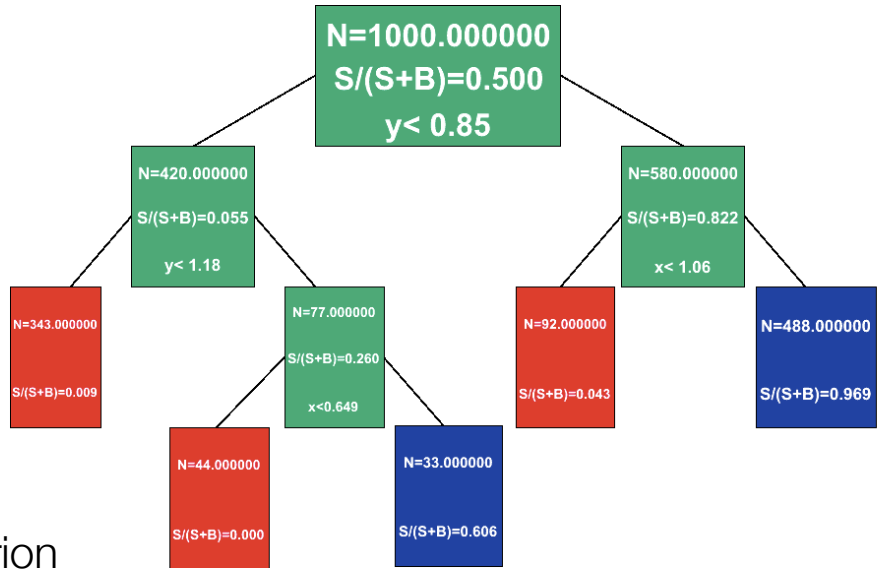
A very different type of Ansatz - Decision Trees

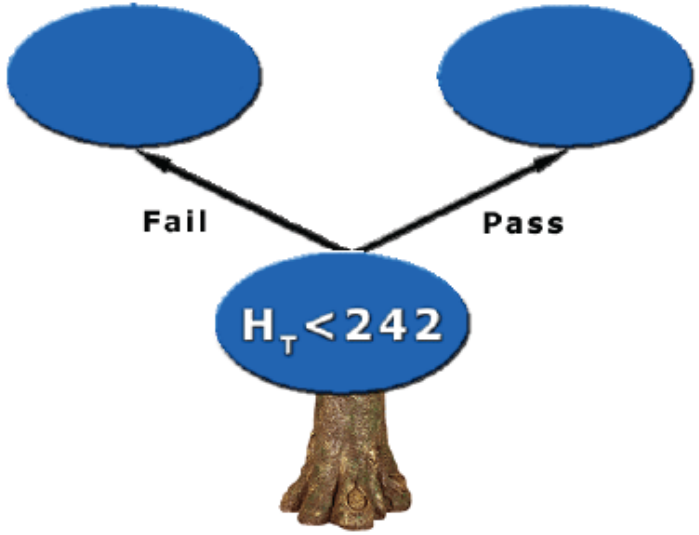
- A Decision Tree encodes sequential rectangular cuts
  - But with a lot of underlying theory on training and optimization
  - Machine-learning technique, widely used in social sciences
  - L. Breiman et al., “Classification and Regression Trees” (1984)



- 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

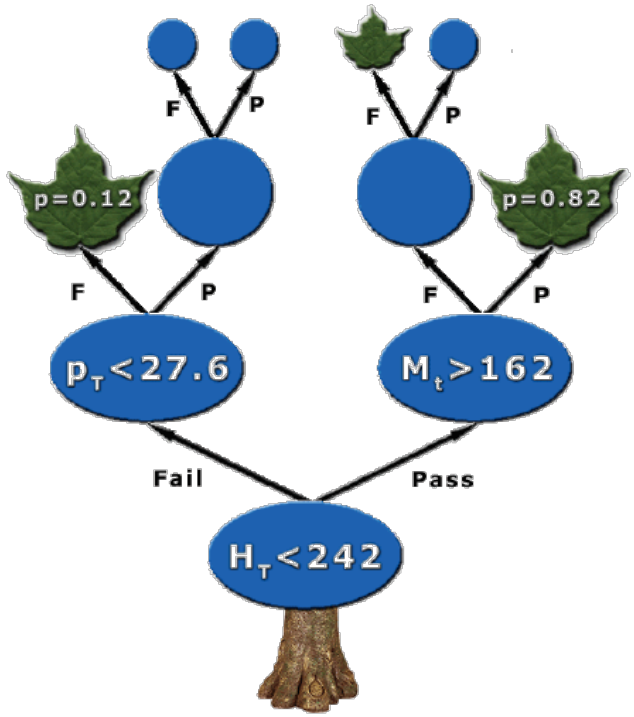
- Essential operation :
  splitting the data in 2 groups using a single cut, e.g. H_T<242



- Goal: find 'best cut' as quantified through best separation of signal and background (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



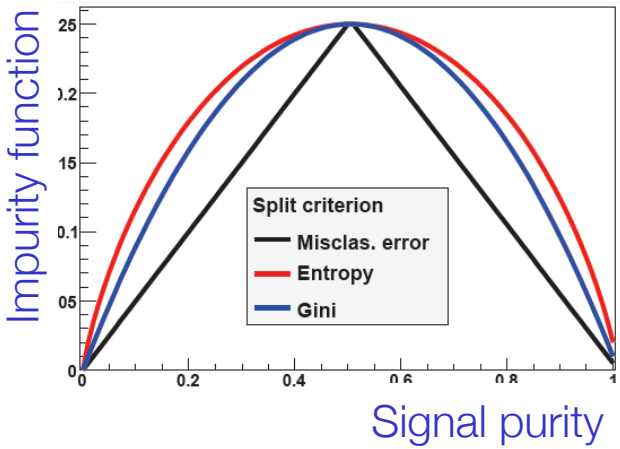
- **Output** of decision tree:
  - 'signal' or 'background' (0/1) or
  - probability based on *expected purity* of leaf (s/s+b)

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

Machine learning with Decision Trees

- Instead of '(Empirical) Risk' **minimize 'Impurity Function'** of leaves
  - Impurity function *i(t)* quantifies (im)purity of a sample, but is not uniquely defined
  - Simplest option: i(t) = misclassification rate



- For a proposed split *s* on a node *t*, decrease of impurity is

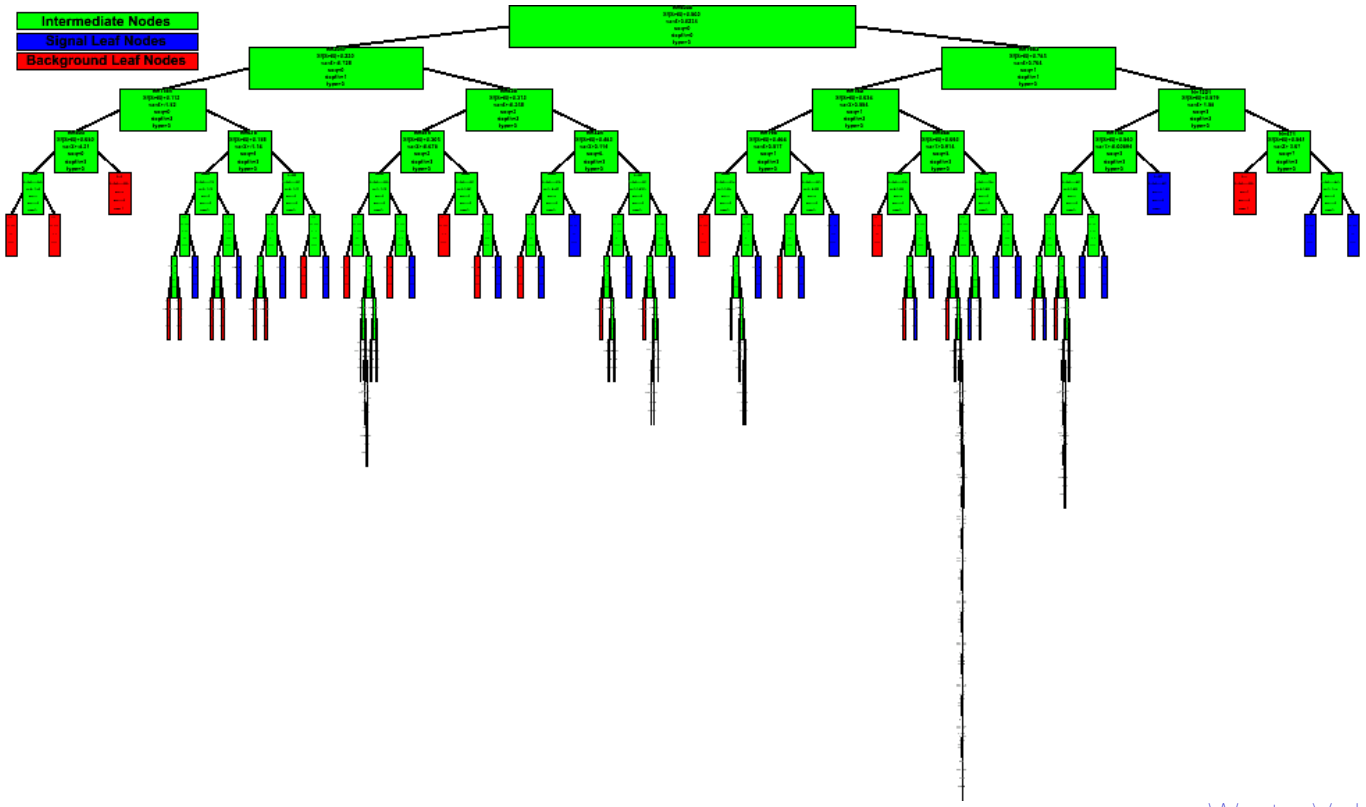
$$\Delta i(s,t) = i(t) - p_L \cdot i(t_L) - p_R \cdot i(t_R)$$

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

- Take split that results in largest Δi

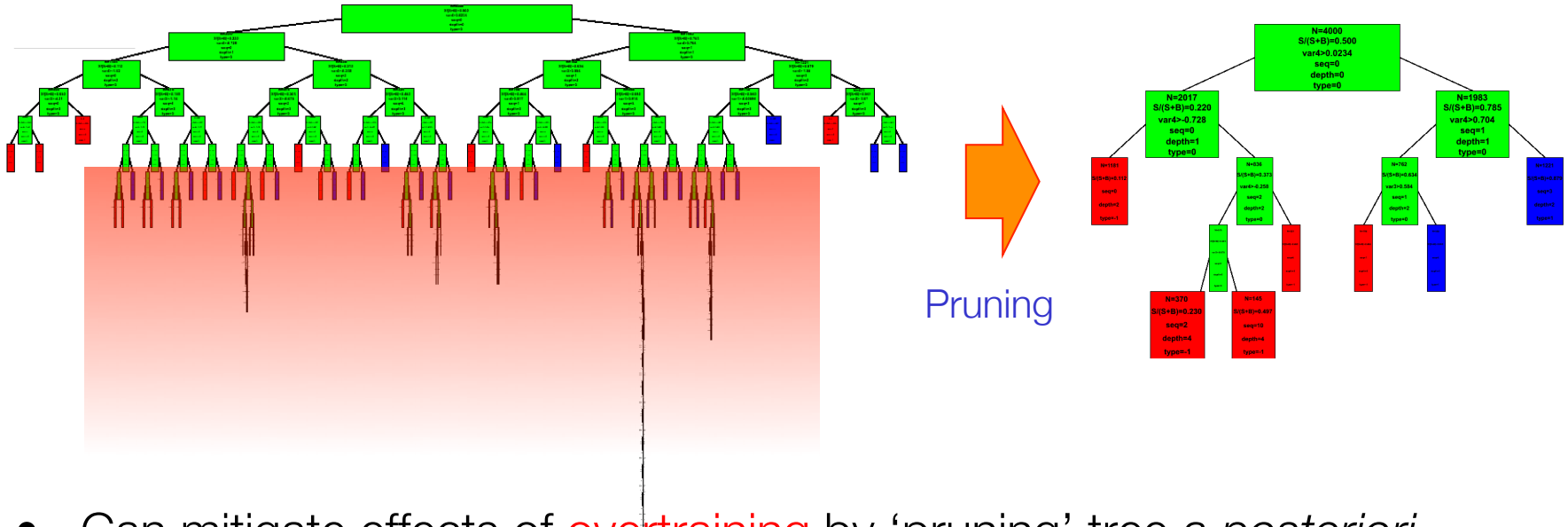
Machine learning with Decision Trees

- Stop splitting when
  - not enough improvement (introduce a cutoff Δi)
  - not enough statistics in sample, or node is pure (signal or background)
- Example decision tree from learning process



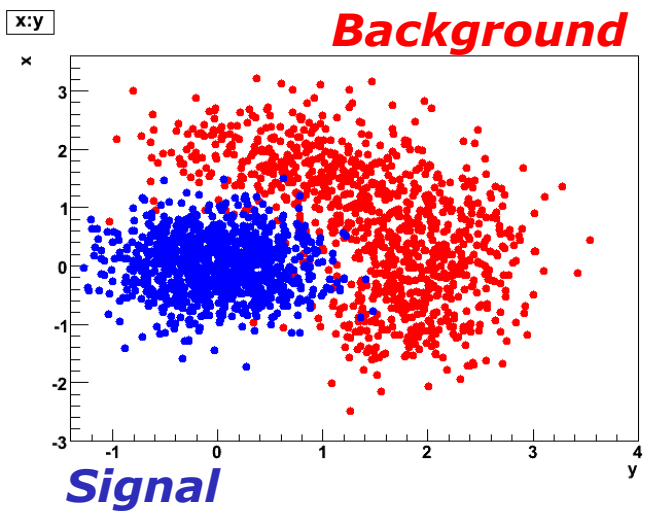
Machine learning with Decision Trees

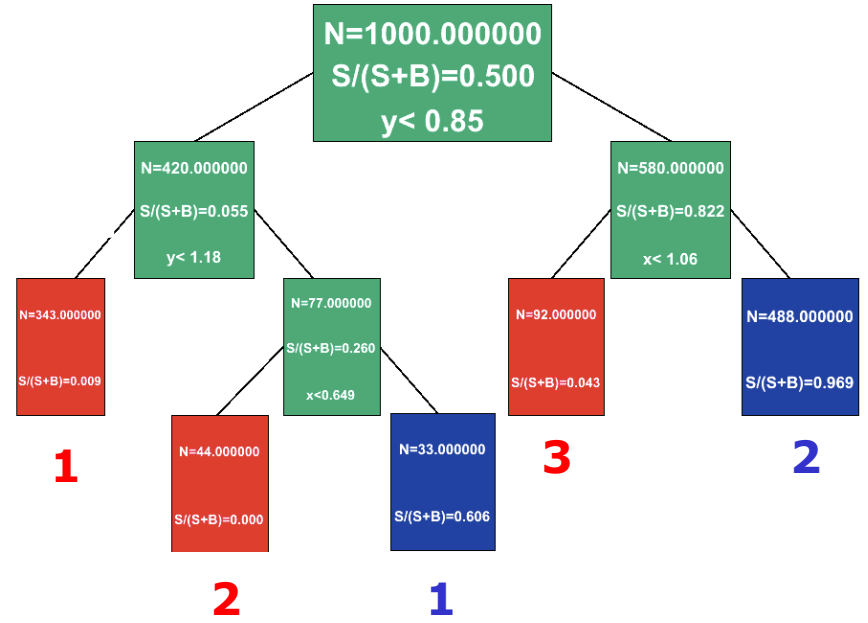
- Given that analytical pdfs f(x|s) and f(x|b) are usually not available, splitting decisions are based on 'empirical impurity' rather than true 'impurity' → risk of overtraining exists

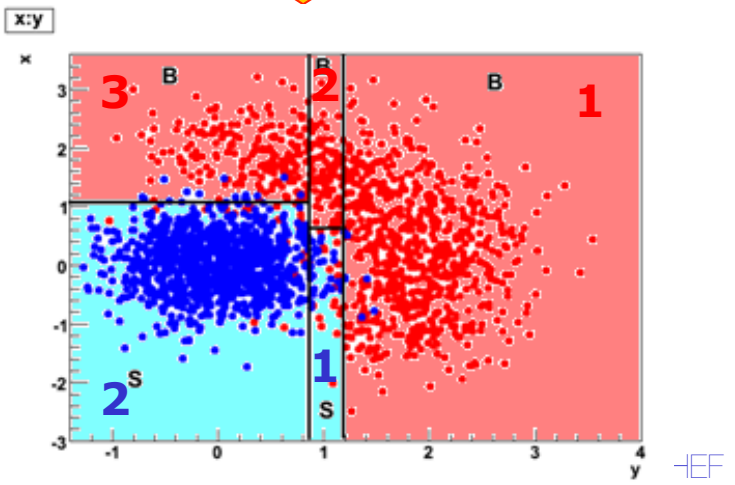


- Can mitigate effects of overtraining by 'pruning' tree *a posteriori*
  - **Expected error pruning** (prune weak splits that are consistent with original leaf within statistical error of training sample)
  - **Cost/Complexity pruning** (generally strategy to trade tree complexity against performance)

Concrete example of a trained Decision Tree







Boosted Decision trees

- Decision trees largely used with 'boosting strategy'
- **Boosting** = strategy to combine multiple weaker classifiers into a single strong classifier

- First provable boosting algorithm by Shapire (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*)

- Most used: 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

AdaBoost

- Schematic view of *iterative* algorithm
  - **Train Decision Tree** on (weighted) signal and background training samples
  - **Calculate misclassification** rate for Tree K (initial tree has k=1)

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

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

  - **Calculate weight of tree K** in 'forest decision' $\alpha_k = \beta \times \ln((1-\epsilon_k)/\epsilon_k)$
  - **Increase weight of misclassified events** in Sample(k) to create Sample(k+1)

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

- Boosted classifier is result is performance-weighted 'forest'

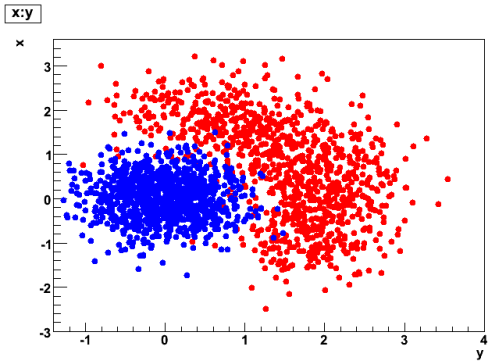
$$T(i) = \sum_{k=1}^{N_{\text{tree}}} \alpha_k T_k(i)$$

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

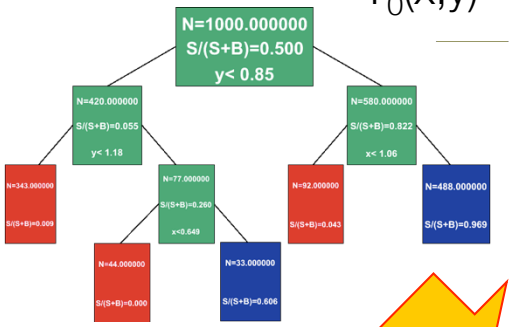
AdaBoost by example

- 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
- Note that boosting usually results in (strong) overtraining
  - Since with misclassification rate will ultimately go to zero

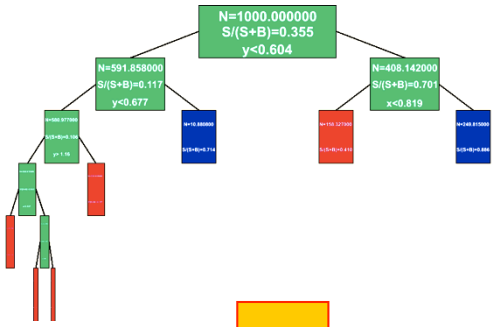
Example of Boosting



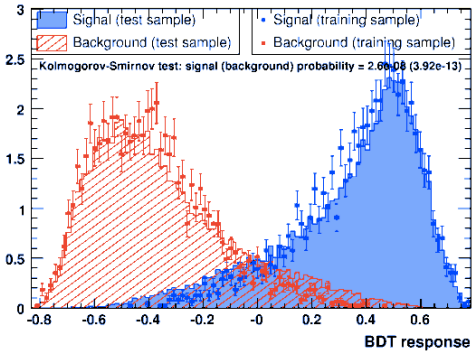
$T_0(x,y)$

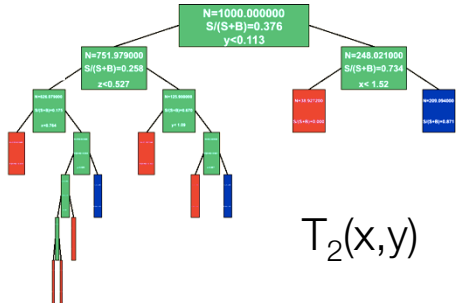


$T_1(x,y)$

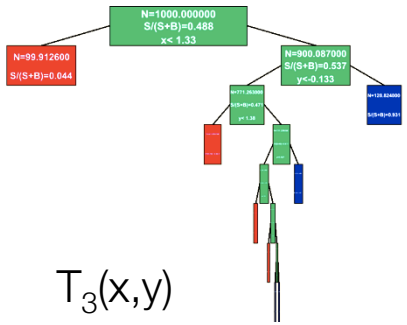


$$B(x,y) = \sum_{i=0}^{4} \alpha_i T_i(x,y)$$

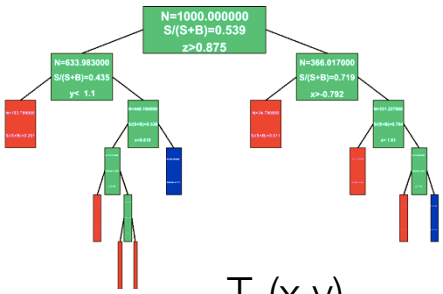




$T_2(x,y)$



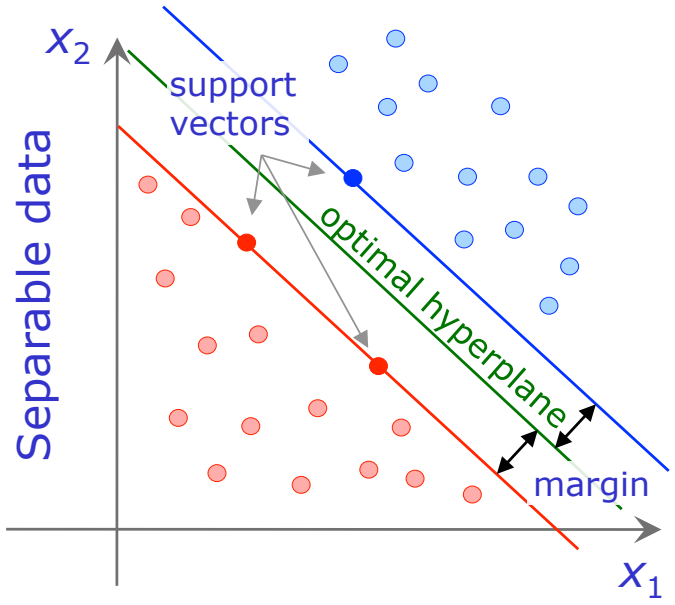
$T_3(x,y)$

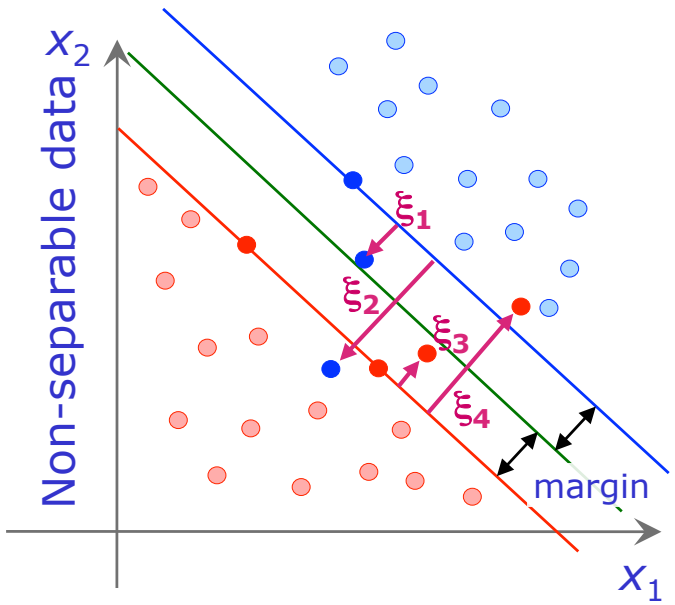


$T_4(x,y)$

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



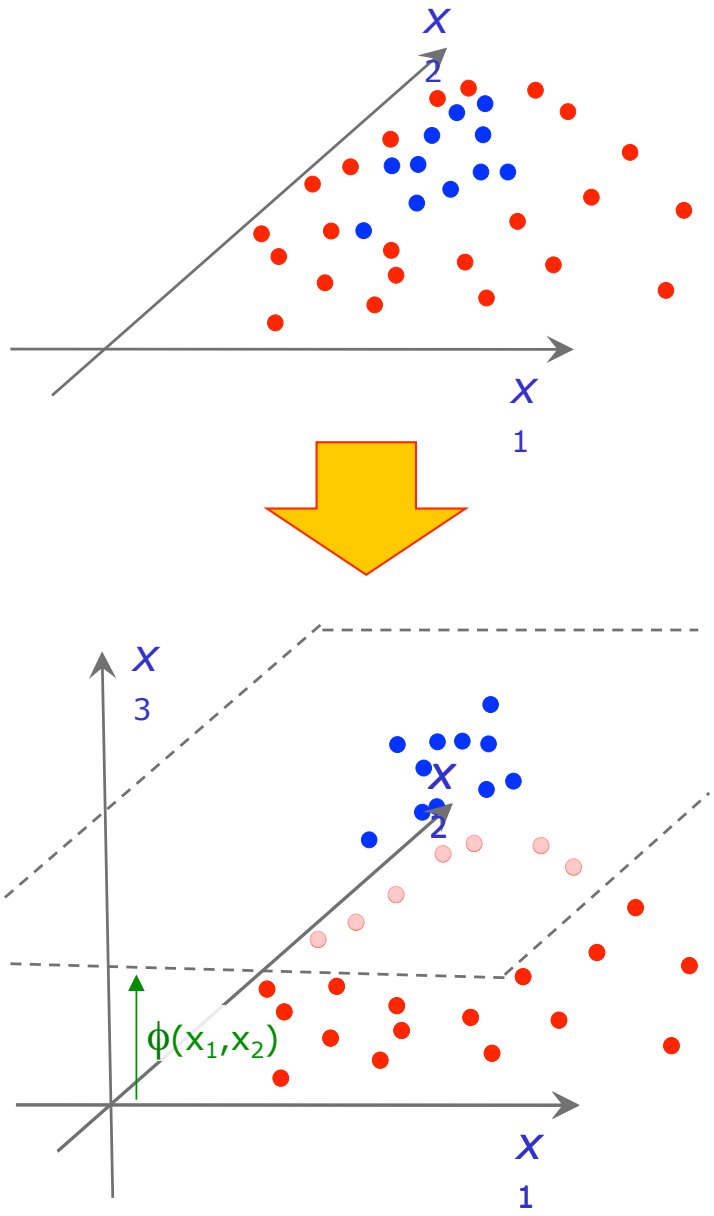


Support Vector Machines

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

    (x,y) → (x,y,z=ϕ(x,y))

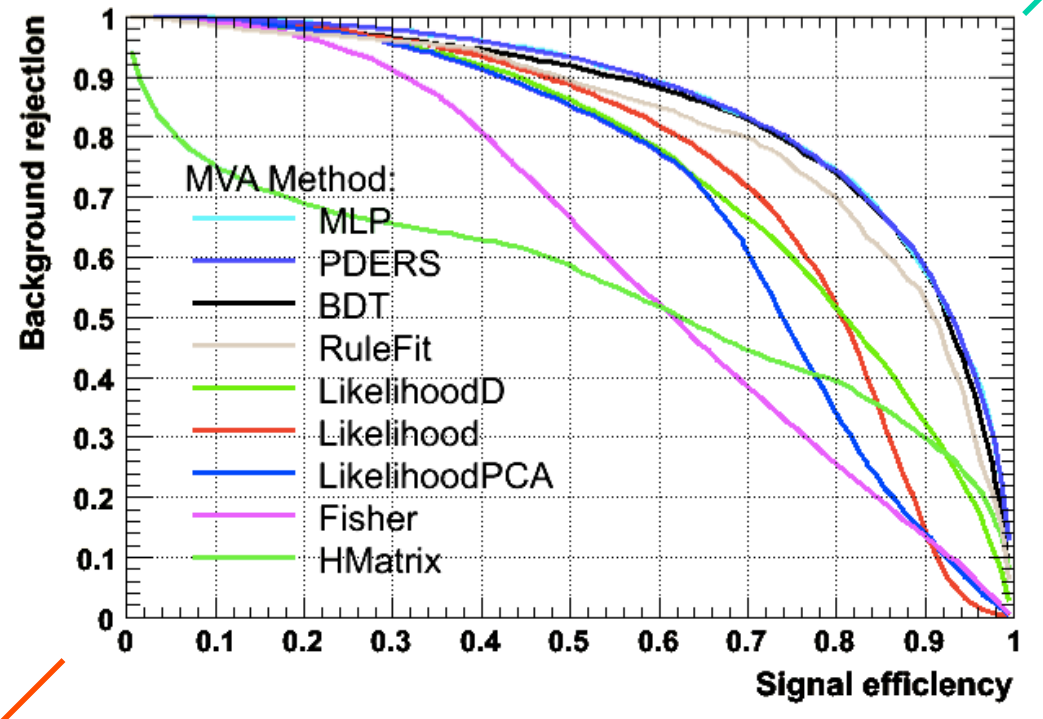
    where again a linear boundary (hyperplane) can separate the data
  - Explicit basis functions not required: use *Kernel Functions* to approximate scalar products between transformed vectors in the higher dimensional feature space
  - Choose Kernel and use the hyperplane using the linear techniques developed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



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

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

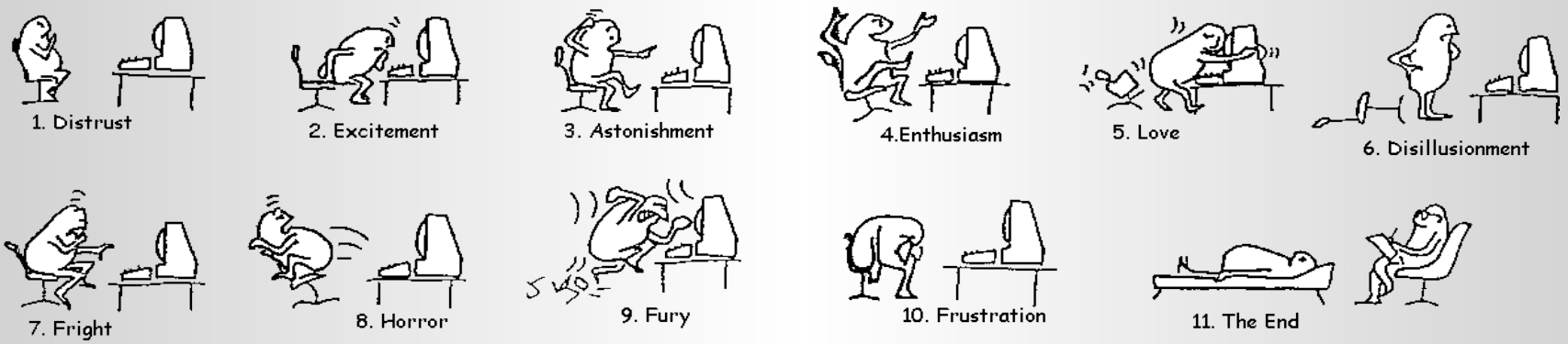
- Currently implemented data preprocessing stages:
  - Decorrelation
  - Principal Value Decomposition
  - Transformation to uniform and Gaussian distributions

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

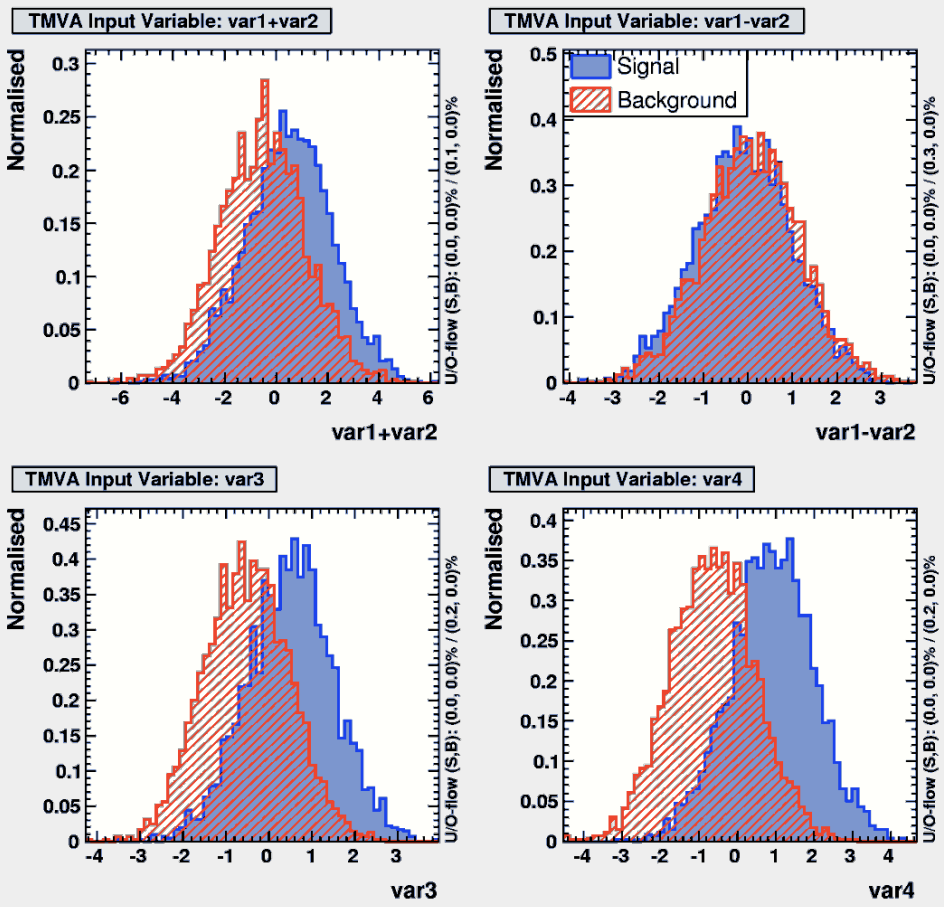
- Illustration of these steps with toy data samples

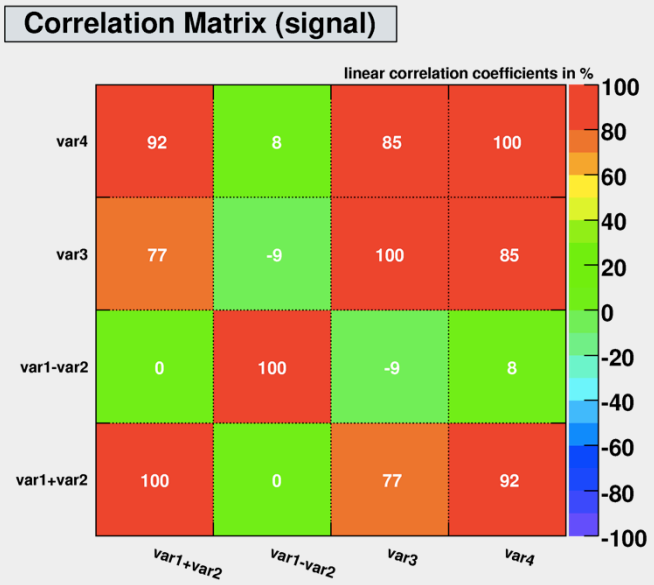


→ *T*MVA tutorial

A Toy Example (idealized)

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

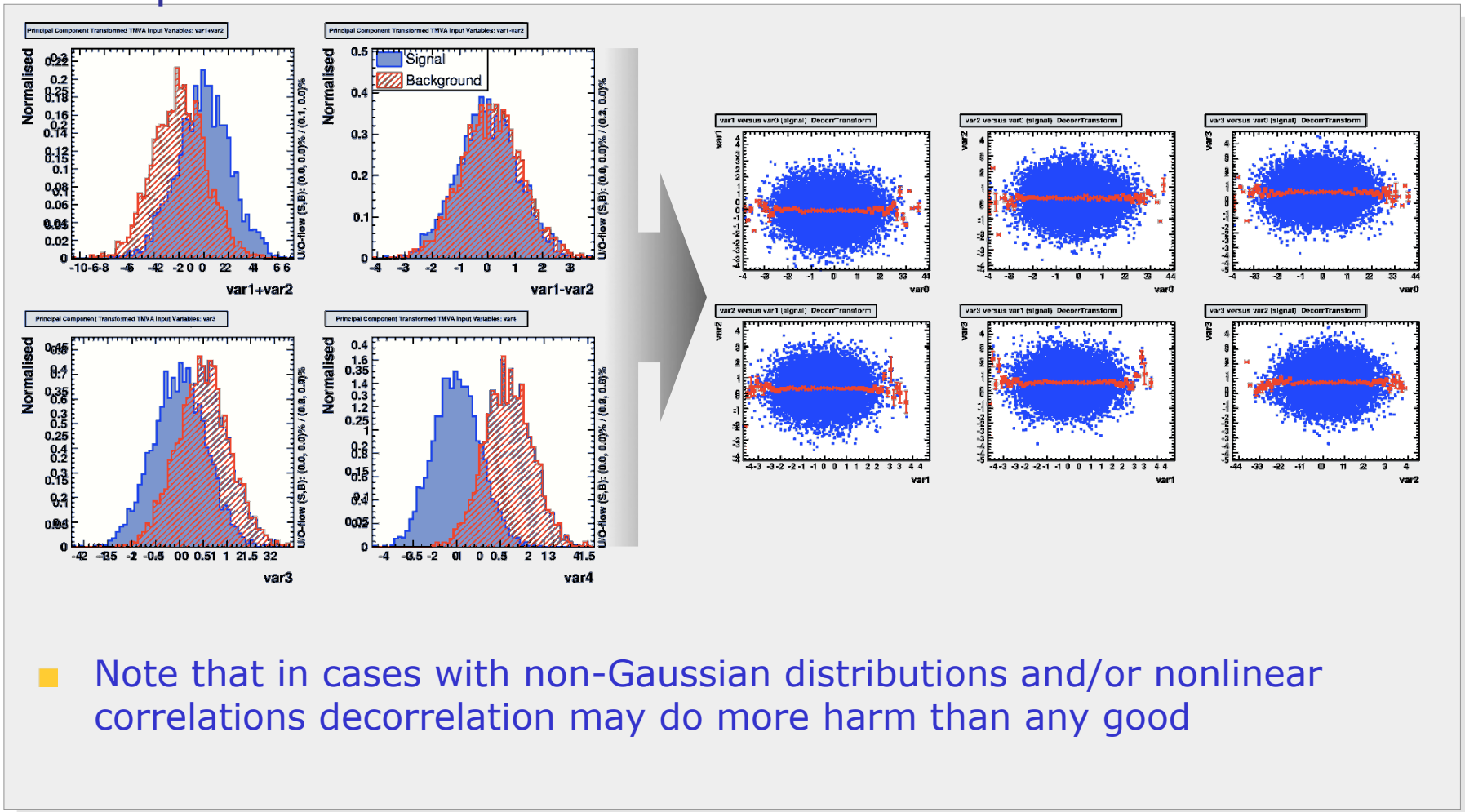





```
-------------------------------------
Rank : Variable   : Separation
-------------------------------------
   1 : var4       : 0.606
   2 : var1+var2  : 0.182
   3 : var3       : 0.173
   4 : var1-var2  : 0.014
-------------------------------------
```

Preprocessing the Input Variables

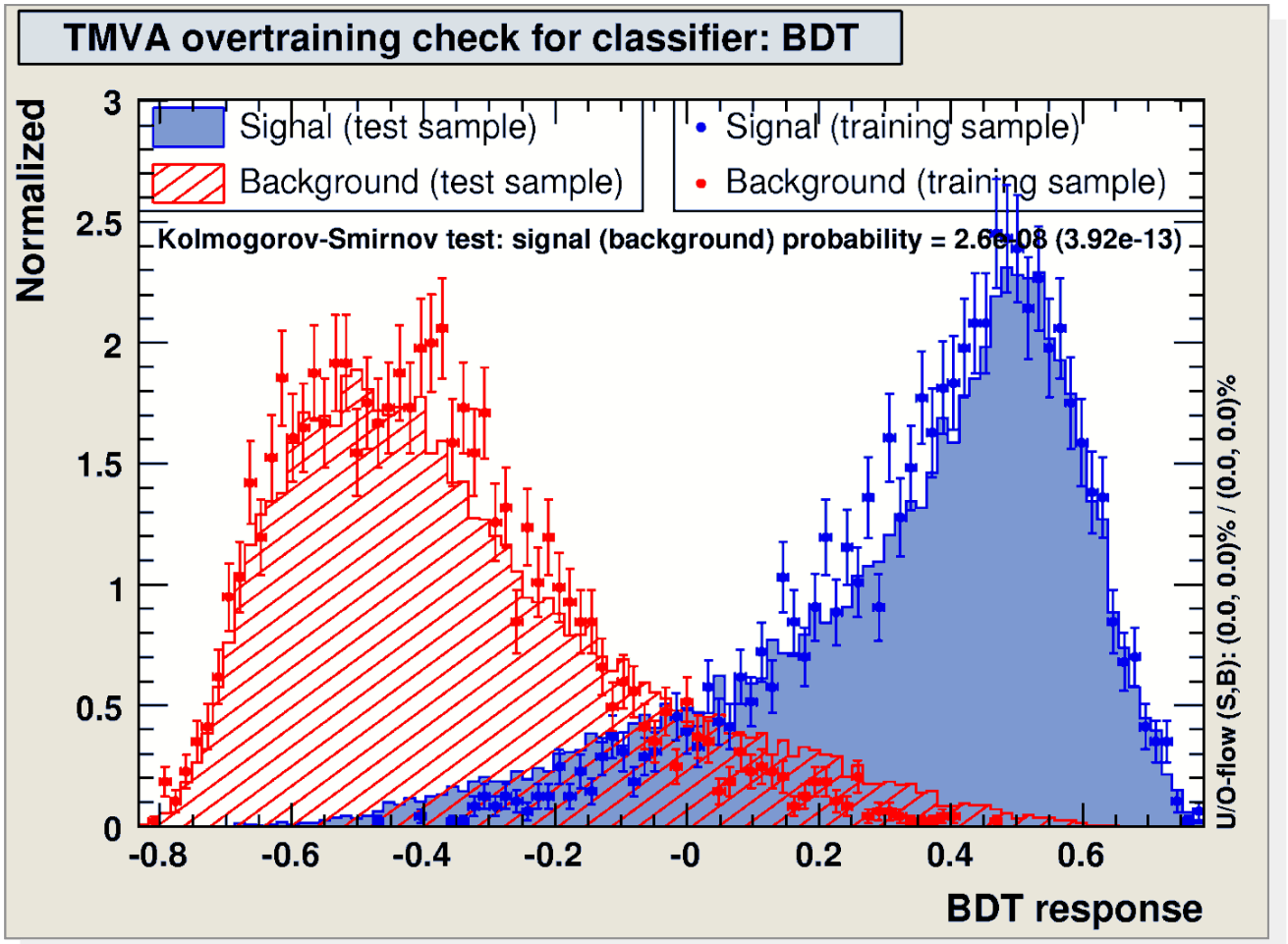
- Decorrelation of variables before training is useful for *this* example



- 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)

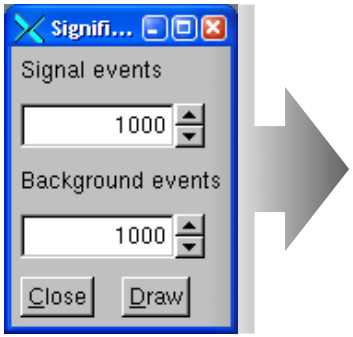
- Check for overtraining: classifier output for test *and* training samples …

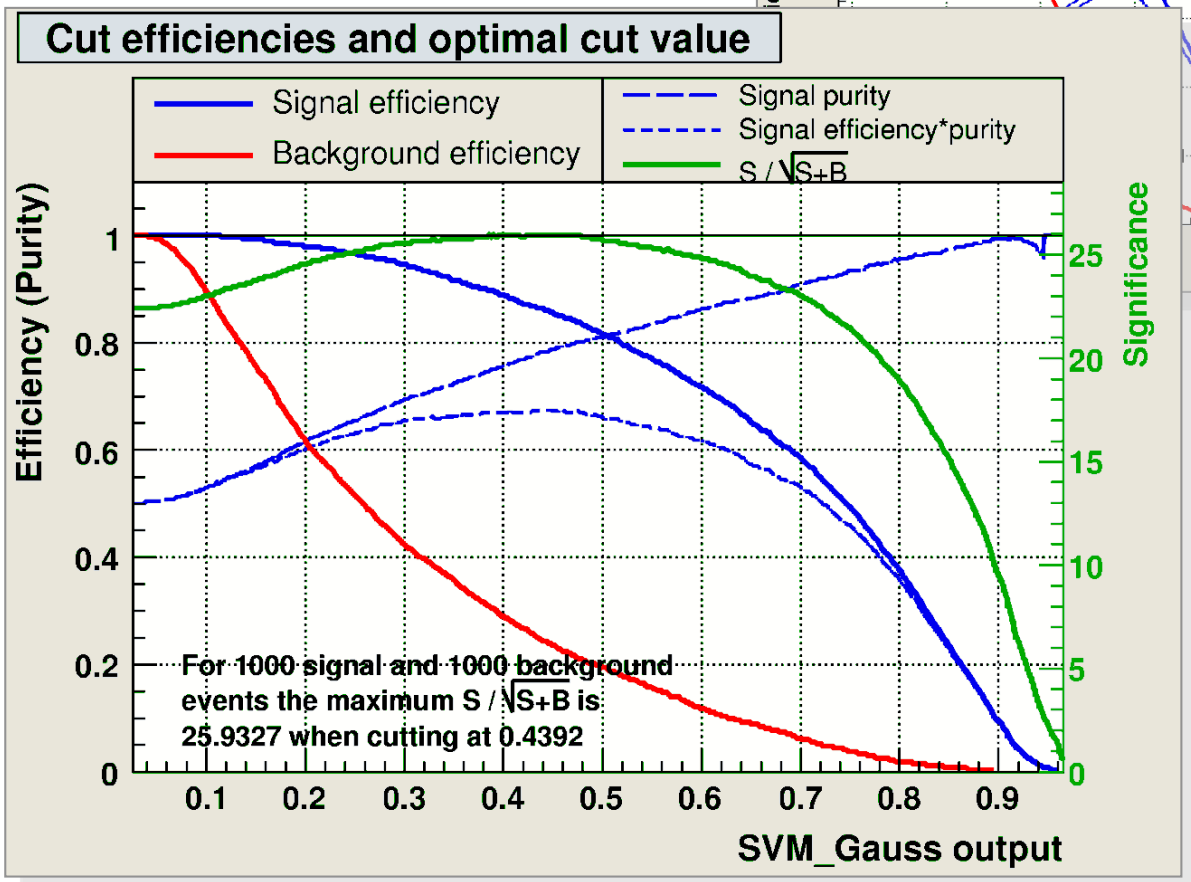


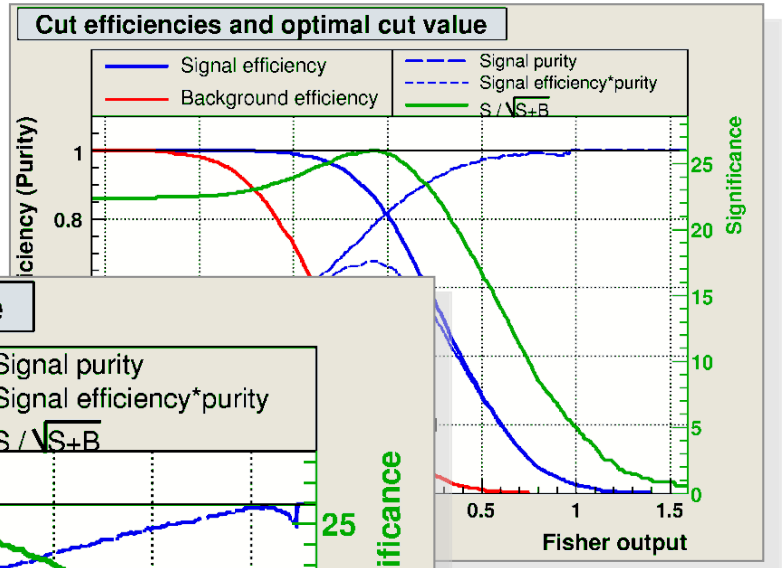
Evaluating the Classifier Training (V)

- Optimal cut for each classifiers …

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

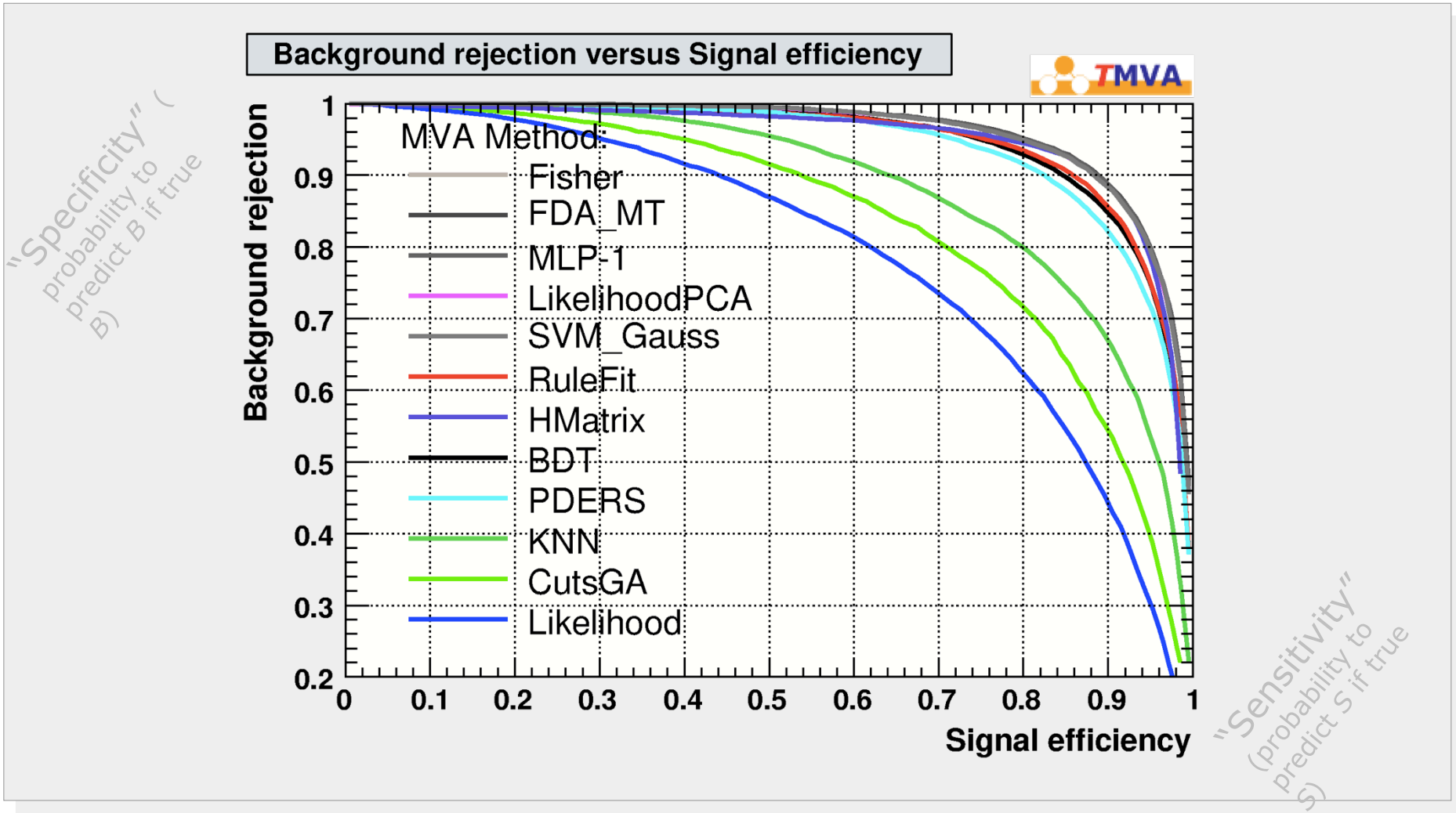






Receiver Operating Characteristics (ROC) Curve

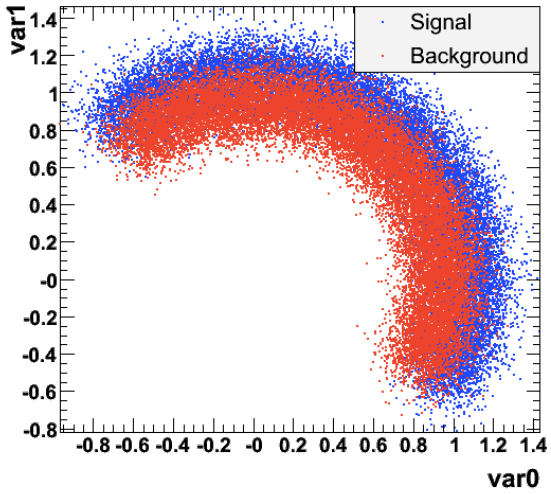
- Smooth background rejection versus signal efficiency curve: (from cut on classifier output)

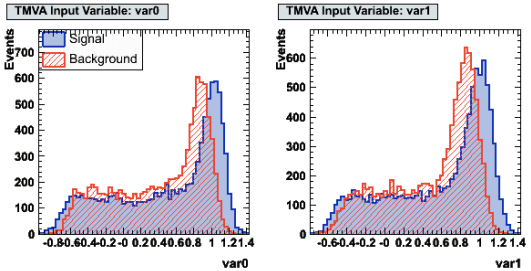


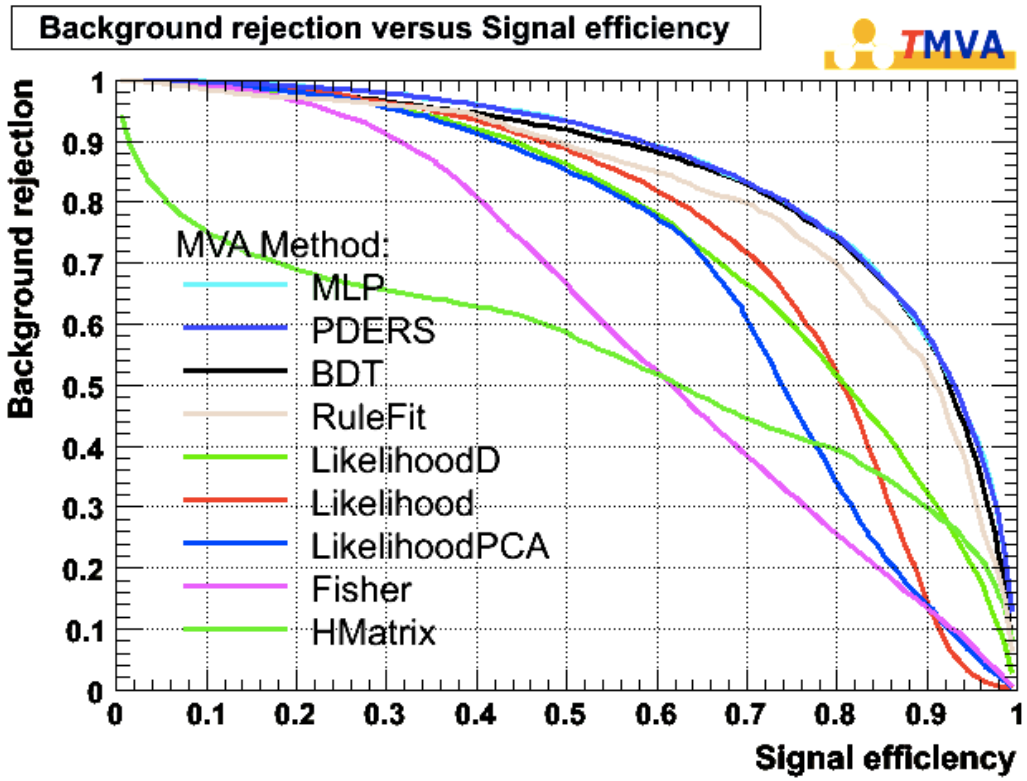
Example: Circular Correlation

- Illustrate the behavior of linear and nonlinear classifiers

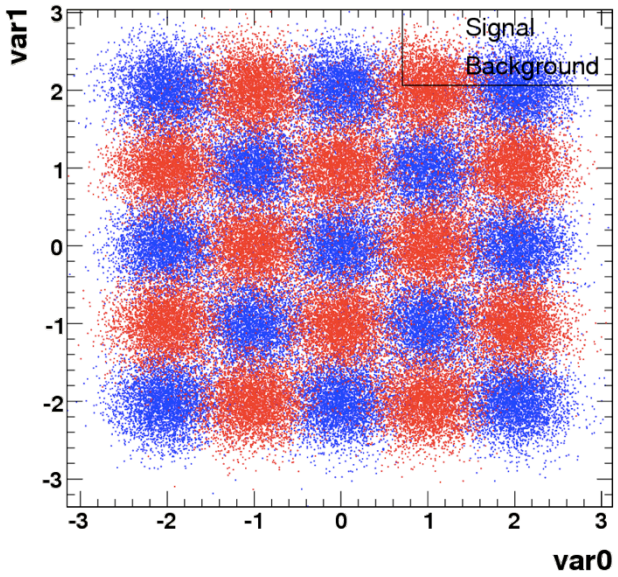
Circular correlations
(same for signal and background)

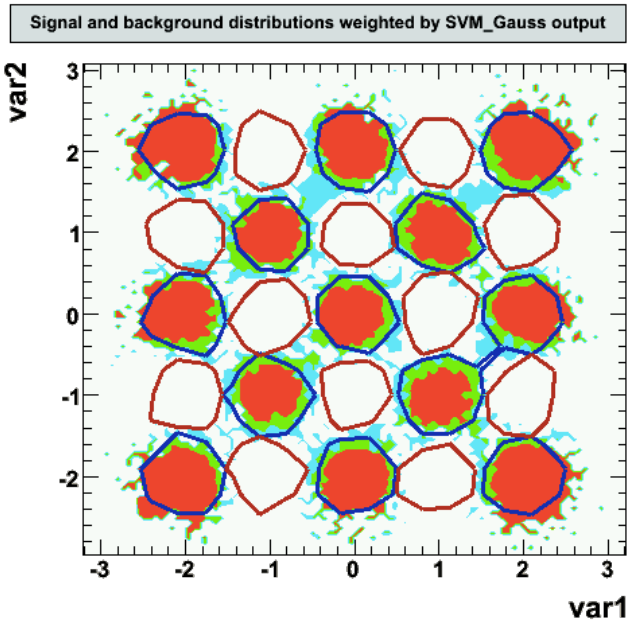




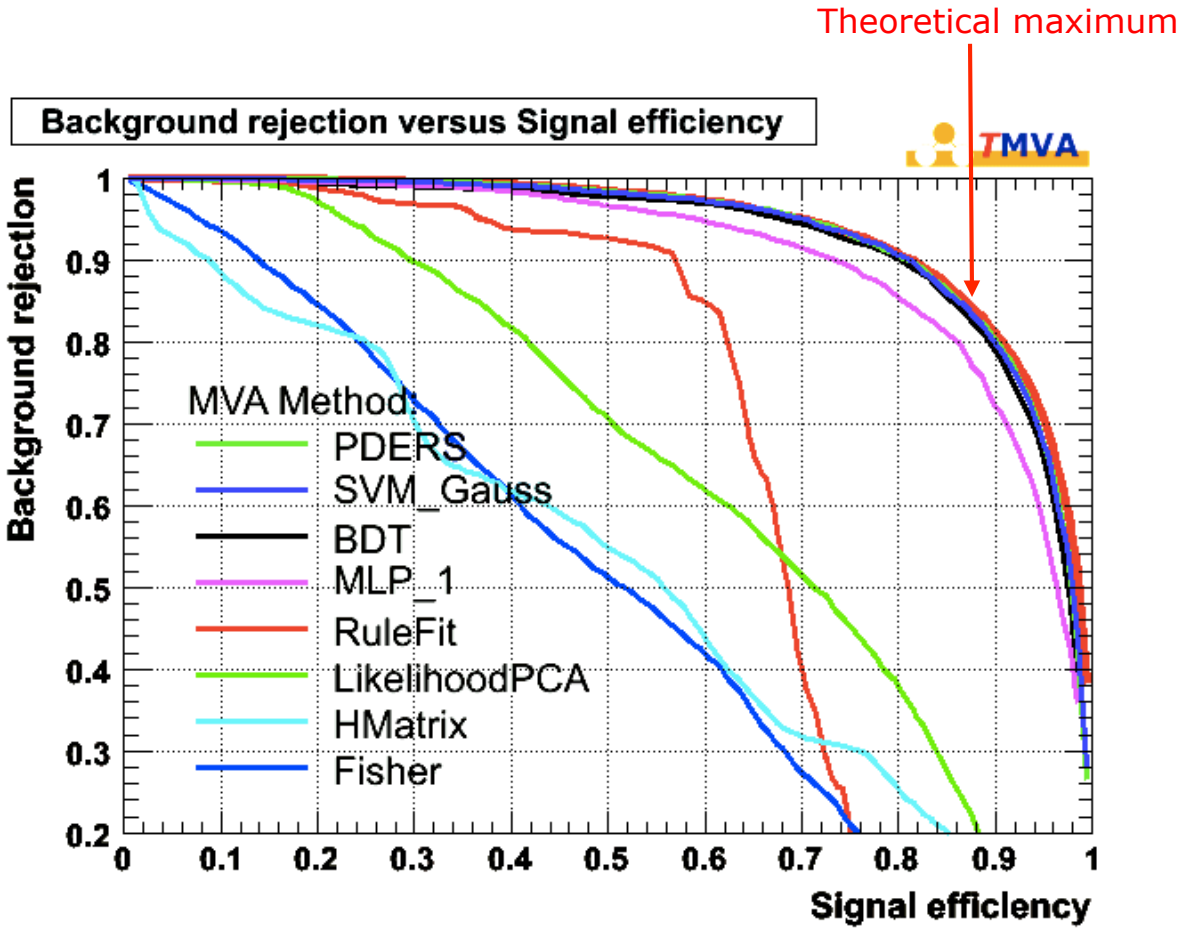


The "Schachbrett" Toy





- 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

HEP workflow versus statistical concepts

"Likelihood"

$$L(\mathbf{x} \mid H_i)$$

$\mathbf{x}_{obs}$

MC Simulated Events (sig,bkg)

All available "real data"

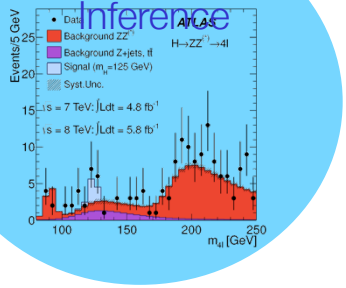
Helps to define selection

Event selection (cuts, NN, BDT)

Final Event Selection (MC)

Final Event Selection (data)

Statistical Inference



"Likelihood Ratio"

$$\lambda(\mathbf{x}) \equiv \frac{L(\mathbf{x} \mid H_{s+b})}{L(\mathbf{x} \mid H_b)} > \alpha$$

Or **approximation of optimal test statistic** with a parametric form from **machine learning**

Remaining question:
What value of α represents 'optimal cut'?

Choosing the optimal cut on the test statistic

"Likelihood"

$$L(\mathbf{x} \mid H_i)$$

$\mathbf{x}_{obs}$

MC Simulated Events (sig,bkg)

All available "real data"

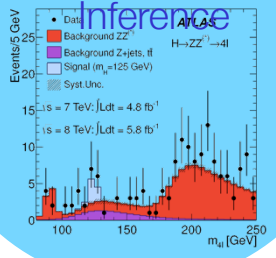
Helps to define selection

Event selection (cuts, NN, BDT)

Final Event Selection (MC)

Final Event Selection (data)

Statistical Inference



Note that in the limit of an optimal test statistic, and when subsequent using LR hypothesis test,
the cut on α has no influence on the statistical inference!

→ *Purely operational decision (ntuple-sizes etc…)*

"Likelihood Ratio"

$$\lambda(\mathbf{x}) \equiv \frac{L(\mathbf{x} \mid H_{s+b})}{L(\mathbf{x} \mid H_b)} > \alpha$$

"p-value from Likelihood Ratio test statistic"

$$p_0(\mathbf{x} \mid H_i) = \int_{\lambda_{obs}}^{\infty} f(\lambda \mid H_i)$$

Choosing the optimal cut on the test statistic

- But reality is usually more complex:
  - Test statistics are usually not optimal,
  - Ingredients to test statistics, i.e. the event selection, are usually not perfectly known (systematic uncertainties)
- *In the subsequent statistical test phase we can account for (systematic) uncertainties in signal and background models in a detailed way*. In the event selection phase we cannot
  - Pragmatically considerations in design of event selection criteria: Ability to estimate level of background from the selected data & Small sensitivity of signal acceptance to selection criteria used
- Result is that Likelihood Ratio used for event selection and final hypothesis test are different ($\lambda_{selection} \neq \lambda_{hypotest}$)
  ➔ Cut on $\lambda_{selection}$ *will* influence statistical test with $\lambda_{hypotest}$
- To be able decide on optimal cut on $\lambda_{selection}$ you need a figure merit that approximates behavior of statistical test using $\lambda_{hypotest}$

Traditional approximate Figures of Merit

- Traditional choices for Figure of Merit

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

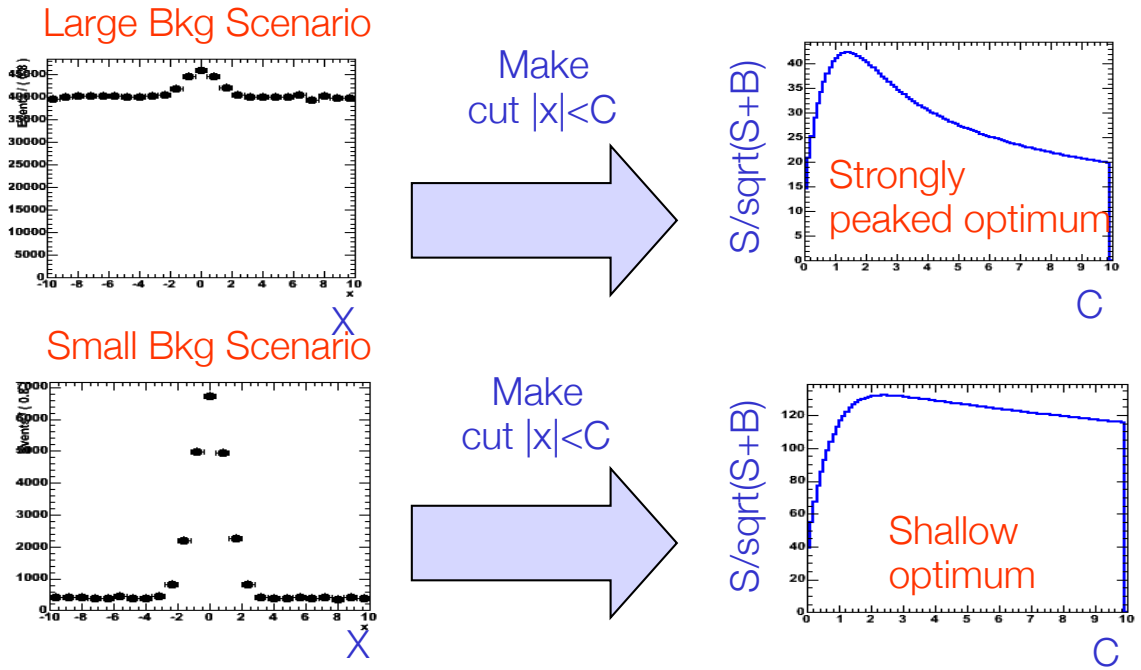
$$F(\alpha) = \frac{S(\alpha)}{\sqrt{B(\alpha)}}$$

'discovery'

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

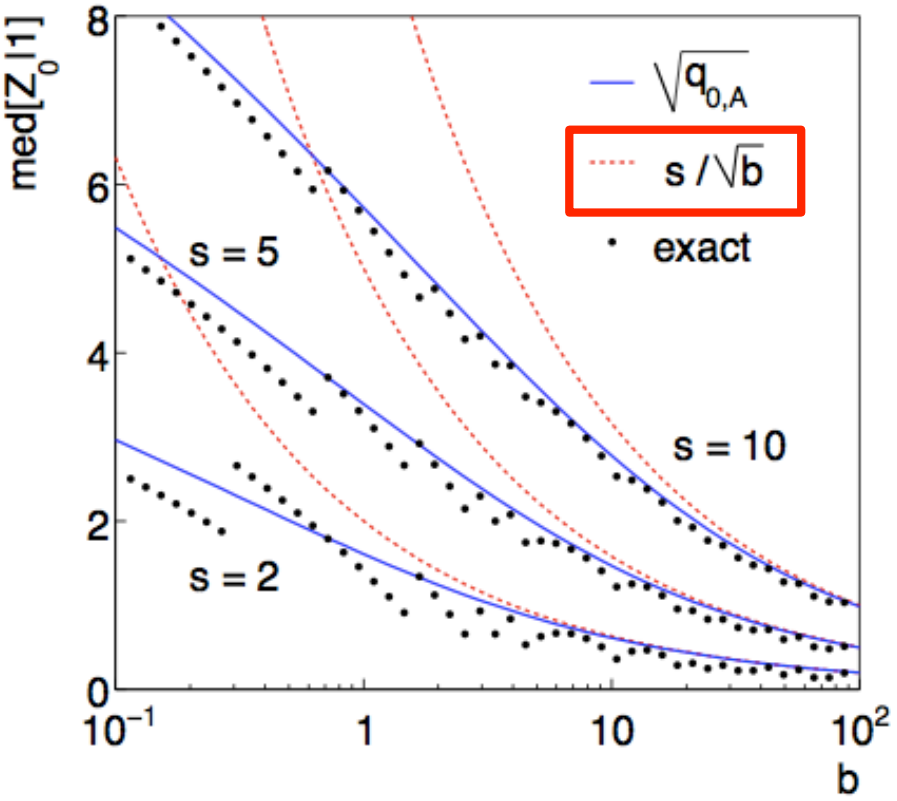
'measurement'

  - Choice of FOM for cut optimization requires assumption on subsequent statistical analysis strategy. These traditional FOMs quantify signal significance for a counting experiment with an known level of expected background, and not e.g. 'strongest upper limit', no accounting for systematic uncertainties



Validity of approximations in Figures of Merit

- Note that approximations made in 'traditional' figure of merit are not always good (even for a counting experiment!)
- E.g. for 'discovery FOM' s/√b
  illustration of approximation for s=2,5,10 and b in range [0.01-100]
  shows significant deviations of s/√b from actual significance at low b



Improved discovery F.O.M ("Asimov Z") suggested for situations where s<<b is not true

$$\sqrt{q_{0,\mathrm{A}}} = \sqrt{2\left((s+b)\ln(1+s/b)-s\right)}\,.$$

$$= \frac{s}{\sqrt{b}}\left(1+\mathcal{O}(s/b)\right)\,.$$

Final comments on event selection

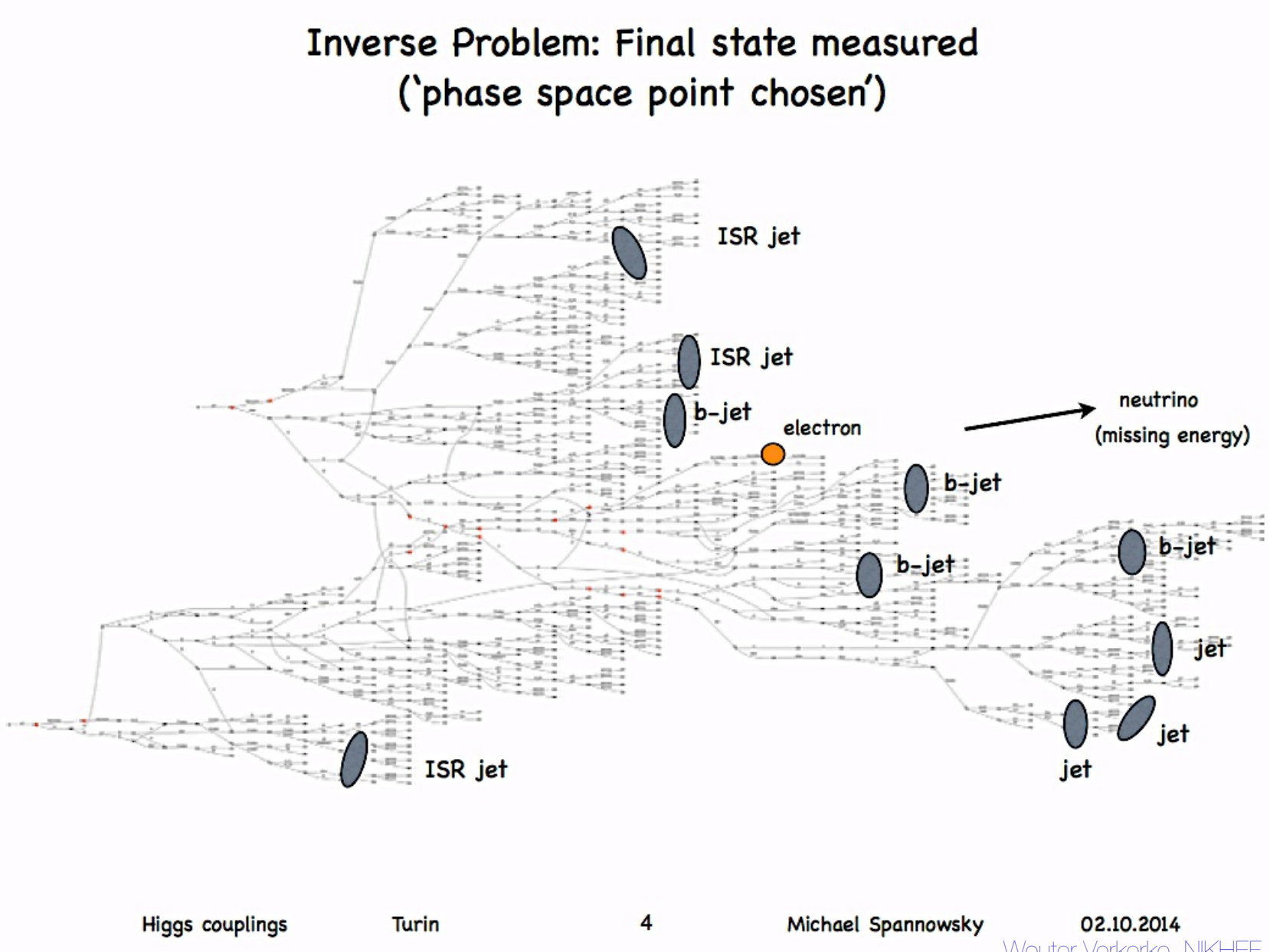
- Main issue with event selection is usually, sensitivity of selection criteria to systematic uncertainties
- What you'd like to avoid is your BDT/NN that is trained to get a small statistical uncertainty has a large sensitivity to a systematic uncertainties
- No easy way to incorporate effect of systematic uncertainties in training process

  → Can insert some knowledge of systematic uncertainties included in figure of merit when deciding where to cut in BDT/NN, but proper calculation usually requires much more information that signal and background event counts and is time consuming
- Use your physics intuition…

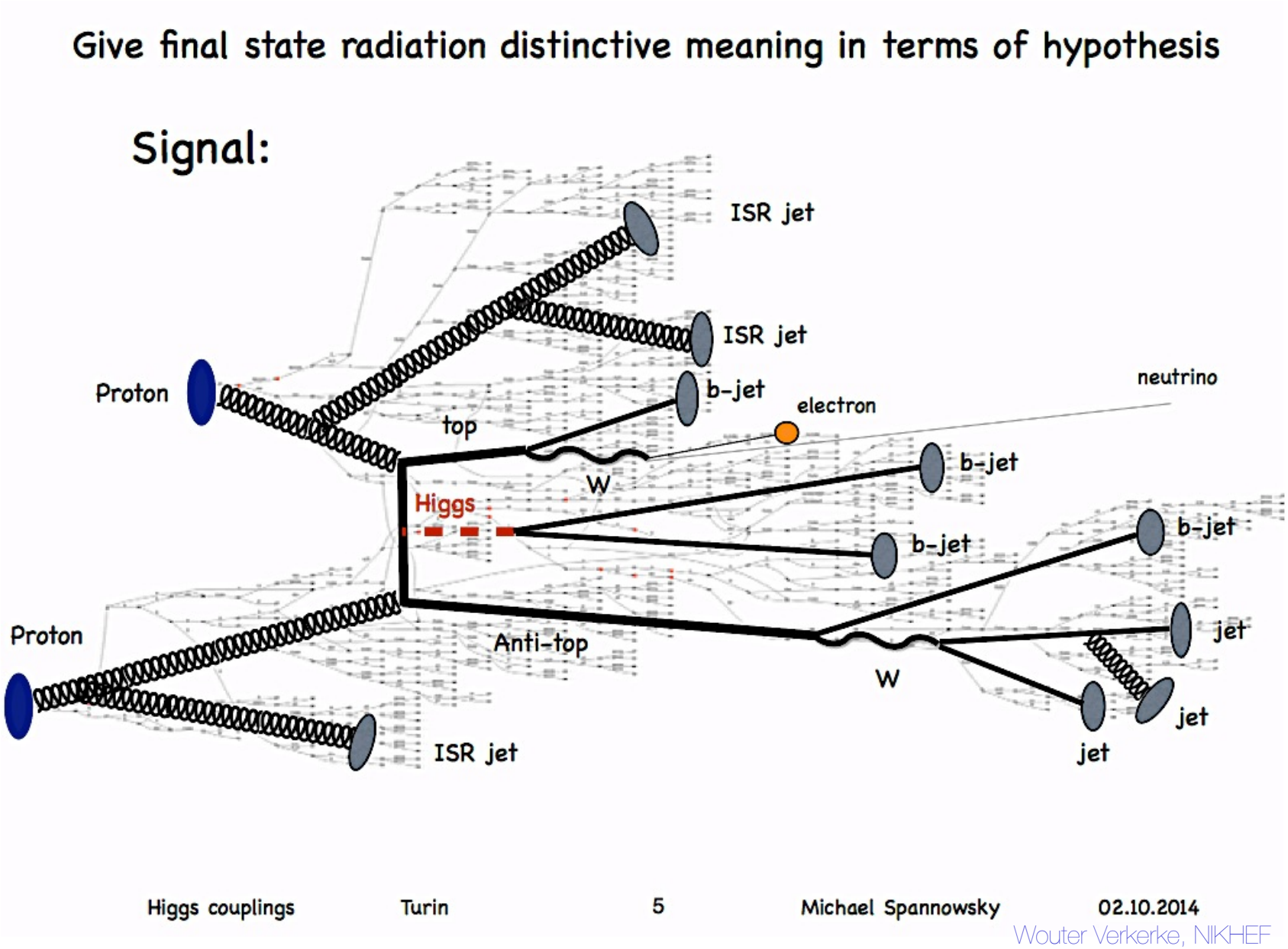
Alternatives to Machine Learning, back to NP optimal discrimant…

- Machine learning or Bayesian learning approach doesn't use detailed physics knowledge of signal and background processes that is available inside simulation to achieve separation
  - Only through final distribution S(x) and B(x) that is implicitly provided through MC simulation samples
- Another approach is to exploit full information of physics simulation process better to construct S(x) and B(x) and construct (optimal) NP discriminant from these → **Matrix Element Methods**
- Idea is to inject knowledge of the hard physics processes as encoded in physics simulation directly into discriminant and approximate effects of detector reconstruction through so-called 'transfer functions'
  - At level of hard physics simulation, calculation of probability model for truth-level quantities still tractable (although still relatively expensive)
  - Add effects of parton showers and detector resolution a posteriori with transfer functions

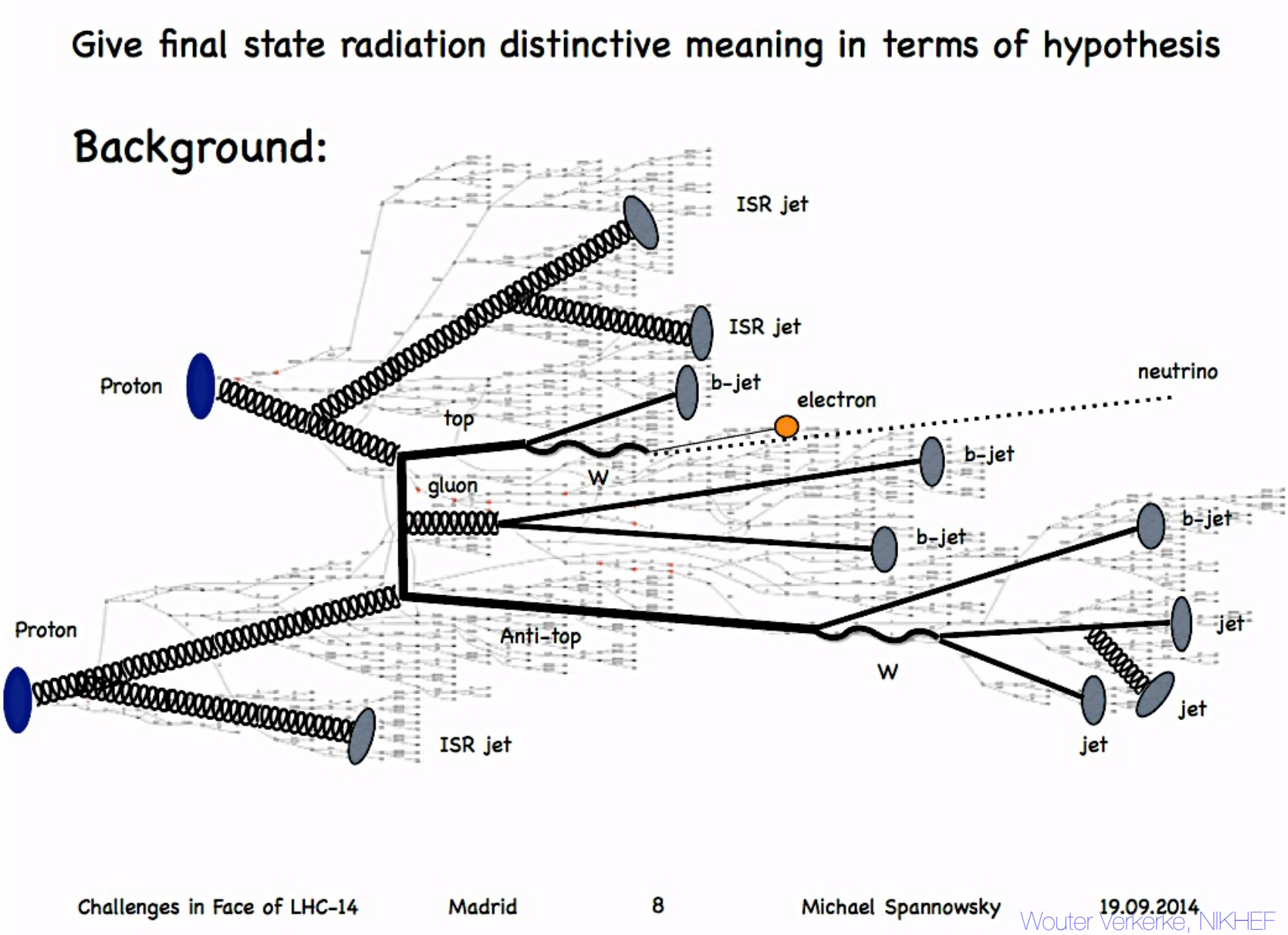
The Matrix Element Method



The Matrix Element Method

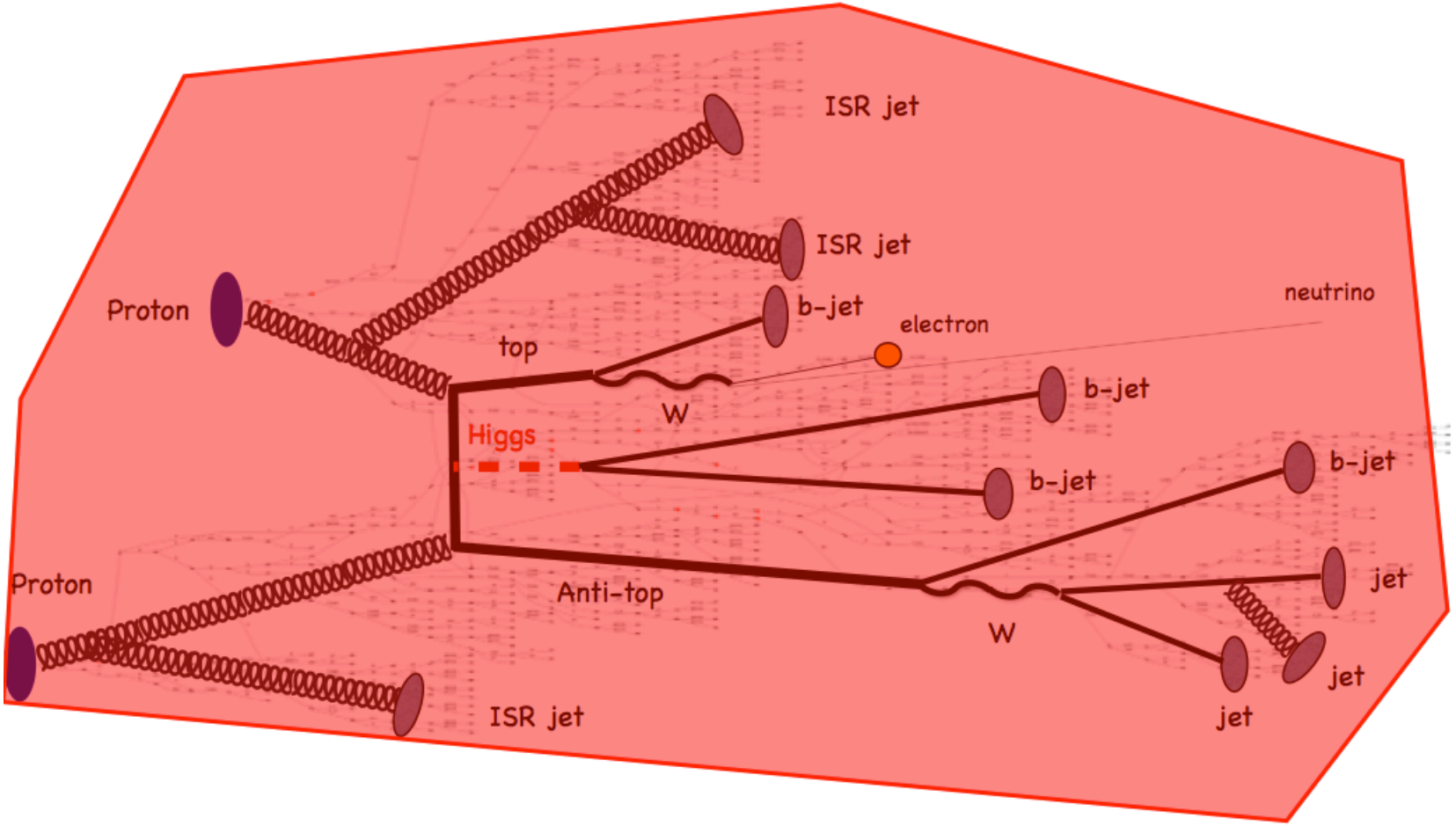


The Matrix Element Method



The Matrix Element Method

Ideally one would like to use all radiation related to hard process to discriminate signal from background



The Matrix Element Method

The matrix element method in a nutshell:

Y = parton-level final state
X = reconstruction-level final state

Given a theoretical assumption α, attach a weight $P(\mathbf{x}, \alpha)$ to each experimental event **x** quantifying the validity of the theoretical assumption α for this event.

$$P(\mathbf{x}, \alpha) = \frac{1}{\sigma} \int d\phi(\mathbf{y}) |M_\alpha|^2(\mathbf{y}) W(\mathbf{x}, \mathbf{y})$$

$P(x,\alpha) = L(x|H_\alpha)$
$\alpha = S,B$

$|M_\alpha|^2$ is squared matrix element = S(y) or B(y) from theory (=calculable!)

$W(\mathbf{x}, \mathbf{y})$ is the resolution or transfer function

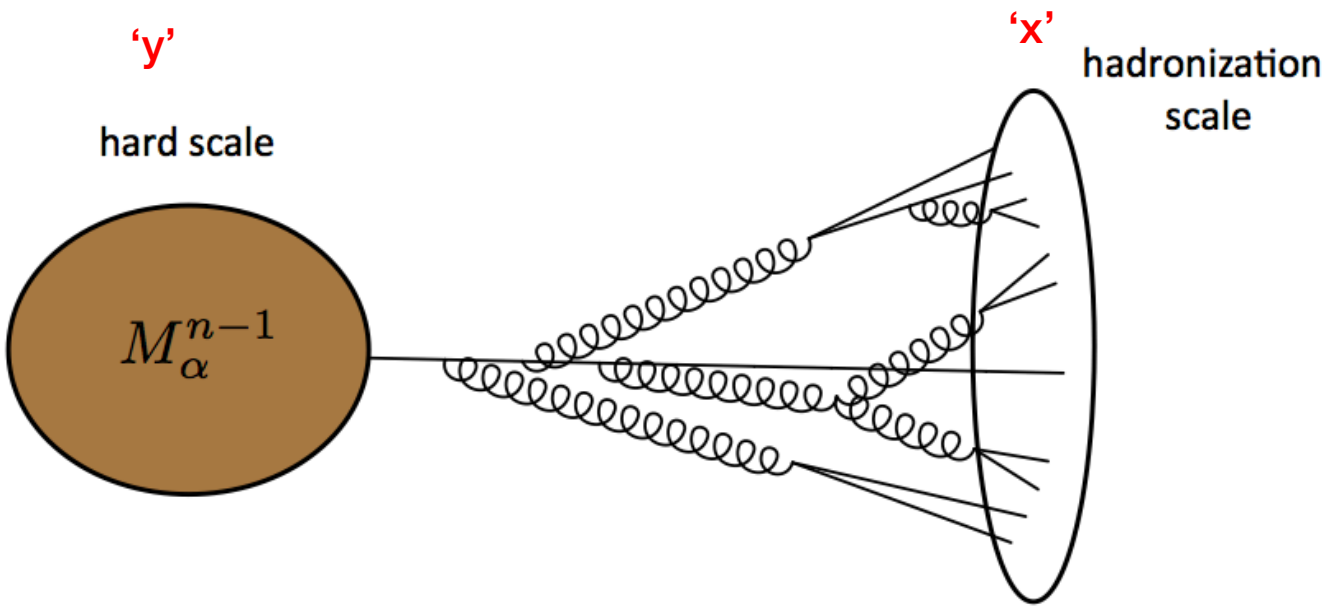
$d\phi(\mathbf{y})$ is the parton-level phase-space measure

The value of the weight $P(\mathbf{x}, \alpha)$ is the probability to observe the experimental event **x** in the theoretical frame α

$\lambda_{MEM} = P(x,S)/p(x,B)$

The Matrix Element Method

Purpose of the transfer function is to match jets to partons



Probability density function: $\int d\mathbf{y}\ W(\mathbf{x},\mathbf{y}) = 1$

The Matrix Element Method

W(x|y) = p.d.f for observable quantities x, given parton-level theory observables y

The form of the transfer function:

resolution in

$$W(\mathbf{x},\mathbf{y}) \approx \Pi_i \frac{1}{\sqrt{2\pi}\sigma_{E,i}} e^{-\frac{(E_i^{rec}-E_i^{gen})^2}{2\sigma_{E,i}^2}}$$ Energy

$$\times \frac{1}{\sqrt{2\pi}\sigma_{\phi,i}} e^{-\frac{(\phi_i^{rec}-\phi_i^{gen})^2}{2\sigma_{\phi,i}^2}}$$ azimuthal angle

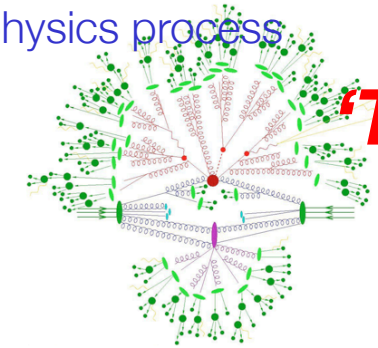
$$\times \frac{1}{\sqrt{2\pi}\sigma_{y,i}} e^{-\frac{(y_i^{rec}-y_i^{gen})^2}{2\sigma_{y,i}^2}}$$ rapidity

Complex, high-dimensional gaussian distribution!

Transfer function introduces new peaks on top of propagators

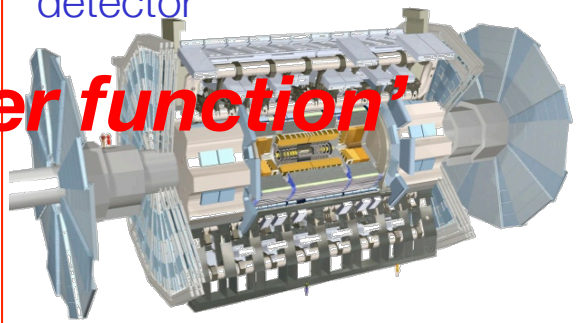
An overview of HEP data analysis procedures

Simulation of 'soft physics' physics process



'Transfer function'

Simulation of ATLAS detector



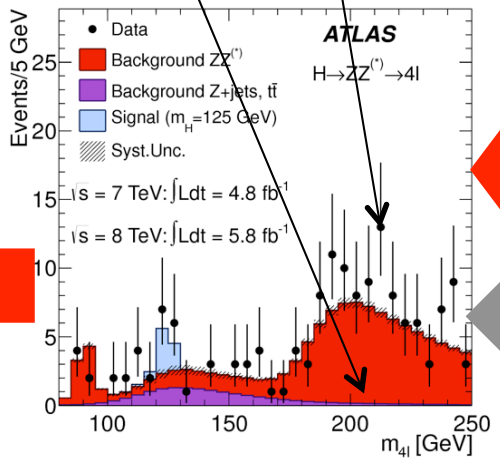
LHC data

Simulation of high-energy physics process

'Matrix Element'

$P(m_{4l}|SM[m_H])$

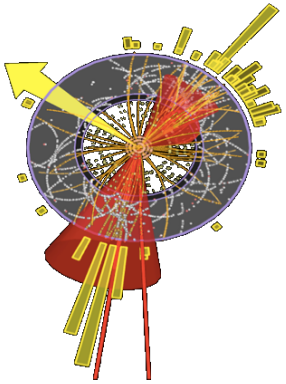
Observed m_{4l}



prob(data|SM)

Analysis Event selection

Reconstruction of ATLAS detector



The Matrix Element Method

Subtleties of the convolution $|M(y)|^2 \times W(y,x)$

1) $|M(y)|^2$

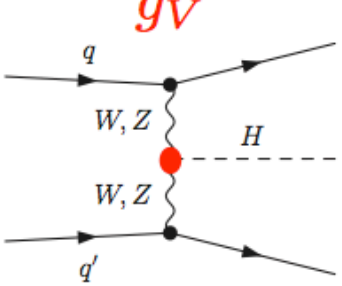
- Can be calculated at different order in pert. series (LO, NLO)
- Final state multiplicity fixed (exclusive process)
- Some kinematic configurations induce large logs (need resummation)

2) $W(y,x)$

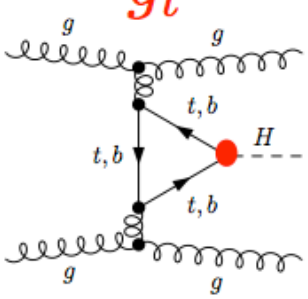
- Number of final state objects limited to exclusive process
- Integration very time consuming -> limits final state multiplicity
- Transfer function fit dependent (input from experiment)

The Matrix Element Method

Example

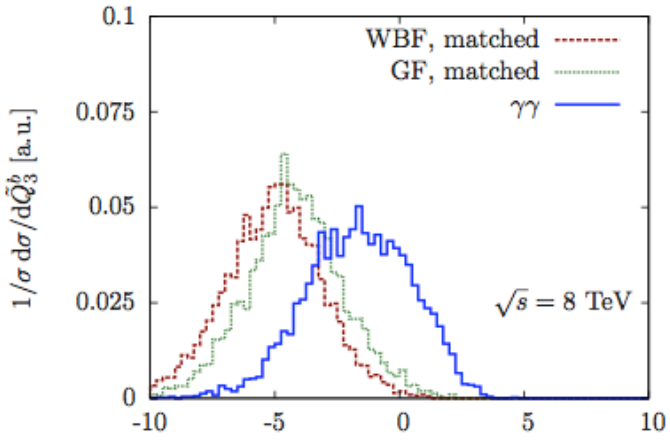


vs



in $H \to \gamma\gamma$

Higgs reconstructed, but no transfer function for jets:



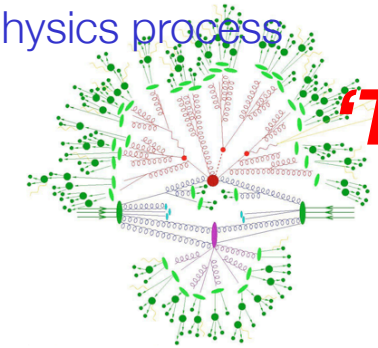
$S/B \nearrow 100\%$

$$\tilde{Q}_n^b(p_1^\gamma, p_2^\gamma, \{p_n^j\}) = -\log\left[\frac{\{|\mathcal{M}^{\mathrm{WBF}}(pp \to (h \to \gamma\gamma)j^n)|^2 + |\mathcal{M}^{\mathrm{GF}}(pp \to (h \to \gamma\gamma)j^n)|^2\}}{|\mathcal{M}^{2\gamma}(pp \to \gamma\gamma j^n)|^2}\right]$$

Likelihood-Ratio based on Matrix Elements

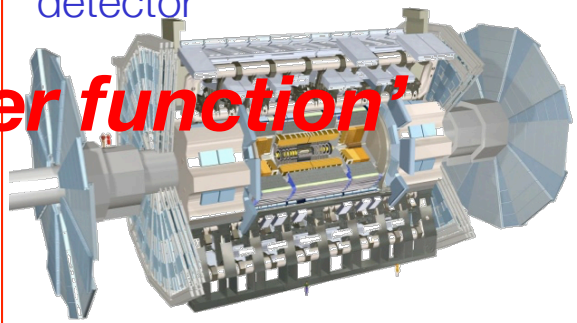
An overview of HEP data analysis procedures

Simulation of 'soft physics' physics process



'Transfer function'

Simulation of ATLAS detector



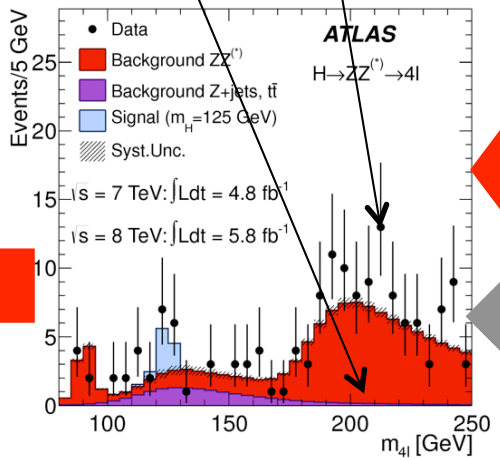
LHC data

Simulation of high-energy physics process

'Matrix Element'

$P(m_{4l}|SM[m_H])$

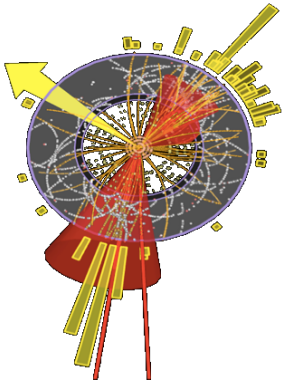
Observed m_{4l}



prob(data|SM)

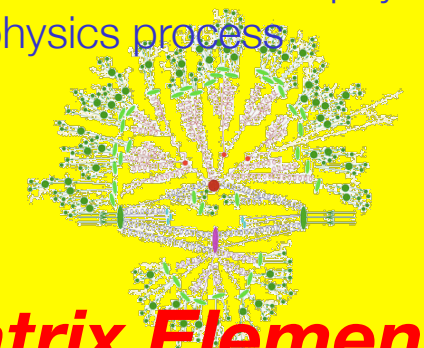
Analysis Event selection

Reconstruction of ATLAS detector



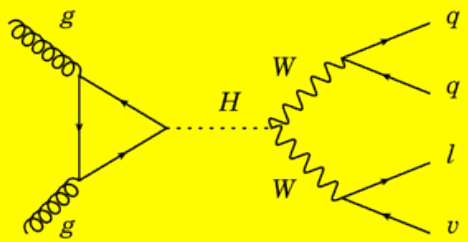
An overview of HEP data analysis procedures

Simulation of 'soft physics' physics process

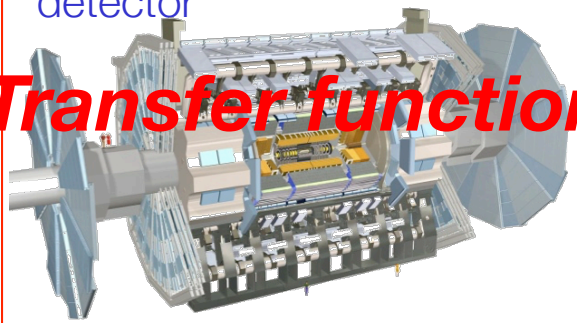


'Matrix Element +Parton Shower'

Simulation of high energy physics process



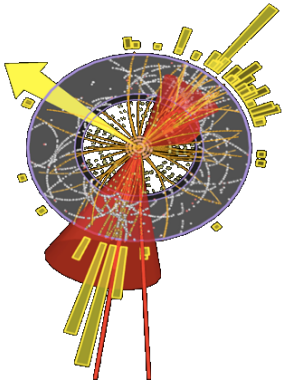
Simulation of ATLAS detector



'Transfer function'

LHC data

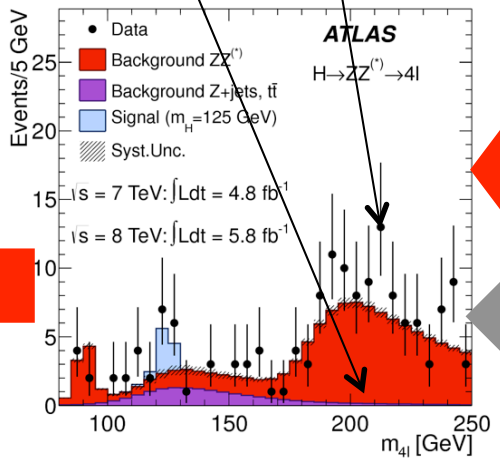
Reconstruction of ATLAS detector



Analysis Event selection

$P(m_{4l}|SM[m_H])$

Observed m_{4l}

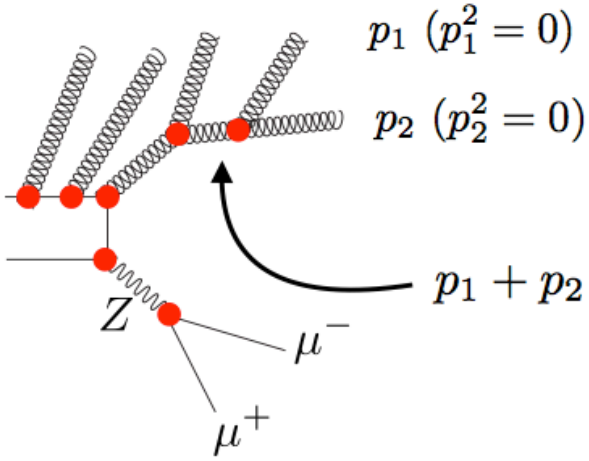


prob(data|SM)

The Matrix Element Method

Remove limitation of final state objects on $|M(y)|^2$

Shower approximation for matrix element, i.e. **shower deconstruction:**



partons from the hard interaction emit other partons (gluons and quarks)

These emissions are enhanced if they are collinear and/or soft with respect to the emitting parton

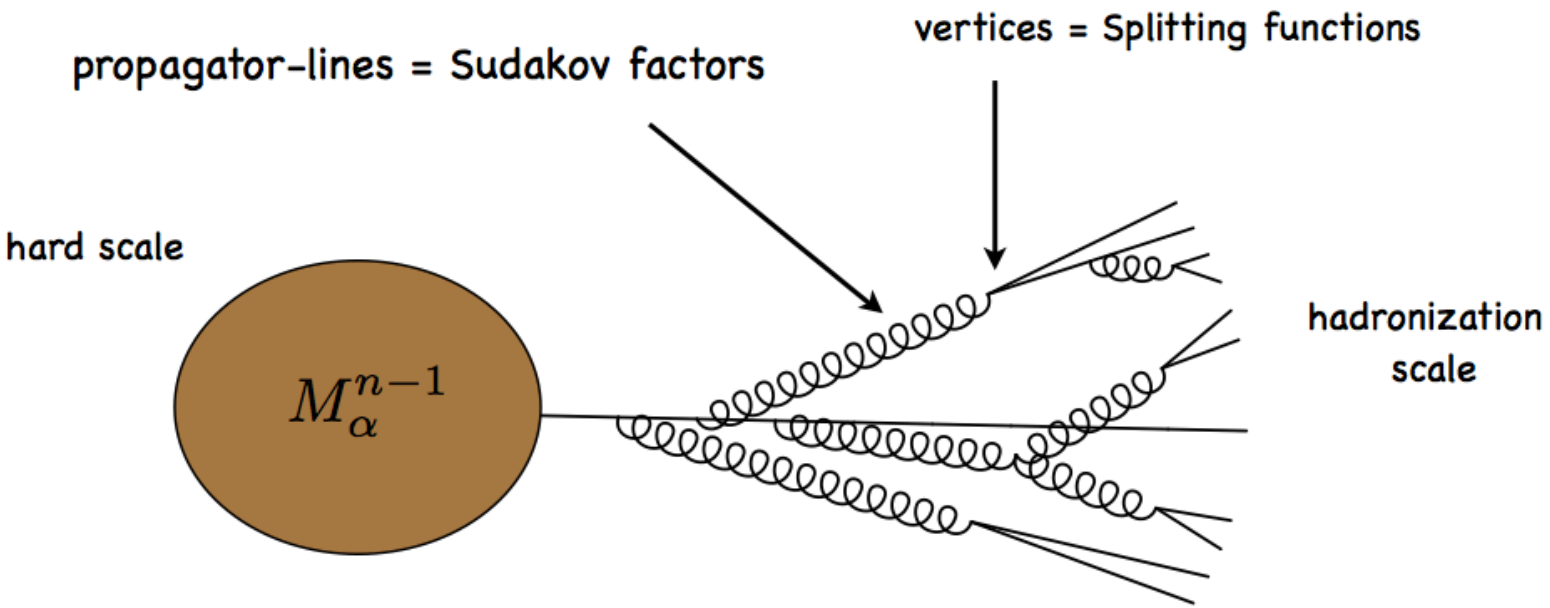
Probability enhanced in soft and collinear region due to ~ $1/(p_1 + p_2)^2$

- If $p_1 \to 0$, then $1/(p_1 + p_2)^2 \to \infty$
- If $p_2 \to 0$, then $1/(p_1 + p_2)^2 \to \infty$
- If $p_2 \to \lambda p_1$, then $1/(p_1 + p_2)^2 \to \infty$

The Matrix Element Method

Factorization of emissions in soft/collinear limit

and Sudakov factors allow semiclassical approximation of quantum process:



Can calculate weight for shower history iteratively

Can use smaller objects and more objects (more information)

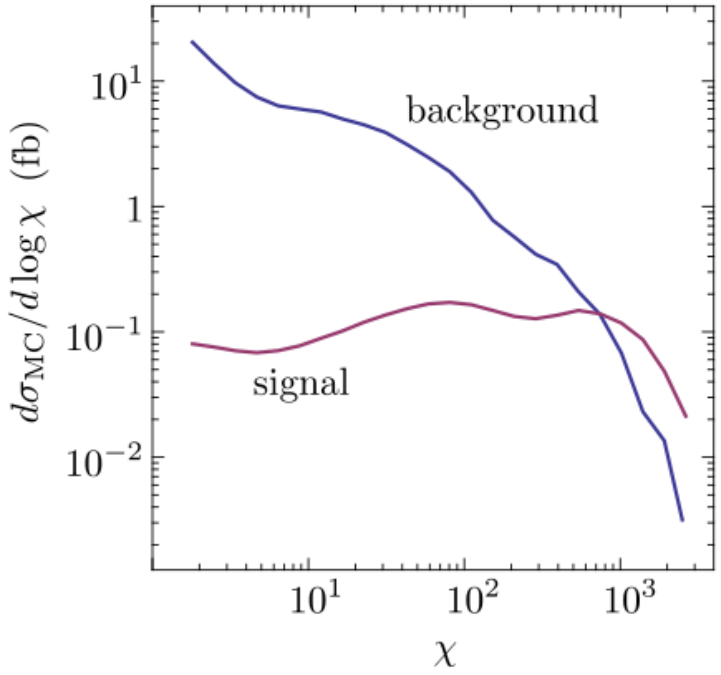
The Matrix Element Method

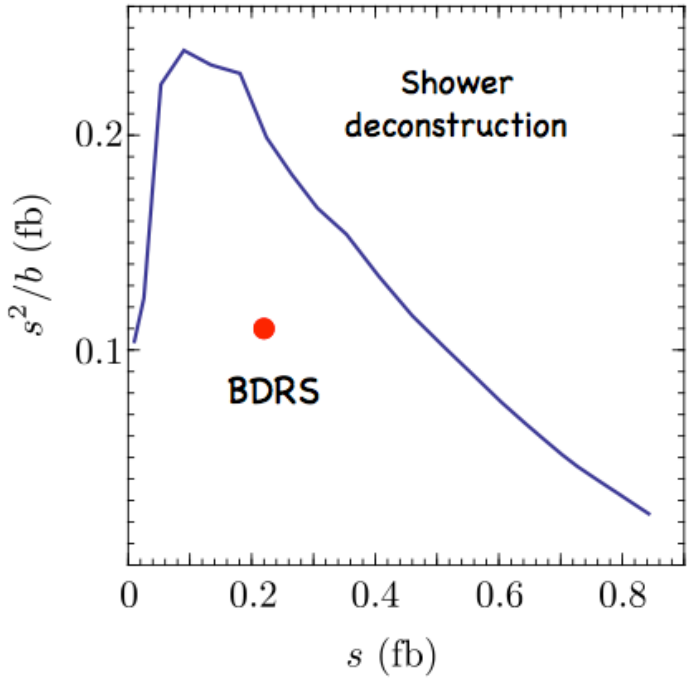
Likelihood-Ratio based on Matrix Elements with Parton Shower deconstruction

Results for Higgs boson:

$$\chi(\{p,t\}_N) = \frac{P(\{p,t\}_N|\mathrm{S})}{P(\{p,t\}_N|\mathrm{B})}$$

[Soper, MS PRD 84 (2011)]

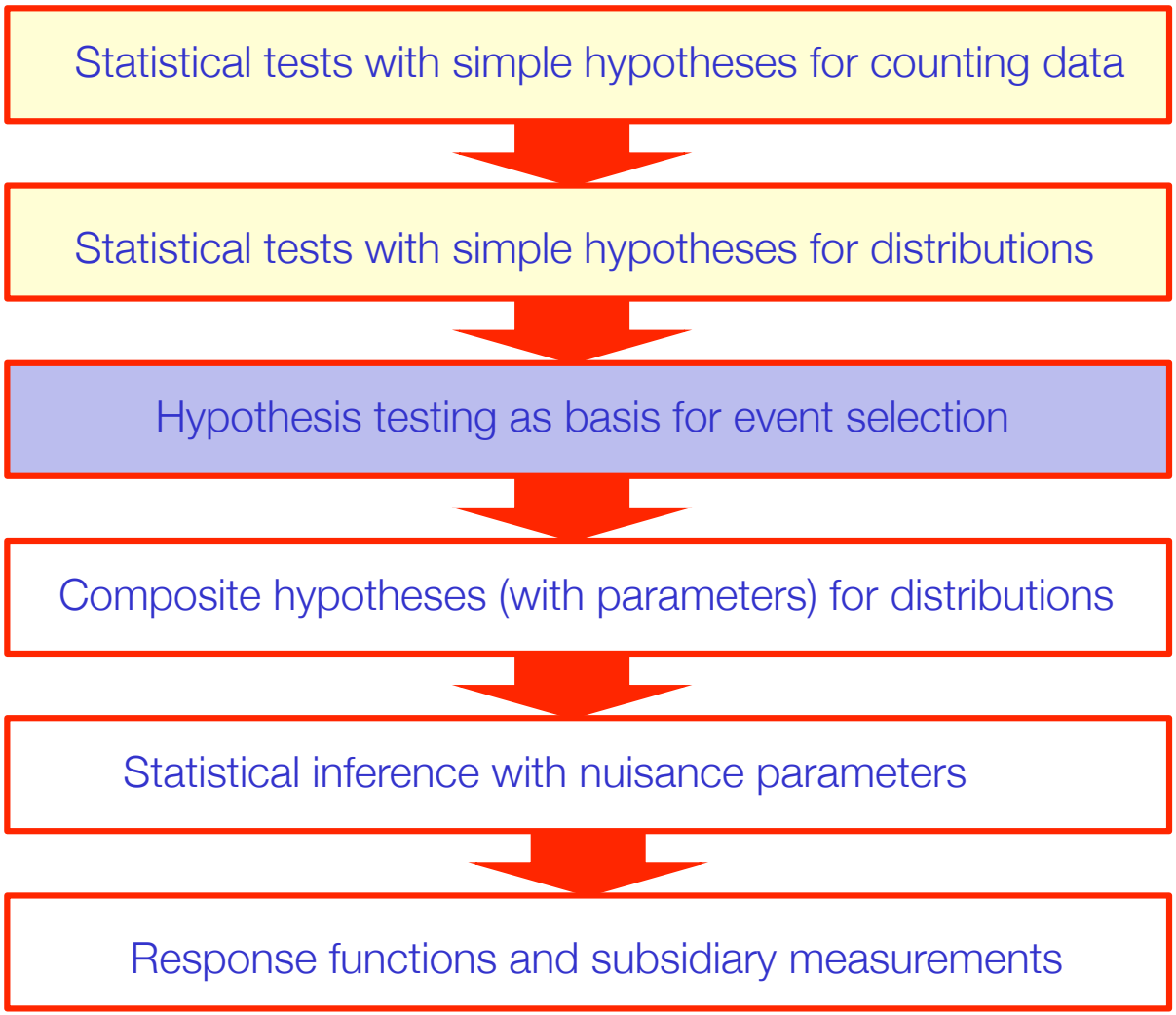




imperfect b-tagging (60%,2%) no b-tag required

Roadmap for this course

- Start with basics, gradually build up to complexity of



"What do we mean with probabilities"

"p-values"

"Optimal event selection & machine learning"

"Confidence intervals, Maximum Likelihood"

"Fitting the background"

"Sideband fits and systematic uncertainties"