Machine learning

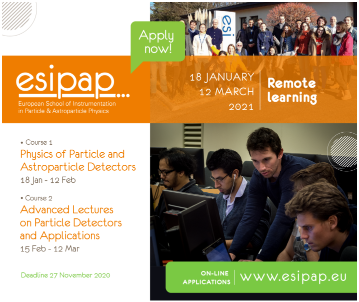
Yann Coadou

CPPM Marseille

European School of Instrumentation
in Particle & Astroparticle Physics

Online edition, 27 January 2021

Outline



Apply now!

esipap...

European School of Instrumentation in Particle & Astroparticle Physics

18 JANUARY
12 MARCH
2021

Remote learning

- Course 1

Physics of Particle and Astroparticle Detectors

18 Jan - 12 Feb

- Course 2

Advanced Lectures on Particle Detectors and Applications

15 Feb - 12 Mar

Deadline 27 November 2020

ON-LINE APPLICATIONS www.esipap.eu

1. Introduction
2. Optimal discrimination
3. Machine learning
4. Random grid search
5. Genetic algorithms
6. Quadratic and linear discriminants
7. Support vector machines
8. Kernel density estimation
9. (Boosted) Decision trees
10. Neural networks
11. Deep neural networks
12. Machine learning and particle physics
13. Conclusion
14. References

Introduction

Typical problems in HEP

- Classification of objects
  - separate real and fake leptons/jets/etc.
- Signal enhancement relative to background
- Regression: best estimation of a parameter
  - lepton energy, $E_{\mathrm{T}}^{\mathrm{miss}}$ value, invariant mass, etc.

Discrimination of signal from background in HEP

- Event level (Higgs searches, ...)
- Cone level (tau-vs-jet reconstruction, ...)
- Lifetime and flavour tagging (*b*-tagging, ...)
- Track level (particle identification, ...)
- Cell level (energy deposit from hard scatter/pileup/noise, ...)

Introduction

Input information from various sources

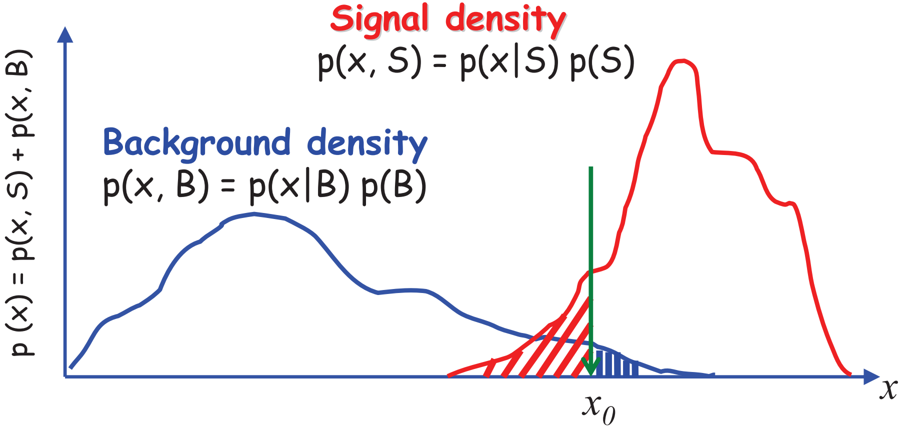
- Kinematic variables (masses, momenta, decay angles, . . .)
- Event properties (jet multiplicity, sum of charges, brightness . . .)
- Event shape (sphericity, aplanarity, . . .)
- Detector response (silicon hits, dE/dx, Cherenkov angle, shower profiles, muon hits, . . .)

Most data are (highly) multidimensional

- Use dependencies between $x = \{x_1, \cdots, x_n\}$ discriminating variables
- Approximate this n-dimensional space with a function $f(x)$ capturing the essential features
- f is a multivariate discriminant
- For most of these lectures, use binary classification:
  - an object belongs to one class (e.g. signal) if $f(x) > q$, where q is some threshold,
  - and to another class (e.g. background) if $f(x) \leq q$

Optimal discrimination: 1-dimension case

- Where to place a cut x_0 on variable x?

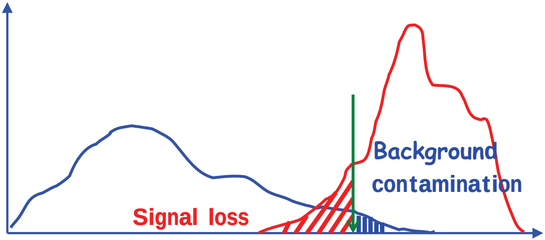


- Optimal choice: minimum misclassification cost at decision boundary $x = x_0$

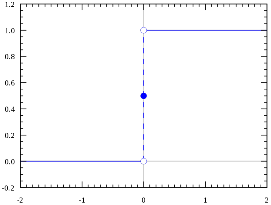
Optimal discrimination: cost of misclassification

$$\begin{aligned} C(x_0) &= C_S \int H(x_0 - x)\,p(x,S)\,dx \qquad \text{signal loss} \\ &+ C_B \int H(x - x_0)\,p(x,B)\,dx \qquad \text{background contamination} \end{aligned}$$

$$\begin{aligned} C_S &= \text{cost of misclassifying signal as background} \\ C_B &= \text{cost of misclassifying background as signal} \end{aligned}$$



- $H(x)$: Heaviside step function
  - $H(x) = 1$ if $x > 0$, 0 otherwise



- Optimal choice: when cost function C is minimum

Optimal discrimination: Bayes discriminant

Minimising the cost

- Minimise
  $C(x_0) = C_S \int H(x_0 - x)p(x,S)dx + C_B \int H(x - x_0)p(x,B)dx$
  with respect to the boundary x_0:

$$\begin{aligned} 0 &= C_S \int \delta(x_0 - x)p(x,S)dx - C_B \int \delta(x - x_0)p(x,B)dx \\ &= C_S p(x_0,S) - C_B p(x_0,B) \end{aligned}$$

- This gives the Bayes discriminant:

$$BD = \frac{C_B}{C_S} = \frac{p(x_0,S)}{p(x_0,B)} = \frac{p(x_0|S)p(S)}{p(x_0|B)p(B)}$$

Optimal discrimination: Bayes limit

Generalising to multidimensional problem

- The same holds when x is an n-dimensional variable:

$$BD = \frac{p(x|S)}{p(x|B)} \times \frac{p(S)}{p(B)}$$

- From Bayes theorem $(p(A|B)p(B) = p(B|A)p(A))$ and sum of probabilities $(p(S|x) + p(B|x) = 1)$:

$$p(S|x) = \frac{BD}{1 + BD}$$

Bayes limit

- $p(S|x) = BD/(1 + BD)$ is what should be achieved to minimise cost, reaching classification with the fewest mistakes
- Fixing relative cost of background contamination and signal loss $q = C_B/(C_S + C_B)$, $q = p(S|x)$ defines decision boundary:
  - signal-rich if $p(S|x) \geq q$
  - background-rich if $p(S|x) < q$
- Any function that approximates conditional class probability $p(S|x)$ with negligible error reaches the Bayes limit

Optimal discrimination: using a discriminant

How to construct $p(S|x)$?

- $k = p(S)/p(B)$ typically unknown
- Problem: $p(S|x)$ depends on k!
- Solution: it's not a problem...
- Define a multivariate discriminant:

$$D(x) = \frac{s(x)}{s(x)+b(x)} = \frac{p(x|S)}{p(x|S)+p(x|B)}$$

- Now:

$$p(S|x) = \frac{D(x)}{D(x) + (1 - D(x))/k}$$

- Cutting on $D(x)$ is equivalent to cutting on $p(S|x)$, implying a corresponding (unknown) cut on $p(S|x)$

Machine learning: learning from examples

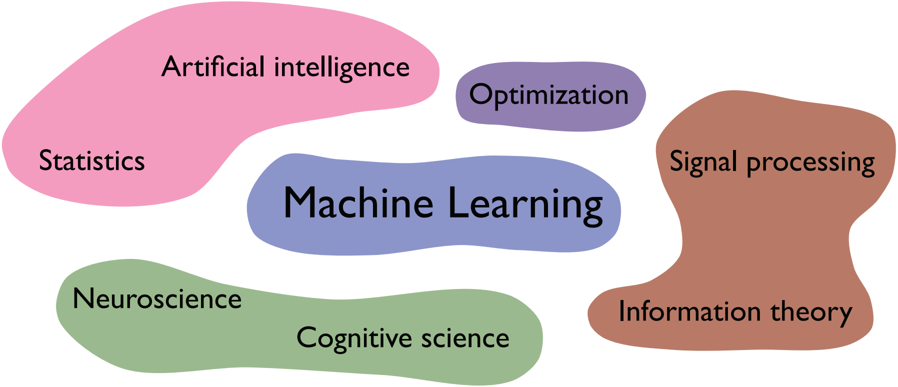
Several types of problems

- Classification/decision:
  - signal or background
  - type Ia supernova or not
  - will pay his/her credit back on time or not
- Regression: estimating a parameter value (energy of a particle, brightness of a supernova, ...) [mostly ignored in these lectures]
- Clustering (cluster analysis):
  - in exploratory data mining, finding features

Our goal

- Teach a machine to learn the discriminant $f(x)$ using examples from a training dataset
- Be careful to not learn too much the properties of the training sample
  - no need to memorise the training sample
  - instead, interested in getting the right answer for new events $\Rightarrow$ generalisation ability

Machine learning and connected fields



Artificial intelligence

Statistics

Optimization

Machine Learning

Signal processing

Information theory

Neuroscience

Cognitive science

Machine learning: (un)supervised learning

Supervised learning

- Training events are labelled: N examples $(x, y)_1, (x, y)_2, \ldots, (x, y)_N$ of (discriminating) feature variables x and class labels y
- The learner uses example classes to know how good it is doing

Machine learning: (un)supervised learning

Supervised learning

- Training events are labelled: N examples $(x, y)_1, (x, y)_2, \ldots, (x, y)_N$ of (discriminating) feature variables x and class labels y
- The learner uses example classes to know how good it is doing

Reinforcement learning

- Instead of labels, some sort of reward system (e.g. game score)
- Goal: maximise future payoff by optimising decision policy
- May not even "learn" anything from data, but remembers what triggers reward or punishment

Machine learning: (un)supervised learning

Supervised learning

- Training events are labelled: N examples $(x, y)_1, (x, y)_2, \ldots, (x, y)_N$ of (discriminating) feature variables x and class labels y
- The learner uses example classes to know how good it is doing

Reinforcement learning

- Instead of labels, some sort of reward system (e.g. game score)
- Goal: maximise future payoff by optimising decision policy
- May not even "learn" anything from data, but remembers what triggers reward or punishment

Unsupervised learning

- e.g. clustering: find similarities in training sample, without having predefined categories
- Discover good internal representation of the input
- Not biased by pre-determined classes $\Rightarrow$ may discover unexpected features!

Finding the multivariate discriminant $y = f(x)$

- Given our N examples $(x, y)_1, \ldots, (x, y)_N$ we need
  - a function class $\mathbb{F} = \left\{f(x, w)\right\}$ (w: parameters of prediction to be found)
  - a constraint $Q(w)$ on $\mathbb{F}$ (regularisation term)
  - a loss or error function $L(y, f)$, encoding what is lost if f is poorly chosen in $\mathbb{F}$ (i.e., $f(x, w)$ far from the desired $y = f(x)$)
- Cannot minimise L directly (would depend on the dataset used), but rather its average over a training sample, the empirical risk:

$$R(w) = \frac{1}{N} \sum_{i=1}^{N} L\big(y_i, f(x_i, w)\big)$$

subject to constraint $Q(w)$, so we minimise the cost function:

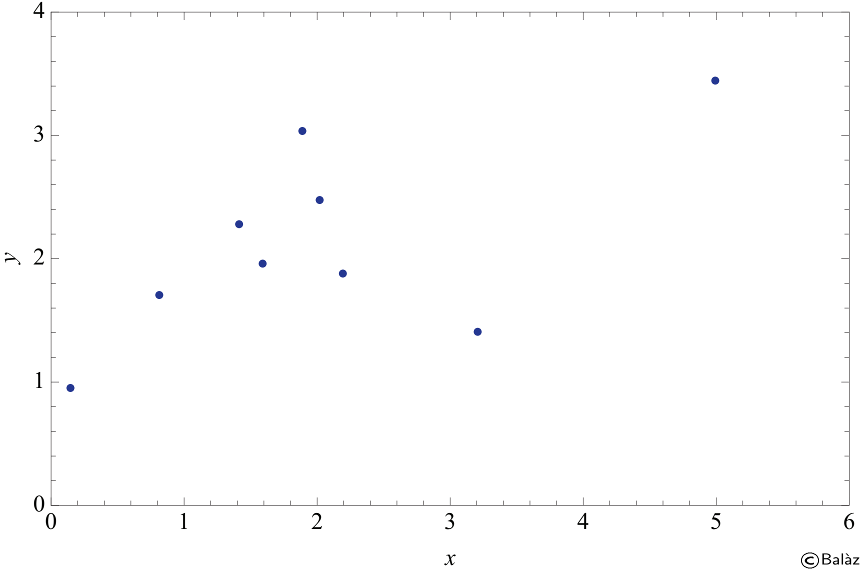
$$C(w) = R(w) + \lambda Q(w)$$

where λ controls the strength of regularisation

- At the minimum of $C(w)$ we select $f(x, w_*)$, our estimate of $y = f(x)$

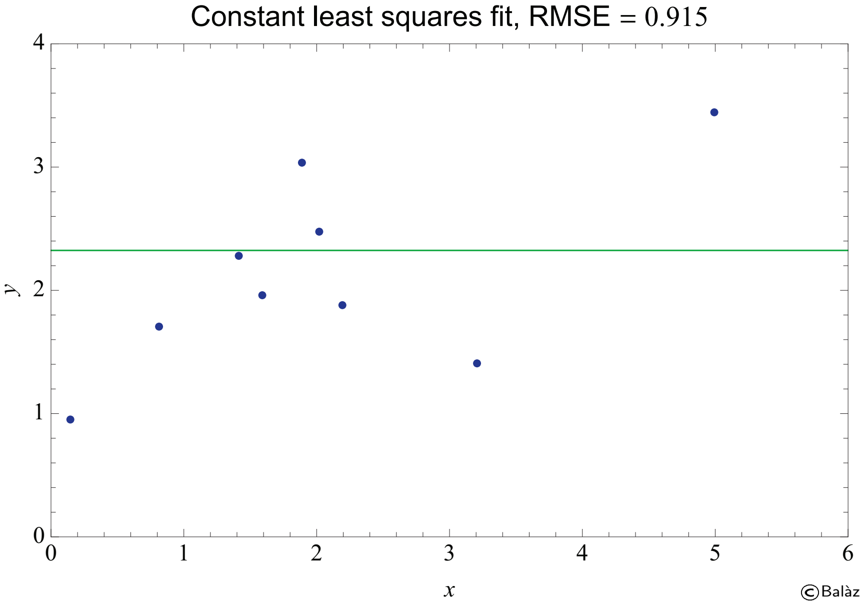
Choice of function class: training

Data generated from an unknown function with unknown noise



©Balàzs Kégl

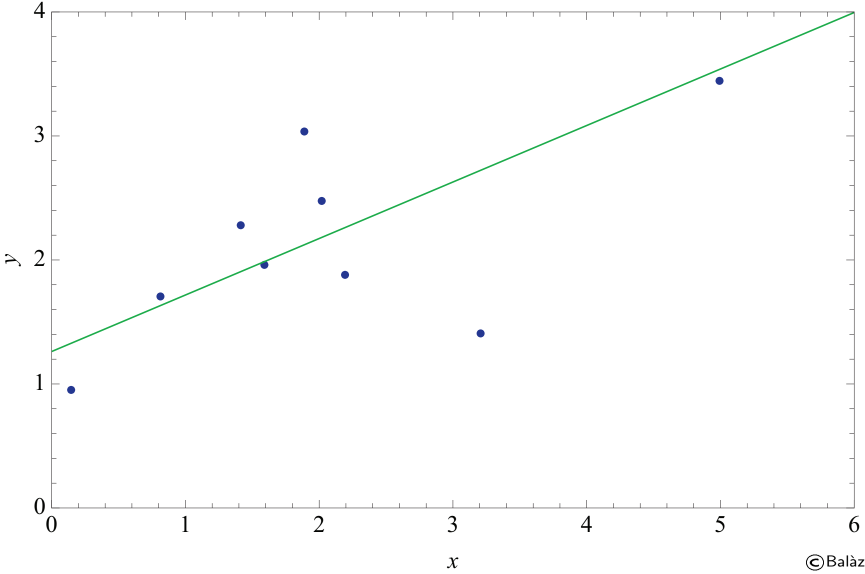
Choice of function class: training



©Balàzs Kégl

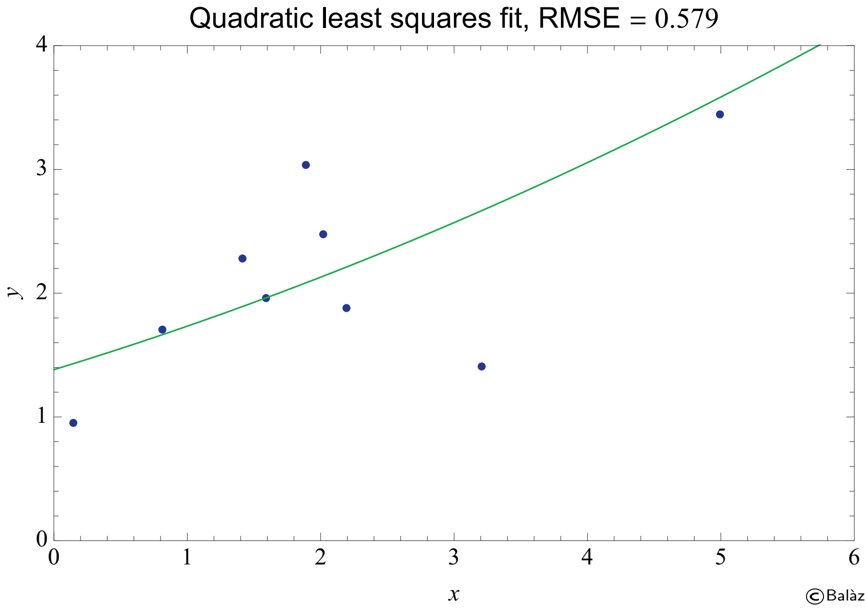
Choice of function class: training

Linear least squares fit, RMSE $= 0.581$



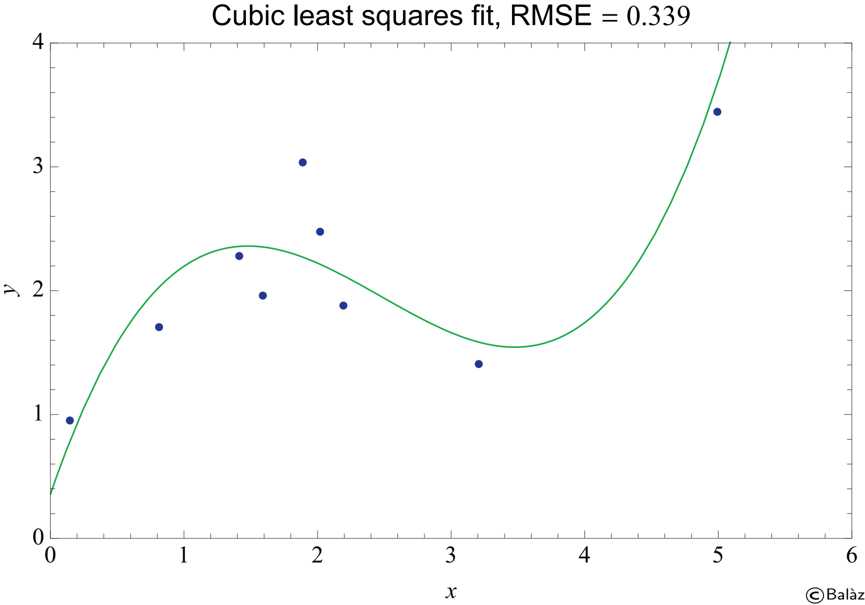
©Balàzs Kégl

Choice of function class: training



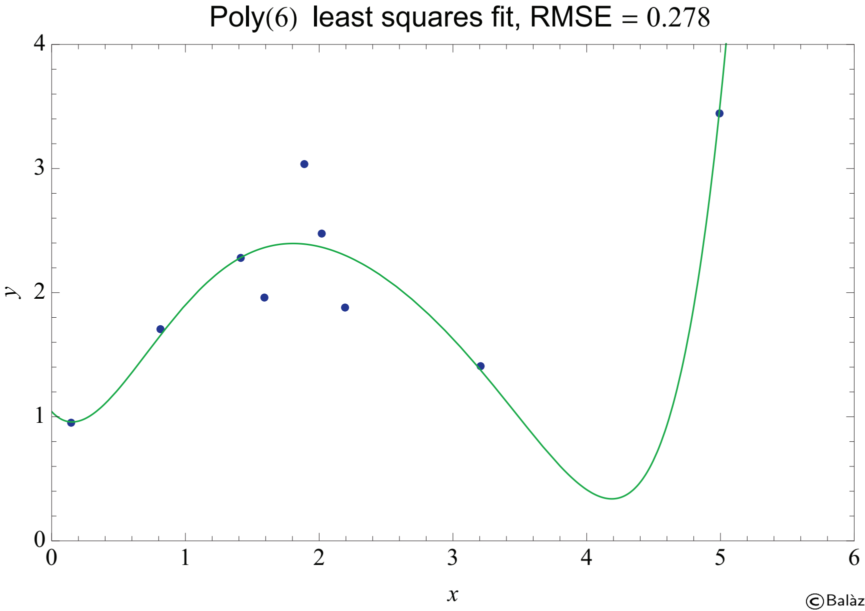
©Balàzs Kégl

Choice of function class: training



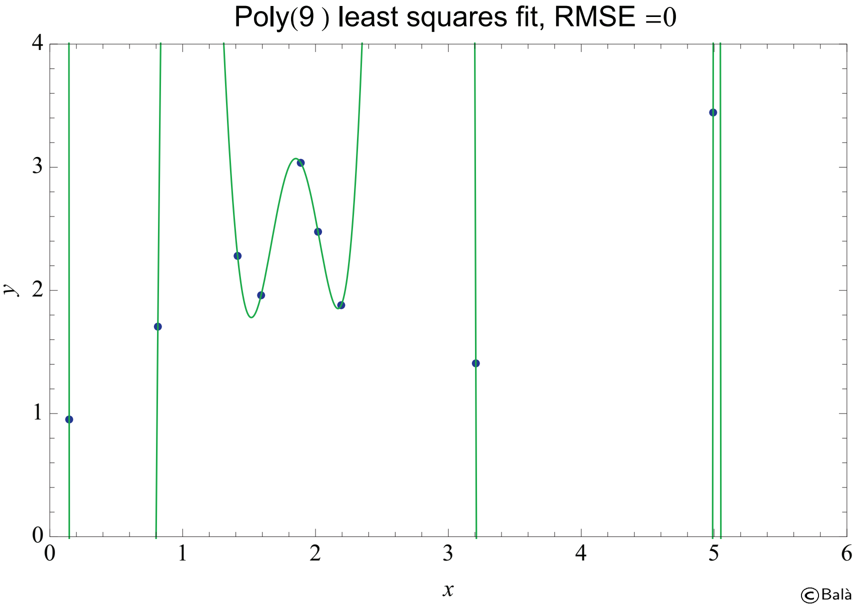
©Balàzs Kégl

Choice of function class: training



©Balàzs Kégl

Choice of function class: training



©Balàzs Kégl

Choice of function class

Quality of fit

- Increasing degree of polynomial increases flexibility of function
- Higher degree $\Rightarrow$ can match more features
- If degree $=$ # points, polynomial passes through each point: perfect match!

Choice of function class

Quality of fit

- Increasing degree of polynomial increases flexibility of function
- Higher degree ⇒ can match more features
- If degree = # points, polynomial passes through each point: perfect match!

Is it meaningful?

- It could be:
  - if there is no noise or uncertainty in the measurement
  - if the true distribution is indeed perfectly described by such a polynomial
- ...not impossible, but not very common...

Choice of function class

Quality of fit

- Increasing degree of polynomial increases flexibility of function
- Higher degree ⇒ can match more features
- If degree = # points, polynomial passes through each point: perfect match!

Is it meaningful?

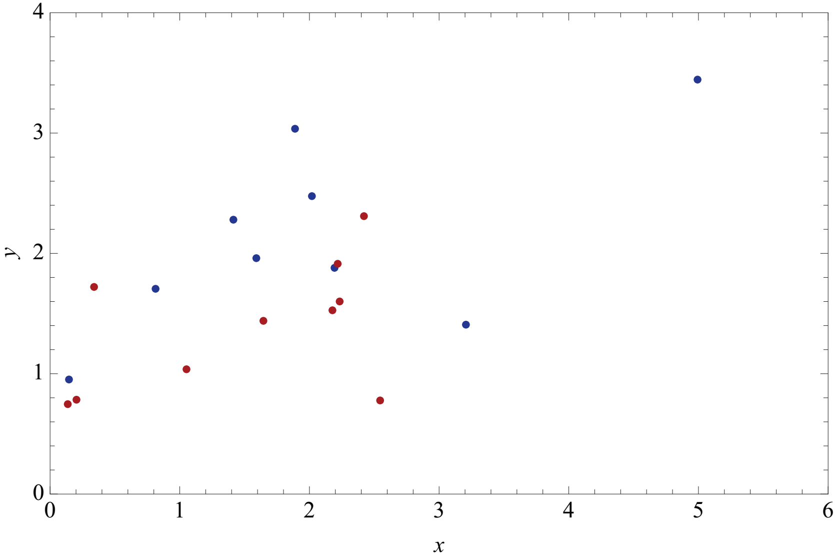
- It could be:
  - if there is no noise or uncertainty in the measurement
  - if the true distribution is indeed perfectly described by such a polynomial
- . . . not impossible, but not very common. . .

Solution: testing and/or validation sample

- Use independent sample to validate the result
- Expected: performance will also increase, go through a maximum and decrease again, while it keeps increasing on the training sample

Choice of function class: testing

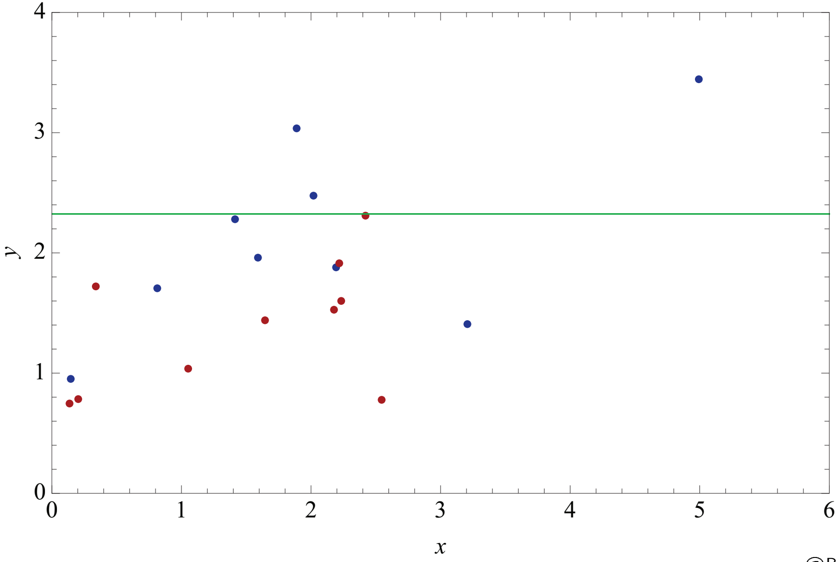
Data generated from an unknown function with unknown noise



©Balàzs Kégl

Choice of function class: testing

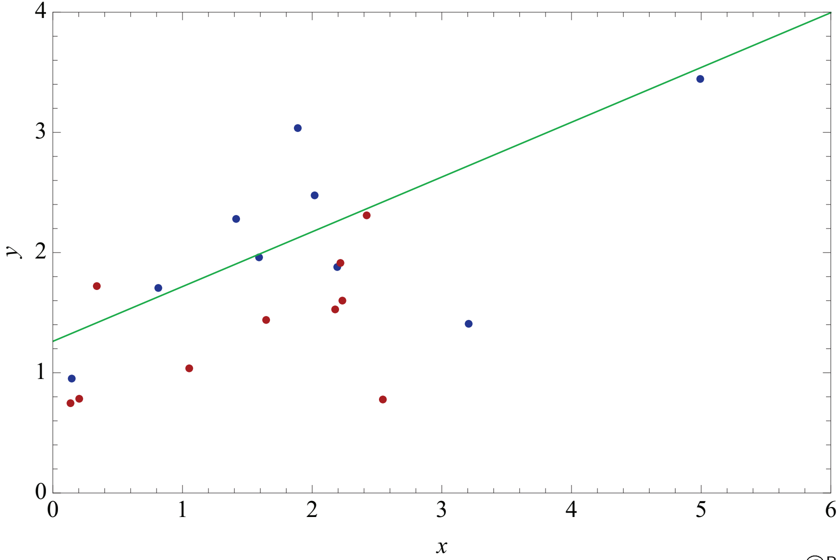
Const. least squares fit, training RMSE $= 0.915$, test RMSE $= 1.067$



©Balàzs Kégl

Choice of function class: testing

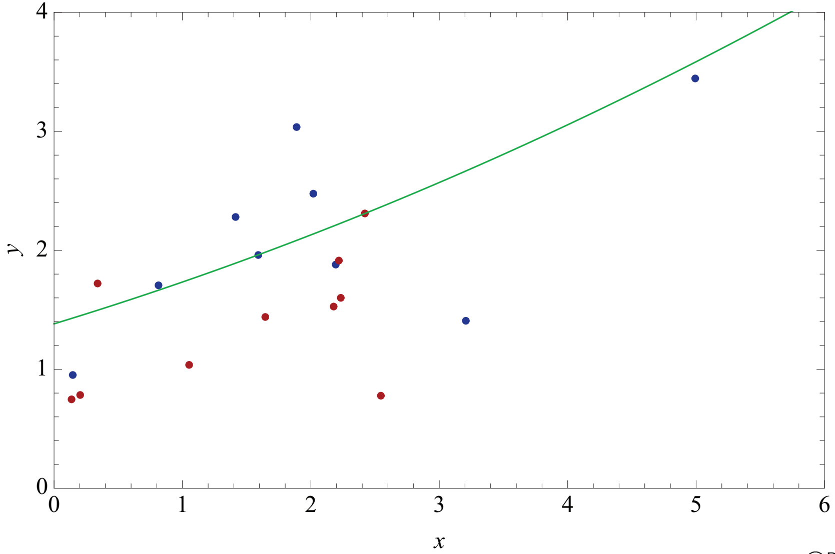
Linear least squares fit, training RMSE $= 0.581$, test RMSE $= 0.734$



ⒸBalàzs Kégl

Choice of function class: testing

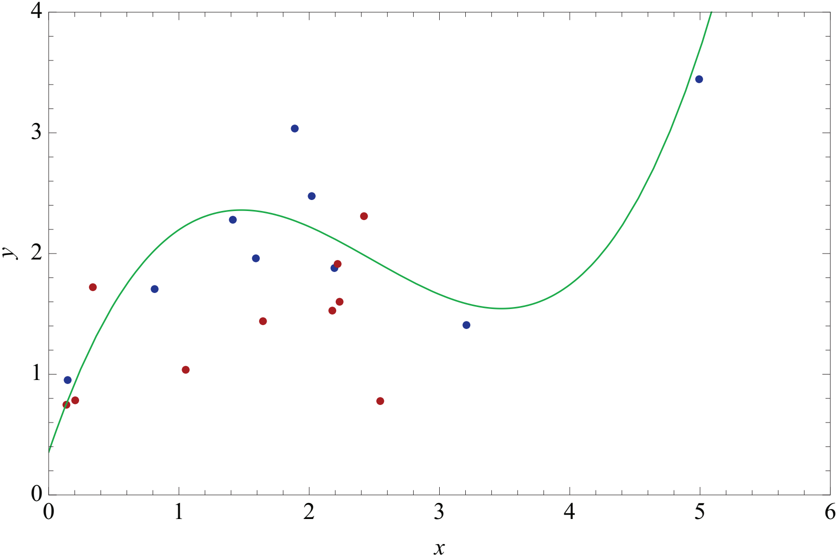
Quadr. least squares fit, training RMSE $= 0.579$, test RMSE $= 0.723$



©Balàzs Kégl

Choice of function class: testing

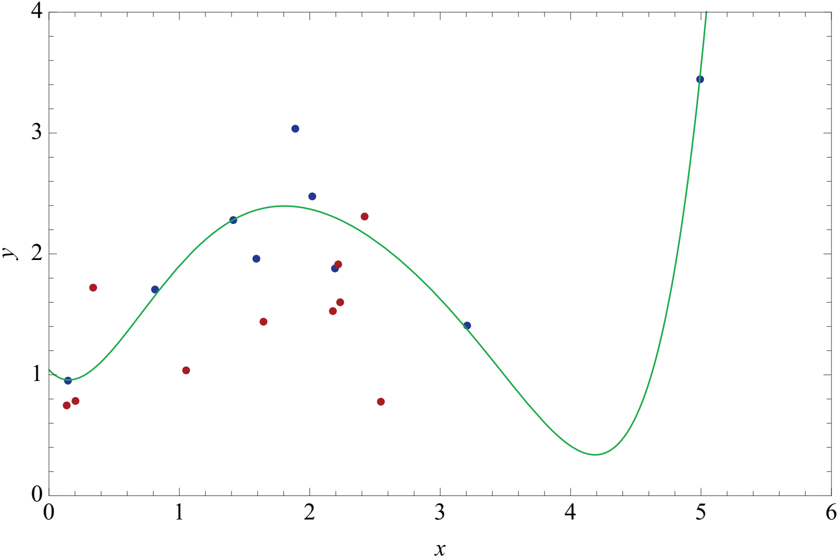
Cubic least squares fit, training RMSE $= 0.339$, test RMSE $= 0.672$



©Balàzs Kégl

Choice of function class: testing

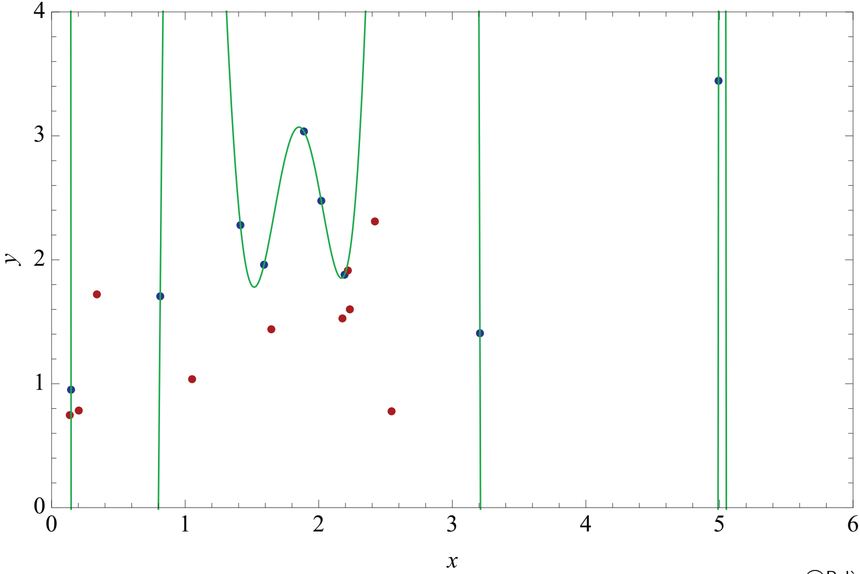
Poly(6) least squares fit, training RMSE $= 0.278$, test RMSE $= 0.72$



©Balàzs Kégl

Choice of function class: testing

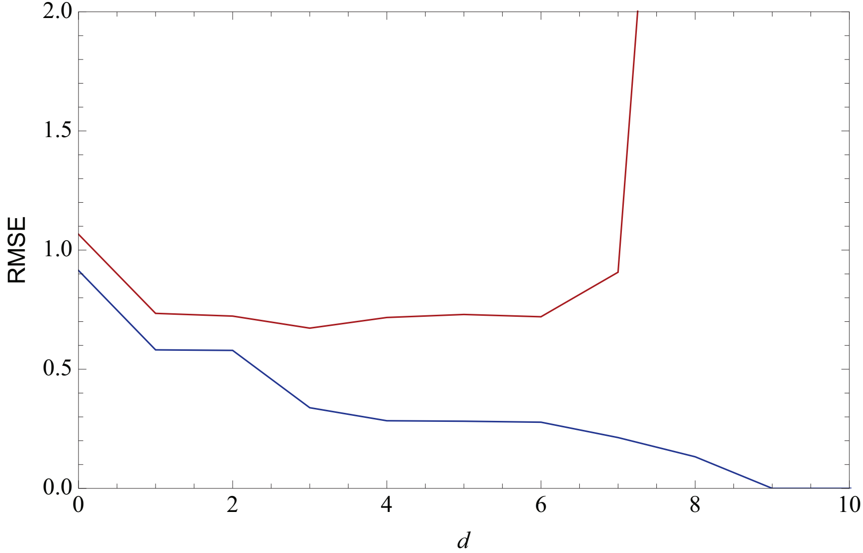
Poly(9) least squares fit, training RMSE = 0, test RMSE = 46.424



ⒸBalàzs Kégl

Choice of function class

Training and test RMSE's for polynomial fits of different degrees



©Balàzs Kégl

Capacity control and regularisation

- Trade-off between approximation error and estimation error
- Take into account sample size
- Measure (and penalise) complexity
- Use independent test sample
- In practice, no need to correctly guess the function class, but need enough flexibility in your model, balanced with complexity cost

Multivariate discriminants

4. Random grid search
5. Genetic algorithms
6. Quadratic and linear discriminants
7. Support vector machines
8. Kernel density estimation
9. (Boosted) Decision trees
10. Neural networks
11. Deep neural networks
12. Machine learning and particle physics

Multivariate discriminants

Reminder

- To solve binary classification problem with the fewest number of mistakes, sufficient to compute the multivariate discriminant:

$$D(x) = \frac{s(x)}{s(x) + b(x)}$$

where:

  - $s(x) = p(x|S)$ signal density
  - $b(x) = p(x|B)$ background density

- Cutting on $D(x)$ is equivalent to cutting on probability $p(S|x)$ that event with x values is of class S

Which approximation to choose?

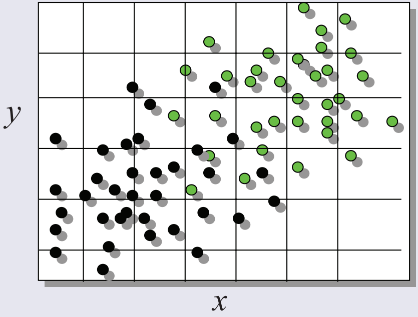
- Best possible choice: cannot beat Bayes limit (but usually impossible to define)
- No single method can be proven to surpass all others in particular case
- Advisable to try several and use the best one

Cut-based analysis and grid search

Cut-based analysis

- Simple approach: cut on each discriminating variable
- Difficulty: how to optimise the cuts?

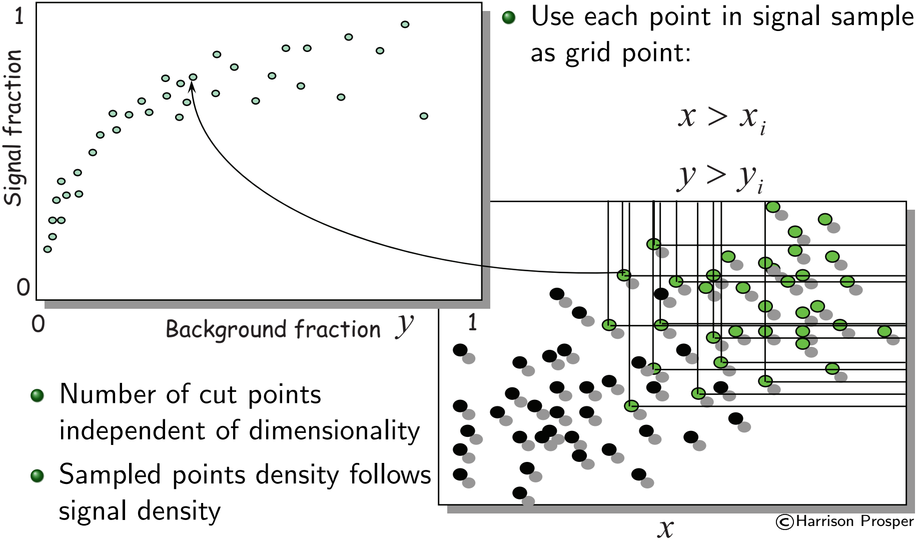
Grid search



©Harrison Prosper

- Split each variable in K values
- Apply cuts at each grid point: $x > x_i,\ y > y_i$
- Number of points scales with K^n: curse of dimensionality

Random grid search



- Use each point in signal sample as grid point:

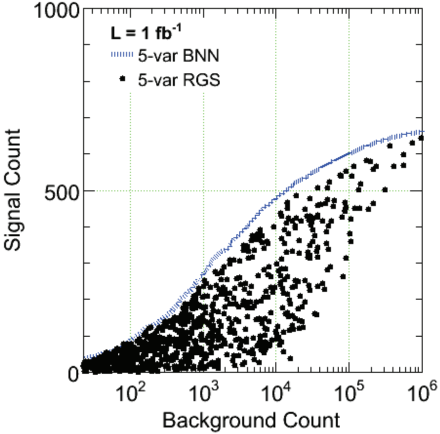
$$x > x_i$$

$$y > y_i$$

- Number of cut points independent of dimensionality
- Sampled points density follows signal density

©Harrison Prosper

Random grid search example



©Harrison Prosper

Comparison to BNN

- Blue: 5-dim Bayesian neural network discriminant
- Points: each cut point from a 5-dim RGS calculation
- Conclusions:
  - RGS can find very good criteria with high discrimination
  - but it usually cannot compete with a full-blown multivariate discriminant
  - and never outsmarts it

Genetic algorithms: survival of the fittest

- Inspired by biological evolution
- Model: group (population) of abstract representations (genome/discriminating variables) of possible solutions (individuals/list of cuts)
- Typical processes at work in evolutionary processes:
  - inheritance
  - mutation
  - sexual recombination (a.k.a. crossover)
- Fitness function: value representing the individual's goodness, or comparison of two individuals
- For cut optimisation:
  - good background rejection and high signal efficiency
  - compare individuals in each signal efficiency bin and keep those with higher background rejection

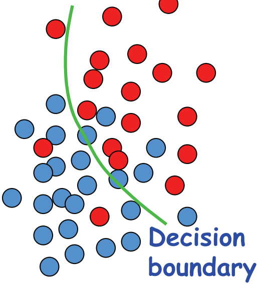
Genetic algorithms

- Better solutions more likely to be selected for mating and mutations, carrying their genetic code (cuts) from generation to generation
- Algorithm:
  1. Create initial random population (cut ensemble)
  2. Select fittest individuals
  3. Create offsprings through crossover (mix best cuts)
  4. Mutate randomly (change some cuts of some individuals)
  5. Repeat from 2 until convergence (or fixed number of generations)
- Good fitness at one generation ⇒ average fitness in the next
- Algorithm focuses on region with higher potential improvement

Quadratic discriminants: Gaussian problem

- Suppose densities $s(x)$ and $b(x)$ are multivariate Gaussians:
$$\text{Gaussian}(x|\mu,\Sigma) = \frac{1}{\sqrt{(2\pi)^n|\Sigma|}} \exp^{\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)}$$
with vector of means μ and covariance matrix Σ

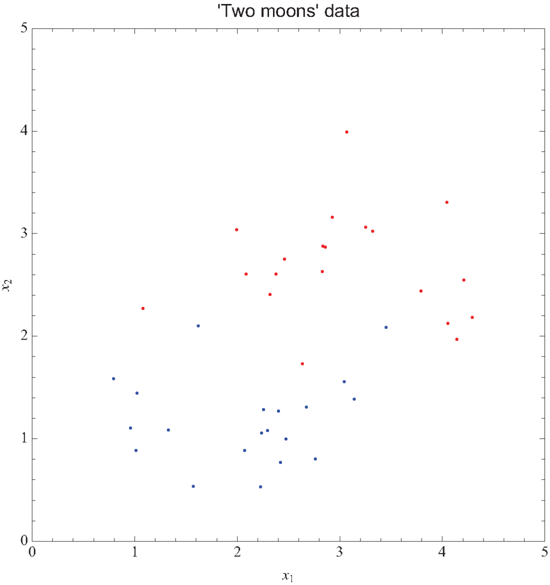
- Then Bayes factor $B(x) = s(x)/b(x)$ (or its logarithm) can be expressed explicitly:
$$\ln B(x) = \lambda(x) \equiv \chi^2(\mu_B,\Sigma_B) - \chi^2(\mu_S,\Sigma_S)$$



with $\chi^2(\mu,\Sigma) = (x-\mu)^T\Sigma^{-1}(x-\mu)$

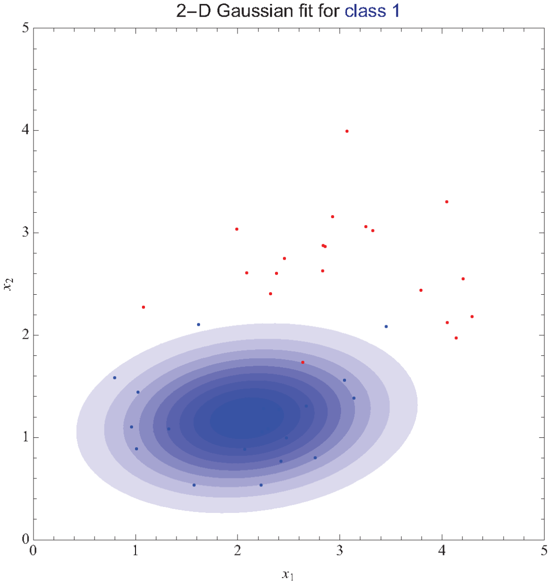
- Fixed value of $\lambda(x)$ defines quadratic hypersurface partitioning n-dimensional space into signal-rich and background-rich regions
- Optimal separation if $s(x)$ and $b(x)$ are indeed multivariate Gaussians

Quadratic discriminant



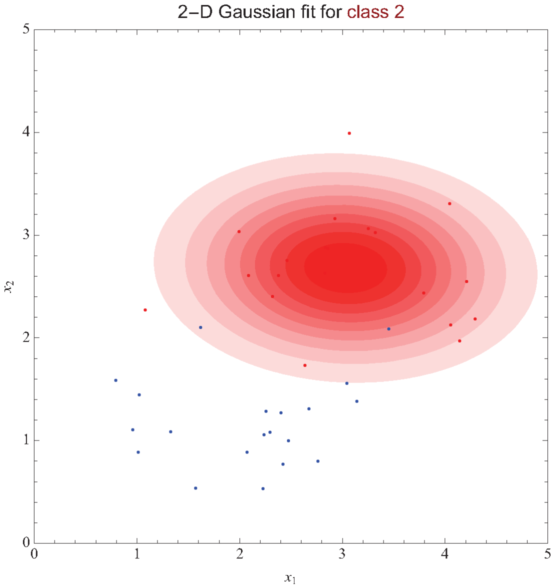
ⒸBalàzs Kégl

Quadratic discriminant



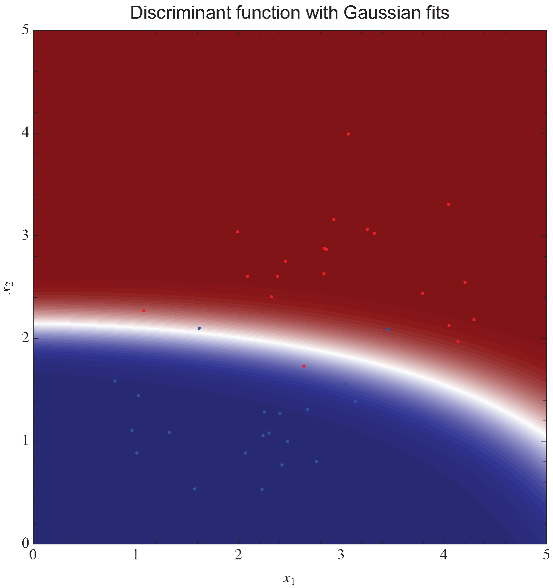
©Balàzs Kégl

Quadratic discriminant



©Balàzs Kégl

Quadratic discriminant

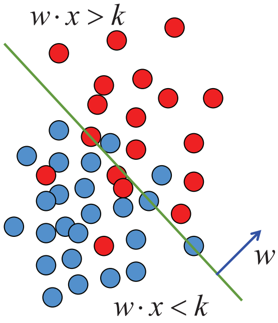


ⒸBalàzs Kégl

Linear discriminant: Fisher's discriminant

- If in $\lambda(x)$ the same covariance matrix is used for each class (e.g. $\Sigma = \Sigma_S + \Sigma_B$) one gets Fisher's discriminant:

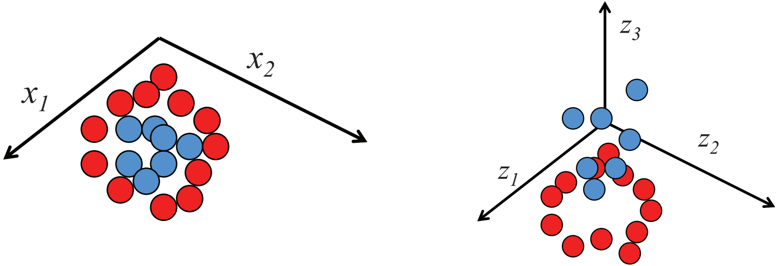
$$\lambda(x) = w \cdot x \quad \text{with} \quad w \propto \Sigma^{-1}(\mu_S - \mu_B)$$



- Optimal linear separation
- Works only if signal and background have different means!
- Optimal classifier (reaches the Bayes limit) for linearly correlated Gaussian-distributed variables

Support vector machines

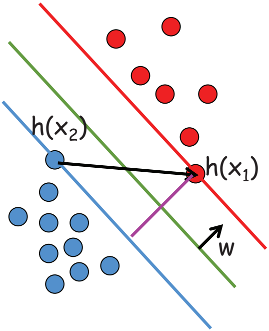
- Fisher discriminant: may fail completely for highly non-Gaussian densities
- But linearity is good feature ⇒ try to keep it
- Generalising Fisher discriminant: data non-separable in n-dim space $\mathbb{R}^n$, but better separated if mapped to higher dimension space $\mathbb{R}^H$: $h : x \in \mathbb{R}^n \to z \in \mathbb{R}^H$
- Use hyper-planes to partition higher dim space: $f(x) = w \cdot h(x) + b$
- Example:$h : (x_1, x_2) \to (z_1, z_2, z_3) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$



Support vector machines: separable data

- Consider separable data in $\mathbb{R}^H$, and three parallel hyper-planes:

$$\begin{aligned} w \cdot h(x) + b &= 0 \text{ (separating hyper-plane between red and blue)} \\ w \cdot h(x_1) + b &= +1 \text{ (contains } h(x_1)) \\ w \cdot h(x_2) + b &= -1 \text{ (contains } h(x_2)) \end{aligned}$$



- Subtract blue from red: $w \cdot \big(h(x_1) - h(x_2)\big) = 2$
- With unit vector $\hat{w} = w/\|w\|$: $\hat{w} \cdot \big(h(x_1) - h(x_2)\big) = 2/\|w\| = m$
- Margin m is distance between red and blue planes
- Best separation: maximise margin
- $\Rightarrow$ empirical risk margin to minimise: $R(w) \propto \|w\|^2$

Support vector machines: constraints

- When minimising $R(w)$, need to keep signal and background separated
- Label red dots $y=+1$ ("above" red plane) and blue dots $y=-1$ ("below" blue plane)
- Since:

$$\begin{aligned} w\cdot h(x)+b &> 1 \text{ for red dots} \\ w\cdot h(x)+b &< -1 \text{ for blue dots} \end{aligned}$$

  all correctly classified points will satisfy constraints:

$$y_i\big(w\cdot h(x_i)+b\big)\geq 1,\ \forall i=1,\ldots,N$$

- Using Lagrange multipliers $\alpha_i>0$, cost function can be written:

$$C(w,b,\alpha)=\frac{1}{2}\|w\|^2-\sum_{i=1}^{N}\alpha_i\big[y_i\big(w\cdot h(x_i)+b\big)-1\big]$$

Support vector machines

Minimisation

- Minimise cost function $C(w, b, \alpha)$ with respect to w and b:

$$C(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \big(h(x_i) \cdot h(x_j)\big)$$

- At minimum of $C(\alpha)$, only non-zero α_i correspond to points on red and blue planes: support vectors

Kernel functions

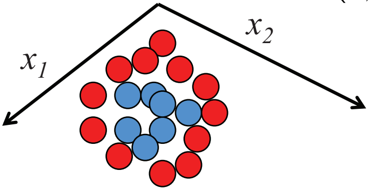
- Issues:
  - need to find h mappings (potentially of infinite dimension)
  - need to compute scalar products $h(x_i) \cdot h(x_j)$
- Fortunately $h(x_i) \cdot h(x_j)$ are equivalent to some kernel function $K(x_i, x_j)$ that does the mapping and the scalar product:

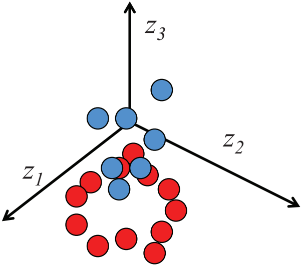
$$C(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$

Support vector machines: example

- $h : (x_1, x_2) \rightarrow (z_1, z_2, z_3) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$

$$\begin{aligned} h(x)\cdot h(y) &= (x_1^2, \sqrt{2}x_1x_2, x_2^2)\cdot(y_1^2, \sqrt{2}y_1y_2, y_2^2) \\ &= (x\cdot y)^2 \\ &= K(x,y) \end{aligned}$$



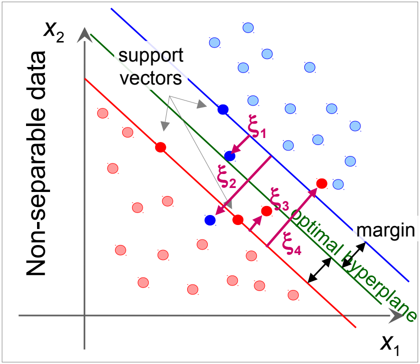


- In reality: do not know a priori the right kernel
- ⇒ have to test different standard kernels and use the best one

Support vector machines: non-separable data

- Even in infinite dimension space, data are often non-separable
- Need to relax constraints:

$$y_i\left(w\cdot h(x_i)+b\right)\geq 1-\xi_i$$



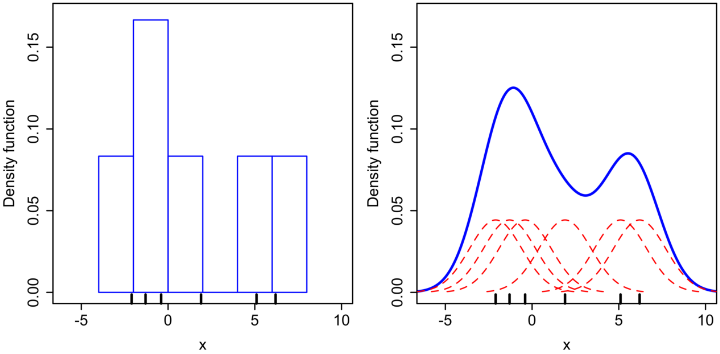
with slack variables $\xi_i > 0$

- $C(w,b,\alpha,\xi)$ depends on ξ, modified $C(\alpha,\xi)$ as well
- Values determined during minimisation

Kernel density estimation (KDE)

- Introduced by E. Parzen in the 1960s
- Place a kernel $K(x, \mu)$ at each training point μ
- Density $p(x)$ at point x approximated by:

$$p(x) \approx \hat{p}(x) = \frac{1}{N}\sum_{j=1}^{N} K(x, \mu_j)$$



Kernel density estimation (KDE)

Choice of kernel

- Any kernel can be used
- In practice, often product of Gaussians:

$$K(x,\mu) = \prod_{i=1}^{n} Gaussian(x_i|\mu, h_i)$$

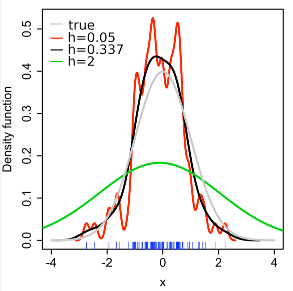
  each with bandwidth (width) h_i

Optimal bandwidth

- Too narrow: noisy approximation
- Too wide: loose fine structure
- In principle found by minimising risk function $R(\hat{p},p) = \int \left(\hat{p}(x) - p(x)\right)^2 dx$
- For Gaussian densities:

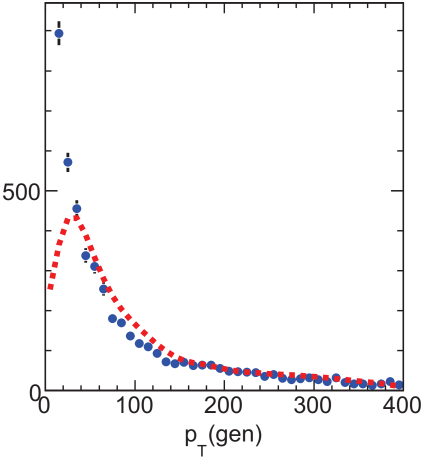
$$h = \sigma\left(\frac{4}{(n+2)N}\right)^{1/(n+4)}$$

- Far from optimal for non-Gaussian densities

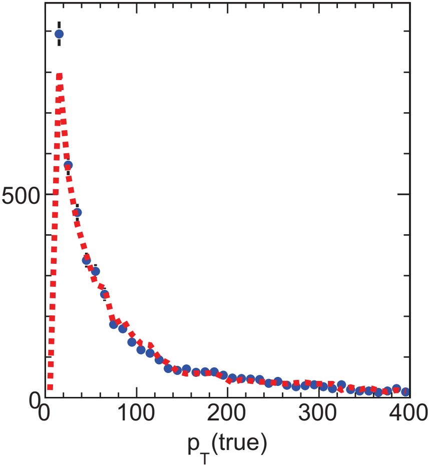


Kernel density estimation (KDE): example

with Gaussian optimal bandwidth



with optimised bandwidth



Kernel density estimation (KDE)

Why does it work?

- When $N \to \infty$:

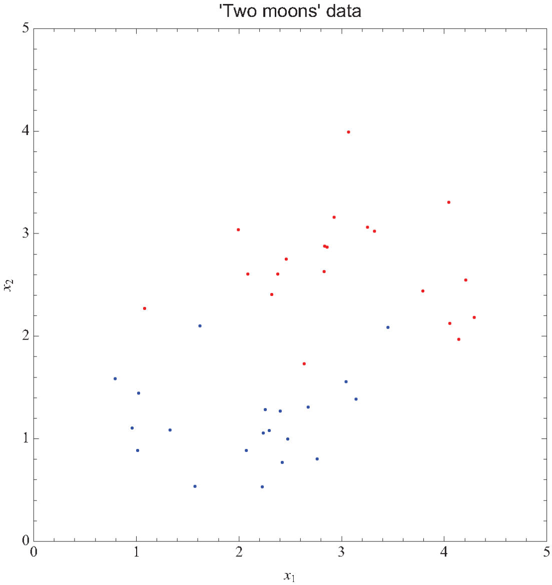
$$\hat{p}(x) = \int K(x,\mu)p(\mu)d\mu$$

- $p(\mu)$: true density of x
- Kernel bandwidth getting smaller with N, so when $N \to \infty$, $K(x,\mu) \to \delta^n(x-\mu)$ and $\hat{p}(x) = p(x)$
- KDE gives consistent estimate of probability density $p(x)$

Limitations

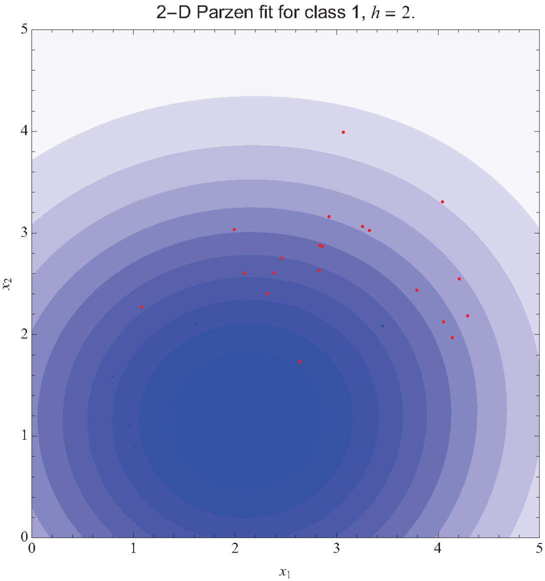
- Choice of bandwidth non-trivial
- Difficult to model sharp structures (e.g. boundaries)
- Kernels too far apart in regions of low point density
- (both can be mitigated with adaptive bandwidth choice)
- Requires evaluation of N n-dimensional kernels

Kernel density estimation (KDE)



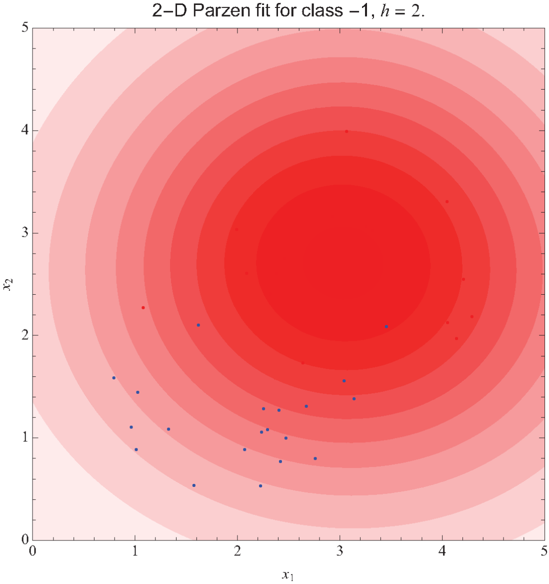
©Balàzs Kégl

Kernel density estimation (KDE)



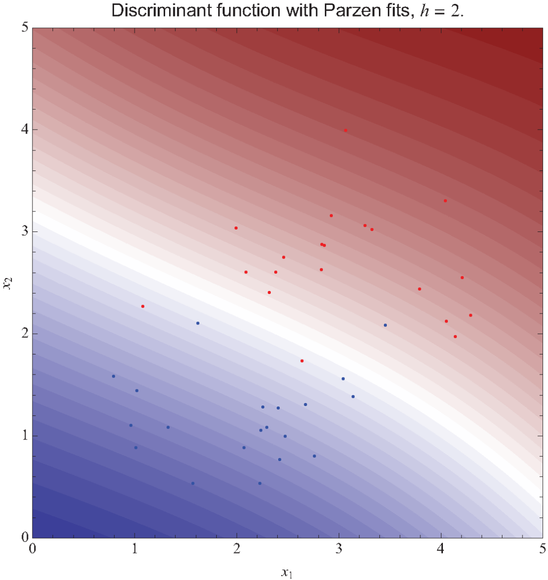
ⒸBalàzs Kégl

Kernel density estimation (KDE)



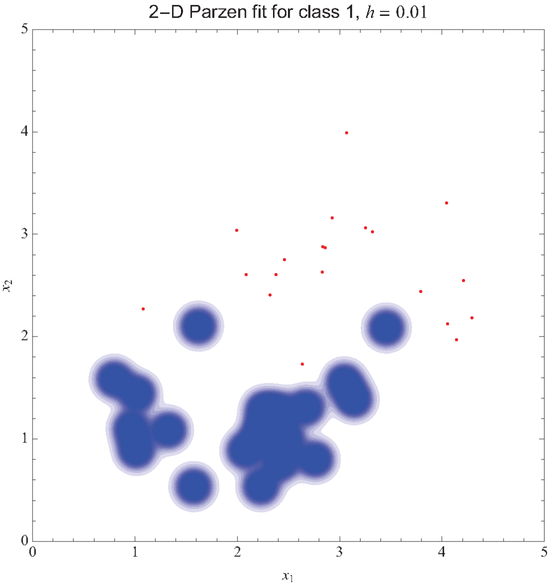
©Balàzs Kégl

Kernel density estimation (KDE)



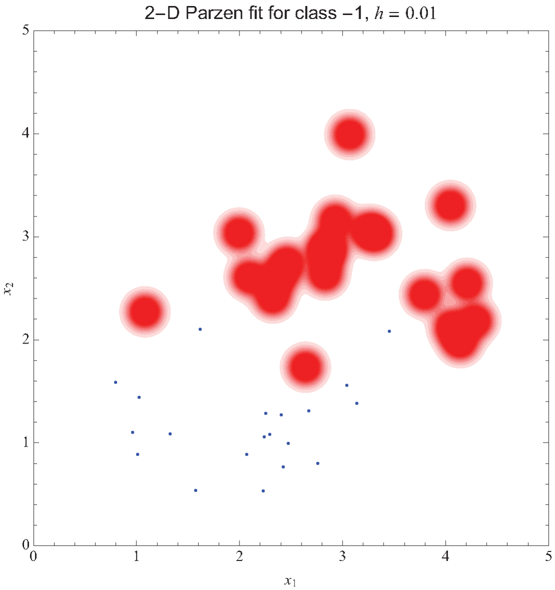
©Balàzs Kégl

Kernel density estimation (KDE)



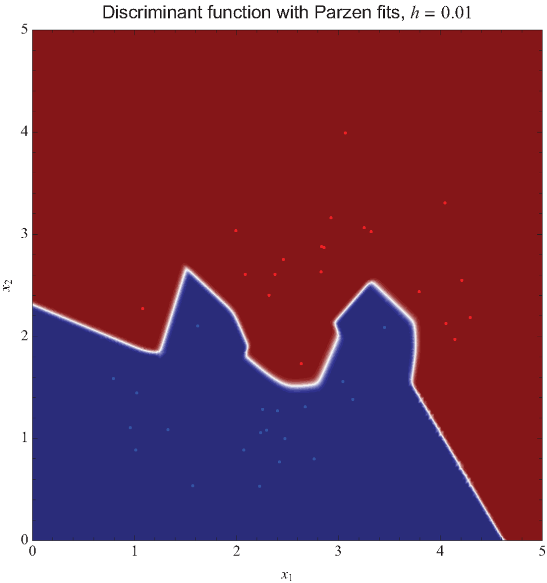
©Balàzs Kégl

Kernel density estimation (KDE)



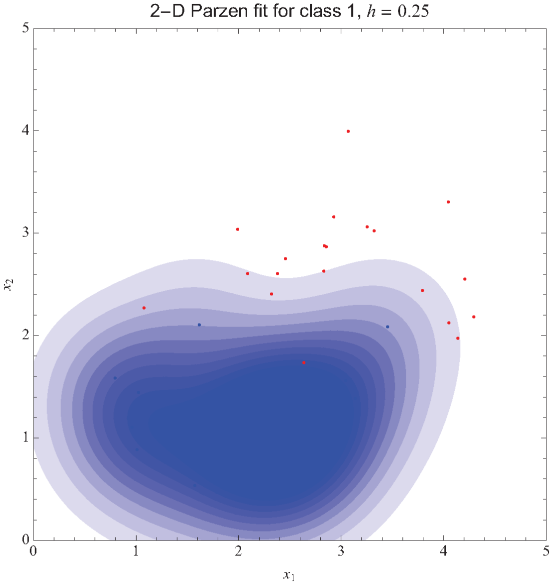
©Balàzs Kégl

Kernel density estimation (KDE)



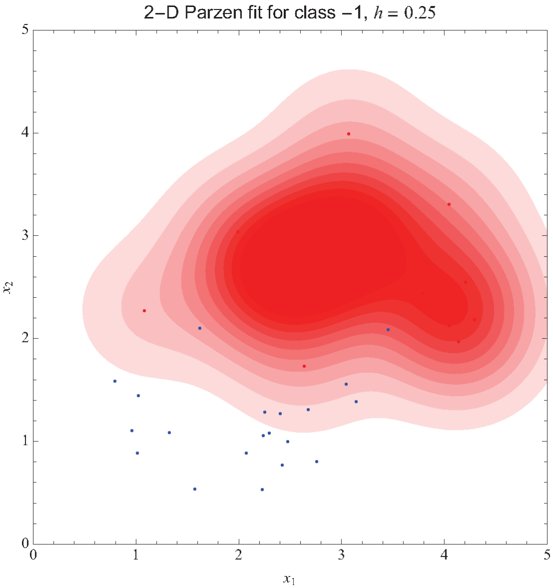
©Balàzs Kégl

Kernel density estimation (KDE)



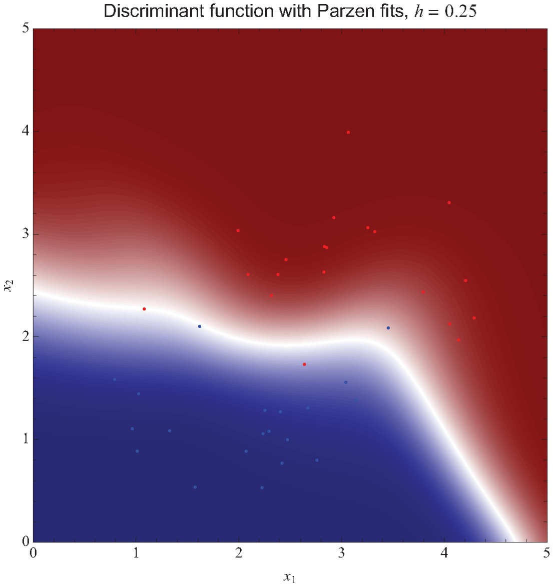
©Balàzs Kégl

Kernel density estimation (KDE)



©Balàzs Kégl

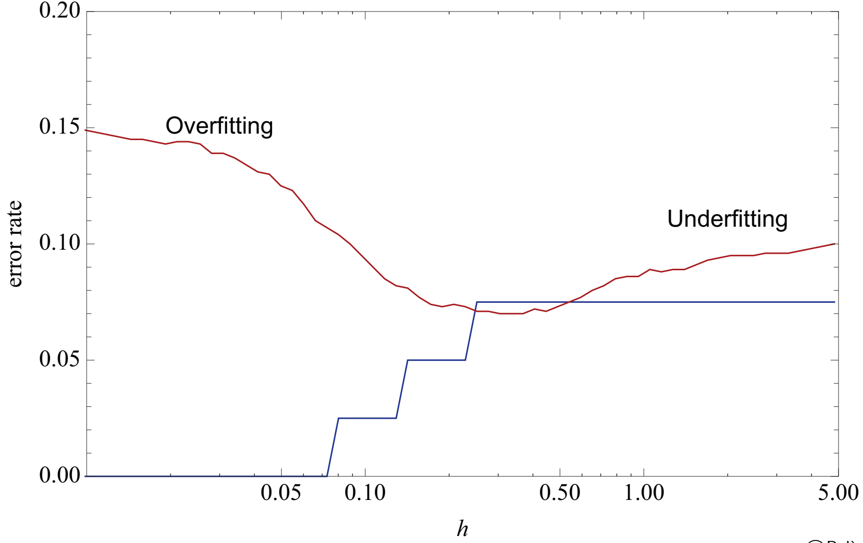
Kernel density estimation (KDE)



©Balàzs Kégl

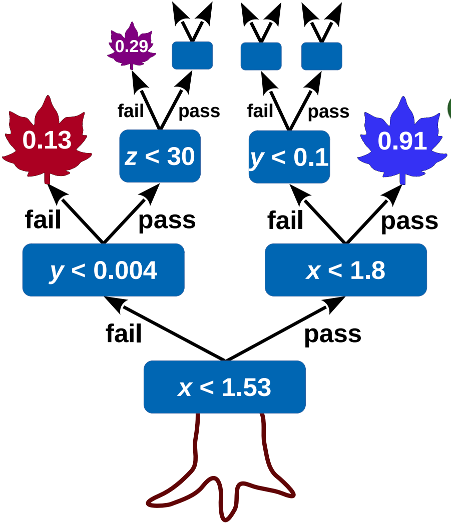
KDE: choice of bandwidth

Training and test error rates



©Balàzs Kégl

(Boosted) Decision trees



9 (Boosted) Decision trees

- Decision trees
  - Algorithm
  - Tree hyperparameters
  - Splitting a node
  - Variable selection
- Limitations
- Boosted decision trees
- Performance examples
- BDTs in real physics cases
- Software and example code

Introduction

Decision tree origin

- Machine-learning technique, widely used in social sciences. Originally data mining/pattern recognition, then medical diagnosis, insurance/loan screening, etc.

📕 L. Breiman *et al.*, “Classification and Regression Trees” (1984)

Basic principle

- Extend cut-based selection
  - many (most?) events do not have *all* characteristics of signal or background
  - 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

Binary trees

- Trees can be built with branches splitting into many sub-branches
- In this lecture: mostly binary trees

Tree building algorithm

Start with all events (signal and background) = first (root) node

- sort all events by each variable
- for each variable, find splitting value with best separation between two children
  - mostly signal in one child
  - mostly background in the other
- select variable and splitting value with best separation, produce two branches (nodes)
  - events failing criterion on one side
  - events passing it on the other

Keep splitting

- Now have two new nodes. Repeat algorithm recursively on each node
- Can reuse the same variable
- Iterate until stopping criterion is reached
- Splitting stops: terminal node = leaf

Algorithm example

- Consider signal (s_i) and background (b_j) events described by 3 variables: p_T of leading jet, top mass M_t and scalar sum of p_T's of all objects in the event H_T

Algorithm example

- Consider signal (s_i) and background (b_j) events described by 3 variables: p_T of leading jet, top mass M_t and scalar sum of p_T's of all objects in the event H_T
  - sort all events by each variable:
    - $p_T^{s_1} \leq p_T^{b_{34}} \leq \cdots \leq p_T^{b_2} \leq p_T^{s_{12}}$
    - $H_T^{b_5} \leq H_T^{b_3} \leq \cdots \leq H_T^{s_{67}} \leq H_T^{s_{43}}$
    - $M_t^{b_6} \leq M_t^{s_8} \leq \cdots \leq M_t^{s_{12}} \leq M_t^{b_9}$

Algorithm example

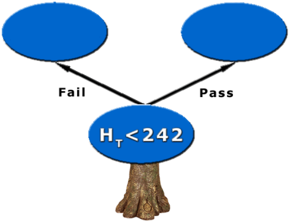
- Consider signal (s_i) and background (b_j) events described by 3 variables: p_T of leading jet, top mass M_t and scalar sum of p_T's of all objects in the event H_T
  - sort all events by each variable:
    - $p_T^{s_1} \leq p_T^{b_{34}} \leq \cdots \leq p_T^{b_2} \leq p_T^{s_{12}}$
    - $H_T^{b_5} \leq H_T^{b_3} \leq \cdots \leq H_T^{s_{67}} \leq H_T^{s_{43}}$
    - $M_t^{b_6} \leq M_t^{s_8} \leq \cdots \leq M_t^{s_{12}} \leq M_t^{b_9}$
  - best split (arbitrary unit):
    - $p_T < 56$ GeV, separation $= 3$
    - $H_T < 242$ GeV, separation $= 5$
    - $M_t < 105$ GeV, separation $= 0.7$

Algorithm example

- Consider signal (s_i) and background (b_j) events described by 3 variables: p_T of leading jet, top mass M_t and scalar sum of p_T's of all objects in the event H_T
  - sort all events by each variable:
    - $p_T^{s_1} \leq p_T^{b_{34}} \leq \cdots \leq p_T^{b_2} \leq p_T^{s_{12}}$
    - $H_T^{b_5} \leq H_T^{b_3} \leq \cdots \leq H_T^{s_{67}} \leq H_T^{s_{43}}$
    - $M_t^{b_6} \leq M_t^{s_8} \leq \cdots \leq M_t^{s_{12}} \leq M_t^{b_9}$
  - best split (arbitrary unit):
    - $p_T < 56$ GeV, separation $= 3$
    - $H_T < 242$ GeV, separation $= 5$
    - $M_t < 105$ GeV, separation $= 0.7$

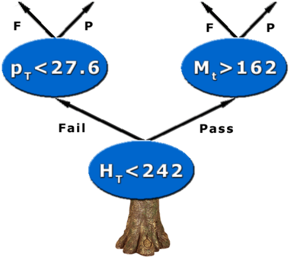
Algorithm example

- Consider signal (s_i) and background (b_j) events described by 3 variables: p_T of leading jet, top mass M_t and scalar sum of p_T's of all objects in the event H_T
  - sort all events by each variable:
    - $p_T^{s_1} \leq p_T^{b_{34}} \leq \cdots \leq p_T^{b_2} \leq p_T^{s_{12}}$
    - $H_T^{b_5} \leq H_T^{b_3} \leq \cdots \leq H_T^{s_{67}} \leq H_T^{s_{43}}$
    - $M_t^{b_6} \leq M_t^{s_8} \leq \cdots \leq M_t^{s_{12}} \leq M_t^{b_9}$
  - best split (arbitrary unit):
    - $p_T < 56$ GeV, separation $= 3$
    - $H_T < 242$ GeV, separation $= 5$
    - $M_t < 105$ GeV, separation $= 0.7$
  - split events in two branches: pass or fail $H_T < 242$ GeV



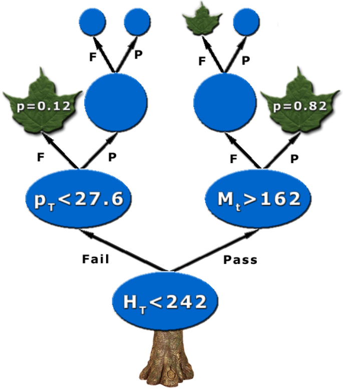
Algorithm example

- Consider signal (s_i) and background (b_j) events described by 3 variables: p_T of leading jet, top mass M_t and scalar sum of p_T's of all objects in the event H_T
    - sort all events by each variable:
        - $p_T^{s_1} \leq p_T^{b_{34}} \leq \cdots \leq p_T^{b_2} \leq p_T^{s_{12}}$
        - $H_T^{b_5} \leq H_T^{b_3} \leq \cdots \leq H_T^{s_{67}} \leq H_T^{s_{43}}$
        - $M_t^{b_6} \leq M_t^{s_8} \leq \cdots \leq M_t^{s_{12}} \leq M_t^{b_9}$
    - best split (arbitrary unit):
        - $p_T < 56$ GeV, separation $= 3$
        - $H_T < 242$ GeV, separation $= 5$
        - $M_t < 105$ GeV, separation $= 0.7$
    - split events in two branches: pass or fail $H_T < 242$ GeV
- Repeat recursively on each node



Algorithm example

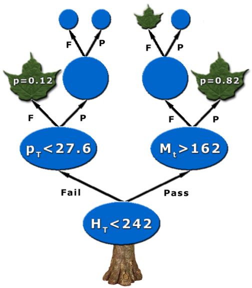
- Consider signal (s_i) and background (b_j) events described by 3 variables: p_T of leading jet, top mass M_t and scalar sum of p_T's of all objects in the event H_T
  - sort all events by each variable:
    - $p_T^{s_1} \leq p_T^{b_{34}} \leq \cdots \leq p_T^{b_2} \leq p_T^{s_{12}}$
    - $H_T^{b_5} \leq H_T^{b_3} \leq \cdots \leq H_T^{s_{67}} \leq H_T^{s_{43}}$
    - $M_t^{b_6} \leq M_t^{s_8} \leq \cdots \leq M_t^{s_{12}} \leq M_t^{b_9}$
  - best split (arbitrary unit):
    - $p_T < 56$ GeV, separation $= 3$
    - $H_T < 242$ GeV, separation $= 5$
    - $M_t < 105$ GeV, separation $= 0.7$
  - split events in two branches: pass or fail $H_T < 242$ GeV
- Repeat recursively on each node
- Splitting stops: e.g. events with $H_T < 242$ GeV and $M_t > 162$ GeV are signal like ($p = 0.82$)



Decision tree output

Run event through tree

- Start from root node
- Apply first best cut
- Go to left or right child node
- Apply best cut for this node
- ...Keep going until...
- Event ends up in leaf



DT Output

- Purity $\left(\frac{s}{s+b}\right.$, with weighted events$\left.\right)$ of leaf, close to 1 for signal and 0 for background
- or binary answer (discriminant function $+1$ for signal, -1 or 0 for background) based on purity above/below specified value (e.g. $\frac{1}{2}$) in leaf
- E.g. events with $H_T < 242$ GeV and $M_t > 162$ GeV have a DT output of 0.82 or $+1$

Tree construction parameters

Normalization of signal and background before training

- Balanced classes: same total weight for signal and background events ($p = 0.5$, maximal mixing)

Selection of splits

- list of questions ($variable_i < cut_i$?, “Is the sky blue or overcast?”)
- goodness of split (separation measure)

Decision to stop splitting (declare a node terminal)

- minimum leaf size (for statistical significance, e.g. 100 events)
- insufficient improvement from further splitting
- perfect classification (all events in leaf belong to same class)
- maximal tree depth (like-size trees choice or computing concerns)

Assignment of terminal node to a class

- signal leaf if purity > 0.5, background otherwise

Splitting a node

Impurity measure $i(t)$

- maximal for equal mix of signal and background
- symmetric in p_{signal} and $\mathrm{p}_{background}$
- minimal for node with either signal only or background only
- strictly concave ⇒ reward purer nodes (favours end cuts with one smaller node and one larger node)

Optimal split: figure of merit

- Decrease of impurity for split s of node t into children t_P and t_F (goodness of split):
  $\Delta i(s,t) = i(t) - p_P \cdot i(t_P) - p_F \cdot i(t_F)$
- Aim: find split s^* such that:

$$\Delta i(s^*,t) = \max_{s \in \{\text{splits}\}} \Delta i(s,t)$$

- Maximising $\Delta i(s,t) \equiv$ minimizing overall tree impurity

Splitting a node: examples

Node purity

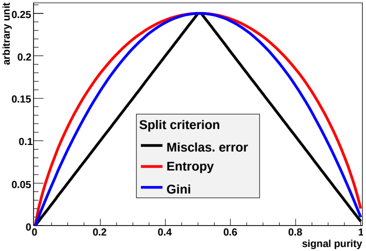
- Signal (background) event i with weight w_s^i (w_b^i)

$$p = \frac{\sum_{i \in signal} w_s^i}{\sum_{i \in signal} w_s^i + \sum_{j \in bkg} w_b^j}$$

- Signal purity (= purity)
  $p_s = p = \frac{s}{s+b}$
- Background purity
  $p_b = \frac{b}{s+b} = 1 - p_s = 1 - p$

Common impurity functions

- misclassification error
  $= 1 - max(p, 1-p)$
- (cross) entropy
  $= -\sum_{i=s,b} p_i \log p_i$
- Gini index



- Also cross section ($-\frac{s^2}{s+b}$) and excess significance ($-\frac{s^2}{b}$)

Splitting a node: Gini index of diversity

Defined for many classes

- Gini $= \sum_{i,j\in\{\text{classes}\}}^{i\neq j} p_i p_j$

Statistical interpretation

- Assign random object to class i with probability p_i.
- Probability that it is actually in class j is p_j
- $\Rightarrow$ Gini = probability of misclassification

For two classes (signal and background)

- $i = s, b$ and $p_s = p = 1 - p_b$
- $\Rightarrow$ Gini $= 1 - \sum_{i=s,b} p_i^2 = 2p(1-p) = \frac{2sb}{(s+b)^2}$
- Most popular in DT implementations
- Usually similar performance to e.g. entropy

Variable selection I

Reminder

- Need model giving good description of data

Variable selection I

Reminder

- Need model giving good description of data

Playing with variables

- Number of variables:
  - not affected too much by "curse of dimensionality"
  - CPU consumption scales as $nN \log N$ with n variables and N training events
- Insensitive to duplicate variables (give same ordering ⇒ same DT)
- Variable order does not matter: all variables treated equal
- Order of training events is irrelevant (batch training)
- Irrelevant variables:
  - no discriminative power ⇒ not used
  - only costs a little CPU time, no added noise
- Can use continuous and discrete variables, simultaneously

Variable selection II

Transforming input variables

- Completely insensitive to replacement of any subset of input variables by (possibly different) arbitrary strictly monotone functions of them:
  - let $f : x_i \rightarrow f(x_i)$ be strictly monotone
  - if $x > y$ then $f(x) > f(y)$
  - ordering of events by x_i is the same as by $f(x_i)$
  - $\Rightarrow$ produces the same DT
- Examples:
  - convert MeV → GeV
  - no need to make all variables fit in the same range
  - no need to regularise variables (e.g. taking the log)
- $\Rightarrow$ Some immunity against outliers

Variable selection II

Transforming input variables

- Completely insensitive to replacement of any subset of input variables by (possibly different) arbitrary strictly monotone functions of them:
  - let $f : x_i \to f(x_i)$ be strictly monotone
  - if $x > y$ then $f(x) > f(y)$
  - ordering of events by x_i is the same as by $f(x_i)$
  - $\Rightarrow$ produces the same DT
- Examples:
  - convert MeV → GeV
  - no need to make all variables fit in the same range
  - no need to regularise variables (e.g. taking the log)
- $\Rightarrow$ Some immunity against outliers

Note about actual implementation

- The above is strictly true only if testing all possible cut values
- If there is some computational optimisation (e.g., check only 20 possible cuts on each variable), it may not work anymore

Variable selection III

Variable ranking (mean decrease impurity MDI)

- Ranking of x_i: add up decrease of impurity each time x_i is used
- Largest decrease of impurity = best variable

Shortcoming: masking of variables

- 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
  $\Rightarrow$ careful with interpreting ranking

Solution: surrogate split

- Compare which events are sent left or right by optimal split and by any other split
- Give higher score to split that mimics better the optimal split
- Highest score = surrogate split
- Can be included in variable ranking
- Helps in case of missing data: replace optimal split by surrogate

Variable selection IV

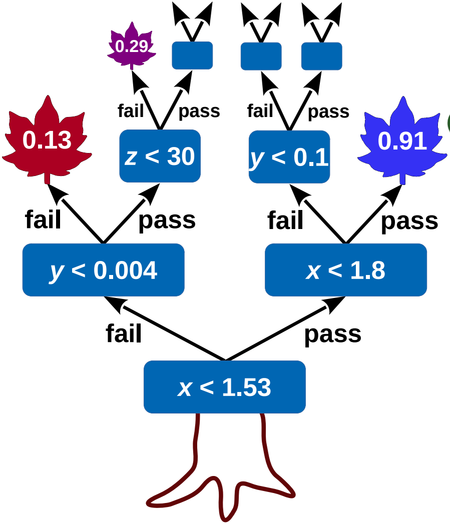
Permutation importance (mean decrease accuracy MDA)

- Applicable to any already trained classifier
- Randomly shuffle each variable in turn and measure decrease of performance
- Important variable ⇒ big loss of performance
- Can also be performed on validation sample
- Beware of correlations [Breiman 2001]

Choosing variables

- Usually try to have as few variables as possible
- But difficult: correlations, possibly large number to consider, large phase space with different properties in different regions
- *Brute force:* with n variables train all n, $n-1$, etc. combinations, pick best
- *Backward elimination:* train with n variables, then train all $n-1$ variables trees and pick best one; now train all $n-2$ variables trees starting from the $n-1$ variable list; etc. Pick optimal cost-complexity tree.
- *Forward greedy selection:* start with $k=1$ variable, then train all $k+1$ variables trees and pick the best; move to $k+2$ variables; etc.

Limitations



9 (Boosted) Decision trees

- Decision trees
- Limitations
  - Training sample composition
  - Pruning a tree
  - Ensemble learning
- Boosted decision trees
- Performance examples
- BDTs in real physics cases
- Software and example code

Tree instability: training sample composition

- Small changes in sample can lead to very different tree structures (high variance)
- Performance on testing events may be as good, or not
- Not optimal to understand data from DT rules
- Does not give confidence in result:
  - DT 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)

Pruning a tree

Why prune a tree?

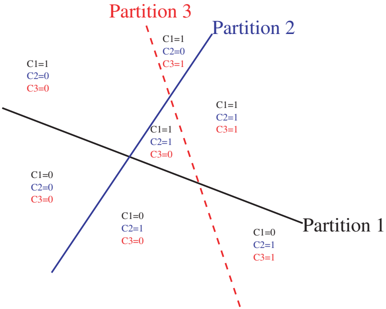
- Possible to get a perfect classifier on training events
- Mathematically misclassification error can be made as little as wanted
- E.g. tree with one class only per leaf (down to 1 event per leaf if necessary)
- Training error is zero
- But run new independent events through tree (testing or validation sample): misclassification is probably > 0, overtraining
- Pruning: eliminate subtrees (branches) that seem too specific to training sample:
  - a node and all its descendants turn into a leaf

Pruning algorithms (details in ▸ backup)

- Pre-pruning (early stopping condition like min leaf size, max depth)
- Expected error pruning (based on statistical error estimate)
- Cost-complexity pruning (penalise "complex" trees with many nodes/leaves)

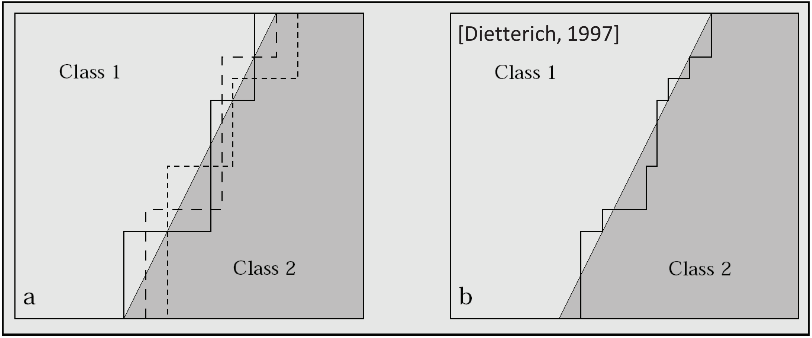
Tree (in)stability: distributed representation

- One tree:
  - one information about event (one leaf)
  - cannot really generalise to variations not covered in training set (at most as many leaves as input size)
- Many trees:
  - distributed representation: number of intersections of leaves exponential in number of trees
  - many leaves contain the event ⇒ richer description of input pattern



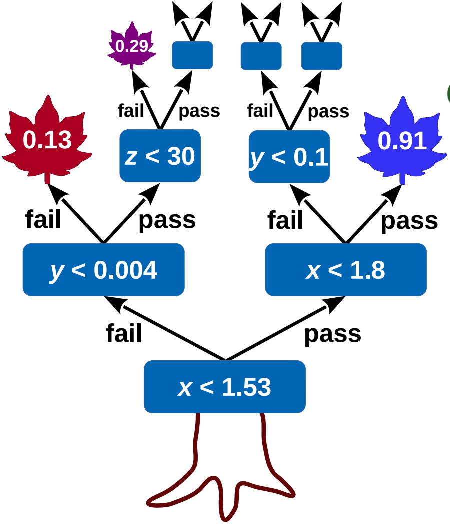
Tree (in)stability solution: averaging

- Build several trees and average the output



- K-fold cross-validation (good for small samples)
  - divide training sample $\mathcal{L}$ in K subsets of equal size: $\mathcal{L} = \bigcup_{k=1..K} \mathcal{L}_k$
  - Train tree T_k on $\mathcal{L} - \mathcal{L}_k$, test on $\mathcal{L}_k$
  - DT output $= \frac{1}{K} \sum_{k=1..K} T_k$
- Bagging, boosting, random forests, etc.

Boosted decision trees



9 (Boosted) Decision trees

- Decision trees
- Limitations
- Boosted decision trees
  - Introduction
  - AdaBoost
  - Overtraining?
  - Clues to boosting performance
  - Gradient boosting [Friedman 2001]
- Performance examples
- BDTs in real physics cases
- Software and example code

Boosting: a brief history

First provable algorithm [Schapire 1990]

- Train classifier T_1 on N events
- Train T_2 on new N-sample, half of which misclassified by T_1
- Build T_3 on events where T_1 and T_2 disagree
- Boosted classifier: $\text{MajorityVote}(T_1, T_2, T_3)$

Boosting: a brief history

First provable algorithm [Schapire 1990]

- Train classifier T_1 on N events
- Train T_2 on new N-sample, half of which misclassified by T_1
- Build T_3 on events where T_1 and T_2 disagree
- Boosted classifier: MajorityVote(T_1, T_2, T_3)

Then

- Variation [Freund 1995]: boost by majority (combining many learners with fixed error rate)
- Freund&Schapire joined forces: 1st functional model AdaBoost (1996)

Boosting: a brief history

First provable algorithm [Schapire 1990]

- Train classifier T_1 on N events
- Train T_2 on new N-sample, half of which misclassified by T_1
- Build T_3 on events where T_1 and T_2 disagree
- Boosted classifier: MajorityVote(T_1, T_2, T_3)

Then

- Variation [Freund 1995]: boost by majority (combining many learners with fixed error rate)
- Freund&Schapire joined forces: 1st functional model AdaBoost (1996)

When it really picked up in HEP

- MiniBooNe compared performance of different boosting algorithms and neural networks for particle ID [MiniBooNe 2005]
- D0 claimed first evidence for single top quark production [D0 2006]
- CDF copied 😉 (2008). Both used BDT for single top observation

Principles of boosting

What is 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

Algorithm

- Training sample $\mathbb{T}_k$ of N events. For i^{th} event:
  - weight w_i^k
  - vector of discriminative variables x_i
  - class label $y_i = +1$ for signal, -1 for background
- Pseudocode:

```
Initialise $\mathbb{T}_1$
for $k$ in 1..$N_{tree}$
  train classifier $T_k$ on $\mathbb{T}_k$
  assign weight $\alpha_k$ to $T_k$
  modify $\mathbb{T}_k$ into $\mathbb{T}_{k+1}$
```

- Boosted output: $F(T_1, .., T_{N_{tree}})$

AdaBoost [Freund&Schapire 1996]

- Introduced by Freund&Schapire in 1996
- Stands for *adaptive boosting*
- Learning procedure adjusts to training data to classify it better
- Many variations on the same theme for actual implementation
- Most common boosting algorithm around
- Usually leads to better results than without boosting

AdaBoost algorithm

- Check which events of training sample $\mathbb{T}_k$ are misclassified by T_k:
  - $\mathbb{I}(X) = 1$ if X is true, 0 otherwise
  - for DT output in $\{\pm 1\}$: $\text{isMisclassified}_k(i) = \mathbb{I}\big(y_i \times T_k(x_i) \leq 0\big)$
  - or $\text{isMisclassified}_k(i) = \mathbb{I}\big(y_i \times (T_k(x_i) - 0.5) \leq 0\big)$ in purity convention
  - misclassification rate:

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

- Derive tree weight $\alpha_k = \beta \times \ln((1 - \varepsilon_k)/\varepsilon_k)$
- Increase weight of misclassified events in $\mathbb{T}_k$ to create $\mathbb{T}_{k+1}$:

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

- Train T_{k+1} on $\mathbb{T}_{k+1}$
- Boosted result of event i:

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

AdaBoost by example

- Assume $\beta = 1$

Not-so-good classifier

- Assume error rate $\varepsilon = 40\%$
- Then $\alpha = \ln \frac{1-0.4}{0.4} = 0.4$
- Misclassified events get their weight multiplied by $e^{0.4}$=1.5
- $\Rightarrow$ next tree will have to work a bit harder on these events

Good classifier

- Error rate $\varepsilon = 5\%$
- Then $\alpha = \ln \frac{1-0.05}{0.05} = 2.9$
- Misclassified events get their weight multiplied by $e^{2.9}$=19 (!!)
- $\Rightarrow$ 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

AdaBoost error rate

Misclassification rate ε on training sample

- Can be shown to be bound:

$$\varepsilon \leq \prod_{k=1}^{N_{tree}} 2\sqrt{\varepsilon_k(1-\varepsilon_k)}$$

- If each tree has $\varepsilon_k \neq 0.5$ (i.e. better than random guessing):

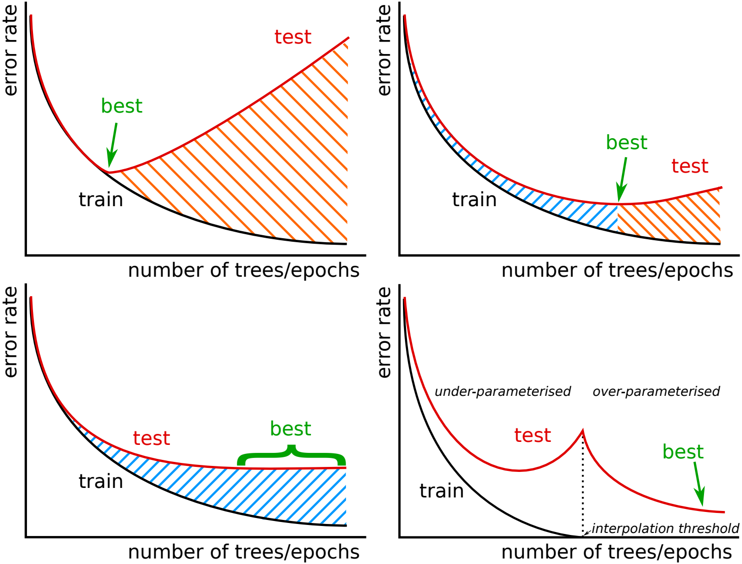
the error rate falls to zero for sufficiently large N_{tree}

- Corollary: training data is overfitted

Overtraining?

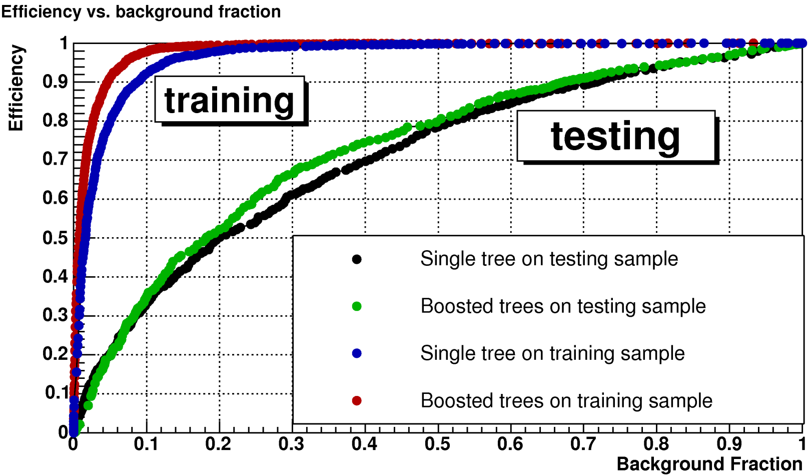
- Error rate on test sample may reach a minimum and then potentially rise. Stop boosting at the minimum.
- In principle AdaBoost *must* overfit training sample
- In many cases in literature, no loss of performance due to overtraining
  - may have to do with fact that successive trees get in general smaller and smaller weights
  - trees that lead to overtraining contribute very little to final DT output on validation sample

Overtraining estimation: good or bad?



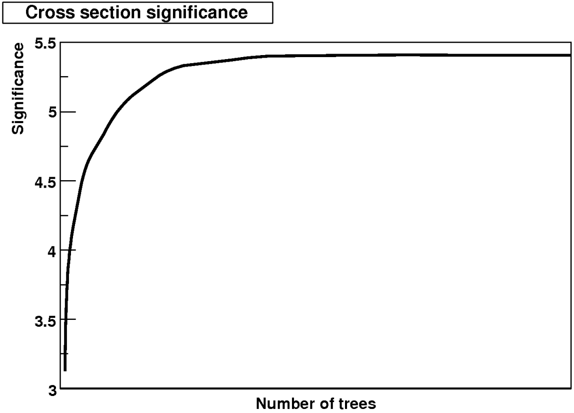
"bad" overtraining (overfitting) / "good" overtraining (still underfitting)

Training and generalisation error



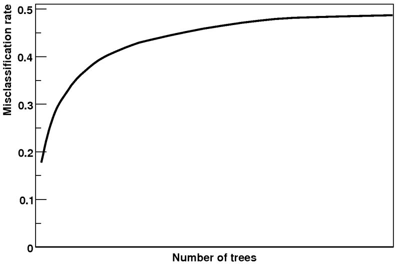
- Clear overtraining, but still better performance after boosting

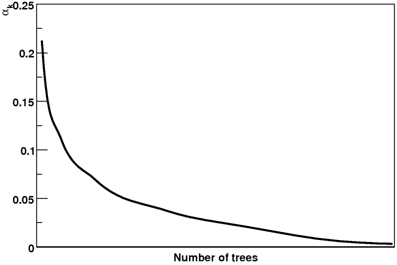
Cross section significance ($s/\sqrt{s+b}$)



- More relevant than testing error
- Reaches plateau
- Afterwards, boosting does not hurt (just wasted CPU)
- Applicable to any other figure of merit of interest for your use case

Clues to boosting performance


Misclassification rate for each tree

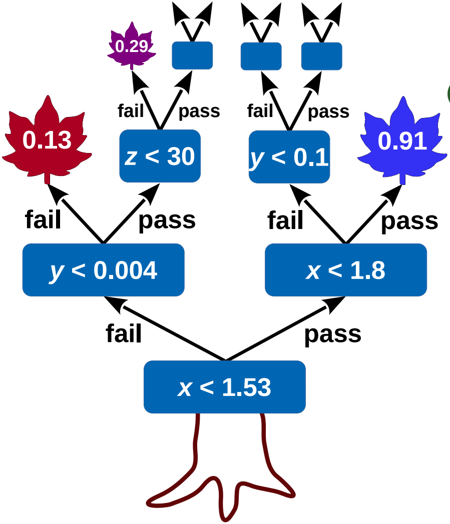

Tree weight α_k

- First tree is best, others are minor corrections
- Specialised trees do not perform well on most events ⇒ decreasing tree weight and increasing misclassification rate
- Last tree is not better evolution of first tree, but rather a pretty bad DT that only does a good job on few cases that the other trees could not get right
- But adding trees may increase reliability of prediction: margins explanation [Shapire&Freund 2012]
- Double descent risk curve and interpolation regime [Belkin 2019]

Gradient boosting

- AdaBoost recast in a statistical framework: corresponds to minimising an exponential loss
- Generalisation: formulate boosting as numerical optimisation problem, minimise loss function by adding trees using gradient descent procedure
- Build imperfect model F_k at step k (sometimes $F_k(x) \neq y$)
- Improve model: $F_{k+1}(x) = F_k(x) + h_k(x) = y$, or residual $h_k(x) = y - F_k(x)$
- Train new classifier on residual
- Example: mean squared error loss function $L_{\mathrm{MSE}}(x,y) = \frac{1}{2}\left(y - F_k(x)\right)^2$
    - minimising loss $J = \sum_i L_{\mathrm{MSE}}(x_i, y_i)$ leads to $\frac{\partial J}{\partial F_k(x_i)} = F_k(x_i) - y_i$ $\Rightarrow$ residual as negative gradient: $h_k(x_i) = y_i - F_k(x_i) = -\frac{\partial J}{\partial F_k(x_i)}$
- Generalised to any differentiable loss function

Performance examples

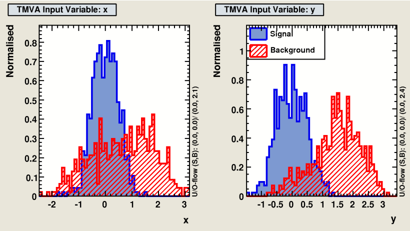


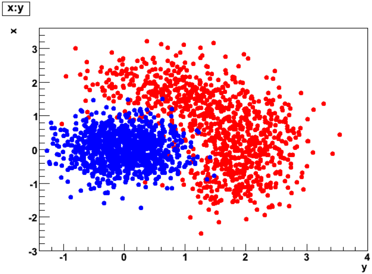
9 (Boosted) Decision trees

- Decision trees
- Limitations
- Boosted decision trees
- Performance examples
  - First is best
  - XOR problem
  - Circular correlation
  - Many small trees or fewer large trees?
- BDTs in real physics cases
- Software and example code

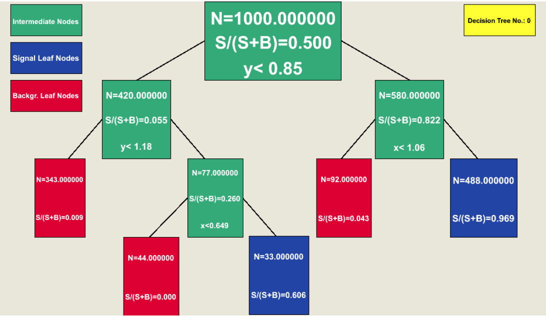
Concrete example

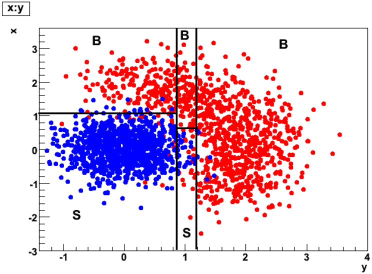
- Using ROOT and TMVA with basic code to make examples (more later)



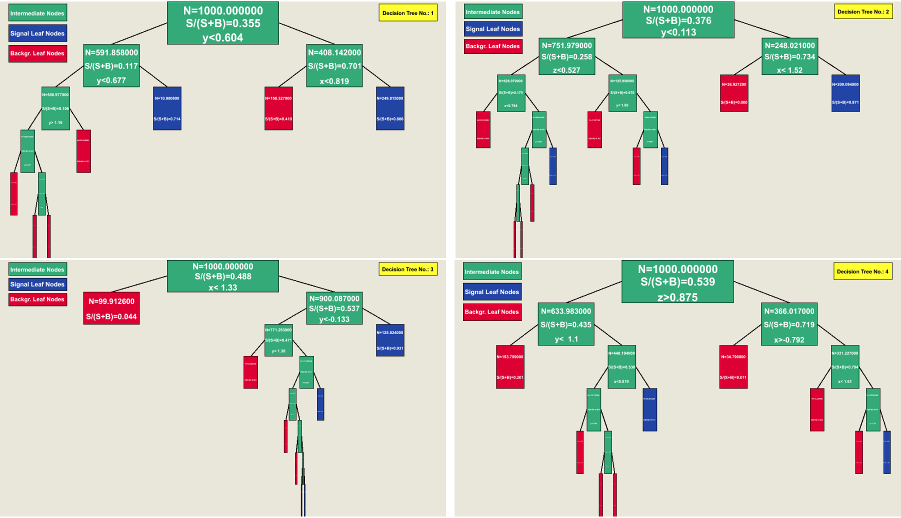


Concrete example



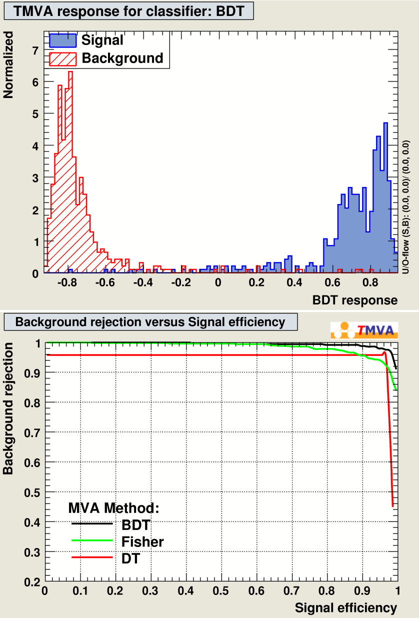


Concrete example



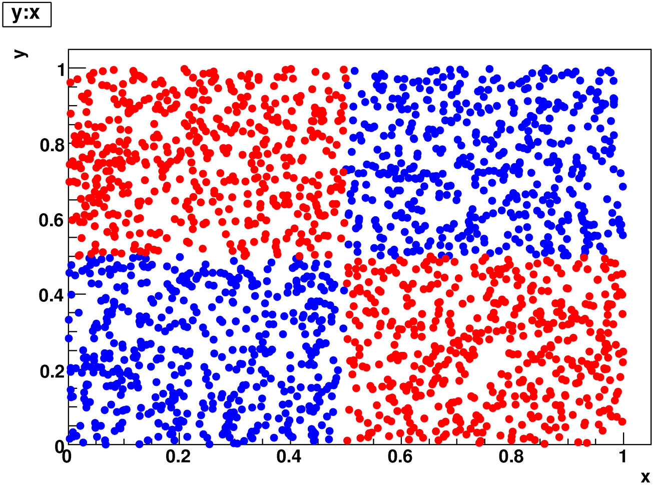
- Specialised trees

Concrete example

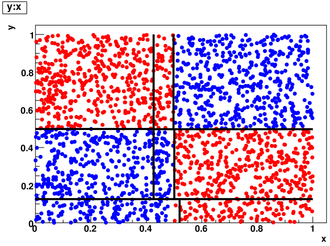

TMVA response for classifier: BDT

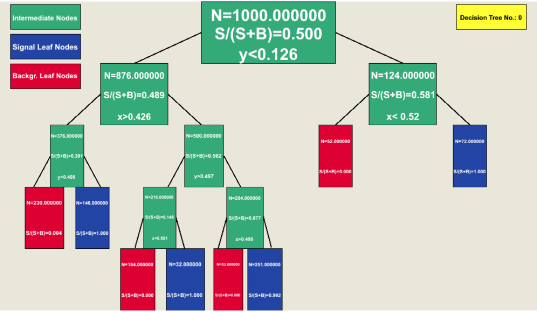
Background rejection versus Signal efficiency

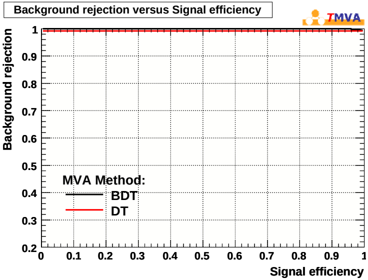
Concrete example: XOR



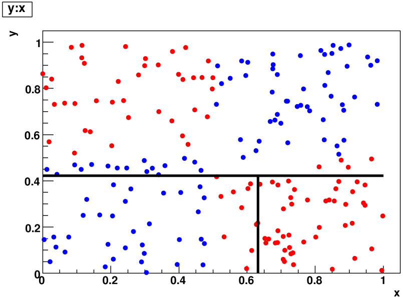
Concrete example: XOR





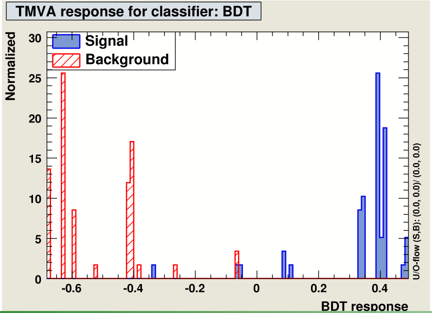


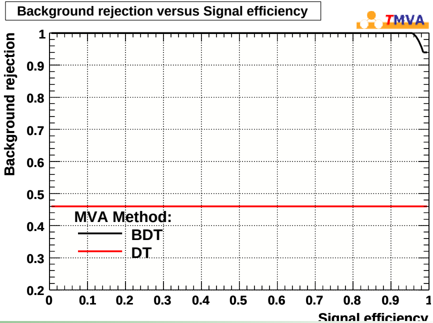
Concrete example: XOR with 100 events



Small statistics

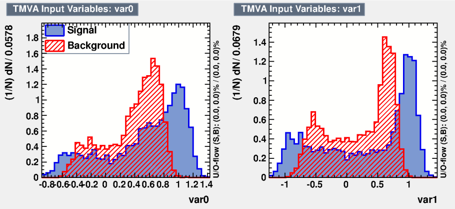
- Single tree not so good
- BDT very good: high performance discriminant from combination of weak classifiers

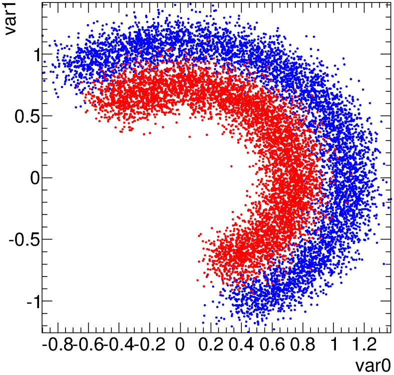




Circular correlation

- Using TMVA and `create_circ` macro from `$ROOTSYS/tutorials/tmva/createData.C` to generate dataset
- Plots: `TMVA::TMVAGui("filename");`

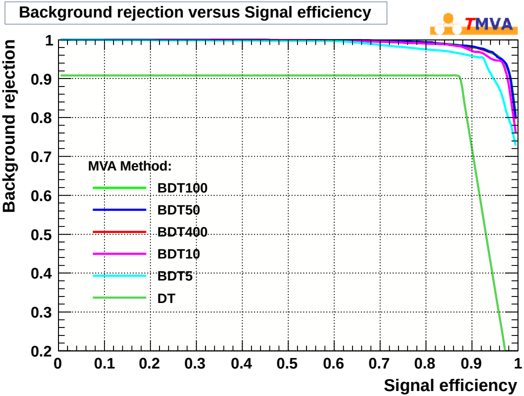




Circular correlation

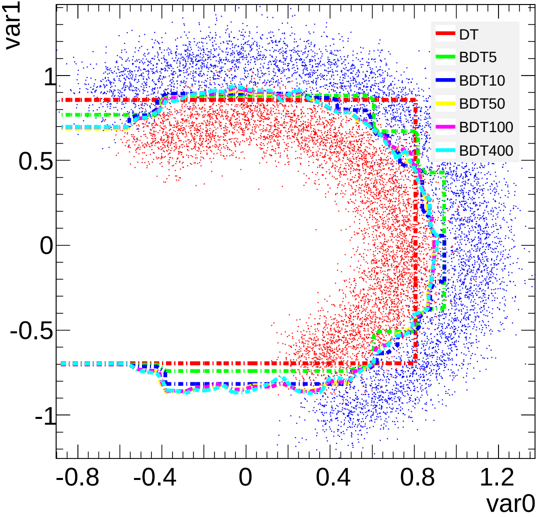
Boosting longer (TMVA: `NTrees`)

- Compare performance of single DT and BDT with more and more trees (5 to 400)
- All other parameters at TMVA default (would be 400 trees)



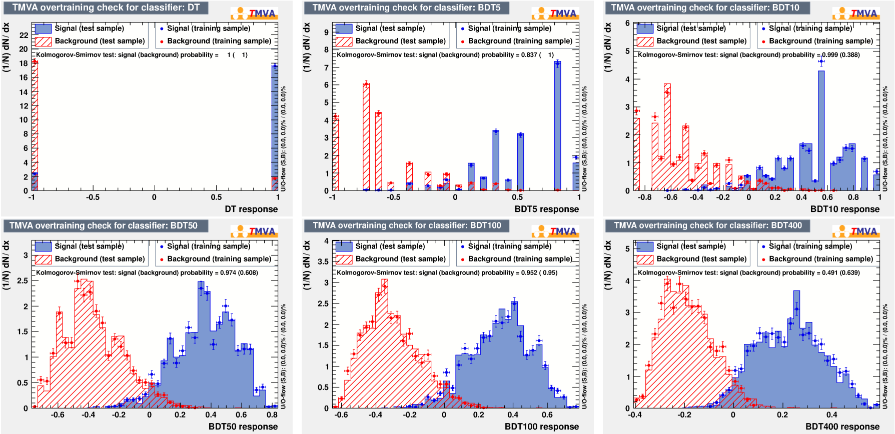
- Single (small) DT: not so good
- More trees ⇒ improve performance until saturation

Decision contours



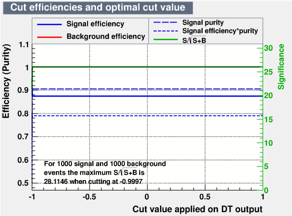
- Note: max tree depth = 3
- Single (small) DT: not so good. Note: a larger tree would solve this problem
- More trees ⇒ improve performance (less step-like, closer to optimal separation) until saturation
- Largest BDTs: wiggle a little around the contour ⇒ picked up features of training sample, that is, overtraining

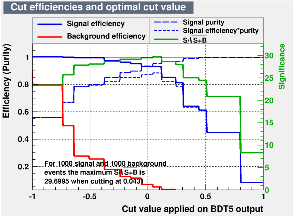
Training/testing output

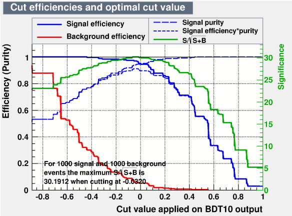


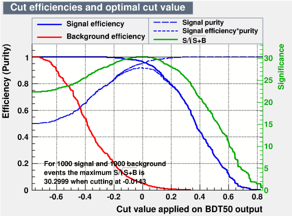
- Better shape with more trees: quasi-continuous
- Overtraining because of disagreement between training and testing? Let's see

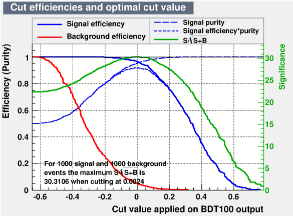
Performance in optimal significance

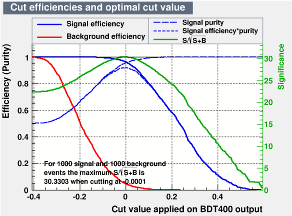






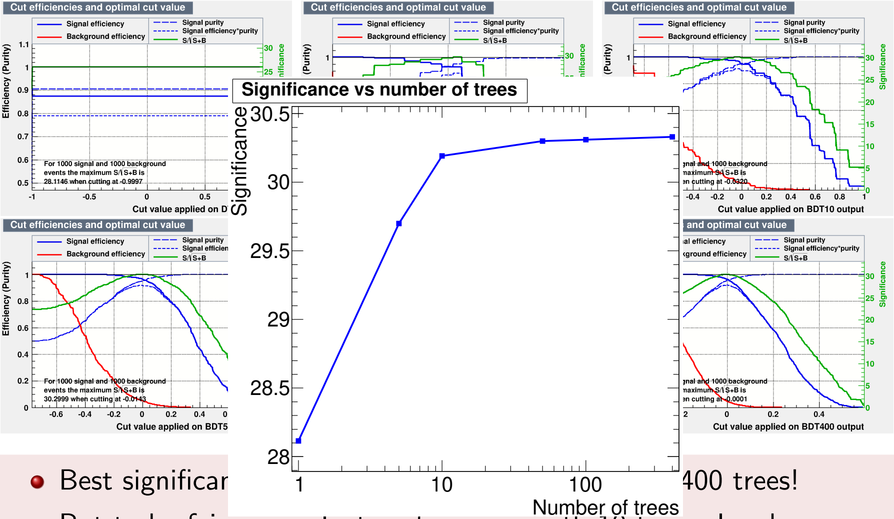






- Best significance actually obtained with last BDT, 400 trees!
- But to be fair, equivalent performance with 10 trees already
- Less “stepped” output desirable? ⇒ maybe 50 is reasonable

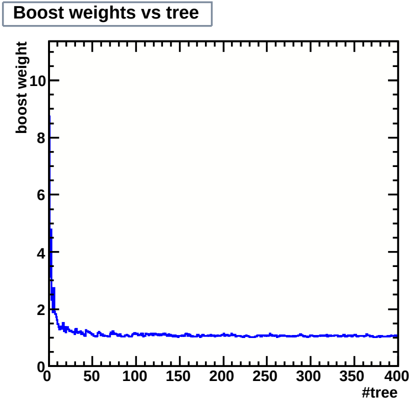
Performance in optimal significance

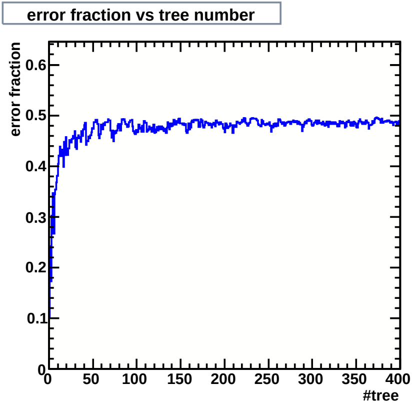


- Best significar 400 trees!
- But to be fair, equivalent performance with 10 trees already
- Less “stepped” output desirable? ⇒ maybe 50 is reasonable

Control plots

- Boosting weight decreases fast and stabilises
- First trees have small error fractions, then increases towards 0.5 (random guess)
- ⇒ confirms that best trees are first ones, others are small corrections

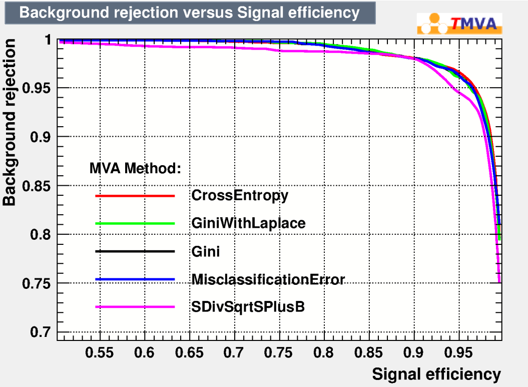




Circular correlation

Separation criterion for node splitting (TMVA: `SeparationType`)

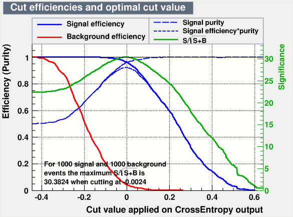
- Compare performance of Gini, entropy, misclassification error, $\frac{s}{\sqrt{s+b}}$
- All other parameters at TMVA default



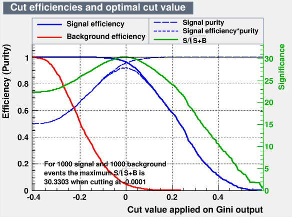
- Very similar performance (even zooming on corner)
- Small degradation (in this particular case) for $\frac{s}{\sqrt{s+b}}$: only criterion that does not respect good properties of impurity measure (see earlier: maximal for equal mix of signal and bkg, symmetric in p_{sig} and p_{bkg}, minimal for node with either signal only or bkg only, strictly concave)

Circular correlation

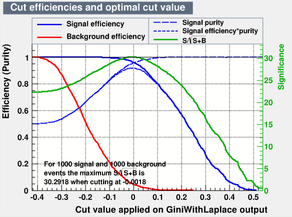
Performance in optimal significance


Cut efficiencies and optimal cut value

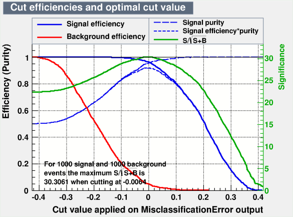
For 1000 signal and 1000 background events the maximum $S/\sqrt{S+B}$ is 30.3824 when cutting at -0.0024


Cut efficiencies and optimal cut value

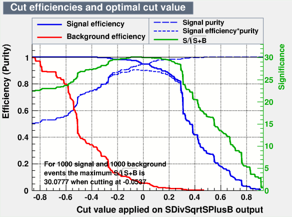
For 1000 signal and 1000 background events the maximum $S/\sqrt{S+B}$ is 30.3303 when cutting at -0.0001


Cut efficiencies and optimal cut value

For 1000 signal and 1000 background events the maximum $S/\sqrt{S+B}$ is 30.2918 when cutting at -0.0018


Cut efficiencies and optimal cut value

For 1000 signal and 1000 background events the maximum $S/\sqrt{S+B}$ is 30.3061 when cutting at -0.0064

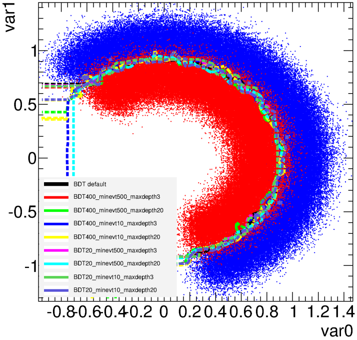

Cut efficiencies and optimal cut value

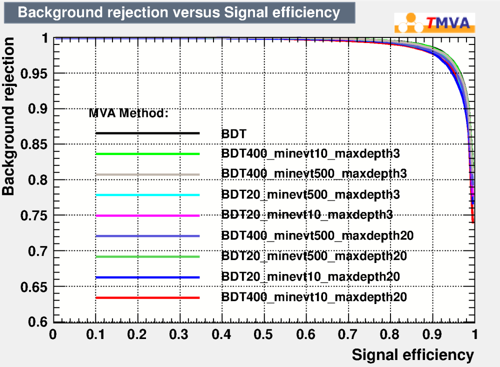
For 1000 signal and 1000 background events the maximum $S/\sqrt{S+B}$ is 30.0777 when cutting at -0.0537

- Confirms previous page: very similar performance, worse for BDT optimised with significance!

Many small trees or fewer large trees?

- Using same `create_circ` macro but generating larger dataset to avoid stats limitations
- 20 or 400 trees; minimum leaf size: 10 or 500 events (`MinNodeSize`)
- Maximum depth (max # of cuts to reach leaf): 3 or 20 (`MaxDepth`)





- Overall: very comparable performance. Depends on use case.

Other boosting algorithms

ε-Boost (shrinkage)

- reweight misclassified events by a fixed $e^{2\varepsilon}$ factor
- $T(i) = \sum_{k=1}^{N_{\text{tree}}} \varepsilon T_k(i)$

ε-LogitBoost

- reweight misclassified events by logistic function $\frac{e^{-y_i T_k(x_i)}}{1+e^{-y_i T_k(x_i)}}$
- $T(i) = \sum_{k=1}^{N_{\text{tree}}} \varepsilon T_k(i)$

Real AdaBoost

- DT output is $T_k(i) = 0.5 \times \ln \frac{p_k(i)}{1-p_k(i)}$ where $p_k(i)$ is purity of leaf on which event i falls
- reweight events by $e^{-y_i T_k(i)}$
- $T(i) = \sum_{k=1}^{N_{\text{tree}}} T_k(i)$

- ε-HingeBoost, LogitBoost, Gentle AdaBoost, etc.

Other averaging techniques

Bagging (Bootstrap aggregating) [Breiman 1996]

- Before building tree T_k take random sample of N events from training sample with replacement
- Train T_k on it
- Events not picked form “out of bag” validation sample
- Applicable to other techniques than DT
    - tends to produce more stable and better classifier

Other averaging techniques

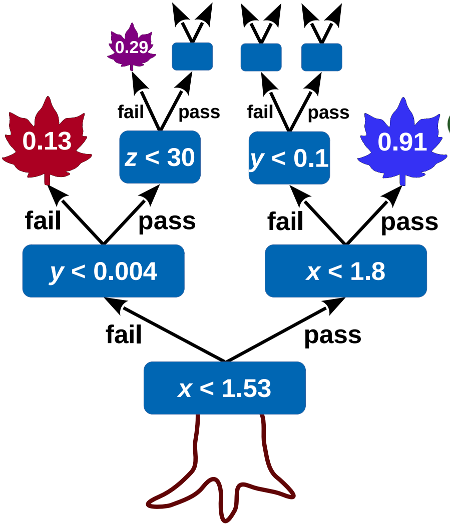
Bagging (Bootstrap aggregating) [Breiman 1996]

- Before building tree T_k take random sample of N events from training sample with replacement
- Train T_k on it
- Events not picked form "out of bag" validation sample
- Applicable to other techniques than DT
  - tends to produce more stable and better classifier

Random forests [Breiman 2001]

- Same as bagging
- In addition, pick random subset of variables to consider for each node split
- Two levels of randomisation, much more stable output
- Often as good as boosting

BDTs in real physics cases



9 (Boosted) Decision trees

- Decision trees
- Limitations
- Boosted decision trees
- Performance examples
- BDTs in real physics cases
  - Single top search at D0
  - LHC examples
  - BDT systematics
- Software and example code

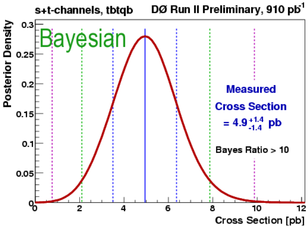
Single top production evidence at D0 (2006)

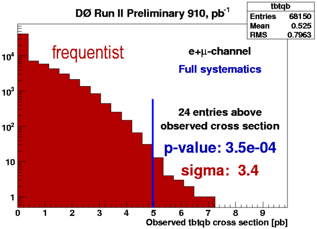
- Three multivariate techniques: BDT, Matrix Elements, BNN
- Most sensitive: BDT

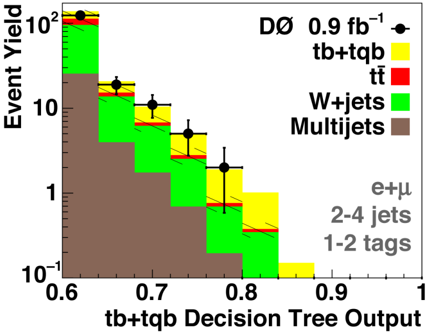
$$\sigma_{s+t} = 4.9 \pm 1.4 \text{ pb}$$

p-value = 0.035% (3.4σ)

SM compatibility: 11% (1.3σ)







$$\sigma_s = 1.0 \pm 0.9 \text{ pb}$$

$$\sigma_t = 4.2^{+1.8}_{-1.4} \text{ pb}$$

▸ Phys. Rev. D**78**, 012005 (2008)

Decision trees — 49 input variables

Object Kinematics

p_T(jet1)
p_T(jet2)
p_T(jet3)
p_T(jet4)
p_T(best1)
p_T(notbest1)
p_T(notbest2)
p_T(tag1)
p_T(untag1)
p_T(untag2)

Angular Correlations

ΔR(jet1,jet2)
cos(best1,lepton)$_{\rm besttop}$
cos(best1,notbest1)$_{\rm besttop}$
cos(tag1,alljets)$_{\rm alljets}$
cos(tag1,lepton)$_{\rm btaggedtop}$
cos(jet1,alljets)$_{\rm alljets}$
cos(jet1,lepton)$_{\rm btaggedtop}$
cos(jet2,alljets)$_{\rm alljets}$
cos(jet2,lepton)$_{\rm btaggedtop}$
cos(lepton,Q(lepton)$\times z$)$_{\rm besttop}$
cos(lepton$_{\rm besttop}$,besttop$_{\rm CMframe}$)
cos(lepton$_{\rm btaggedtop}$,btaggedtop$_{\rm CMframe}$)
cos(notbest,alljets)$_{\rm alljets}$
cos(notbest,lepton)$_{\rm besttop}$
cos(untag1,alljets)$_{\rm alljets}$
cos(untag1,lepton)$_{\rm btaggedtop}$

Event Kinematics

Aplanarity(alljets,W)
M(W,best1) ("best" top mass)
M(W,tag1) ("*b*-tagged" top mass)
H_T(alljets)
H_T(alljets$-$best1)
H_T(alljets$-$tag1)
H_T(alljets,W)
H_T(jet1,jet2)
H_T(jet1,jet2,W)
M(alljets)
M(alljets$-$best1)
M(alljets$-$tag1)
M(jet1,jet2)
M(jet1,jet2,W)
M_T(jet1,jet2)
M_T(W)
Missing E_T
p_T(alljets$-$best1)
p_T(alljets$-$tag1)
p_T(jet1,jet2)
Q(lepton)$\times\eta$(untag1)
$\sqrt{\hat{s}}$
Sphericity(alljets,W)

- Adding variables did not degrade performance
- Tested shorter lists, lost some sensitivity
- Same list used for all channels

Decision trees — 49 input variables

Object Kinematics

p_T(jet1)
p_T(jet2)
p_T(jet3)
p_T(jet4)
p_T(best1)
p_T(notbest1)
p_T(notbest2)
p_T(tag1)
p_T(untag1)
p_T(untag2)

Angular Correlations

ΔR(jet1,jet2)
$\cos(\text{best1,lepton})_{\mathrm{besttop}}$
$\cos(\text{best1,notbest1})_{\mathrm{besttop}}$
$\cos(\text{tag1,alljets})_{\mathrm{alljets}}$
$\cos(\text{tag1,lepton})_{\mathrm{btaggedtop}}$
$\cos(\text{jet1,alljets})_{\mathrm{alljets}}$
$\cos(\text{jet1,lepton})_{\mathrm{btaggedtop}}$
$\cos(\text{jet2,alljets})_{\mathrm{alljets}}$
$\cos(\text{jet2,lepton})_{\mathrm{btaggedtop}}$
$\cos(\text{lepton}, Q(\text{lepton})\times z)_{\mathrm{besttop}}$
$\cos(\text{lepton}_{\mathrm{besttop}}, \text{besttop}_{\mathrm{CMframe}})$
$\cos(\text{lepton}_{\mathrm{btaggedtop}}, \text{btaggedtop}_{\mathrm{CMframe}})$
$\cos(\text{notbest,alljets})_{\mathrm{alljets}}$
$\cos(\text{notbest,lepton})_{\mathrm{besttop}}$
$\cos(\text{untag1,alljets})_{\mathrm{alljets}}$
$\cos(\text{untag1,lepton})_{\mathrm{btaggedtop}}$

Event Kinematics

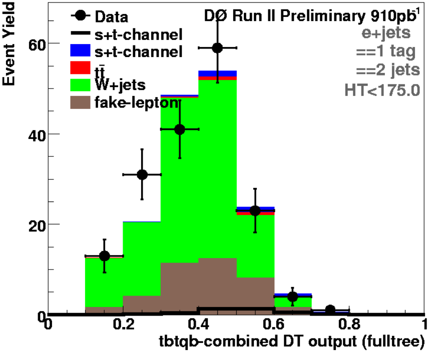
Aplanarity(alljets,W)
M(W,best1) ("best" top mass)
M(W,tag1) ("*b*-tagged" top mass)
H_T(alljets)
H_T(alljets−best1)
H_T(alljets−tag1)
H_T(alljets,W)
H_T(jet1,jet2)
H_T(jet1,jet2,W)
M(alljets)
M(alljets−best1)
M(alljets−tag1)
M(jet1,jet2)
M(jet1,jet2,W)
M_T(jet1,jet2)
M_T(W)
Missing E_T
p_T(alljets−best1)
p_T(alljets−tag1)
p_T(jet1,jet2)
Q(lepton)$\times\eta$(untag1)
$\sqrt{\hat{s}}$
Sphericity(alljets,W)

- Adding variables did not degrade performance
- Tested shorter lists, lost some sensitivity
- Same list used for all channels
- Best theoretical variable: H_T(alljets,W). But detector not perfect ⇒ capture the essence from several variations usually helps "dumb" MVA

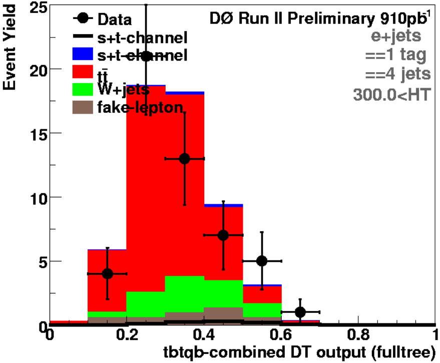
Cross-check samples

- Validate method on data in no-signal region

- **"W+jets":** = 2 jets, $H_T(\text{lepton}, E_T^{\text{miss}}, \text{alljets}) < 175$ GeV



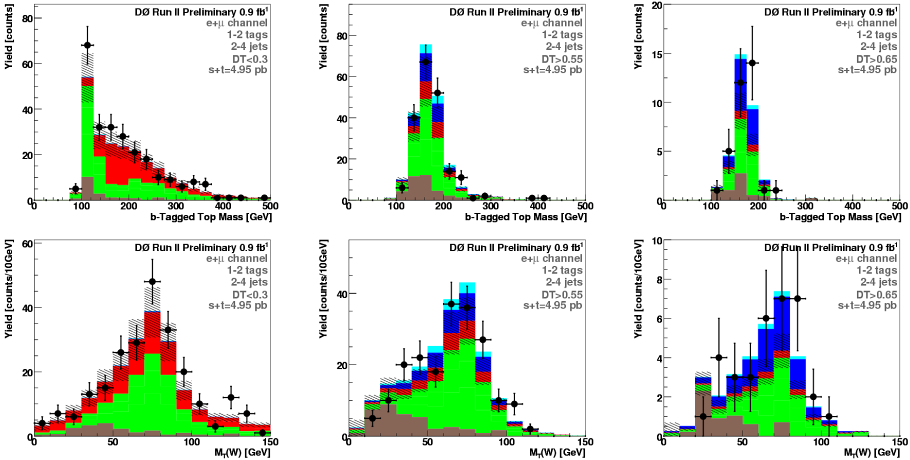
- **"ttbar":** = 4 jets, $H_T(\text{lepton}, E_T^{\text{miss}}, \text{alljets}) > 300$ GeV



- Good agreement

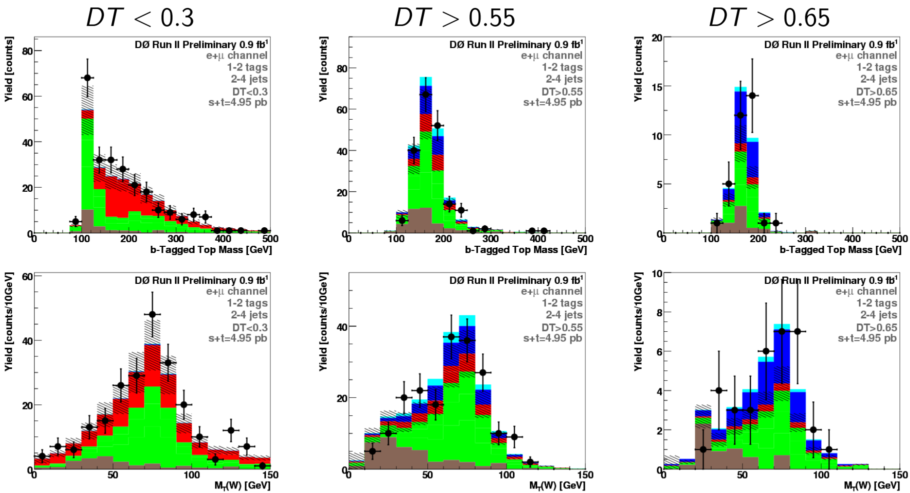
Boosted decision tree event characteristics

$DT < 0.3$ $DT > 0.55$ $DT > 0.65$



- High BDT region = shows masses of real t and W ⇒ expected
- Low BDT region = background-like ⇒ expected

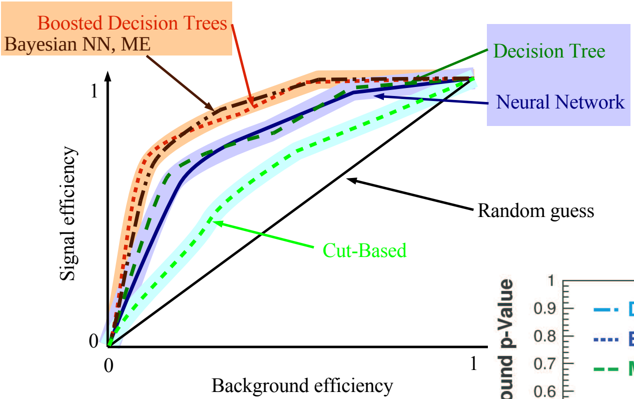
Boosted decision tree event characteristics



$DT < 0.3$ $DT > 0.55$ $DT > 0.65$

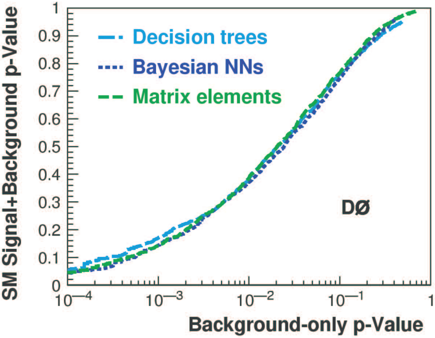
- High BDT region = shows masses of real t and W ⇒ expected
- Low BDT region = background-like ⇒ expected
- Above does NOT tell analysis is ok, but not seeing this could be a sign of a problem

Comparison for D0 single top evidence



- Cannot know *a priori* which method will work best
- ⇒ Need to experiment with different techniques

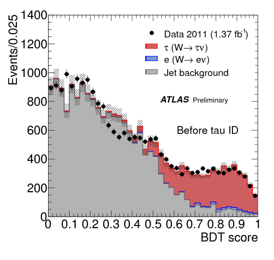
Power curve

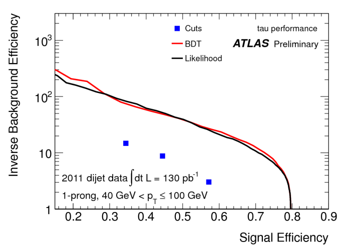


BDT in HEP

ATLAS tau identification

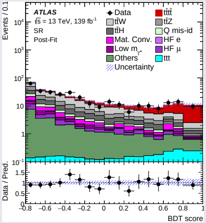
- Now used both offline and online
- Systematics: propagate various detector/theory effects to BDT output and measure variation





ATLAS $t\bar{t}t\bar{t}$ production evidence

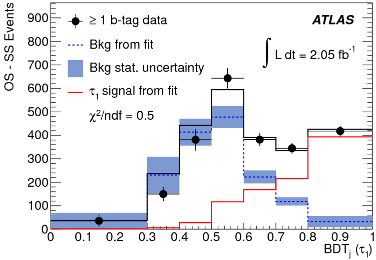
- ‣ Eur. Phys. J. C **80** (2020) 1085 ‣ arXiv:2007.14858 [hep-ex]
- BDT output used in final fit to measure cross section
- Constraints on systematic uncertainties from profiling

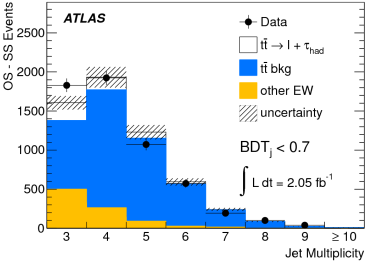


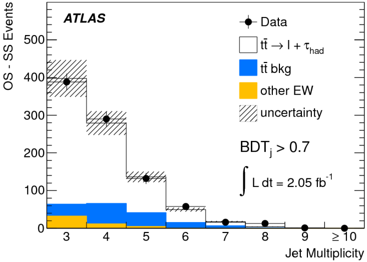
BDT in HEP: ATLAS $t\bar{t} \to e/\mu + \tau$+jets

▸ Phys.Lett. B717 (2012) 89-108

- BDT for tau ID: one to reject electrons, one against jets
- Fit BDT output to get tau contribution in data





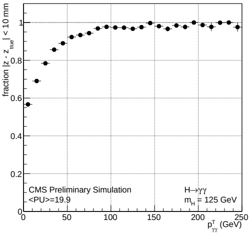


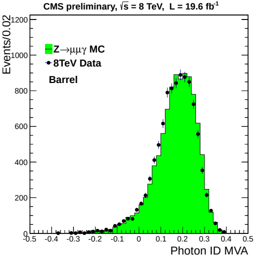
BDT in HEP: CMS $H \to \gamma\gamma$ result

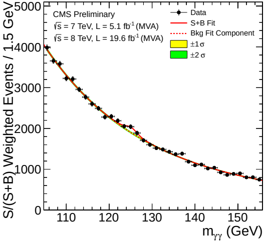
‣ CMS-PAS-HIG-13-001

Hard to use more BDT in an analysis:

- vertex selected with BDT
- 2nd vertex BDT to estimate probability to be within 1cm of interaction point
- photon ID with BDT
- photon energy corrected with BDT regression
- event-by-event energy uncertainty from another BDT
- several BDT to extract signal in different categories



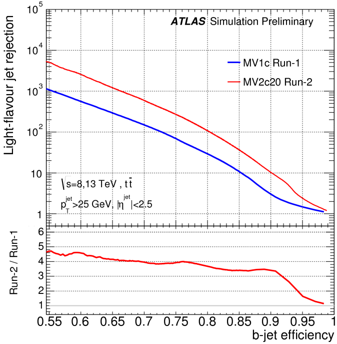


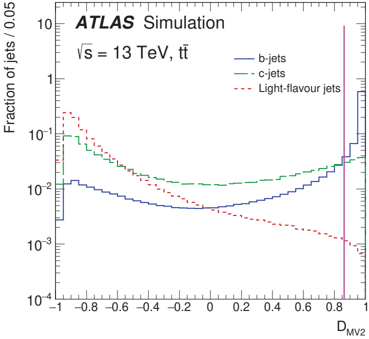


BDT in HEP: ATLAS *b*-tagging in Run 2

‣ ATL-PHYS-PUB-2015-022 ‣ Eur. Phys. J. C **79** (2019) 970 ‣ arXiv:1907.05120 [hep-ex]

- Run 1 MV1c: NN trained from output of other taggers
- Run 2 MV2c20: BDT using feature variables of underlying algorithms (impact parameter, secondary vertices) and p_T, η of jets
- Run 2: introduced IBL (new innermost pixel layer)
  $\Rightarrow$ explains part of the performance gain, but not all



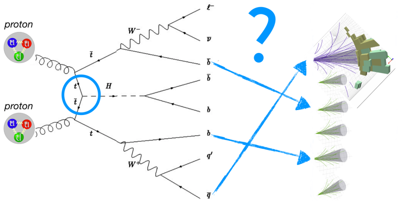


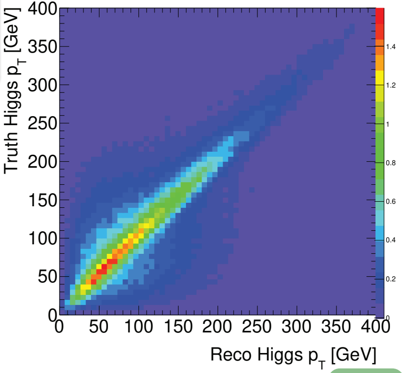
BDT in HEP: final state reconstruction

$t\bar{t}H(b\bar{b})$ reconstruction

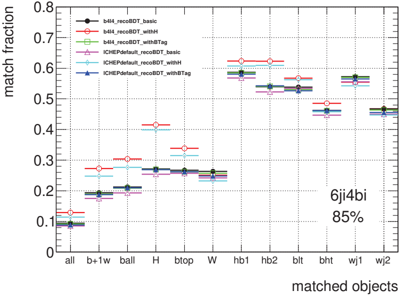
- Match jets and partons in high-multiplicity final state
- BDT trained on all combinations
- New inputs to classification BDT
- Access to Higgs p_T, origin of b-jets

▸ Phys. Rev. D 97, 072016 (2018)





▸ thesis

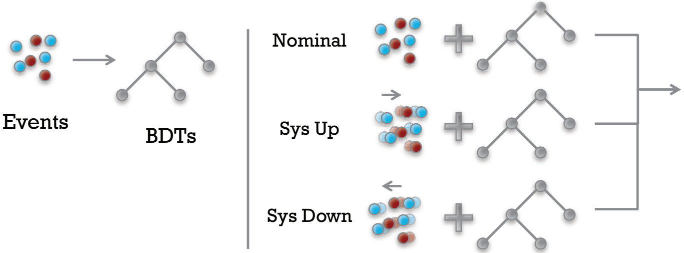


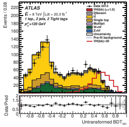
▸ thesis

BDT and systematics

- No particular rule
- BDT output can be considered as any other cut variable (just more powerful). Evaluate systematics by:
  - varying cut value
  - retraining
  - calibrating, etc.
- Most common (and appropriate, I think): propagate other uncertainties (detector, theory, etc.) up to BDT ouput and check how much the analysis is affected
- More and more common: profiling.
  Watch out:
  - BDT output powerful
  - signal region (high BDT output) probably low statistics
    ⇒ potential recipe for disaster if modelling is not good
- May require extra systematics, not so much on technique itself, but because it probes specific corners of phase space and/or wider parameter space (usually loosening pre-BDT selection cuts)

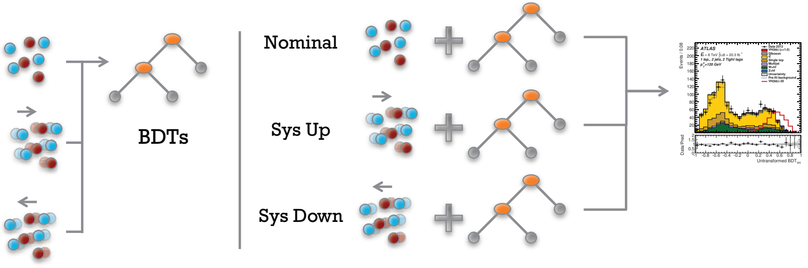
BDT and systematics





S. Hageböck

BDT and systematics



BDTs

Nominal

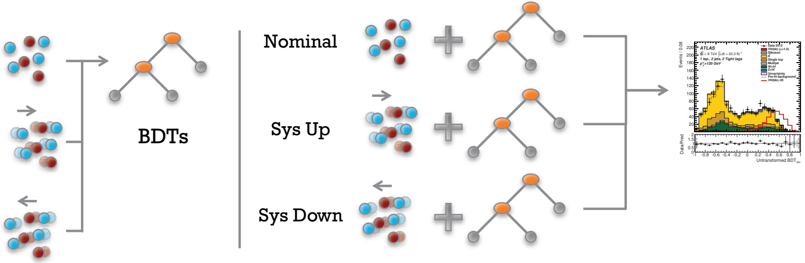
Sys Up

Sys Down

S. Hageböck

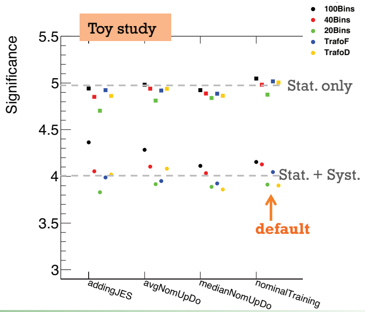
- Hope: seeing systematics-affected events during training may make the BDT less sensitive to systematic effects (data augmentation)

BDT and systematics



- Hope: seeing systematics-affected events during training may make the BDT less sensitive to systematic effects (data augmentation)

S. Hageböck



(Boosted decision tree) software

- Go for a fully integrated solution
  - use different multivariate techniques easily
  - spend your time on understanding your data and model
- Examples:
  - *T***MVA** (Toolkit for MultiVariate Analysis)
    Integrated in ROOT, complete manual ▸ https://root.cern/tmva
    - Example code in ▸ backup
  - scikit-learn (python) ▸ http://scikit-learn.org
- Dedicated to BDT:
  - XGBoost (popular in HEP) ▸ arXiv:1603.02754 ▸ https://github.com/dmlc/xgboost
    (note: cannot handle negative weights)
  - LightGBM (Microsoft) ▸ https://lightgbm.readthedocs.io
  - CatBoost (Yandex) ▸ https://catboost.ai/

Decision trees are not dead! e.g. NeurIPS2019

- PIDForest: Anomaly Detection via Partial Identification ▸ NeurIPS
- A Debiased MDI (Mean Decrease of Impurity) Feature Importance Measure for Random Forests ▸ NeurIPS
- MonoForest framework for tree ensemble analysis ▸ NeurIPS
- Faster Boosting with Smaller Memory (Yoav S Freund) ▸ NeurIPS
- Minimal Variance Sampling in Stochastic Gradient Boosting ▸ NeurIPS
- Regularized Gradient Boosting ▸ NeurIPS
- Partitioning Structure Learning for Segmented Linear Regression Trees ▸ NeurIPS
- Random Tessellation Forests ▸ NeurIPS
- Optimal Sparse Decision Trees ▸ NeurIPS
- Provably robust boosted decision stumps and trees against adversarial attacks ▸ NeurIPS
- Robustness Verification of Tree-based Models ▸ NeurIPS

References I: boosted decision trees

- L. Breiman, J.H. Friedman, R.A. Olshen and C.J. Stone, *Classification and Regression Trees*, Wadsworth, Stamford, 1984
- R.E. Schapire, “The strength of weak learnability” ▸ Machine Learning **5** (1990) 197
- Y. Freund, “Boosting a weak learning algorithm by majority” ▸ Information and computation **121** (1995) 256
- Y. Freund and R.E. Schapire, “Experiments with a New Boosting Algorithm” in *Machine Learning: Proceedings of the Thirteenth International Conference*, edited by L. Saitta (Morgan Kaufmann, San Fransisco, 1996) p. 148
- Y. Freund and R.E. Schapire, “A short introduction to boosting” ▸ Journal of Japanese Society for Artificial Intelligence 14 (1999) 771
- R. E. Schapire and Y. Freund, “Boosting: Foundations and Algorithms”, MIT Press, 2012.
- Y. Freund and R.E. Schapire, “A decision-theoretic generalization of on-line learning and an application to boosting” ▸ Journal of Computer and System Sciences **55** (1997) 119
- J.H. Friedman, T. Hastie and R. Tibshirani, “Additive logistic regression: a statistical view of boosting” ▸ Annals of Statistics **28** (2000) 377

References II: boosted decision trees

- J. H. Friedman, “Greedy function approximation: A gradient boosting machine” ▸ Annals of Statistics **29** (2001) 1189
- T. Hastie, R. Tibshirani, and J. Friedman, “The Elements of Statistical Learning: Data Mining, Inference, and Prediction (2nd edition)” ▸ Springer Series in Statistics, 2009
- S. Shalev-Shwartz and S. Ben-David, “Understanding Machine Learning: From Theory to Algorithms” ▸ Cambridge University Press, 2014
- M. Belkin, D. Hsu, S. Ma, and S. Mandal, “Reconciling modern machine-learning practice and the classical bias–variance trade-off” ▸ PNAS **116** (2019) 15849, ▸ arXiv:1812.11118 [stat.ML]
- L. Breiman, “Bagging Predictors” ▸ Machine Learning **24** (1996) 123
- L. Breiman, “Random forests” ▸ Machine Learning **45** (2001) 5
- B. P. Roe, H.-J. Yang, J. Zhu, Y. Liu, I. Stancu, and G. McGregor ▸ Nucl. Instr. Meth. A **555** (2005) 370; H.-J. Yang, B.P. Roe, and J. Zhu ▸ Nucl. Instr. Meth. A **555** (2005) 370
- V. M. Abazov *et al.* [D0 Collaboration], “Evidence for production of single top quarks” ▸ Phys. Rev. D **78** (2008) 012005

Beyond the standard slides

Backup

Pruning a tree I

Pre-pruning

- Stop tree growth during building phase
- Already seen: minimum leaf size, minimum separation improvement, maximum depth, etc.
- Careful: early stopping condition may prevent from discovering further useful splitting

Expected error pruning

- Grow full tree
- When result from children not significantly different from result of parent, prune children
- Can measure statistical error estimate with binomial error $\sqrt{p(1-p)/N}$ for node with purity p and N training events
- No need for testing sample
- Known to be “too aggressive”

Pruning a tree II: cost-complexity pruning

- Idea: penalise "complex" trees (many nodes/leaves) and find compromise between good fit to training data (larger tree) and good generalisation properties (smaller tree)
- With misclassification rate $R(T)$ of subtree T (with N_T nodes) of fully grown tree T_{max}:

$$\text{cost complexity } R_\alpha(T) = R(T) + \alpha N_T$$

  $\alpha =$ complexity parameter
- Minimise $R_\alpha(T)$:
  - small α: pick T_{max}
  - large α: keep root node only, T_{max} fully pruned
- First-pass pruning, for terminal nodes t_L, t_R from split of t:
  - by construction $R(t) \geq R(t_L) + R(t_R)$
  - if $R(t) = R(t_L) + R(t_R)$ prune off t_L and t_R

Pruning a tree III: cost-complexity pruning

- For node t and subtree T_t:
  - if t non-terminal, $R(t) > R(T_t)$ by construction
  - $R_\alpha(\{t\}) = R_\alpha(t) = R(t) + \alpha\ (N_T = 1)$
  - if $R_\alpha(T_t) < R_\alpha(t)$ then branch has smaller cost-complexity than single node and should be kept
  - at critical $\alpha = \rho_t$, node is preferable
  - to find ρ_t, solve $R_{\rho_t}(T_t) = R_{\rho_t}(t)$, or: $\rho_t = \dfrac{R(t) - R(T_t)}{N_T - 1}$
  - node with smallest ρ_t is *weakest link* and gets pruned
  - apply recursively till you get to the root node
- This generates sequence of decreasing cost-complexity subtrees
- Compute their true misclassification rate on validation sample:
  - will first decrease with cost-complexity
  - then goes through a minimum and increases again
  - pick this tree at the minimum as the best pruned tree
- Note: best pruned tree may not be optimal in a forest

Introduction to TMVA (ROOT v6.20.06)

- TMVA: Toolkit for MultiVariate Analysis

  ▸ https://root.cern/tmva ▸ https://github.com/root-project/root/tree/master/tmva
- Written by physicists
- In C++ (also python API), integrated in ROOT
- Quite complete manual
- Includes many different multivariate/machine learning techniques
- To compile, add appropriate header files in your code (e.g., `#include "TMVA/Factory.h"`) and this to your compiler command line: `` `root-config --cflags --libs` -lTMVA ``
- More complete examples of code: `$ROOTSYS/tutorials/tmva`
  - `createData.C` macro to make example datasets
  - classification and regression macros
  - also includes Keras examples (deep learning)
- Sometimes useful performance measures (more in these headers):

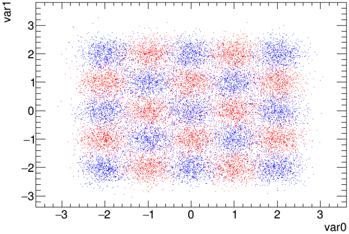
```
#include "TMVA/ROCCalc.h"
TMVA::ROCCalc(TH1* S,TH1* B).GetROCIntegral();
#include "TMVA/Tools.h"
TMVA::gTools().GetSeparation(TH1* S,TH1* B);
```

Training with TMVA (Train.C)

```
TFile* outputFile = TFile::Open("output.root","RECREATE");
TMVA::Factory *factory = new TMVA::Factory( "TMVAClassification", outputFile,
    "!V:Color:DrawProgressBar:Transformations=I:AnalysisType=Classification");
```

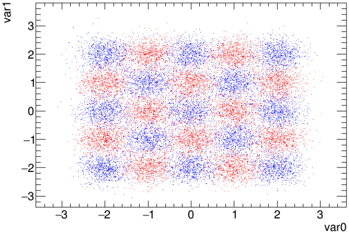
Training with TMVA (Train.C)

```
TFile* outputFile = TFile::Open("output.root","RECREATE");
TMVA::Factory *factory = new TMVA::Factory( "TMVAClassification", outputFile,
    "!V:Color:DrawProgressBar:Transformations=I:AnalysisType=Classification");
TFile* inputFile = new TFile("dataSchachbrett.root")
TTree* sig = (TTree*)inputFile->Get("TreeS");
TTree* bkg = (TTree*)inputFile->Get("TreeB");
double sigWeight = 1.0; double bkgWeight = 1.0;
TMVA::DataLoader *dataloader =
    new TMVA::DataLoader("dataset");
dataloader->AddSignalTree(sig, sigWeight);
dataloader->AddBackgroundTree(bkg, bkgWeight);
```



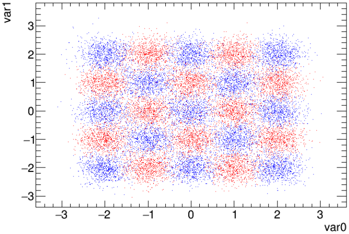
Training with TMVA (Train.C)

```
TFile* outputFile = TFile::Open("output.root","RECREATE");
TMVA::Factory *factory = new TMVA::Factory( "TMVAClassification", outputFile,
    "!V:Color:DrawProgressBar:Transformations=I:AnalysisType=Classification");
TFile* inputFile = new TFile("dataSchachbrett.root")
TTree* sig = (TTree*)inputFile->Get("TreeS");
TTree* bkg = (TTree*)inputFile->Get("TreeB");
double sigWeight = 1.0; double bkgWeight = 1.0;
TMVA::DataLoader *dataloader =
    new TMVA::DataLoader("dataset");
dataloader->AddSignalTree(sig, sigWeight);
dataloader->AddBackgroundTree(bkg, bkgWeight);
dataloader->AddVariable("var0", 'F');
dataloader->AddVariable("var1", 'F');
TCut mycut = "";
```



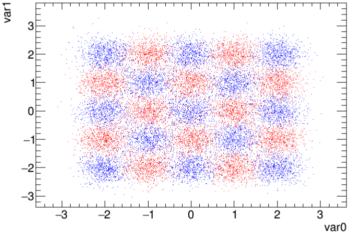
Training with TMVA (Train.C)

```
TFile* outputFile = TFile::Open("output.root","RECREATE");
TMVA::Factory *factory = new TMVA::Factory( "TMVAClassification", outputFile,
    "!V:Color:DrawProgressBar:Transformations=I:AnalysisType=Classification");
TFile* inputFile = new TFile("dataSchachbrett.root");
TTree* sig = (TTree*)inputFile->Get("TreeS");
TTree* bkg = (TTree*)inputFile->Get("TreeB");
double sigWeight = 1.0; double bkgWeight = 1.0;
TMVA::DataLoader *dataloader =
    new TMVA::DataLoader("dataset");
dataloader->AddSignalTree(sig, sigWeight);
dataloader->AddBackgroundTree(bkg, bkgWeight);
dataloader->AddVariable("var0", 'F');
dataloader->AddVariable("var1", 'F');
TCut mycut = "";
dataloader->PrepareTrainingAndTestTree(mycut,"SplitMode=Random");
```



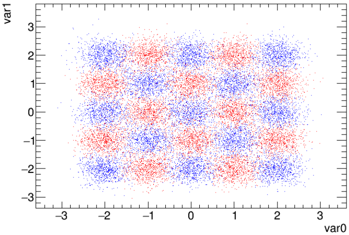
Training with TMVA (Train.C)

```
TFile* outputFile = TFile::Open("output.root","RECREATE");
TMVA::Factory *factory = new TMVA::Factory( "TMVAClassification", outputFile,
    "!V:Color:DrawProgressBar:Transformations=I:AnalysisType=Classification");
TFile* inputFile = new TFile("dataSchachbrett.root")
TTree* sig = (TTree*)inputFile->Get("TreeS");
TTree* bkg = (TTree*)inputFile->Get("TreeB");
double sigWeight = 1.0; double bkgWeight = 1.0;
TMVA::DataLoader *dataloader =
    new TMVA::DataLoader("dataset");
dataloader->AddSignalTree(sig, sigWeight);
dataloader->AddBackgroundTree(bkg, bkgWeight);
dataloader->AddVariable("var0", 'F');
dataloader->AddVariable("var1", 'F');
TCut mycut = "";
dataloader->PrepareTrainingAndTestTree(mycut,"SplitMode=Random");
factory->BookMethod(dataloader, TMVA::Types::kBDT, "BDT", "!H:!V:NTrees=400:
    MinNodeSize=4%:MaxDepth=5:BoostType=AdaBoost:AdaBoostBeta=0.15:nCuts=80");
factory->BookMethod(dataloader, TMVA::Types::kFisher, "Fisher", "!H:!V:Fisher");
```



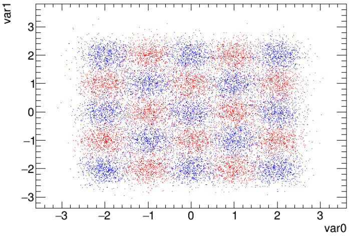
Training with TMVA (Train.C)

```
TFile* outputFile = TFile::Open("output.root","RECREATE");
TMVA::Factory *factory = new TMVA::Factory( "TMVAClassification", outputFile,
    "!V:Color:DrawProgressBar:Transformations=I:AnalysisType=Classification");
TFile* inputFile = new TFile("dataSchachbrett.root")
TTree* sig = (TTree*)inputFile->Get("TreeS");
TTree* bkg = (TTree*)inputFile->Get("TreeB");
double sigWeight = 1.0; double bkgWeight = 1.0;
TMVA::DataLoader *dataloader =
    new TMVA::DataLoader("dataset");
dataloader->AddSignalTree(sig, sigWeight);
dataloader->AddBackgroundTree(bkg, bkgWeight);
dataloader->AddVariable("var0", ’F’);
dataloader->AddVariable("var1", ’F’);
TCut mycut = "";
dataloader->PrepareTrainingAndTestTree(mycut,"SplitMode=Random");
factory->BookMethod(dataloader, TMVA::Types::kBDT, "BDT", "!H:!V:NTrees=400:
    MinNodeSize=4%:MaxDepth=5:BoostType=AdaBoost:AdaBoostBeta=0.15:nCuts=80");
factory->BookMethod(dataloader, TMVA::Types::kFisher, "Fisher", "!H:!V:Fisher");
factory->TrainAllMethods(); // Train MVAs using training events
```



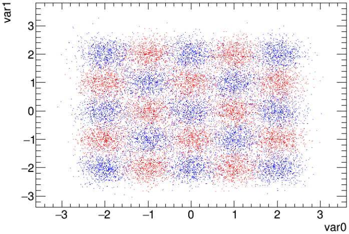
Training with TMVA (Train.C)

```
TFile* outputFile = TFile::Open("output.root","RECREATE");
TMVA::Factory *factory = new TMVA::Factory( "TMVAClassification", outputFile,
    "!V:Color:DrawProgressBar:Transformations=I:AnalysisType=Classification");
TFile* inputFile = new TFile("dataSchachbrett.root")
TTree* sig = (TTree*)inputFile->Get("TreeS");
TTree* bkg = (TTree*)inputFile->Get("TreeB");
double sigWeight = 1.0; double bkgWeight = 1.0;
TMVA::DataLoader *dataloader =
    new TMVA::DataLoader("dataset");
dataloader->AddSignalTree(sig, sigWeight);
dataloader->AddBackgroundTree(bkg, bkgWeight);
dataloader->AddVariable("var0", 'F');
dataloader->AddVariable("var1", 'F');
TCut mycut = "";
dataloader->PrepareTrainingAndTestTree(mycut,"SplitMode=Random");
factory->BookMethod(dataloader, TMVA::Types::kBDT, "BDT", "!H:!V:NTrees=400:
    MinNodeSize=4%:MaxDepth=5:BoostType=AdaBoost:AdaBoostBeta=0.15:nCuts=80");
factory->BookMethod(dataloader, TMVA::Types::kFisher, "Fisher", "!H:!V:Fisher");
factory->TrainAllMethods(); // Train MVAs using training events
factory->TestAllMethods(); // Evaluate all MVAs using test events
```



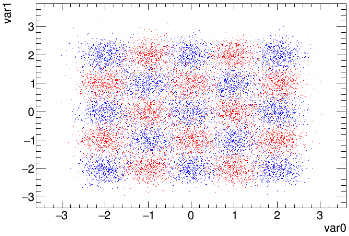
Training with TMVA (Train.C)

```
TFile* outputFile = TFile::Open("output.root","RECREATE");
TMVA::Factory *factory = new TMVA::Factory( "TMVAClassification", outputFile,
    "!V:Color:DrawProgressBar:Transformations=I:AnalysisType=Classification");
TFile* inputFile = new TFile("dataSchachbrett.root");
TTree* sig = (TTree*)inputFile->Get("TreeS");
TTree* bkg = (TTree*)inputFile->Get("TreeB");
double sigWeight = 1.0; double bkgWeight = 1.0;
TMVA::DataLoader *dataloader =
    new TMVA::DataLoader("dataset");
dataloader->AddSignalTree(sig, sigWeight);
dataloader->AddBackgroundTree(bkg, bkgWeight);
dataloader->AddVariable("var0", 'F');
dataloader->AddVariable("var1", 'F');
TCut mycut = "";
dataloader->PrepareTrainingAndTestTree(mycut,"SplitMode=Random");
factory->BookMethod(dataloader, TMVA::Types::kBDT, "BDT", "!H:!V:NTrees=400:
    MinNodeSize=4%:MaxDepth=5:BoostType=AdaBoost:AdaBoostBeta=0.15:nCuts=80");
factory->BookMethod(dataloader, TMVA::Types::kFisher, "Fisher", "!H:!V:Fisher");
factory->TrainAllMethods(); // Train MVAs using training events
factory->TestAllMethods(); // Evaluate all MVAs using test events
// ----- Evaluate and compare performance of all configured MVAs
factory->EvaluateAllMethods();
```



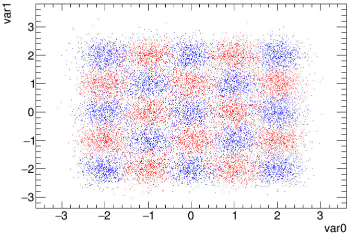
Training with TMVA (Train.C)

```
TFile* outputFile = TFile::Open("output.root","RECREATE");
TMVA::Factory *factory = new TMVA::Factory( "TMVAClassification", outputFile,
    "!V:Color:DrawProgressBar:Transformations=I:AnalysisType=Classification");
TFile* inputFile = new TFile("dataSchachbrett.root");
TTree* sig = (TTree*)inputFile->Get("TreeS");
TTree* bkg = (TTree*)inputFile->Get("TreeB");
double sigWeight = 1.0; double bkgWeight = 1.0;
TMVA::DataLoader *dataloader =
    new TMVA::DataLoader("dataset");
dataloader->AddSignalTree(sig, sigWeight);
dataloader->AddBackgroundTree(bkg, bkgWeight);
dataloader->AddVariable("var0", 'F');
dataloader->AddVariable("var1", 'F');
TCut mycut = "";
dataloader->PrepareTrainingAndTestTree(mycut,"SplitMode=Random");
factory->BookMethod(dataloader, TMVA::Types::kBDT, "BDT", "!H:!V:NTrees=400:
    MinNodeSize=4%:MaxDepth=5:BoostType=AdaBoost:AdaBoostBeta=0.15:nCuts=80");
factory->BookMethod(dataloader, TMVA::Types::kFisher, "Fisher", "!H:!V:Fisher");
factory->TrainAllMethods(); // Train MVAs using training events
factory->TestAllMethods(); // Evaluate all MVAs using test events
// ----- Evaluate and compare performance of all configured MVAs
factory->EvaluateAllMethods();
auto c1 = factory->GetROCCurve(dataloader); // Eager to compare performance
```



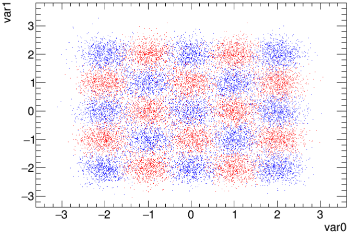
Training with TMVA (Train.C)

```
TFile* outputFile = TFile::Open("output.root","RECREATE");
TMVA::Factory *factory = new TMVA::Factory( "TMVAClassification", outputFile,
    "!V:Color:DrawProgressBar:Transformations=I:AnalysisType=Classification");
TFile* inputFile = new TFile("dataSchachbrett.root")
TTree* sig = (TTree*)inputFile->Get("TreeS");
TTree* bkg = (TTree*)inputFile->Get("TreeB");
double sigWeight = 1.0; double bkgWeight = 1.0;
TMVA::DataLoader *dataloader =
    new TMVA::DataLoader("dataset");
dataloader->AddSignalTree(sig, sigWeight);
dataloader->AddBackgroundTree(bkg, bkgWeight);
dataloader->AddVariable("var0", 'F');
dataloader->AddVariable("var1", 'F');
TCut mycut = "";
dataloader->PrepareTrainingAndTestTree(mycut,"SplitMode=Random");
factory->BookMethod(dataloader, TMVA::Types::kBDT, "BDT", "!H:!V:NTrees=400:
    MinNodeSize=4%:MaxDepth=5:BoostType=AdaBoost:AdaBoostBeta=0.15:nCuts=80");
factory->BookMethod(dataloader, TMVA::Types::kFisher, "Fisher", "!H:!V:Fisher");
factory->TrainAllMethods(); // Train MVAs using training events
factory->TestAllMethods(); // Evaluate all MVAs using test events
// ----- Evaluate and compare performance of all configured MVAs
factory->EvaluateAllMethods();
auto c1 = factory->GetROCCurve(dataloader); // Eager to compare performance
outputFile->Close();
delete factory; delete dataloader;
```



Training with TMVA (Train.C)

```
TFile* outputFile = TFile::Open("output.root","RECREATE");
TMVA::Factory *factory = new TMVA::Factory( "TMVAClassification", outputFile,
    "!V:Color:DrawProgressBar:Transformations=I:AnalysisType=Classification");
TFile* inputFile = new TFile("dataSchachbrett.root")
TTree* sig = (TTree*)inputFile->Get("TreeS");
TTree* bkg = (TTree*)inputFile->Get("TreeB");
double sigWeight = 1.0; double bkgWeight = 1.0;
TMVA::DataLoader *dataloader =
    new TMVA::DataLoader("dataset");
dataloader->AddSignalTree(sig, sigWeight);
dataloader->AddBackgroundTree(bkg, bkgWeight);
dataloader->AddVariable("var0", ’F’);
dataloader->AddVariable("var1", ’F’);
TCut mycut = "";
dataloader->PrepareTrainingAndTestTree(mycut,"SplitMode=Random");
factory->BookMethod(dataloader, TMVA::Types::kBDT, "BDT", "!H:!V:NTrees=400:
    MinNodeSize=4%:MaxDepth=5:BoostType=AdaBoost:AdaBoostBeta=0.15:nCuts=80");
factory->BookMethod(dataloader, TMVA::Types::kFisher, "Fisher", "!H:!V:Fisher");
factory->TrainAllMethods(); // Train MVAs using training events
factory->TestAllMethods(); // Evaluate all MVAs using test events
// ----- Evaluate and compare performance of all configured MVAs
factory->EvaluateAllMethods();
auto c1 = factory->GetROCCurve(dataloader); // Eager to compare performance
outputFile->Close();
delete factory; delete dataloader;          TMVA::TMVAGui("output.root");
```



Apply classifier with TMVA (Apply.C)

```
TFile* inputFile = new TFile("dataSchachbrett.root");
TTree* data = (TTree*)inputFile->Get("TreeS");
Float_t var0=-99., var1=-99.;
data->SetBranchAddress("var0", &var0);
data->SetBranchAddress("var1", &var1);
```

Apply classifier with TMVA (Apply.C)

```
TFile* inputFile = new TFile("dataSchachbrett.root");
TTree* data = (TTree*)inputFile->Get("TreeS");
Float_t var0=-99., var1=-99.;
data->SetBranchAddress("var0", &var0);
data->SetBranchAddress("var1", &var1);
TMVA::Reader *reader = new TMVA::Reader();
reader->AddVariable( "var0", &var0 );
reader->AddVariable( "var1", &var1 );
```

Apply classifier with TMVA (Apply.C)

```
TFile* inputFile = new TFile("dataSchachbrett.root");
TTree* data = (TTree*)inputFile->Get("TreeS");
Float_t var0=-99., var1=-99.;
data->SetBranchAddress("var0", &var0);
data->SetBranchAddress("var1", &var1);
TMVA::Reader *reader = new TMVA::Reader();
reader->AddVariable( "var0", &var0 );
reader->AddVariable( "var1", &var1 );
reader->BookMVA( "My BDT", "dataset/weights/TMVAClassification_BDT.weights.xml");
reader->BookMVA( "Fisher discriminant",
  "dataset/weights/TMVAClassification_Fisher.weights.xml");
```

Apply classifier with TMVA (Apply.C)

```
TFile* inputFile = new TFile("dataSchachbrett.root");
TTree* data = (TTree*)inputFile->Get("TreeS");
Float_t var0=-99., var1=-99.;
data->SetBranchAddress("var0", &var0);
data->SetBranchAddress("var1", &var1);
TMVA::Reader *reader = new TMVA::Reader();
reader->AddVariable( "var0", &var0 );
reader->AddVariable( "var1", &var1 );
reader->BookMVA( "My BDT", "dataset/weights/TMVAClassification_BDT.weights.xml");
reader->BookMVA( "Fisher discriminant",
  "dataset/weights/TMVAClassification_Fisher.weights.xml");
// ------- start your event loop
for (Long64_t ievt=0; ievt<10; ++ievt) {
  data->GetEntry(ievt);
  double bdt = reader->EvaluateMVA("My BDT");
  double fisher = reader->EvaluateMVA("Fisher discriminant");
  cout<<"var0="<<var0<<" var1="<<var1<<" BDT="<<bdt<<" Fisher="<<fisher<<endl;
}
delete reader;
inputFile->Close();
```

Apply classifier with TMVA (Apply.C)

```
TFile* inputFile = new TFile("dataSchachbrett.root");
TTree* data = (TTree*)inputFile->Get("TreeS");
Float_t var0=-99., var1=-99.;
data->SetBranchAddress("var0", &var0);
data->SetBranchAddress("var1", &var1);
TMVA::Reader *reader = new TMVA::Reader();
reader->AddVariable( "var0", &var0 );
reader->AddVariable( "var1", &var1 );
reader->BookMVA( "My BDT", "dataset/weights/TMVAClassification_BDT.weights.xml");
reader->BookMVA( "Fisher discriminant",
  "dataset/weights/TMVAClassification_Fisher.weights.xml");
// ------- start your event loop
for (Long64_t ievt=0; ievt<10; ++ievt) {
  data->GetEntry(ievt);
  double bdt = reader->EvaluateMVA("My BDT");
  double fisher = reader->EvaluateMVA("Fisher discriminant");
  cout<<"var0="<<var0<<" var1="<<var1<<" BDT="<<bdt<<" Fisher="<<fisher<<endl;
}
delete reader;
inputFile->Close();
```

- More complete tutorials:
  ▸ https://github.com/lmoneta/tmva-tutorial

Compiling TMVA with C++

- To make code compilable (and MUCH faster)
  - Need ROOT and TMVA corresponding header files
  - e.g., for Train.C:

```
#include "TFile.h"
#include "TTree.h"
#include "TMVA/Factory.h"
#include "TMVA/DataLoader.h"
#include "TMVA/TMVAGui.h"
```

  - Need a “main” function

```
int main() {
  Train();
  return 0;
}
```

  - Compilation:

```
g++  Train.C ‘root-config --cflags --libs‘ -lTMVA  -lTMVAGui  -o  TMVATrainer
```

  - Train.C: file to compile
  - TMVATrainer: name of executable
  - -lTMVAGui: just because of `TMVA::TMVAGui("output.root");`

TMVA: training refinements

- Common technique: train on even event numbers, test on odd event numbers (and vice versa)
- Can also think of more than two-fold
- Achieve in TMVA by replacing:

```
dataloader->AddSignalTree(sig, sigWeight);
dataloader->AddBackgroundTree(bkg, bkgWeight);
```

- with:

```
TString trainString = "(eventNumber % 2 == 0)";
TString testString = "!"+trainString;
dataloader->AddTree(sig, "Signal", sigWeight, trainString.Data(), "Training");
dataloader->AddTree(sig, "Signal", sigWeight, testString.Data(), "Test");
dataloader->AddTree(bkg, "Background", bkgWeight, trainString.Data(), "Training");
dataloader->AddTree(bkg, "Background", bkgWeight, testString.Data(), "Test");
```

- Use individual event weights:

```
string eventWeight = "TMath::Abs(eventWeight)"; //Compute event weight
dataloader->SetSignalWeightExpression(eventWeight);
dataloader->SetBackgroundWeightExpression(eventWeight); //Can differ
```