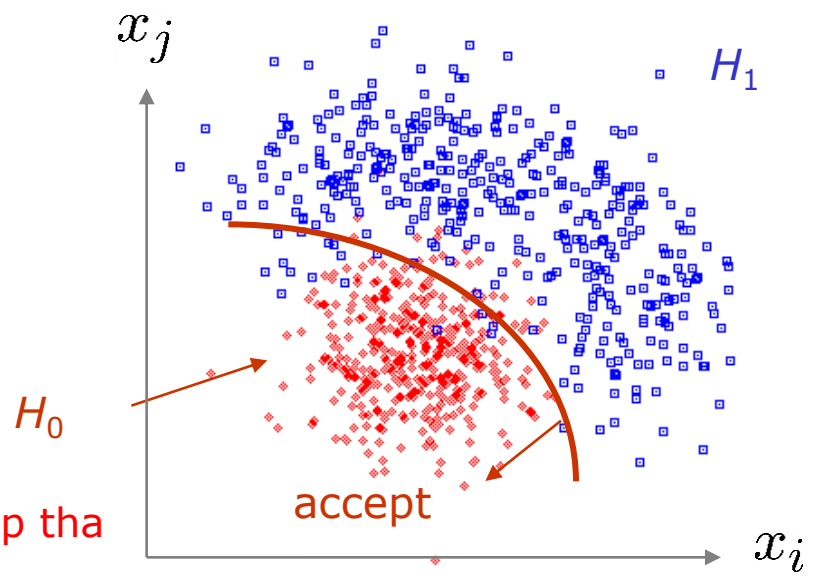


$$P(F) = \frac{1}{1+e^{-F}}$$

'Logistic sigmoid function'

Non-linear test statistics

- The optimal decision boundary may not be a hyperplane → non-linear test statistic
- Large variety of Ansatzes
  - Neural network
  - Support Vector Machines
  - Rule ensembles
  - Decision Trees
  - Kernel density estimates
  - Most of these test statistics *t(x,p)* have many free parameters p tha maximize their performance



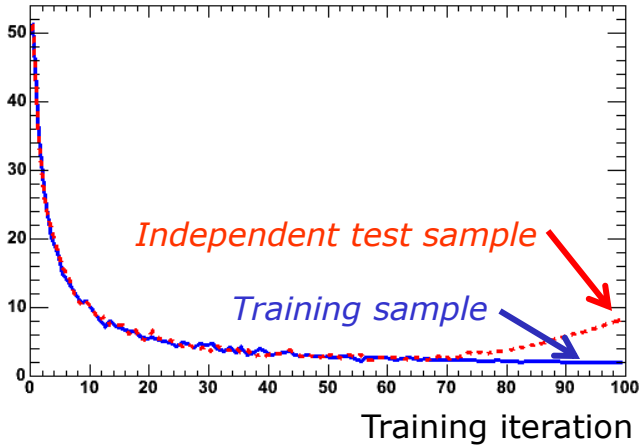
- Unlike Fisher Discriminant, no analytical calculation possible → Computational solution
  - Machine Learning
  - Bayesian Learning

Common approximations made in machine learning

- Machine learning approach most popular for training multivariate non-linear test statistics
- *Approximation of knowledge* of true signal and background distributions with *sample* of signal and background events
  - Finite statistics limit precision (in itself usually not a problem)

- Risks of overtraining
  - *For finite datasets it is always possible to construct a perfect discriminant*

    (i.e. better than Neyman-Pearson optimum for true signal and bkg)
    - E.g. tiny rectangular box cuts around each signal events will do the job
  - Control overtraining by measuring performance of test statistic on independent 'test sample'.



- *Never train on data*! (unless control sample)
  - Overtraining on data → Better selection efficiency than on simulated events that are used to calculate efficiency → Bias towards positive fluctuations

Multivariate data selection – Neural networks

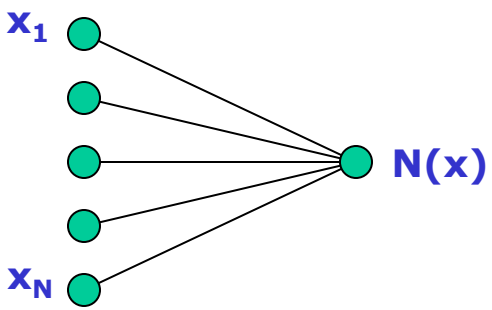
- Neural networks are used in neurobiology, pattern recognition, financial forecasting (and also HEP)

$$N(\vec{x}) = s\left(a_0 + \sum_i a_i x_i\right)$$

s(t) is the *activation function*, usually a logistic sigmoid

$$s(t) = \frac{1}{1+e^{-t}}$$

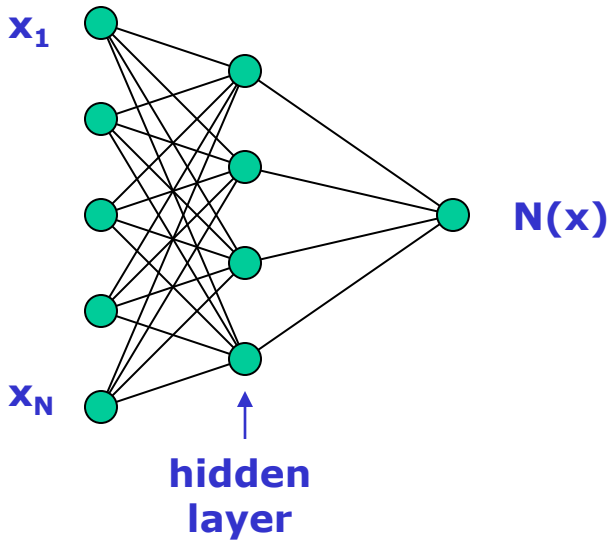
- This formula corresponds to the 'single layer perceptron'
  - Visualization of single layer network topology



Since activation function s(t) is monotonic, **the single layer N(x) is equivalent to the Fisher discriminant F(x)**

Neural networks – general structure

- The single layer model and easily be generalized to a ***multilayer*** perceptron



$$N(\vec{x}) = s(a_0 + \sum_{i=1,}^{m} a_i h_i(\vec{x}))$$

$$with\ h_i(\vec{x}) = s(w_{i0} + \sum_{j=1}^{n} w_{ij} x_j)$$

with a_i and w_{ij} weights (connection strengths)

- Easy to generalize to arbitrary number of layers
- Feed-forward net: values of a node depend only on earlier layers (usually only on preceding layer) 'the network architecture'
- More nodes bring N(x) close to optimal t(x)=S(x)/B(x) but with much more parameters to be determined

Neural networks – Training

- Parameters of NN usually determined by minimizing the error function

$$\varepsilon = \int \left(N(\vec{x}) - 0\right)^2 B(\vec{x}) d\vec{x} \quad + \quad \int \left(N(\vec{x}) - 1\right)^2 S(\vec{x}) d\vec{x}$$

NN target value for background

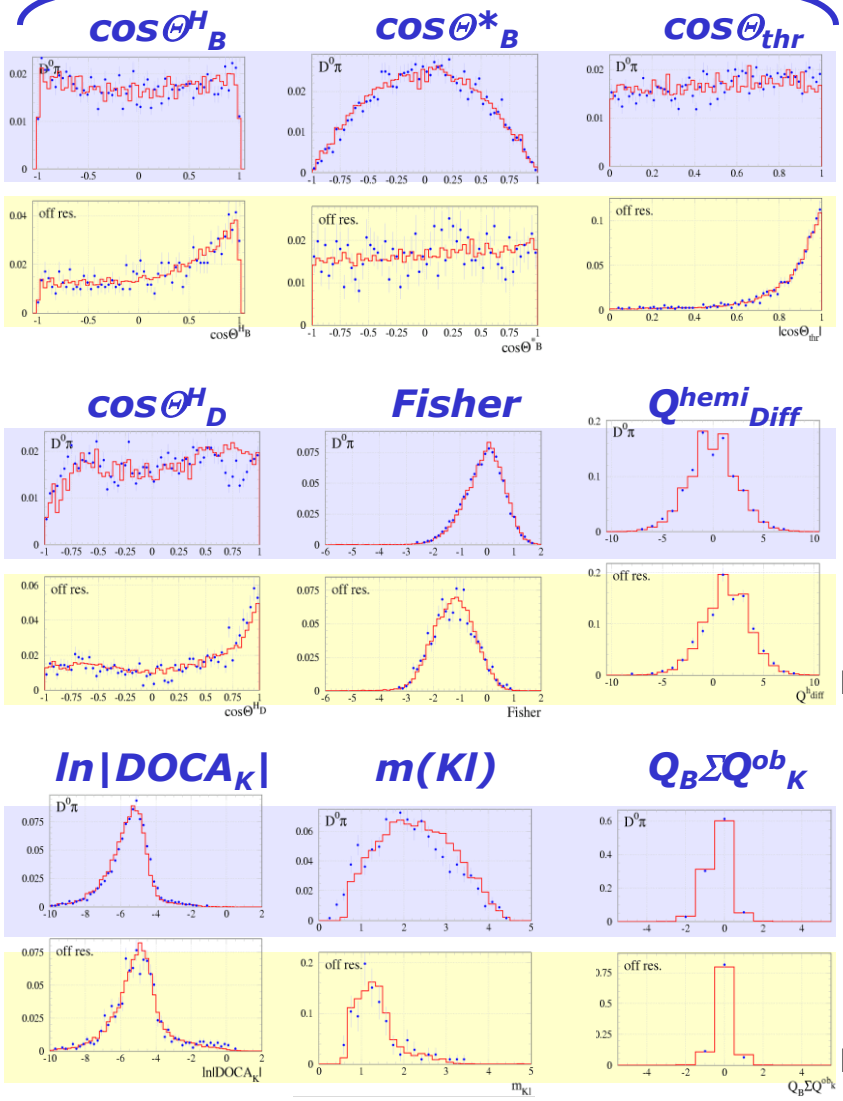
NN target value for signal

- Same principle as Fisher discriminant, but cannot solve analytically for general case
  - In practice replace ε with averages from training data from MC (Adjusting parameters → 'Learning')
  - Generally difficult, but many programs exist to do this for you ('error back propagation' technique most common)

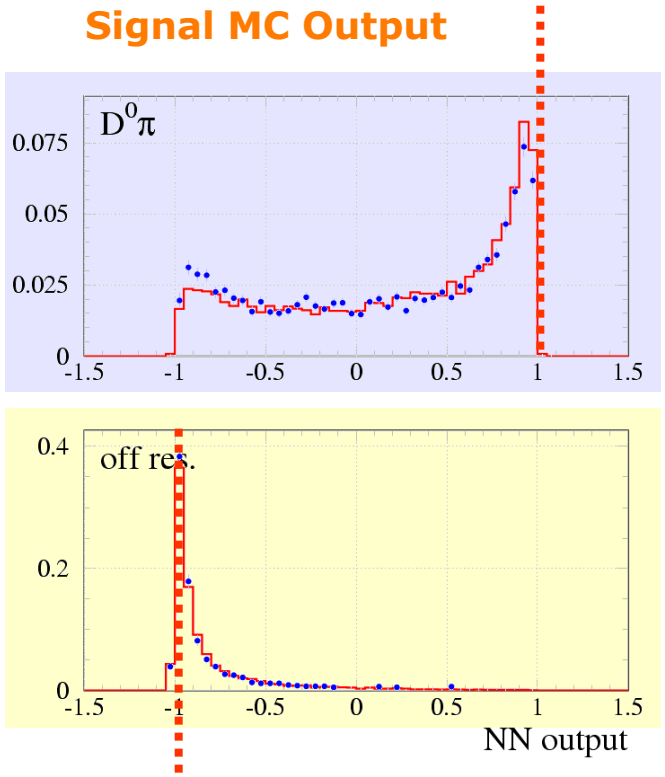
Neural networks – training example

Input Variables (9)

Output Variables (1)



N(x)



Practical aspects of Neural Net training

- **Choose input variables sensibly**
  - Don't include badly understood observables (such as #tracks/evt)
  - Some input variables may be highly correlated → drop or decorrelate
  - Some input variables may contain little or no discriminating power → drop them
  - Transform strongly peaked distributions into smooth ones (rarity transforms / Gaussianization)
  - Fewer inputs → fewer parameters to be adjusted → parameters better determined for finite training data

- **Choose architecture sensibly**
  - No 'rules' for number of hidden layers, nodes
  - Usually better to start simple and gradually increase compexity and see how that pays off

- **Verify sensible behavior**
  - NN are not magic, understand what your trained NN is doing

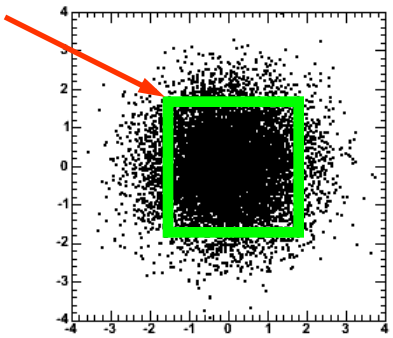
Improving performance of non-linear test statistics

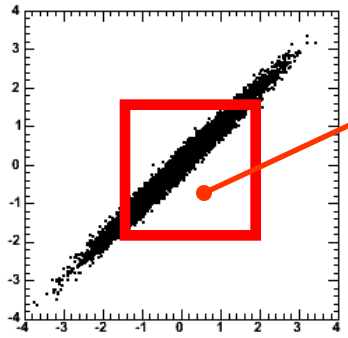
- Strong correlations may adversely impact training and performance of non-linear test statistics
  - Extreme example with rectangular cut optimization, but other algorithm are also affected to lesser degree

Scenario with uncorrelated X,Y in sig,bkg

Scenario with strongly cor-related X,Y in sig

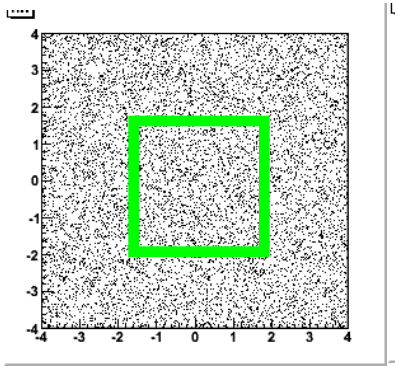
Signal

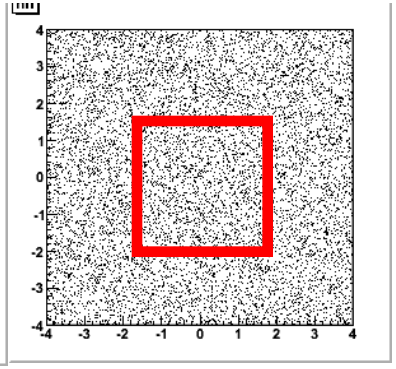




Additional background could have been reduced at no cost with a differently shaped cut

Background



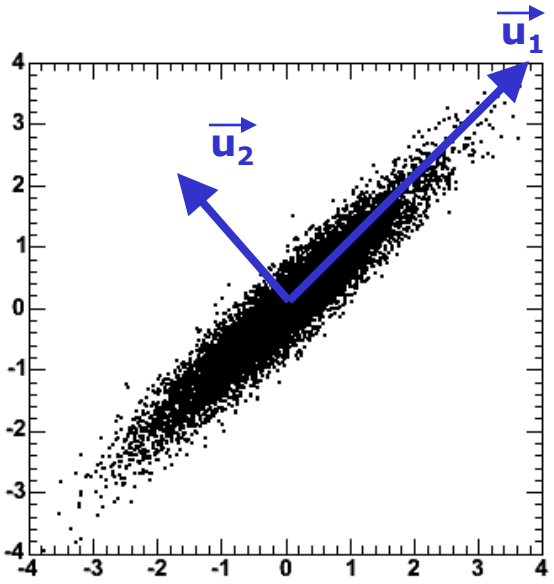


Need different approach…

Decorrelation of input variables – two ways

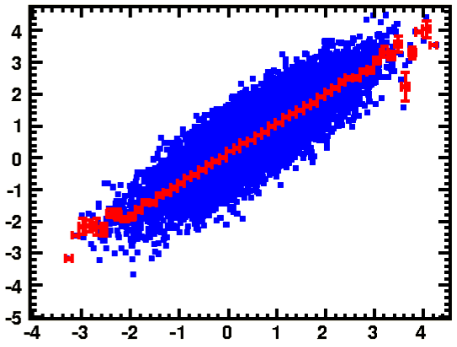
- Removal of linear correlations by rotating input variables
  - Cholesky decomposition: determine *square-root* C' of covariance matrix C, *i.e.*, $C = C'C'$
  - Transform orig (x) into decorrelated variable space (x') by: $x' = C'^{-1}x$

- Principal component analysis
  1) Compute variance matrix **Cov(X)**
  2) Compute eigenvalues λ_i and eigenvectors $\mathbf{v_i}$
  3) Construct rotation matrix $\mathbf{T = Col(v_i)^T}$
  4) Finally calculate $\mathbf{u_i = Tx_i}$

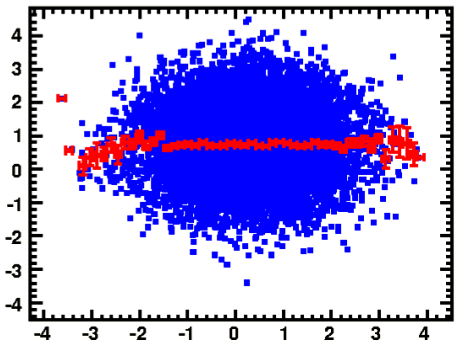


Example of decorrelation

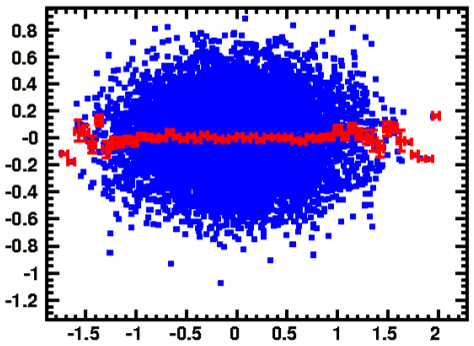
Original



Cholesky (√C)



Principal Comp. Ana.



- Note that decorrelation is only complete, if
  - Correlations are linear
  - Input variables are Gaussian distributed
  - Not very accurate conjecture in general (but valid for above example)
- Can decorrelate signal or background, not both

Gaussianization

- Decorrelation can be improved by applying a transformation to each observable that results in a Gaussian distribution
  - Can Gaussianize either signal or background sample (not both…)
- Two-step transformation: First apply probability integral transform: Given continuous x ∈(a,b), and its pdf p(x), let

$$y(x) = \int_a^x p(x')\,dx'.$$

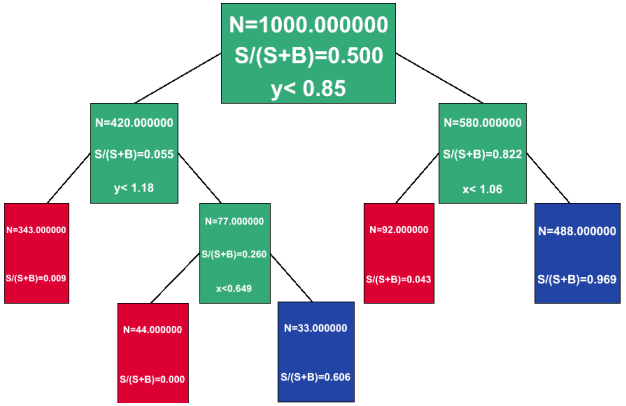
- Then y ∈(0,1) and p(y) = 1 (uniform) for all y. (!)
- Next apply Gaussian transform using inverse error function

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

$$x_k^{\mathrm{Gauss}}(i_{\mathrm{event}}) = \sqrt{2} \cdot \mathrm{erf}^{-1}\left(2x_k^{\mathrm{flat}}(i_{\mathrm{event}}) - 1\right) \quad , \forall k \in \{\mathrm{variables}\}$$

Decision Trees

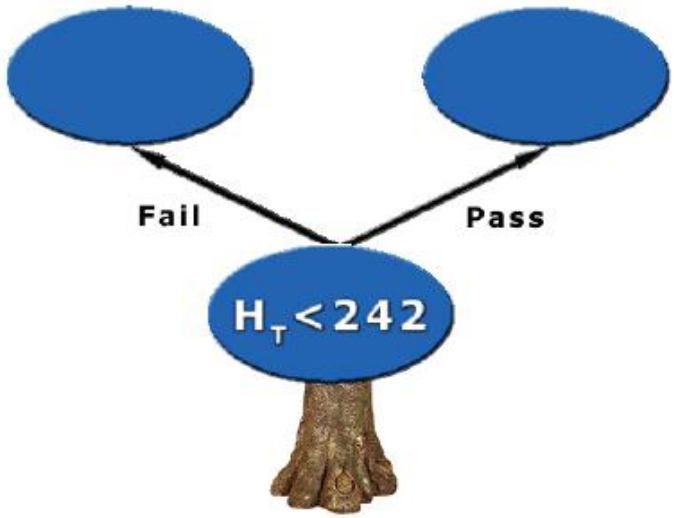
- A Decision Tree encodes sequential rectangular cuts
  - But with a lot of underlying theory on training and optimization
  - Machine-learning technique, widely used in social sciences
  - L. Breiman et al., "Classification and Regression Trees" (1984)



- Basic principle
  - Extend cut-based selection
  - Try not to rule out events failing a particular criterion
  - Keep events rejected by one criterion and see whether other criteria could help classify them properly

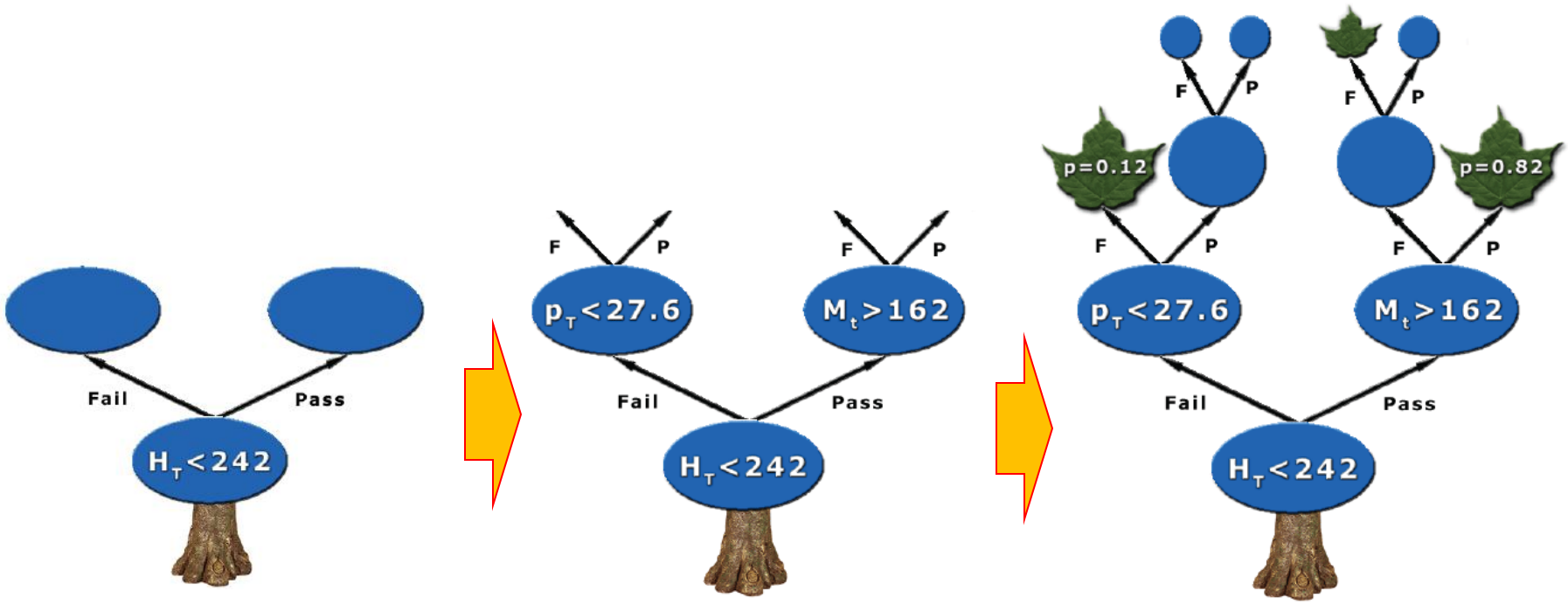
Building a tree – splitting the data

- Essential operation : splitting the data in 2 groups using a single cut, e.g. $H_T<242$
  - Need to find 'best cut' as quantified through best separation of signal and background
    i.e. most signal passes, most background fails
    (requires some metric to quantify this)
  - Procedure:
    1) Find cut value with best separation for *each* observable
    2) Apply **only** cut on observable that results in best separation



Building a tree – recursive splitting

- Repeat splitting procedure on sub-samples of previous split
- Requires algorithm to decide when to *stop* splitting



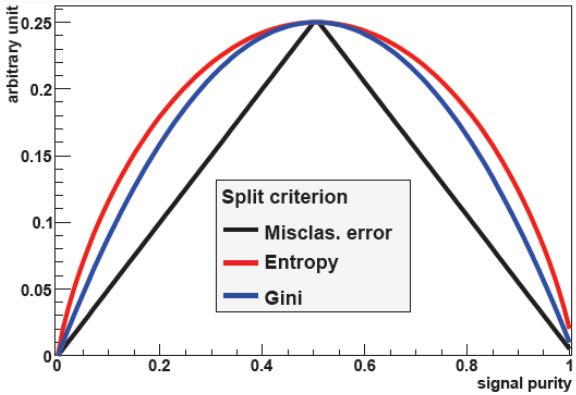
- Output of decision tree:
  - true/false or
  - probability based on expected purity of leaf (s/s+b)

Parameters in the construction of a decision tree

- Normalization of signal and background before training
  - Usually *same total weight* for signal and background events
- In the selection of splits
  - list of questions ($var_i < cut_i$) to consider
  - Separation metric (quantifies how good the split is)
- Decision to stop splitting (declare a node terminal)
  - Minimum leaf size (e.g. 100 events)
  - Insufficient improvement from splitting
  - Perfect classification (all events in leaf belong to same class)
- Assignment of terminal node to a class
  - Usually: purity>0.5 = signal, purity<0.5 = background

Separation metric – The impurity function

- Introduce impurity function *i(t)* to quantify (im)purity of a sample
  - maximum impurity for equal mix of S and B
  - Simplest option: i(t) = misclassification rate
  - strictly concave → reward purer nodes
    - favors end cuts with one smaller node and one larger node



- Definition of optimal split – Minimize *i(t)* on output samples of split
  - Metric = decrease of impurity (increase of purity)
  - For a split *s* of a node $t \rightarrow t_L, t_R$

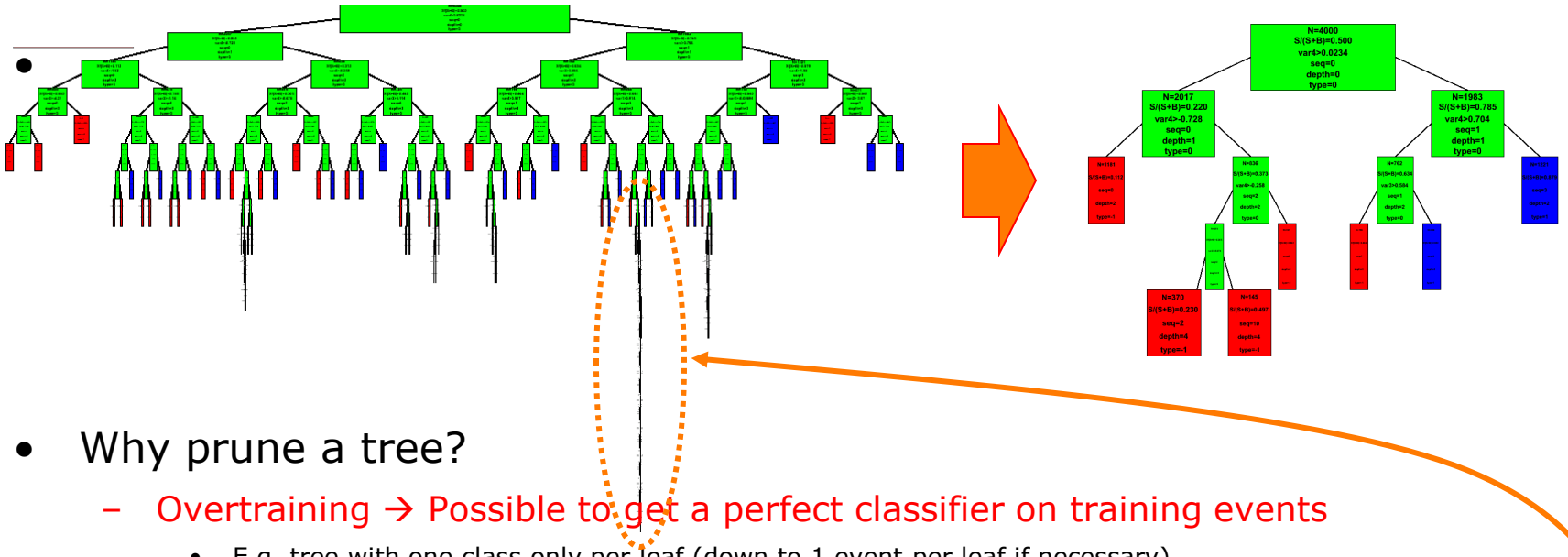
$$\Delta i(s,t) = i(t) - p_L \cdot i(t_L) - p_R \cdot i(t_R)$$

Impurity of sample before split — Impurity of 'left' sample — Impurity of 'right' sample

Aim: find split that results in largest Δi(s,t)

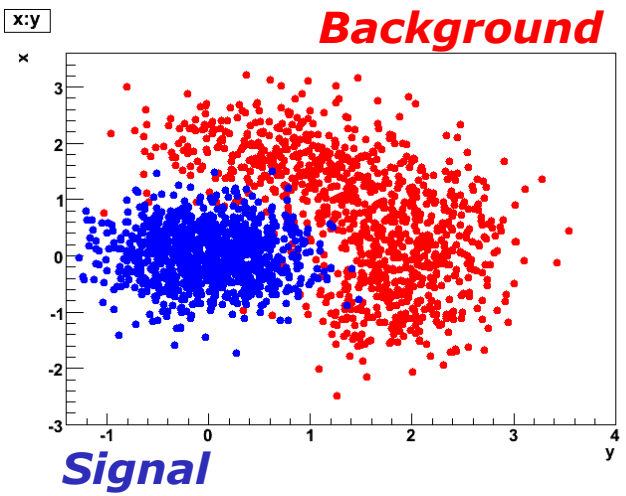
- Stop splitting when
  - not enough improvement (introduce a cutoff parameter β on Δi)
  - not enough statistics in sample
  - node is pure (signal or background)

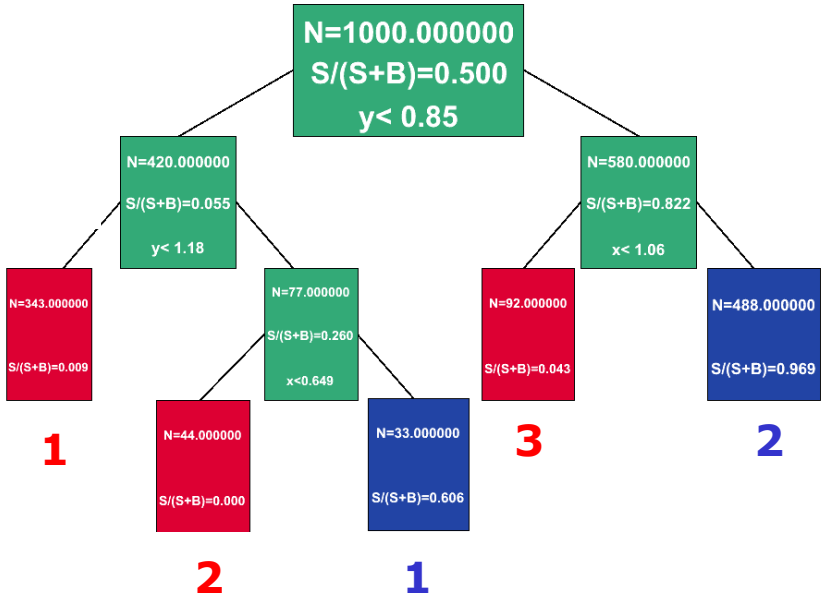
Pruning

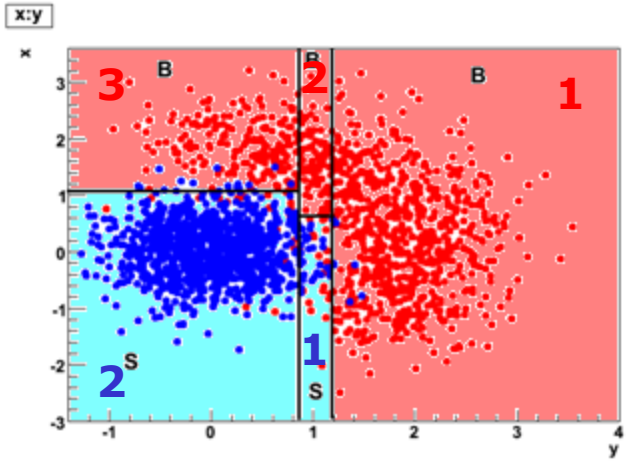


- Why prune a tree?
  - Overtraining → Possible to get a perfect classifier on training events
    - E.g. tree with one class only per leaf (down to 1 event per leaf if necessary)
  - Pruning: eliminate sub-trees (branches) that seem too specific to training sample:
    - a node and all its descendants turn into a leaf
- How?
  - Expected error pruning
    (result from tree with node pruned in consistent *within error* with orig. tree)
    - Statistical error estimate with binomial error $p(1 - p)/N$ for node with purity p and N training events
    - No need for testing sample
  - Cost/complexity pruning
    - Idea: penalize "complex" trees (many nodes/leaves) and find compromise between good fit to training data (larger tree) and good generalization properties (smaller tree)

Concrete example of Decision Tree





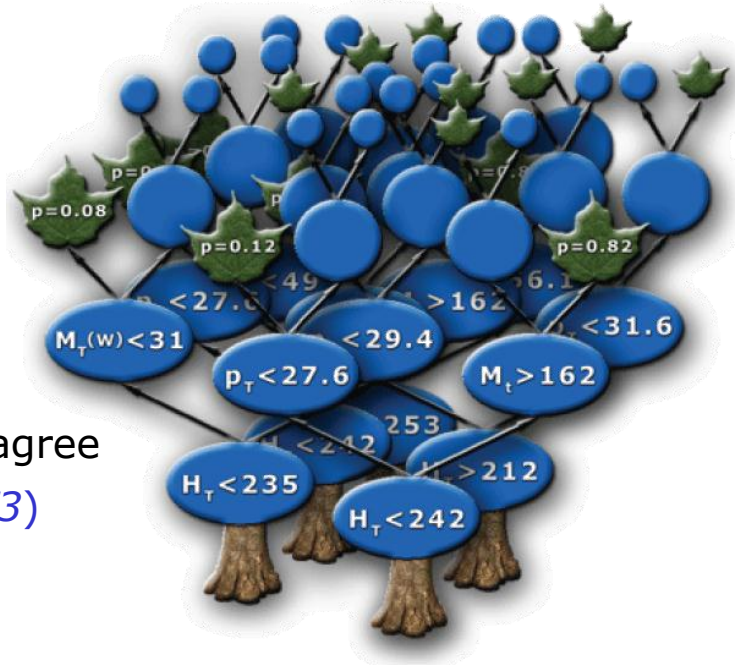


IEF

Decision Tree score card

- Training is fast
- Human readable (not a black box)
- Deals with continuous and discrete variables simultaneously
- No need to transform inputs, resistant to irrelevant variables
  - Irrelevant variables not used, order of training events is irrelevant
- Works well with many variables
  - Ranking of variable x_i : sum impurity at each node where x_i is used → Largest decrease of impurity = best variable
- Good variables can be masked
  - x_j may be just a little worse than x_i but will never be picked → x_j is ranked as irrelevant
  - But remove x_i and x_j becomes very relevant (Solution available (surrogate split, not covered now))
- Very few parameters
- For some time still "original" in HEP
- Unstable tree structure
  - *Small changes in sample can lead to very different tree structures,* easy to overtrain
- Piecewise nature of output (not a continuous distribution)
  - Output distribution discrete by nature
  - granularity related to tree complexity tendency to have spikes at certain purity values (or just two delta functions at ±1 if not using purity)
  - Solution available: averaging over multiple Decision Trees (Boosting)

A brief history of boosting



- First provable algorithm by Schapire (1990)
  - Train classifier *T1* on N events
  - Train *T2* on new N-sample, half of which misclassified by *T1*
  - Build *T3* on events where *T1* and *T2* disagree
  - Boosted classifier: MajorityVote(*T1*,*T2*,*T3*)
- Then
  - Variation by Freund (1995): boost by majority (combining many learners with fixed error rate)
  - Freund & Schapire joined forces: 1st functional model AdaBoost (1996)
- Recently in HEP
  - MiniBooNe compared performance of different boosting algorithms and neural networks for particle ID (2005)
  - D0 claimed first evidence for single top quark production (2006) CDF (2008)

Principles of boosting

- Principles of boosting
  - **General method, not limited to decision trees**
  - Hard to make a very good learner, but easy to make simple, error-prone ones (but still better than random guessing)
  - Goal: combine such weak classifiers into a new more stable one, with smaller error
- General algorithm

- Pseudocode:

  Initialise $\mathbb{T}_1$
  for k in $1..N_{tree}$
    train classifier T_k on $\mathbb{T}_k$
    assign weight α_k to T_k
    modify $\mathbb{T}_k$ into $\mathbb{T}_{k+1}$
- Boosted output: $F(T_1, .., T_{N_{tree}})$

AdaBoost

- **AdaBoost** = Adaptive Boosting (Freund & Shapire '96)
  - Learning procedure adjusts to training data to classify it better
  - Many variations on the same theme for actual implementation
  - Most common boosting algorithm around

- Schematic view of *iterative* algorithm
  - Calculate misclassification rate for Tree K

$$\epsilon_k = \frac{\sum_{i=1}^{N} w_i^k \times \text{isMisclassified}_k(i)}{\sum_{i=1}^{N} w_i^k}$$

*"Weighted average of **isMisclassified** over all training events"*

  - Derive weight of Tree K

$$\alpha_k = \beta \times \ln((1-\epsilon_k)/\epsilon_k)$$

  - Increase weight of misclassified events in Sample(k) to create Sample(k+1)

$$w_i^k \rightarrow w_i^{k+1} = w_i^k \times e^{\alpha_k}$$

  - Train T(k+1) on Sample (k+1)
  - Boosted result

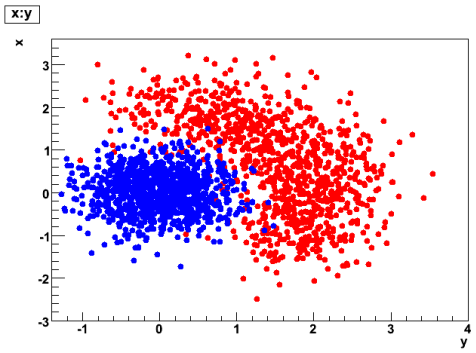
$$T(i) = \sum_{k=1}^{N_{\text{tree}}} \alpha_k T_k(i)$$

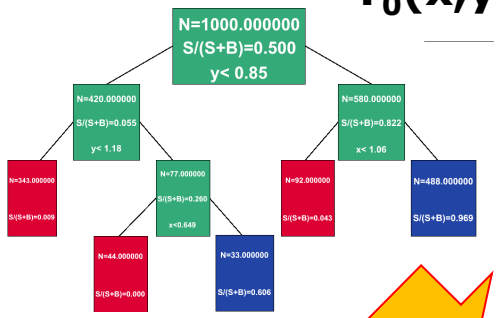
*"Weighted average of **Trees** by their performance*

AdaBoost by example

- So-so classifier (Error rate = 40%) $\alpha = \ln \frac{1-0.4}{0.4} = 0.4$
  - Misclassified events get their weight multiplied by **exp(0.4)=1.5**
  - Next tree will have to work a bit harder on these events
- Good classifier (Error rate = 5%) $\alpha = \ln \frac{1-0.05}{0.05} = 2.9$
  - Misclassified events get their weight multiplied by **exp(2.9)=19** (!!)
  - Being failed by a good classifier means a big penalty: must be a difficult case
  - Next tree will have to pay much more attention to this event and try to get it right
- Overtraining in boosting
  - Misclassification rate on training sample is bound → Rate falls to zero for sufficiently large N(tree)
    - Corollary: training data is over fitted
    - Error rate on test sample may reach a minimum and then potentially rise. Stop boosting at the minimum.
  - In principle AdaBoost must overfit training sample

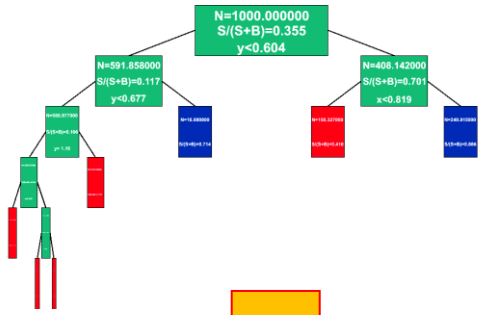
Example of Boosting



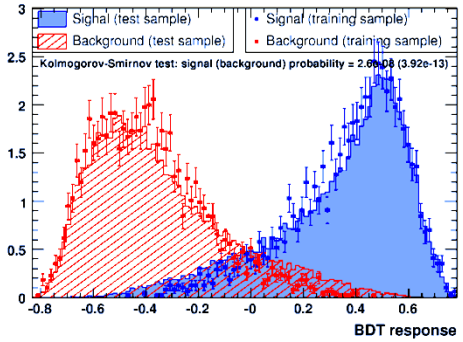


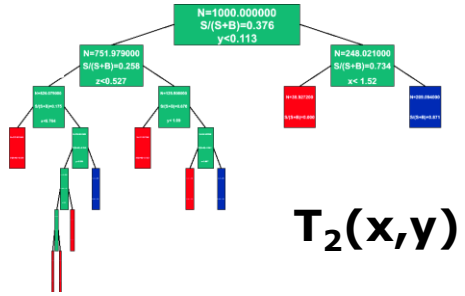
$T_0(x,y)$

$T_1(x,y)$

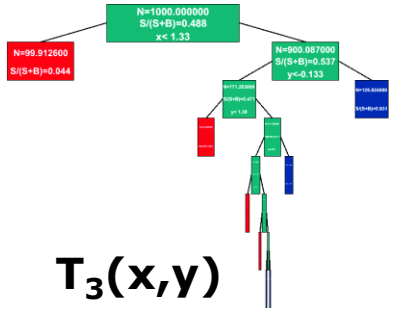


$$B(x,y) = \sum_{i=0}^{4} \alpha_i T_i(x,y)$$

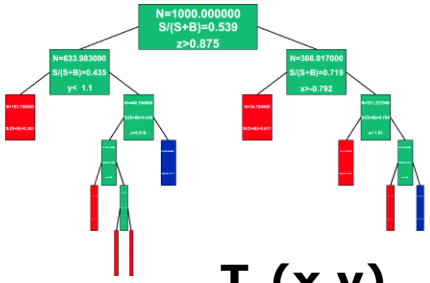




$T_2(x,y)$



$T_3(x,y)$



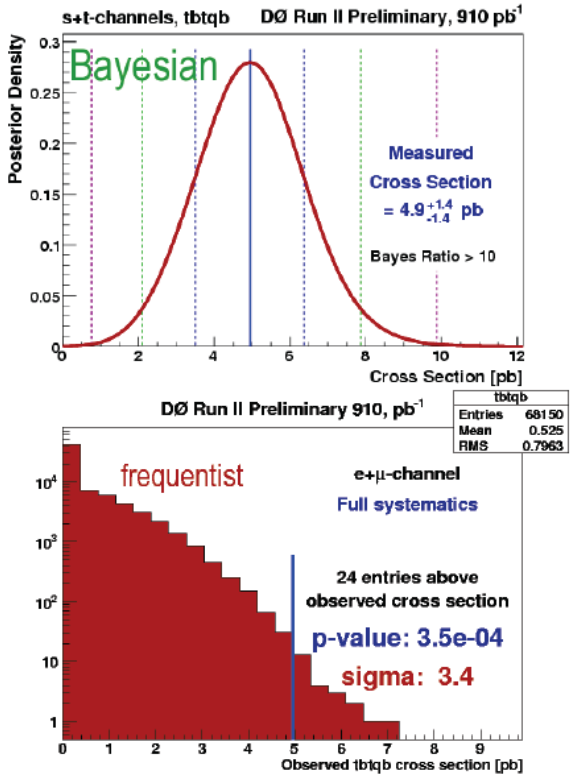
$T_4(x,y)$

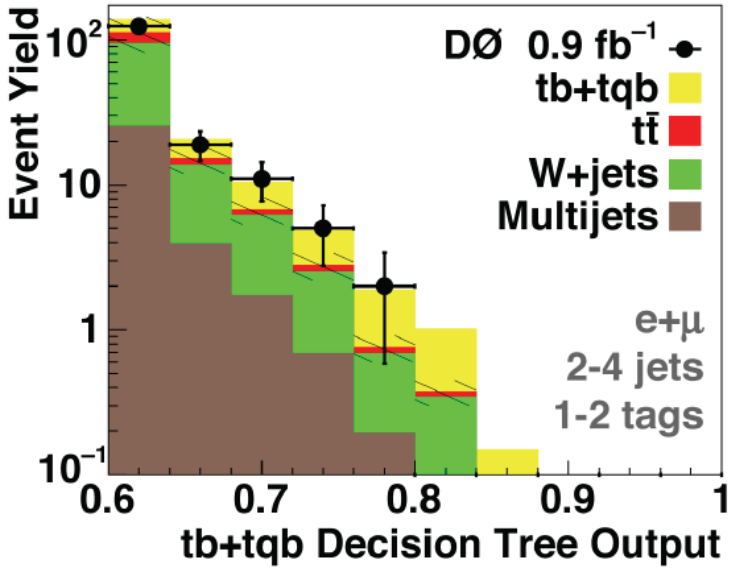
Wouter Verkerke, NIKHEF

Example of BDT analysis of single top quark in D^0

- Three multivariate techniques: BDT, Matrix Elements, BNN
- Most sensitive: BDT

$\sigma_{s+t} = 4.9 \pm 1.4$ pb
p-value $= 0.035\%$ (3.4σ)
SM compatibility: 11% (1.3σ)



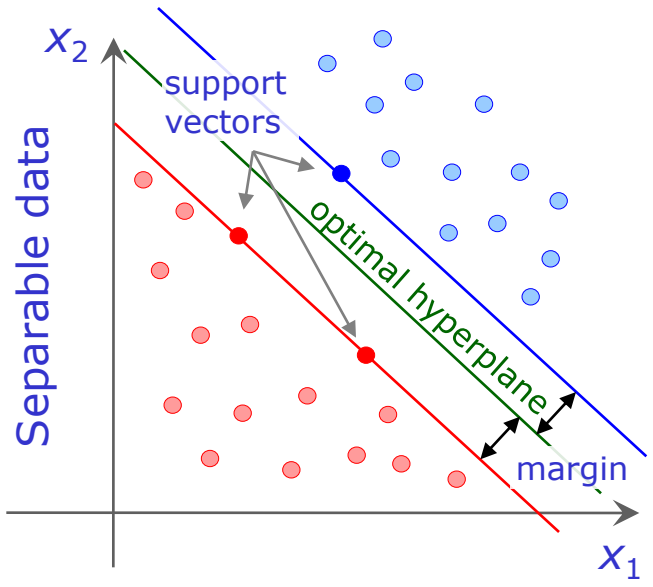


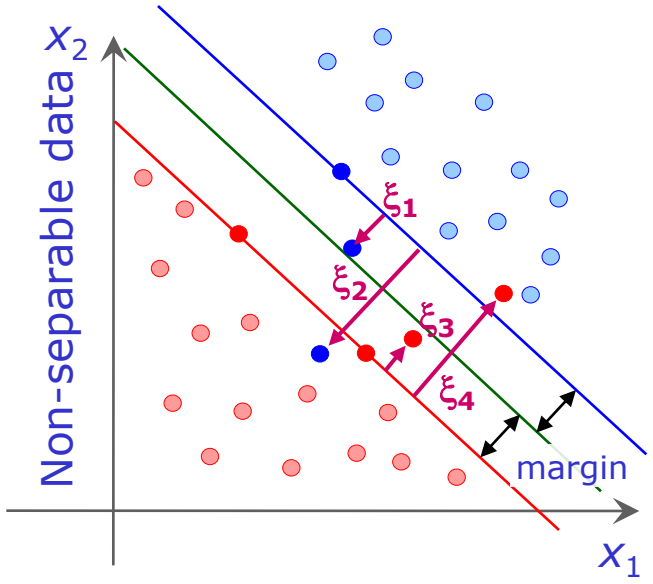
$\sigma_s = 1.0 \pm 0.9$ pb
$\sigma_t = 4.2^{+1.8}_{-1.4}$ pb

Support Vector Machines

- Find hyperplane that best separates signal from background
  - Best separation: maximum distance (margin) between closest events (*support*) to hyperplane
  - Linear decision boundary is defined by solution of a Langrangian
  - *Solution of Lagrangian only depends on inner product of support vectors*

- For non-separable data add misclassification cost
  - add *misclassification cost* parameter $C \cdot \Sigma_i \xi_i$ to minimization function



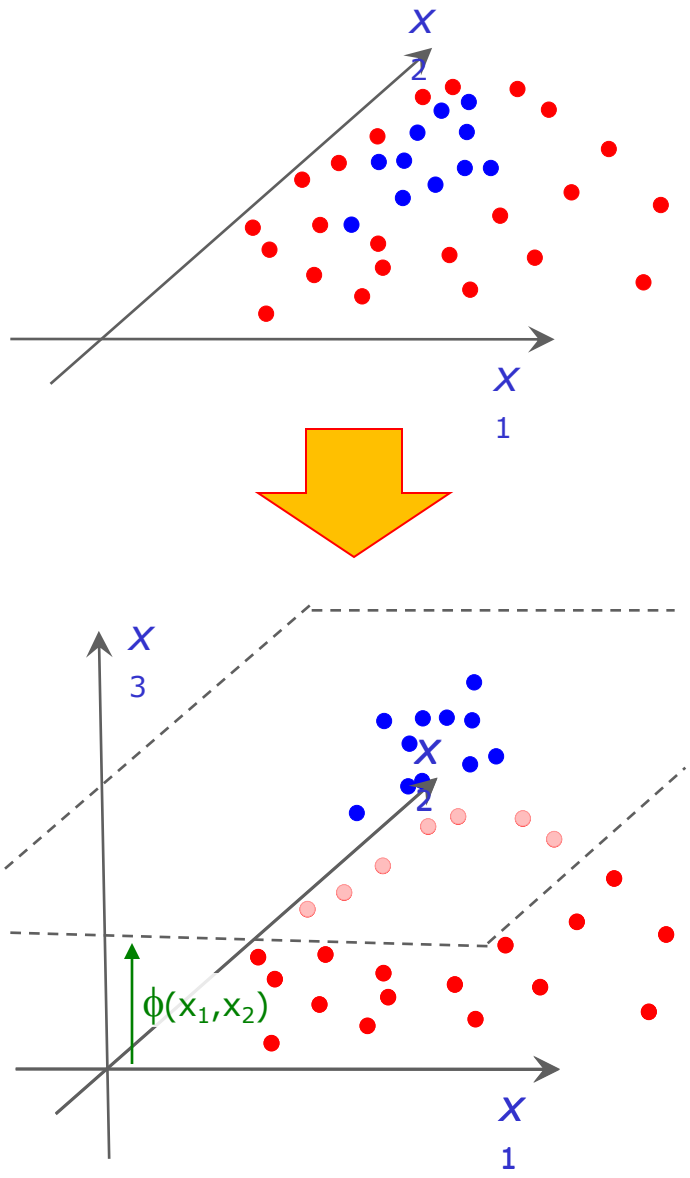


Support Vector Machines

- Non-linear cases
  - Transform variables into higher dimensional feature space

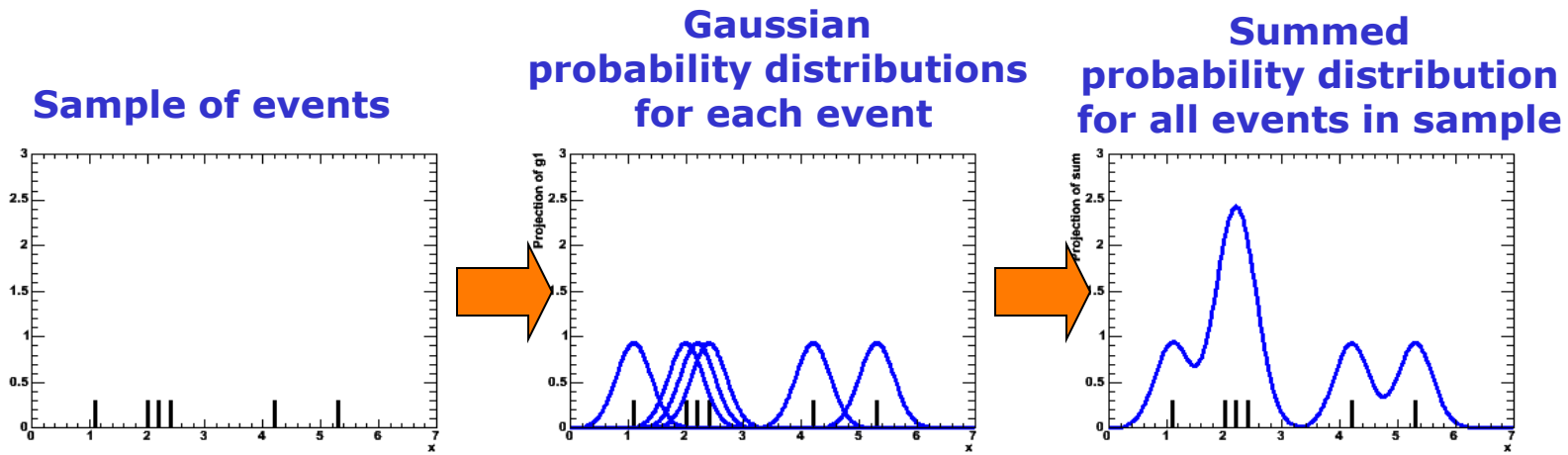
    (x,y) → (x,y,z=ϕ(x,y))

    where again a linear boundary (hyperplane) can separate the data
  - Explicit basis functions not required: use *Kernel Functions* to approximate scalar products between transformed vectors in the higher dimensional feature space
  - Choose Kernel and use the hyperplane using the linear techniques developed above



Probability density estimates

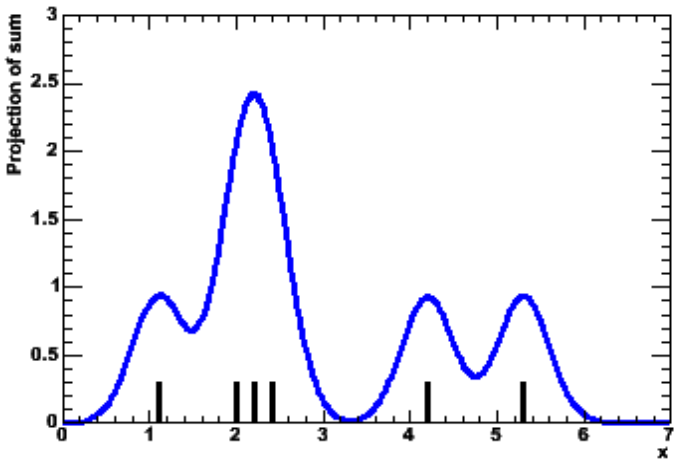
- Instead of constructing a test statistic t(x) using machine learning…
- You can also try to model S(x) and B(x) individually and construct a test statistic as t(x)≡S(x)/B(x)
- Training and parameter-free approach – Probability density estimates from MC samples
  - Idea (1-dim): represent each event of your MC sample as a Gaussian probability distribution
  - Add probability distributions from all events in sample



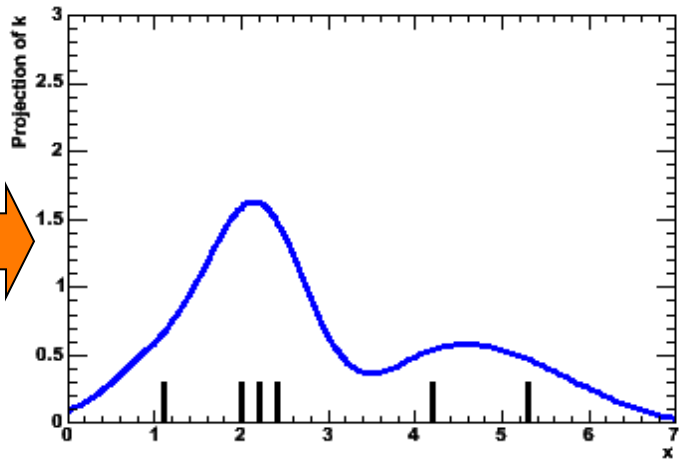
Probability Density Estimates – Adaptive Kernel

- Width of single event Gaussian can of course vary
  - Width of Gaussian tradeoff between smoothness and ability to describe small features
- Idea: 'Adaptive kernel' technique
  - Choose wide Gaussian if local density of events is low
  - Choose narrow Gaussian if local density of events is high
  - Preserves small features in high statistics areas, minimize jitter in low statistics areas

Static Kernel
(with of all Gaussian identical)



Adaptive Kernel
(width of all Gaussian depends on local density of events)

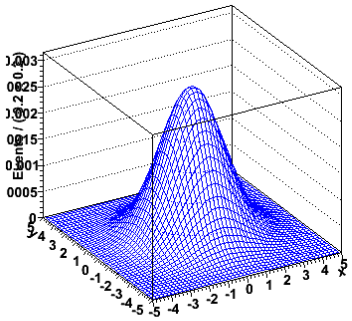


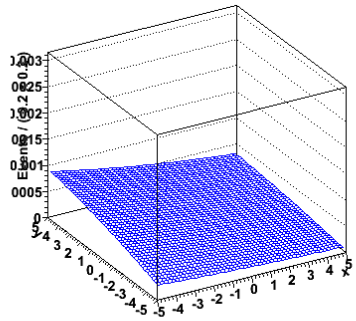
Probability Density Estimates – Some examples

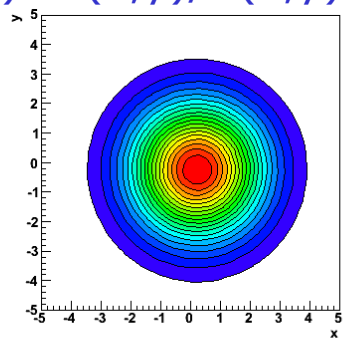
- Example 2D signal and background distribution

S(x,y) B(x,y) D(x,y)=S(x,y)/B(x,y)

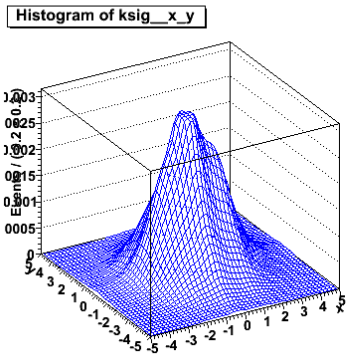
Theoretical distributions

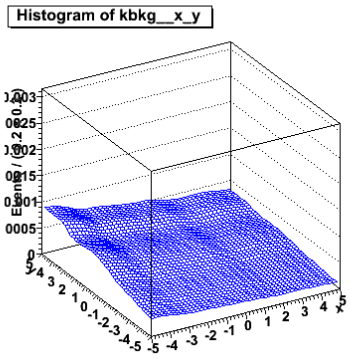


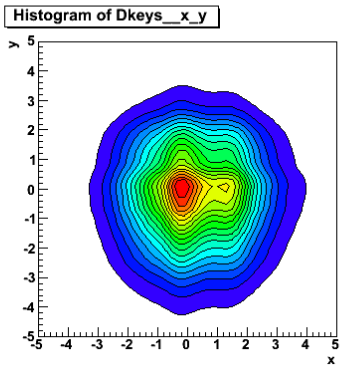




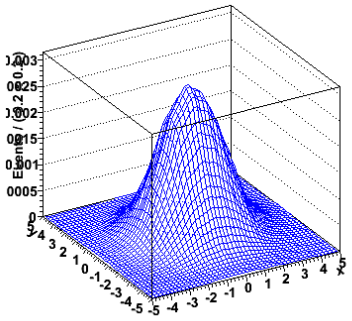
Kernel Estimation (N=1000)

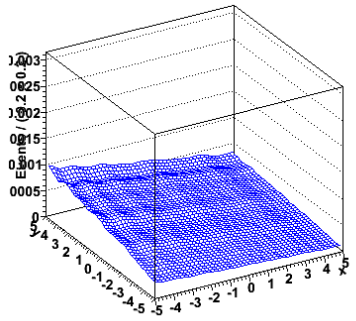


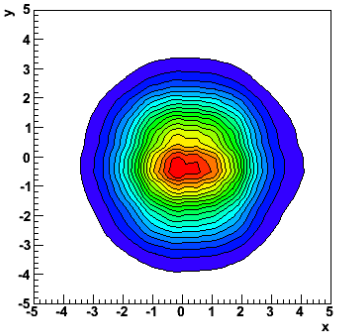




Kernel Estimation (N=10000)







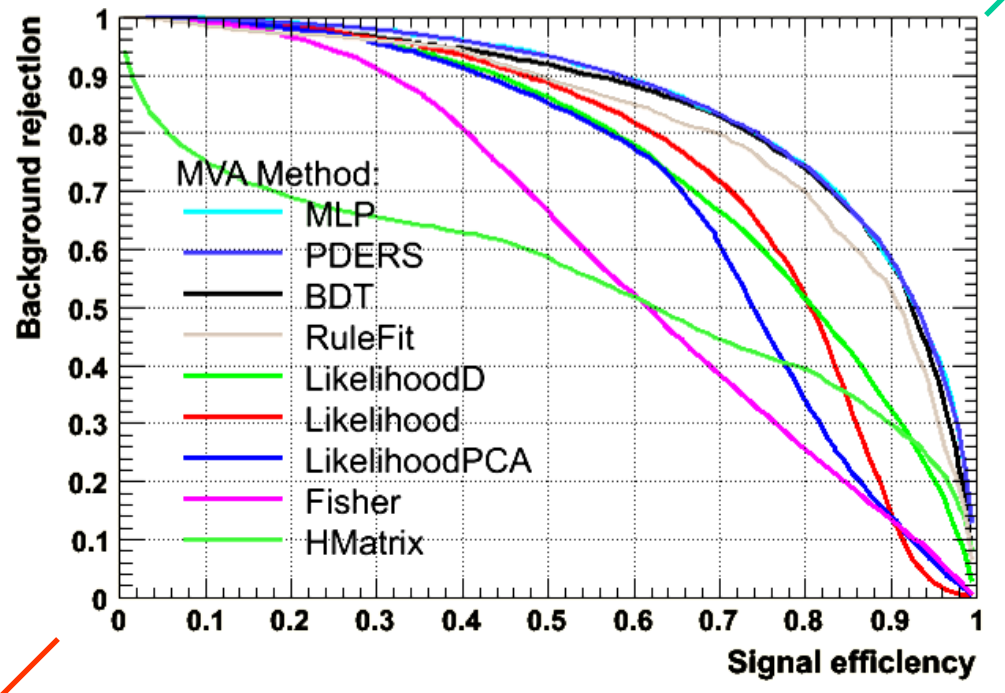
Probability Density Estimates

- Also works in >2 dimensions
- Simplified approach also possible:
  - count events in a (hyper)cube around (**x**)
- Advantages of PDE technique
  - No training necessary
  - Insightful / intuitive
- Disadvantages
  - Prone to effects of low statistics
  - Computationally *very* intense in application phase for multiple dimensions

Characterizing and comparing performance

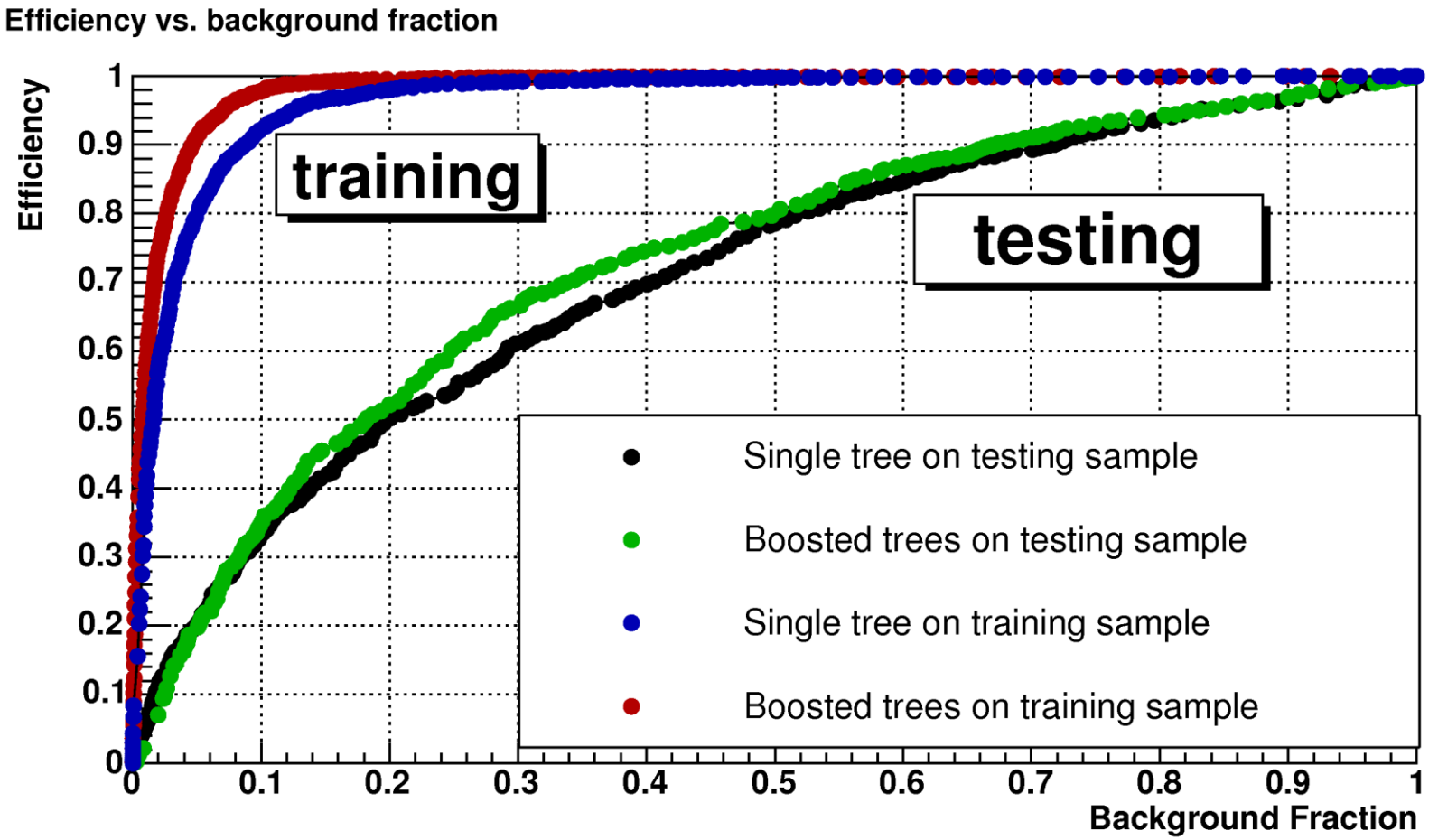
- Performance of a test statistic characterized by ε(sig) vs ε(bkg) curve
  - Curve for theoretical maximum performance can be added if true S(x) and B(x) are known
  - Position on curve determines tradeoff between type-I and type-II errors

Good Performance



Bad Performance

Performance comparison of (boosted) decision trees



Using the compactified output

- Using the output of a of test statistic t(x)
  - **Fit data** to sum of templates $M(x) = N_{sig} \cdot T_S(x) + N_{bkg} \cdot T_B(x)$
    - Optimal use of information, but possibly more sensitive to systematic uncertainties that influence the shape of T_S and T_B
  - **Or select only events with $t(x) > \alpha$** and either count, or fit a distribution of another observable y that was not used in construction of t(x)
    - You throw away some information, but if shapes of signal and background in y are well understood, you have smaller systematic uncertainties
  - **What is the optimal value of α?**
  - Need a 'Figure of Merit' to quantify this

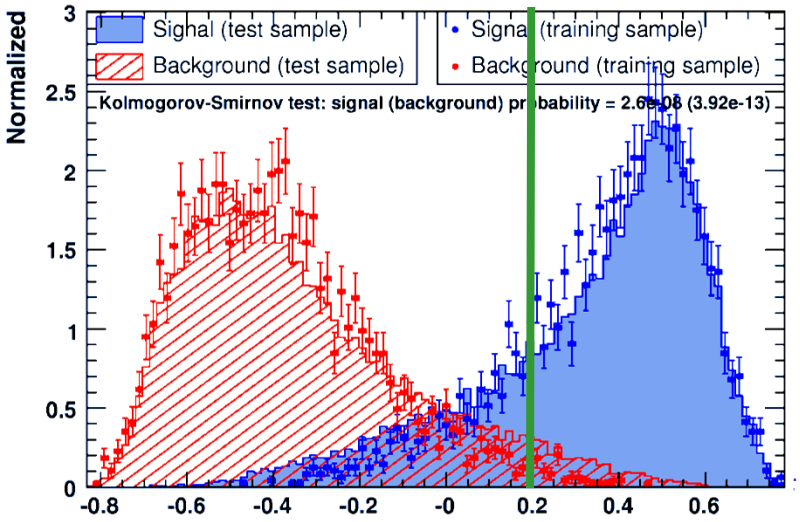


Figure of merit

- Common choices for Figure of Merit
  - Choose position of cut α where F(α) is maximal

$$F(\alpha) = \frac{S(\alpha)}{\sqrt{S(\alpha)+B(\alpha)}}$$

'measurement'

$$F(\alpha) = \frac{S(\alpha)}{\sqrt{B(\alpha)}}$$

'discovery'

Note that position of optimum depends on *a priori* knowledge of signal cross section

Where $S(\alpha)$ and $B(\alpha)$ are number of signal and background events remaining after a cut α

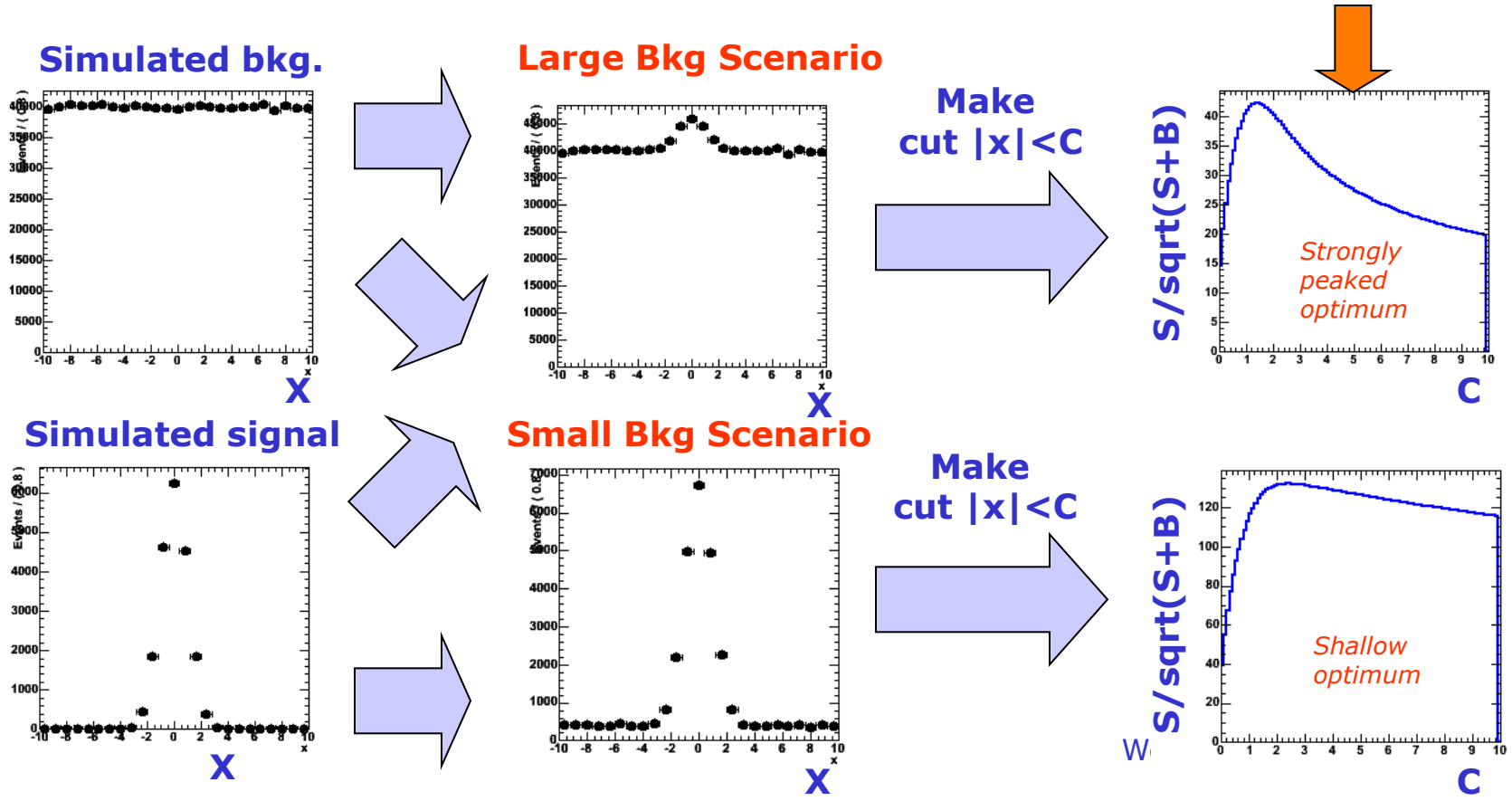
- Note that above FOMs are 'traditional', not statistically rigorous.
  - Better calculations exist, see e.g. Punzi / PhyStat 2003
  - Example: Counting experiment where signal and background are Poisson processes, discovery at *n* sigma

$$F(\alpha) = \frac{\varepsilon(\alpha)}{n/2+\sqrt{B(\alpha)}}$$

Where $\varepsilon(\alpha)$ the signal efficiency and $B(\alpha)$ is number of background events at a cut α

Using a figure of merit

- Application of S/√S+B figure of merit to simple one-dimensional cut

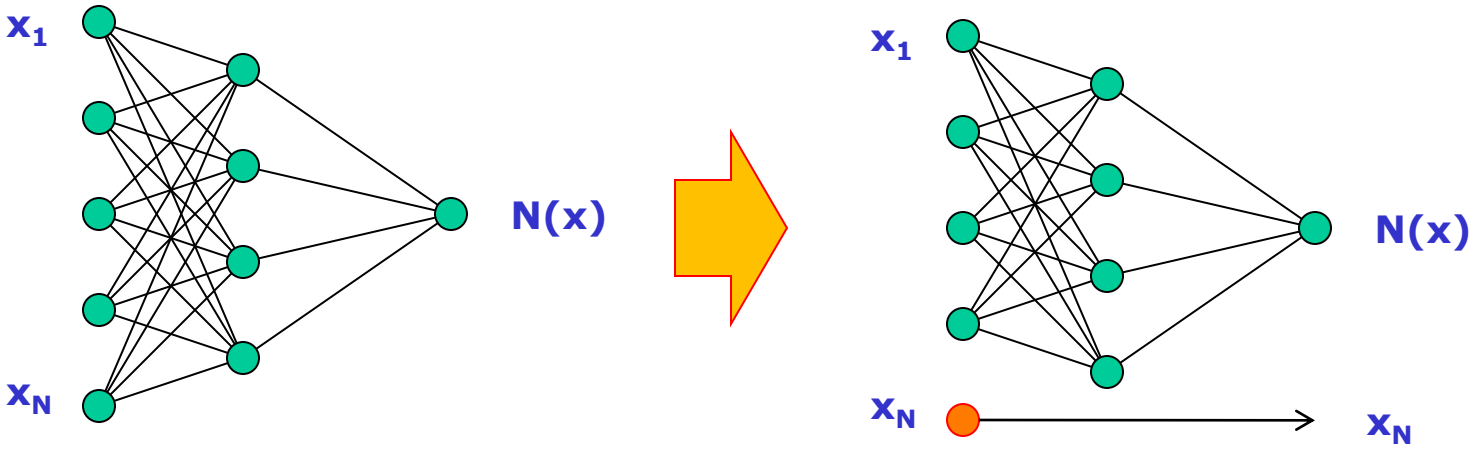


Do you trust your NN / BDT etc?

- Keep in mind that any trained NN / BDT / SVM is just a parameterized decision boundary in *n* dimensions
  - You can visualize it. It may be sub-optimal (or overtrained) but it is really just a boundary.
- Better question: Do you believe that the signal and background *training samples* that were used were an accurate reflection of reality?
  - In general, they don't agree (perfectly) so performance should be interpreted with some care
  - Usually background is harder to get right than signal, because in addition to any imperfections in the simulation process, you also need to guess correctly which physics processes constitute your background
  - In particular, you should not assume that the efficiency of a cut on your test statistic t(x)>a on simulated events is the same as in data → Important source of systematic uncertainty in your final result

Measuring the performance of your *t(x)* on data

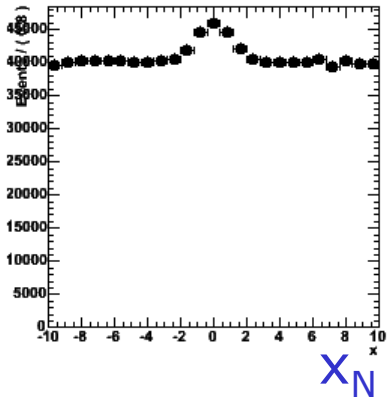
- Common solution for controlling uncertainties in the performance of your *t(x)* is to measure it on data
  - Several ways to do this, will highlight one here
- Keep one (uncorrelated) observable out of your test statistic
  - Example with NN here. Instead of using all observables with good discriminating power, keep one out of your test statistic
  - Choose one that is powerful and maximally uncorrelated with the others (for both signal and background). This may be your 'best' observable



Measuring the performance of your *t(x)* on data

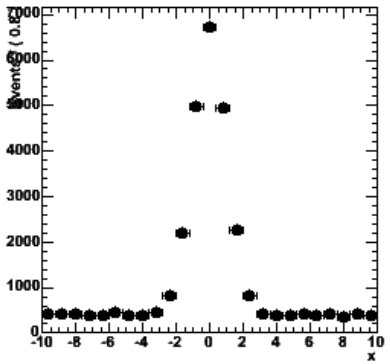
- Analysis strategy
  - Cut on optimal value on N(x) to reduce background as much as possible
  - Make a fit to distribution of x_N to measure residual amount of signal and background
- In the limit of zero correlation between N(x) and x_N the shape of the distribution is invariant under the cut on N(x)

No cut on N(x)



X_N

After cut N(x)>α



X_N

Background much reduced, but you can *measure* how much is left

  - In the limit of complete de-correlation between N(x) and X_N no systematic uncertainty to N(x) performance

Finding the right method

- Which technique is right for your problem? Depends on
  - Complexity of your problem
  - Time scale in which you would like to finish the analysis

- On finding the absolute best set of cuts
  - ***All*** methods for finding discriminants are approximate when used with finite training/tuning statistics
  - Your experiments event simulation is imperfect – your performance on data can be different (usually it is less)
  - You may have systematic error later that might depend on your choice of cuts
  - Don't hunt for upward statistical fluctuations in tuning data
  - If it takes you 6 months of work to reduce your error by 10% keep in mind that your experiment may have accumulated enough additional data by them to reduce your statistical error by a comparable or larger amount

- It is more important to get the right(=unbiased) answer than the smallest possible statistical error
  - Don't use discriminating variables that you know are poorly modeled in simulation
  - Always try to find a way to cross check your performance on data, e.g. by using a control sample, or leaving one observable out of your t(x)

(Software Advertisement #1)
TMVA

Wouter Verkerke, NIKHEF

What is *T*MVA

- ROOT: is the analysis framework used by most (HEP)-physicists
- Idea: rather than just implementing new MVA techniques and making them available in ROOT (*i.e.*, like TMultiLayerPercetron does):
  - Have one 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

Limitations of *T*MVA

- Development started beginning of 2006 – a mature but **not** a final package
- Known limitations / missing features
  - Performs classification only, and only in binary mode: *signal* versus *background*
  - Supervised learning only (no unsupervised "bump hunting")
  - Relatively stiff design – not easy to mix methods, not easy to setup categories
  - Cross-validation not yet generalised for use by all classifiers
  - Optimisation of classifier architectures still requires tuning "by hand"
- Work ongoing in most of these areas → see later in this talk

TMVA Content

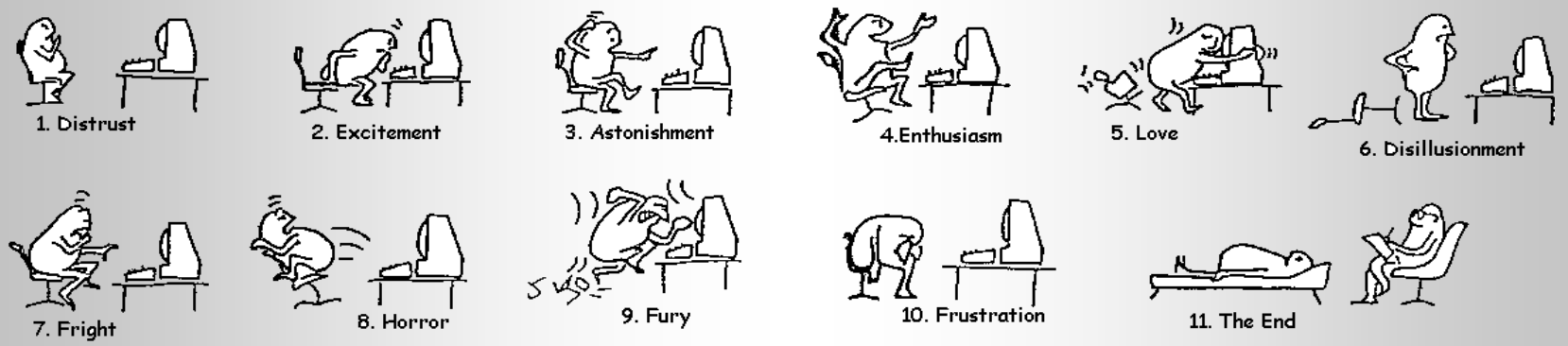
- Currently implemented classifiers
  - Rectangular cut optimisation
  - Projective and multidimensional likelihood estimator
  - k-Nearest Neighbor algorithm
  - Fisher and H-Matrix discriminants
  - Function discriminant
  - Artificial neural networks (3 *multilayer perceptron* impls)
  - Boosted/bagged decision trees
  - RuleFit
  - Support Vector Machine
- Currently implemented data preprocessing stages:
  - Decorrelation
  - Principal Value Decomposition
  - Transformation to uniform and Gaussian distributions (*coming soon*)

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

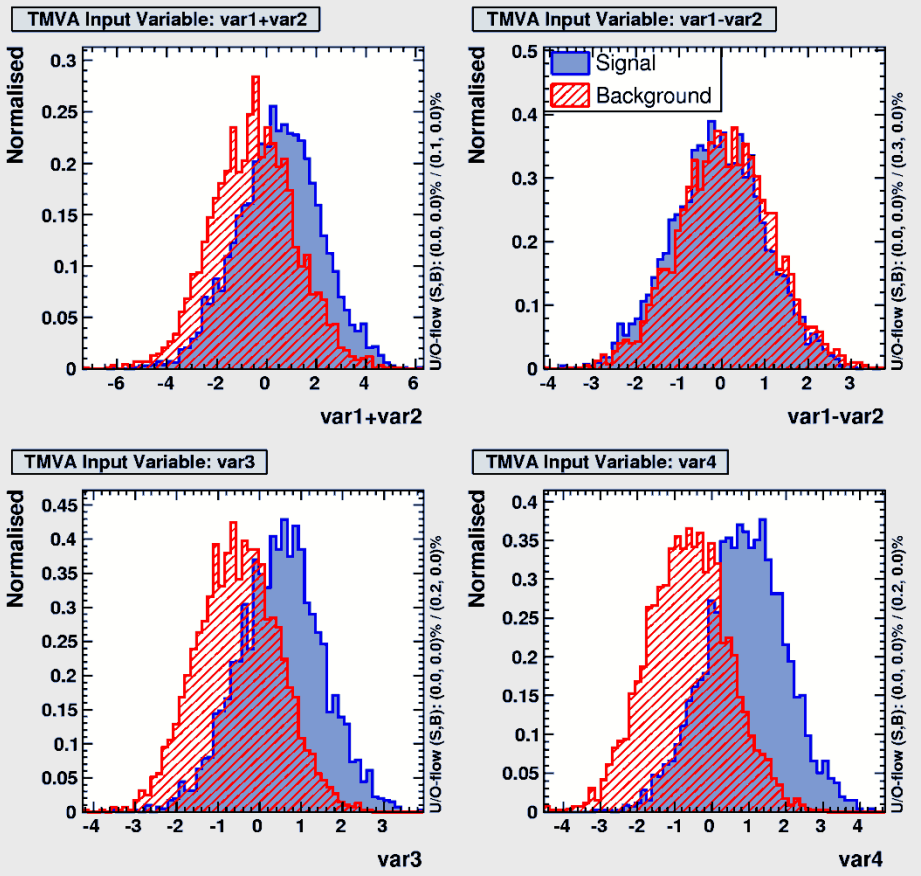
- Illustration of these steps with toy data samples

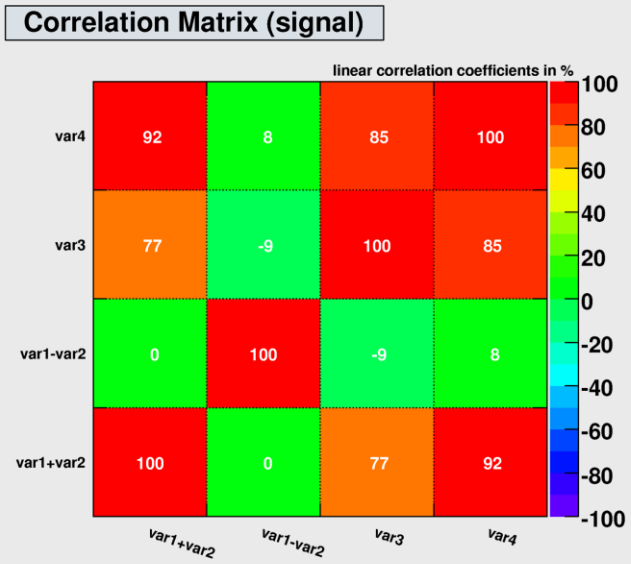


→ *T*MVA tutorial

A Toy Example (idealized)

- Use data set with 4 linearly correlated Gaussian distributed variables:

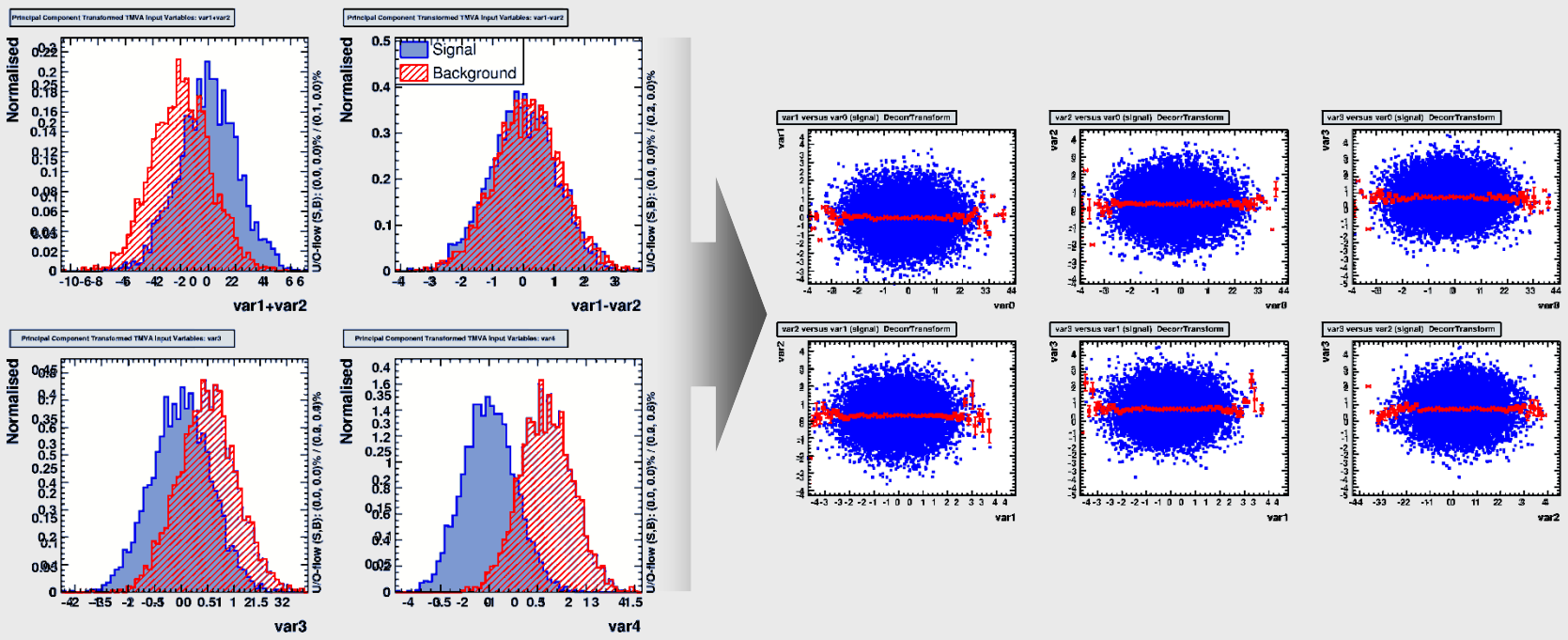




Rank	Variable	Separation
1	var4	0.606
2	var1+var2	0.182
3	var3	0.173
4	var1-var2	0.014

Preprocessing the Input Variables

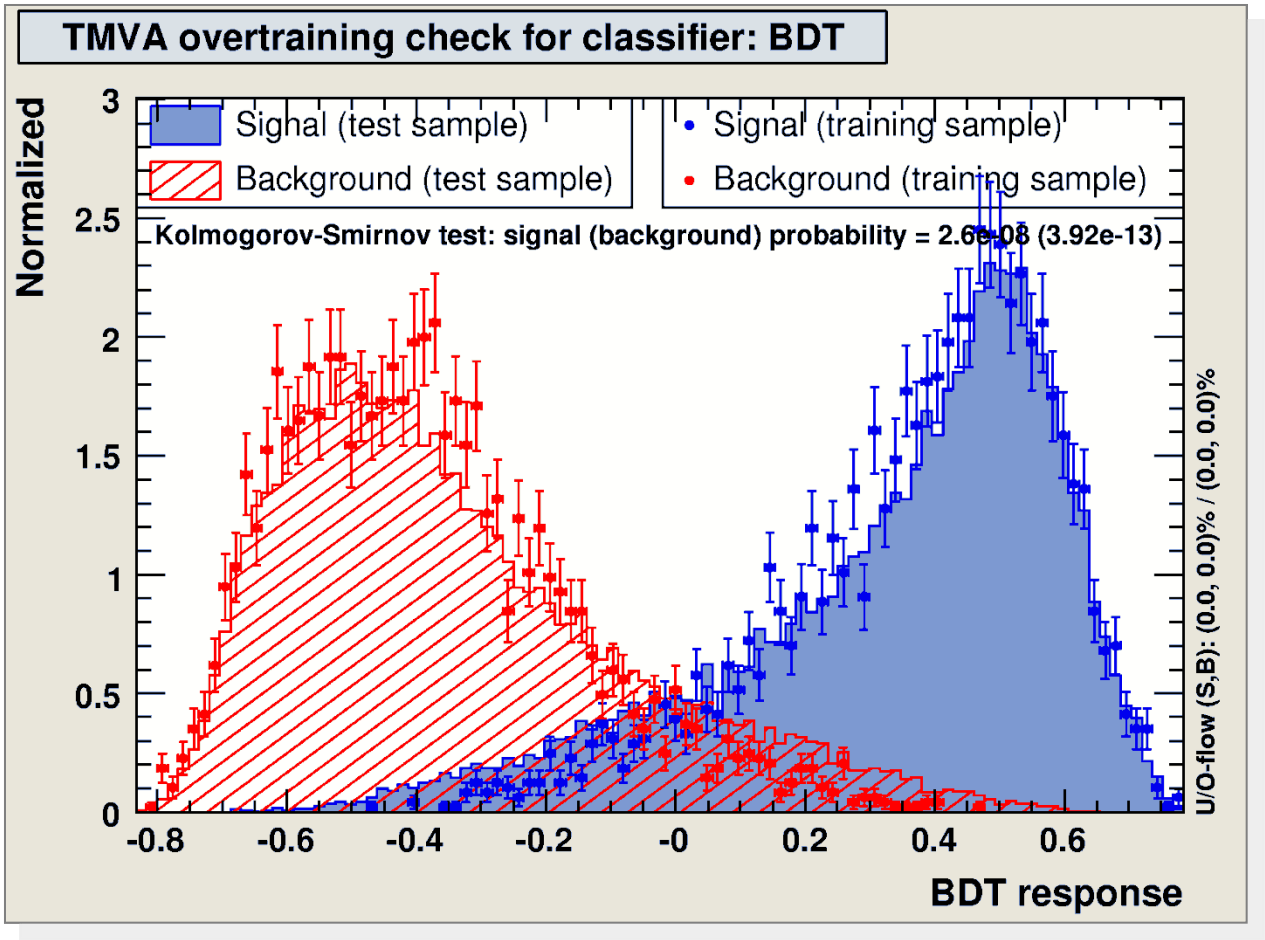
- Decorrelation of variables before training is useful for *this* example



- Note that in cases with non-Gaussian distributions and/or nonlinear correlations decorrelation may do more harm than any good

Evaluating the Classifier Training (II)

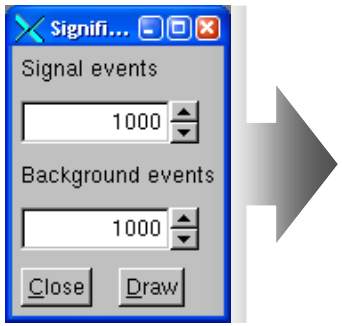
- Check for overtraining: classifier output for test *and* training samples …

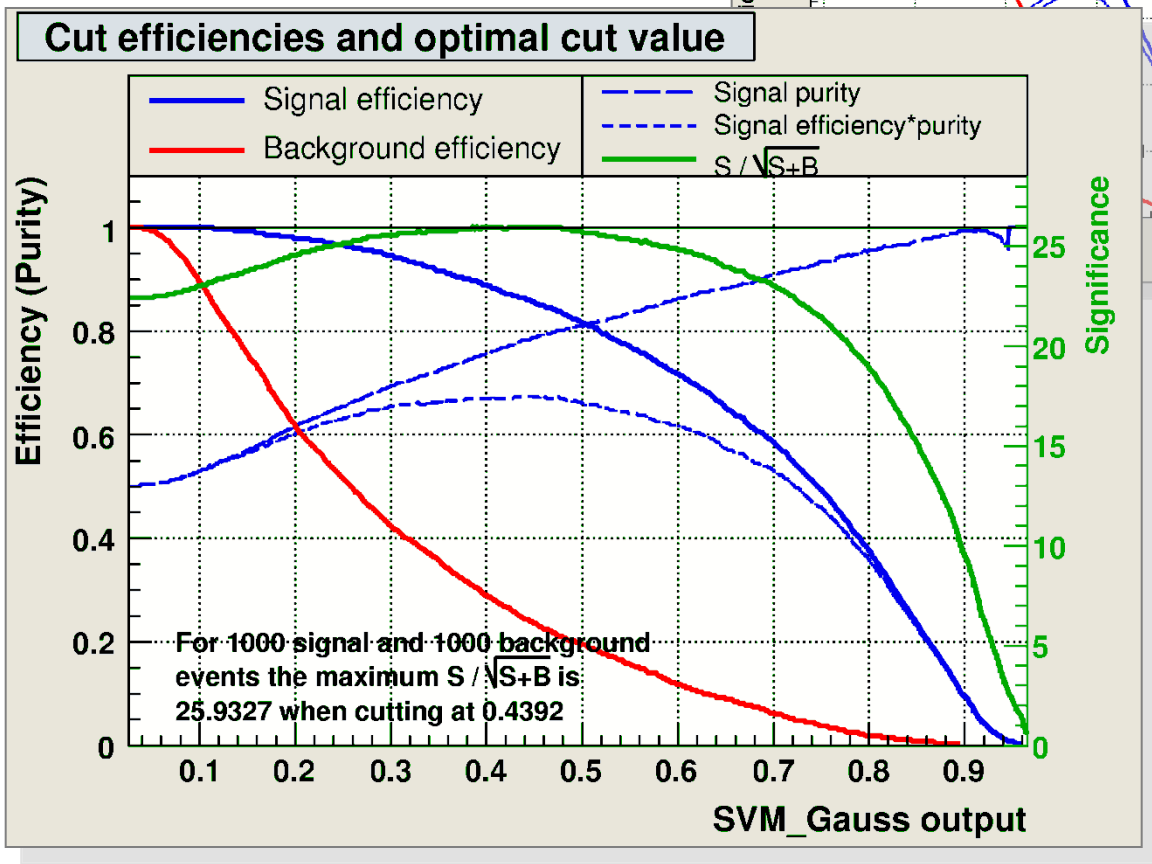


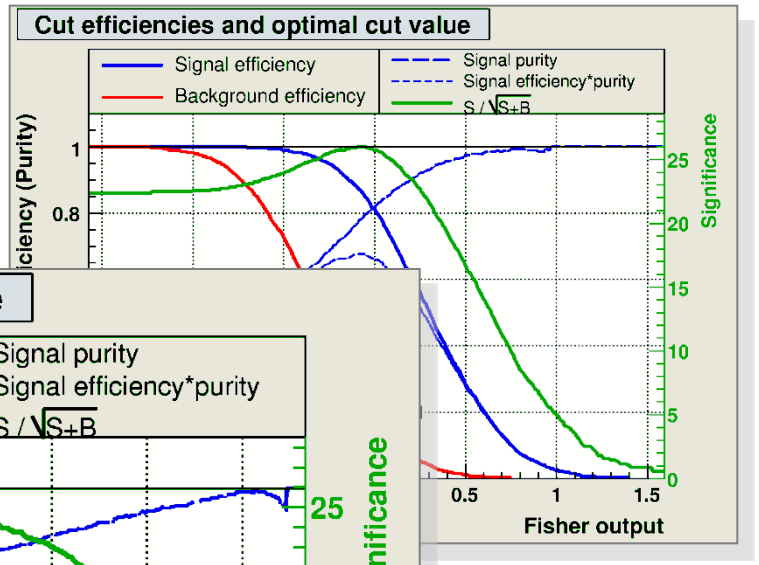
Evaluating the Classifier Training (V)

- Optimal cut for each classifiers …

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

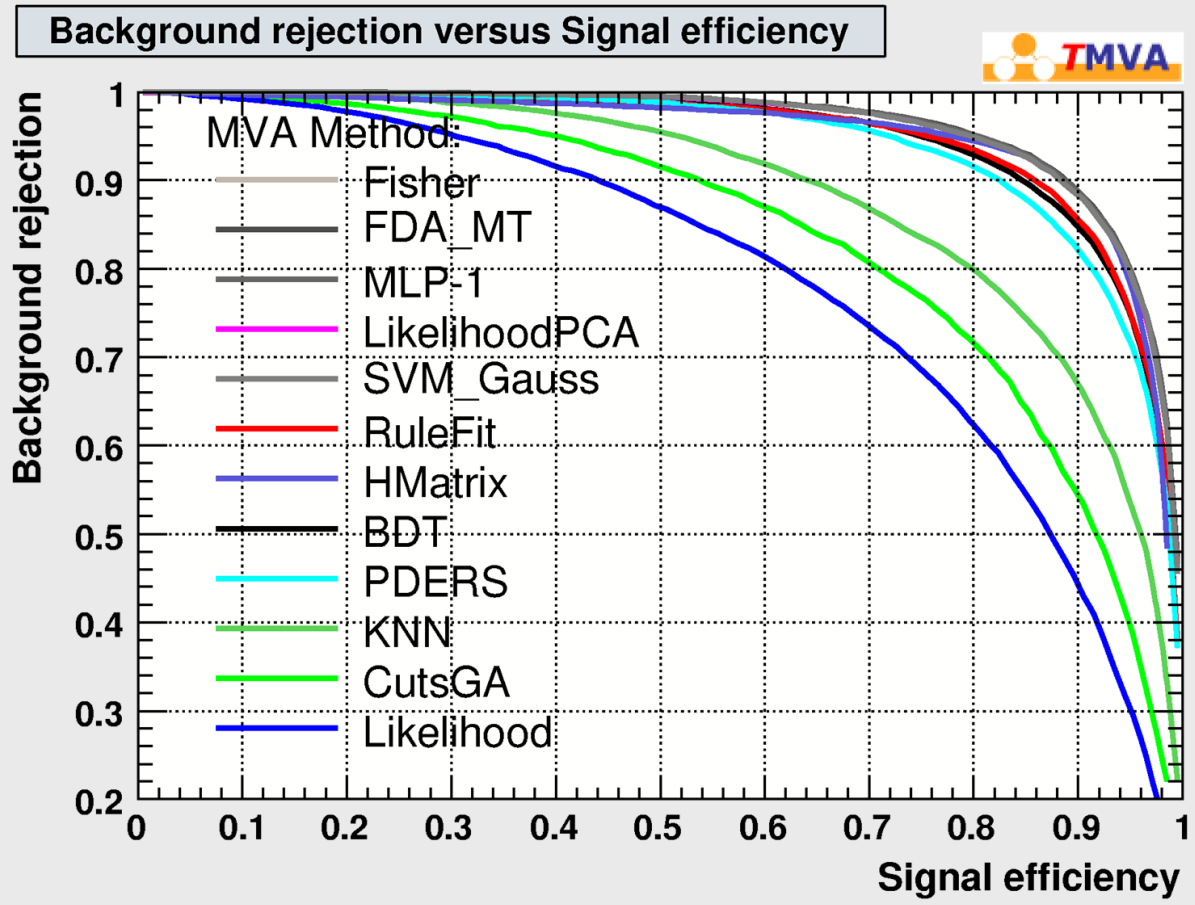






Receiver Operating Characteristics (ROC) Curve

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



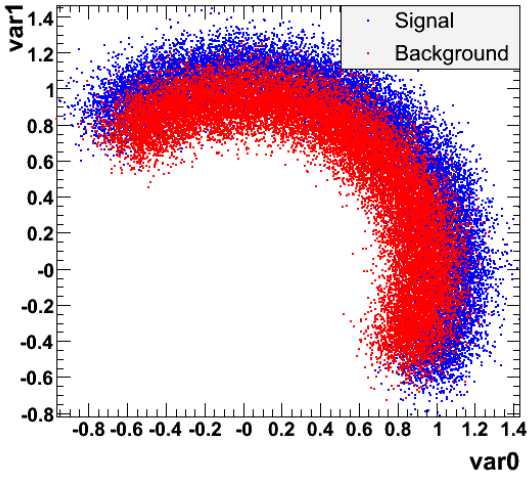
"Specificity" (probability to predict *B* if true *B*)

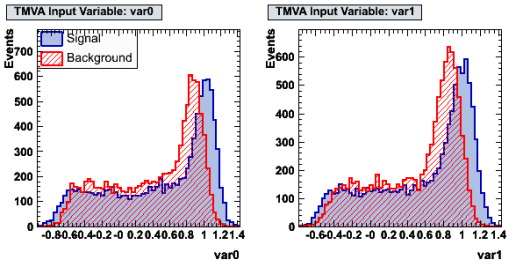
"Sensitivity" (probability to predict *S* if true *S*)

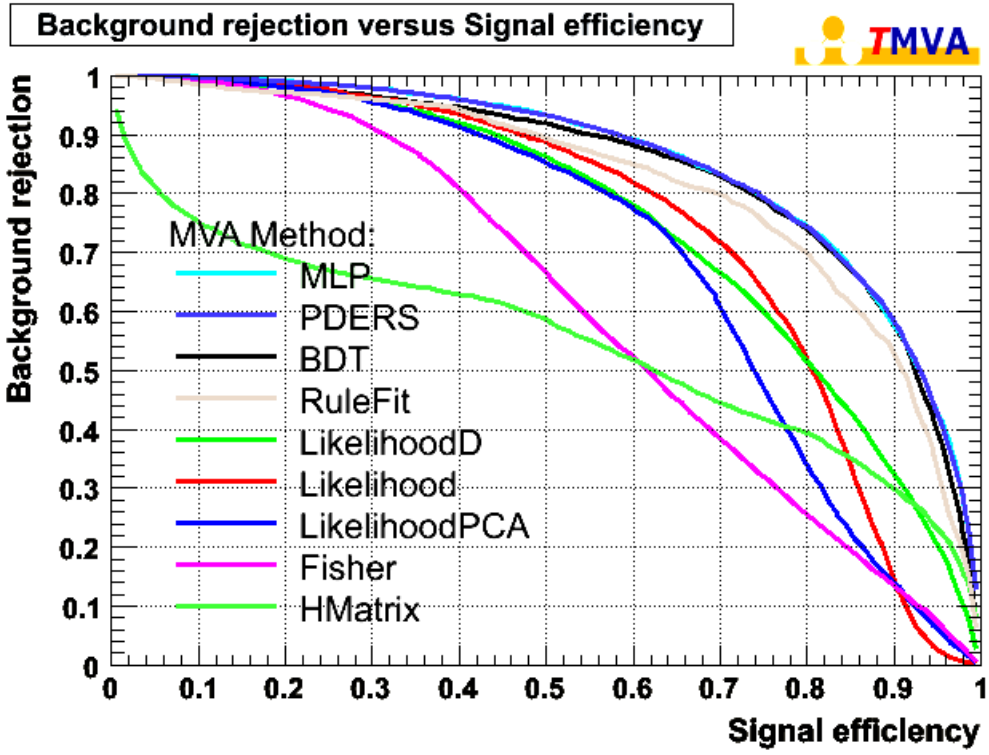
Example: Circular Correlation

- Illustrate the behavior of linear and nonlinear classifiers

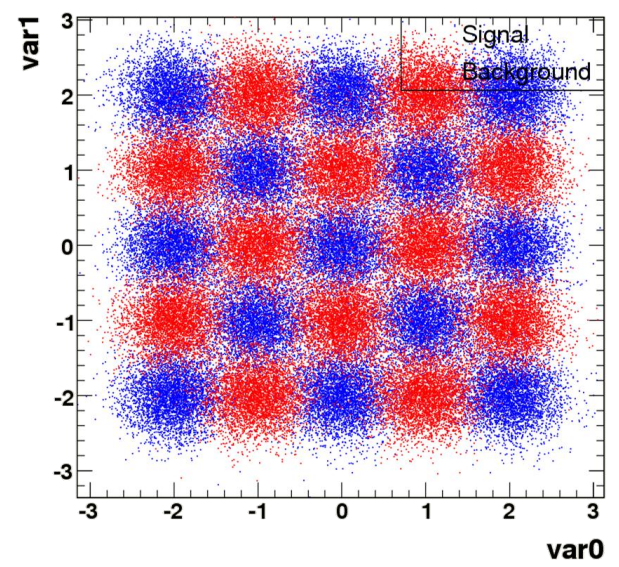
Circular correlations
(same for signal and background)

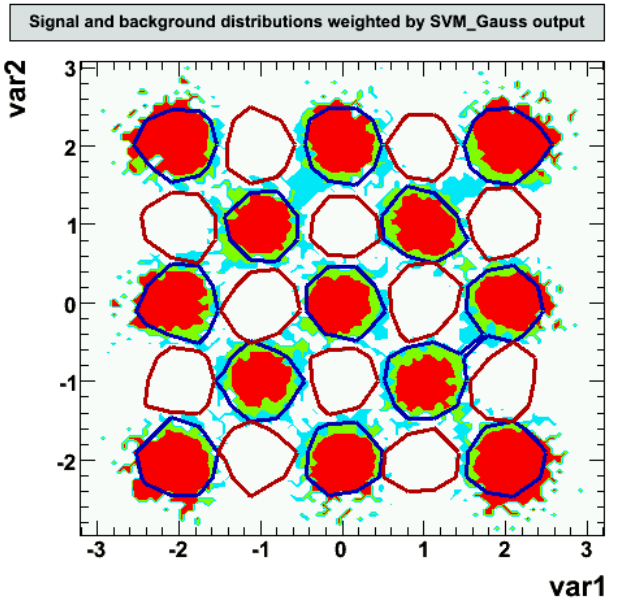




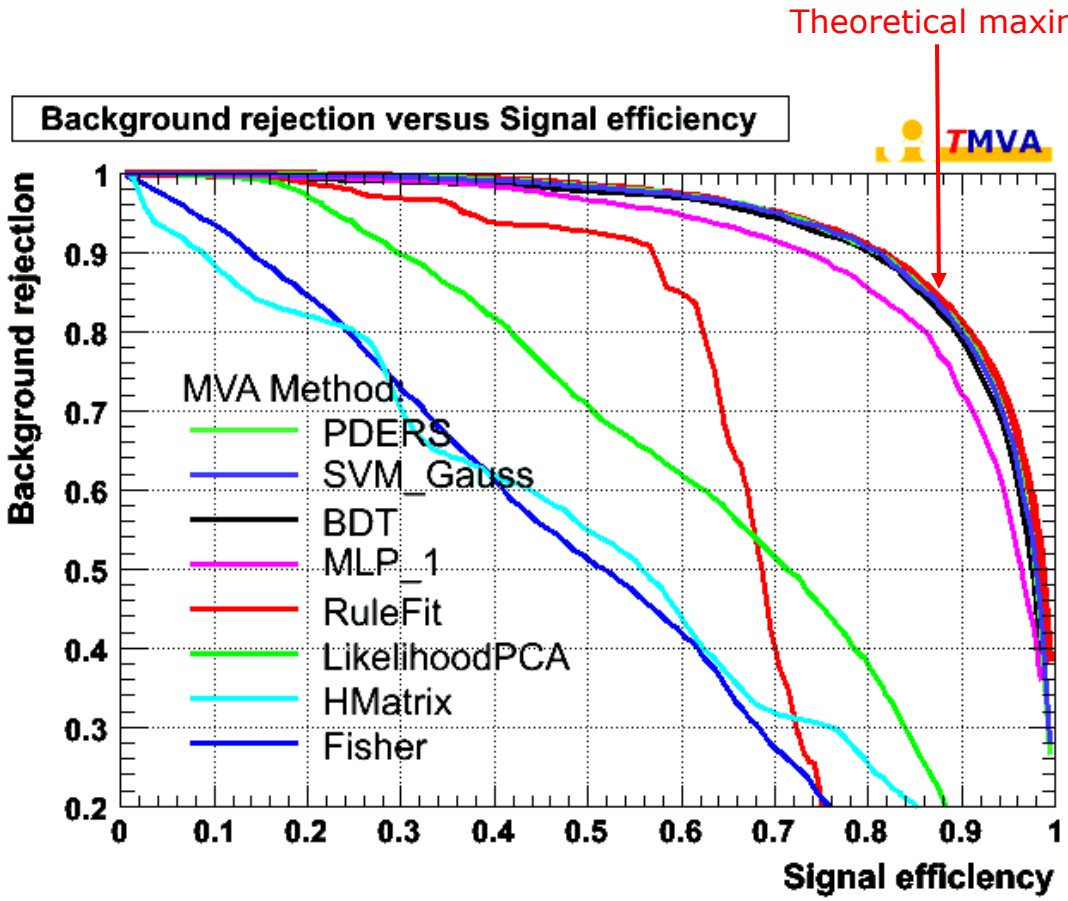


The "Schachbrett" Toy





- Performance achieved without parameter tuning: PDERS and BDT best "out of the box" classifiers
- After specific tuning, also SVM und MLP perform well



Summary of the Classifiers and their Properties

Criteria		Classifiers								
		Cuts	Likeli-hood	PDERS / k-NN	H-Matrix	Fisher	MLP	BDT	RuleFit	SVM
Perfor-mance	no / linear correlations	😐	🙂	🙂	😐	🙂	🙂	😐	🙂	🙂
	nonlinear correlations	😐	☹	🙂	☹	☹	🙂	🙂	😐	🙂
Speed	Training	☹	🙂	🙂	🙂	🙂	😐	☹	😐	☹
	Response	🙂	🙂	☹/😐	🙂	🙂	🙂	😐	😐	😐
Robust-ness	Overtraining	🙂	😐	😐	🙂	🙂	☹	☹	😐	😐
	Weak input variables	🙂	🙂	☹	🙂	🙂	😐	😐	😐	😐
Curse of dimensionality		☹	🙂	☹	🙂	🙂	😐	🙂	😐	😐
Transparency		🙂	🙂	😐	🙂	🙂	☹	☹	☹	☹

The properties of the Function discriminant (FDA) depend on the chosen function

Exercises

- You need ROOT 5.30-03 (or 04)
- You need some small input files
- All files (plus exercise descriptions) are in

http://www.nikhef.nl/~verkerke/statcourse

- Either pick up nbi.tar
  all files individually