Numerical Methods & Data Analysis in HEP

Mike Williams

Department of Physics

Massachusetts Institute of Technology

IDPASC Lectures
January 29 – 31, 2013

Lectures Overview

The main topics covered in these lectures are as follows:

- machine learning (primarily for classification);
- bootstrapping and other resampling methods;
- two-sample testing;
- regression (parametric and non-parametric);
- multivariate goodness-of-fit;
- limit setting.

There are 3 1-hour lectures and a practical session at the end. Clearly not enough time to cover all the details on any topic (let alone all of them), so I'll try and provide references so you can learn more on your own.

Lectures Goal

Statistical methods are *tools* physicists use. Cars are also tools. People use them to get from A to B; they don't care how they work.

Many people think car (or stats) enthusiasts are strange. I'm not going to advocate that you become a gear or stats fan but ...

Lectures Goal

You don't need to be a gearhead to use a car, but it's a good idea to learn how to drive properly! The consequences can be catastrophic.

The same applies to the use of statical methods. You don't need to know all the theory behind them, but it's a good idea to know roughly how they work and how to check that they are working properly.

Machine Learning

Machine learning involves learning to accurately make predictions by studying data provided for training. Here, the physicist is the teacher and provides the data to learn on; the machine is the student.

In these lectures, we will only discuss *supervised learning* (the teacher gives the student the answers during training). The goal here will be to learn how to classify events (*e.g.*, as signal or background) by studying the properties of data samples of each class.

A few notes:

- it's possible to deal with any number of data classes, but in HEP we tend to only use two (signal and background) so that's all I'll discuss;
- these methods can also be used for regression (no time for this here).

Types of Classifiers

The 2 most common classifiers used in HEP are boosted decision trees (BDTs) and artificial neural networks (ANN). There are plenty of others on the market (*e.g.*, support vector machines, k nearest neighbor, *etc.*); however, I only have time to cover BDTs and ANNs.

For most real-world problems, what matters is how you do the training, not which classifier you use. *I.e.*, it's the teacher not the student that matters. There are counter examples, of course, but generally the difference in performance between the major types of classifiers is small.

The choice of which classifier to use then will often be determined by *mundane* details like algorithm speed, availability/usability of software, interpretability *etc.*.

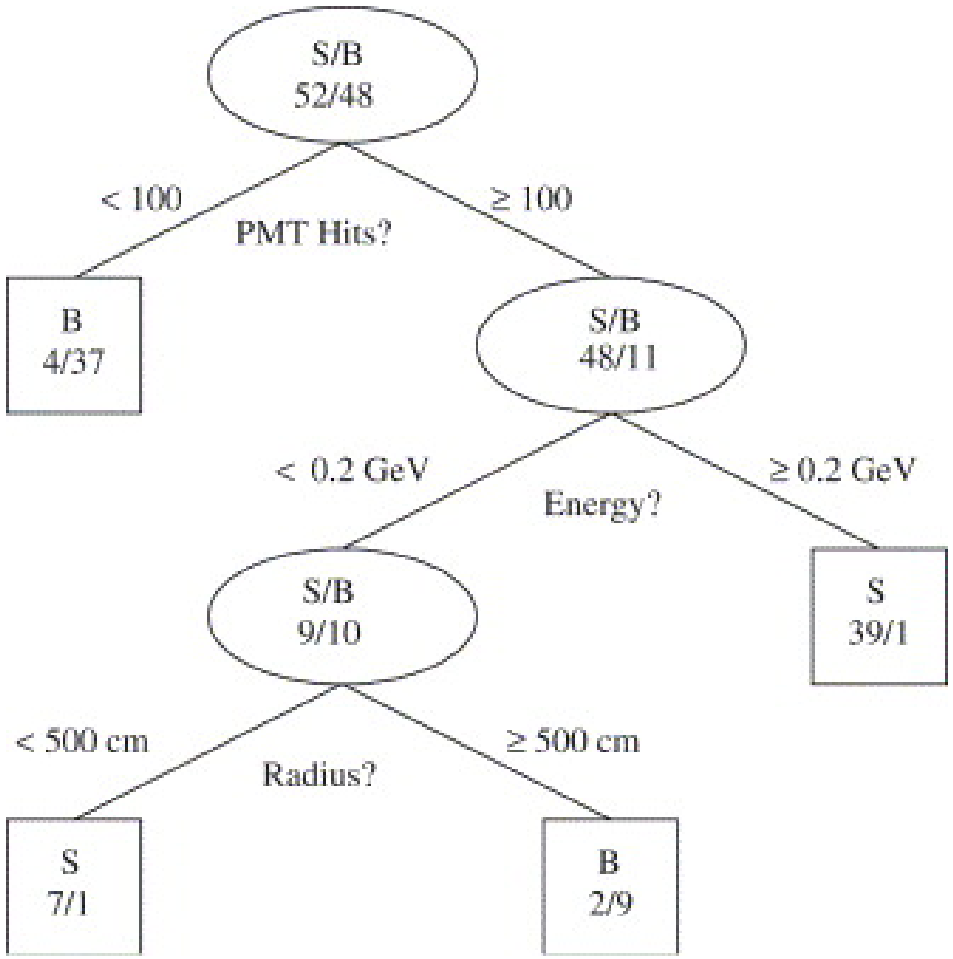
Decision Trees

Training works using provided sample with known classifications:

for each *node*, loop over variables and split if $\mathrm{FOM}_1 + \mathrm{FOM}_2 > \mathrm{FOM}$;

keep splitting until can't improve FOM, or maximum n_{leaf} or minimum events per leaf reached;

score can be discrete or continuous.



See L. Brieman *el al.*, *Classification and regression trees*, Wadsworth International Group, Belmont, California (1984).

Overtraining

Consider a node with 2 events with $x = \{2.1, 2.2\}, \sigma_x = 0.5$; all other variables are comparable (they've made it this far together in the DT). The first is S and the second B.

Splitting at 2.15 makes pure S,B leaves. On the training sample, this improves the performance.

Clearly, however, it will not help on data from an independent sample. In fact, it will most likely **hurt** performance!

Training & Validation

To avoid overtraining, use the following approach:

- Split the *known* data into training and validation samples. Train the tree on the former and evaluate its performance on the latter. If the performance is similar on the two samples (when clearly the first is biased), then the performance on any new data (ignoring systematics) should be predictable.
- If the performance is very different or if you've *over-validated*, it's advisable to use a third (test) data sample to estimate the performance. You can also train on the validation sample and validate on the training sample as another test (if more data isn't available).
- For the DT above, we can make it more robust by requiring a minimum number of events/leaf and maximum number of leaves.
- We can also **boost**!

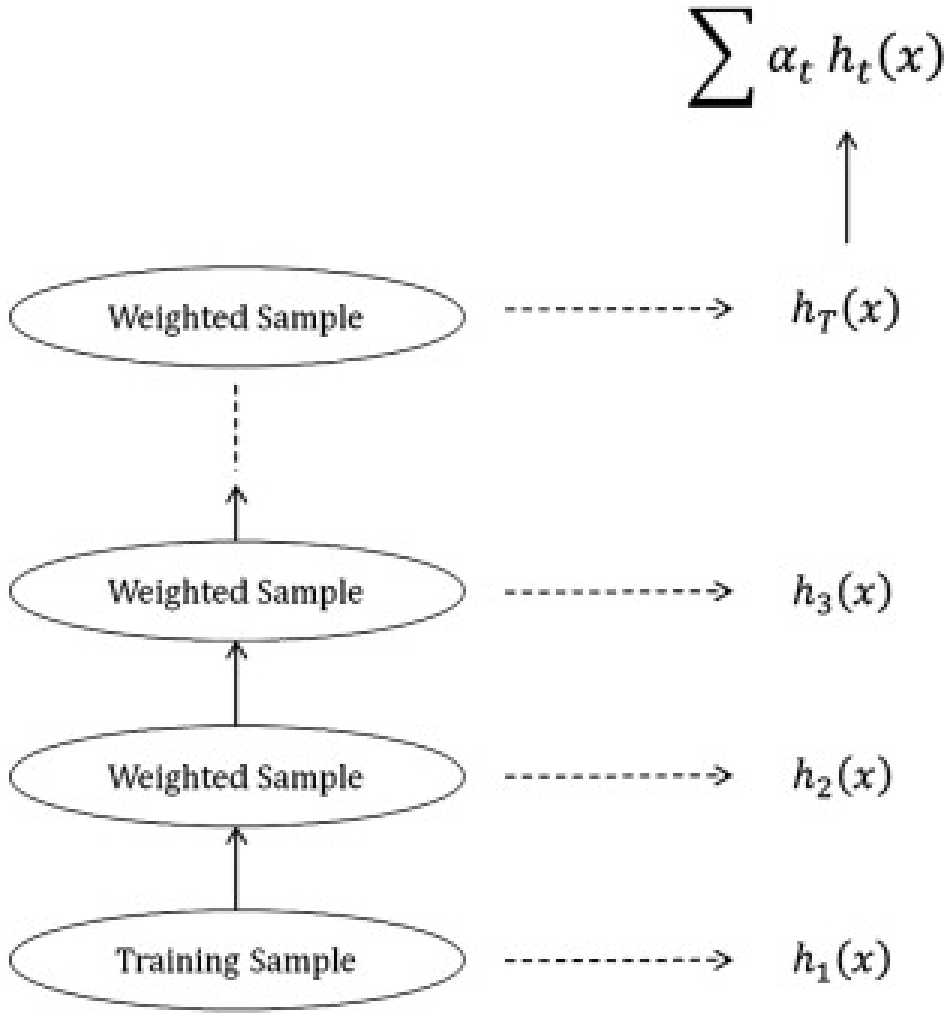
Boosting

Boosting is a family of methods that produce a series of classifiers.

The training sample for each member of the series is determined by the performance of earlier members. Incorrectly classified events are (in some way) given more weight.

The main idea is for each successive classifier to predict where it's predecessors fail and then improve on their performance.

Many examples: Adaboost, ϵ-boost, arcing, *etc.*

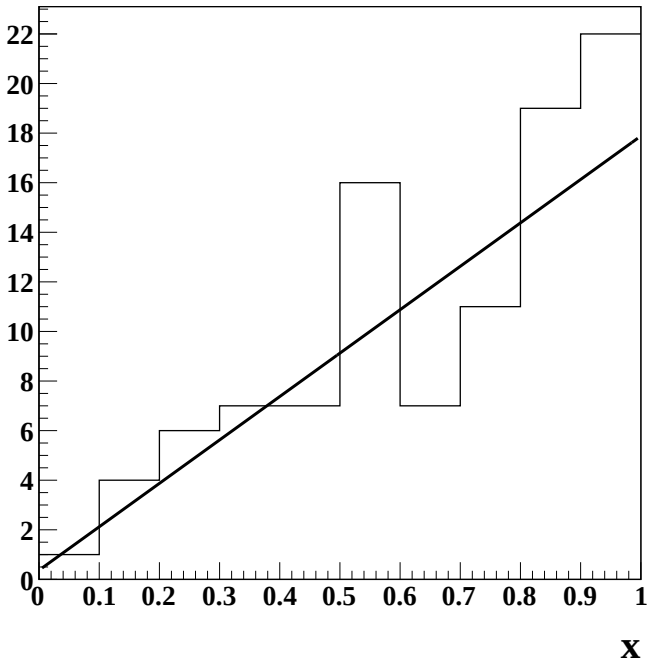


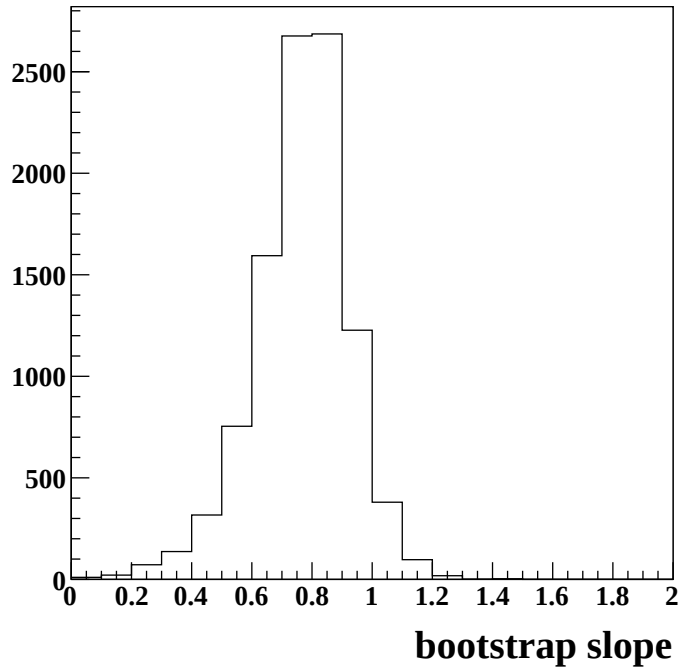
See B.P.Roe *et al*, NIM **A543**, 577 (2005) for a nice explanation of a few of these.

Bootstrapping

Bootstrapping determines estimators by sampling with replacement from the original data (simulates repeated observations using only the data).

Ex) $f(x) = x$, $n = 100$, χ^2 fit $\rightarrow \hat{a} = 0.88 \pm 0.13$, $\sigma^a_{\text{boot}} = 0.15$

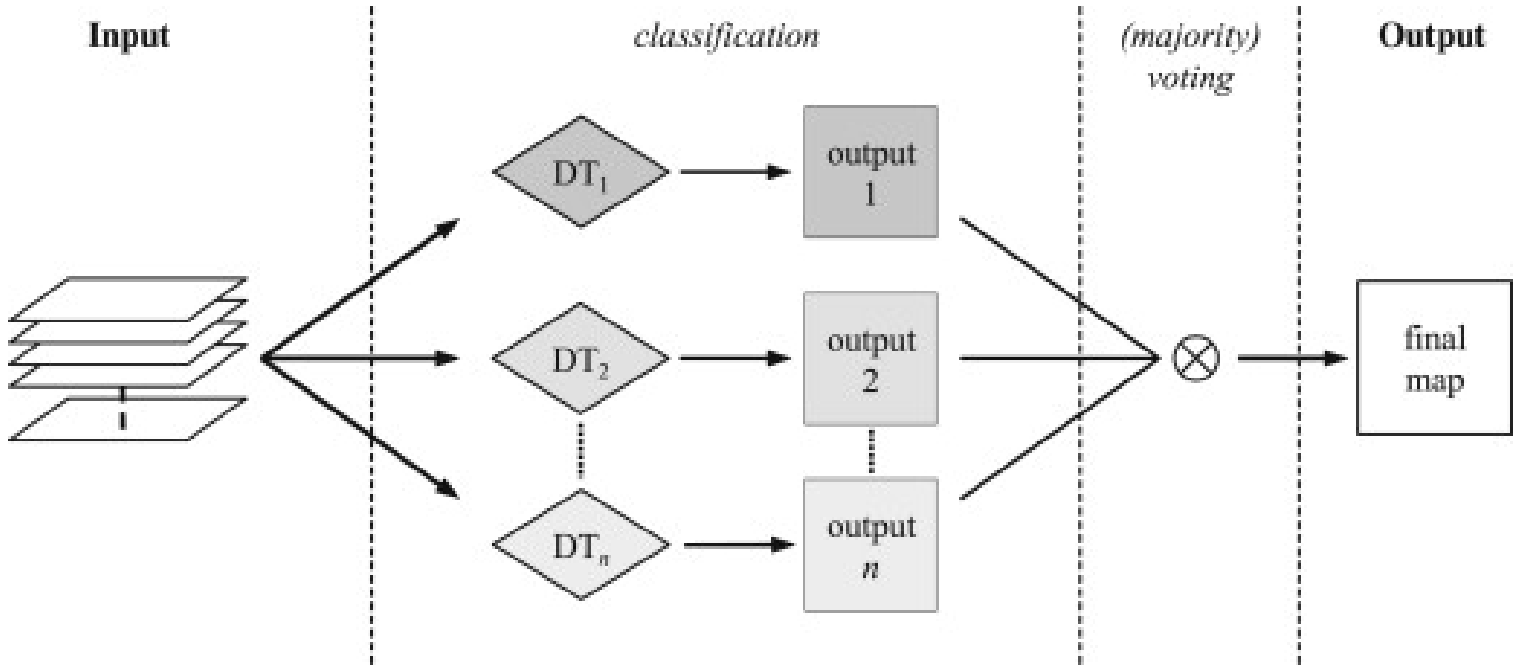




Notice that the bootstrap μ_a is roughly the sample $\hat{a}$. We don't gain information so we can't use the bootstrap to obtain a better $\hat{a}$.

Bootstrapping (Bagging) a DT

Make n_b bootstrap-copy data sets from the training sample; train an independent DT on each. Response is $n_{\text{sig leaf}}(\vec{x})/n_b$.



My experience with bagging is that it works very well but it is slow. If you don't care about timing though it's a very reliable boosting method.

See L. Breiman, *Bagging Predictors*, Machine Learning 26 (1996) 123-140; (1997) 553-568.

Random Forest

For each DT, for every split select a random sample of the variables to consider for splitting:

- normally used together with bagging;
- increases power (in theory) by making the DTs less correlated;
- faster in training than standard DT due to considering less variables at each split.

I have seen small gains from RF over bagging in some cases. An important thing to keep in mind is that RF requires you to allow it to make tiny leaves (few events), while BDTs do better with $\mathcal{O}(10)$ (or more) events/leaf.

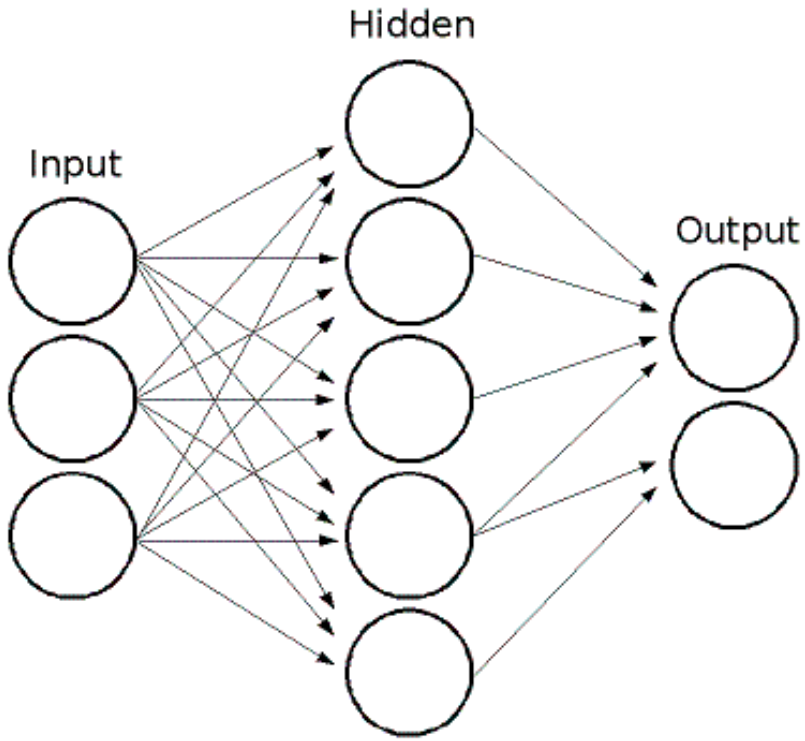
See L.Breiman, *Random Forests*, Machine Learning **45**, 5-32 (2001).

Training

Ph.D. students best excuse for slacking off: Training their ANN/BDT.

Artificial Neural Networks

ANNs send data from input neurons via synapses to a hidden layer of neurons, and then to output neurons via more synapses.



Learning works using forward propagation of weights; determination of classification error; backwards propagation of error to update weights.

More on ANNs

- Of course, the same tricks we used to boost BDTs can be used to boost ANNs to improve performance.
- Can struggle with *mixed-type* inputs and also *useless* inputs.
- Large training time in high dimensions but fast response time.
- Many successful uses in HEP.

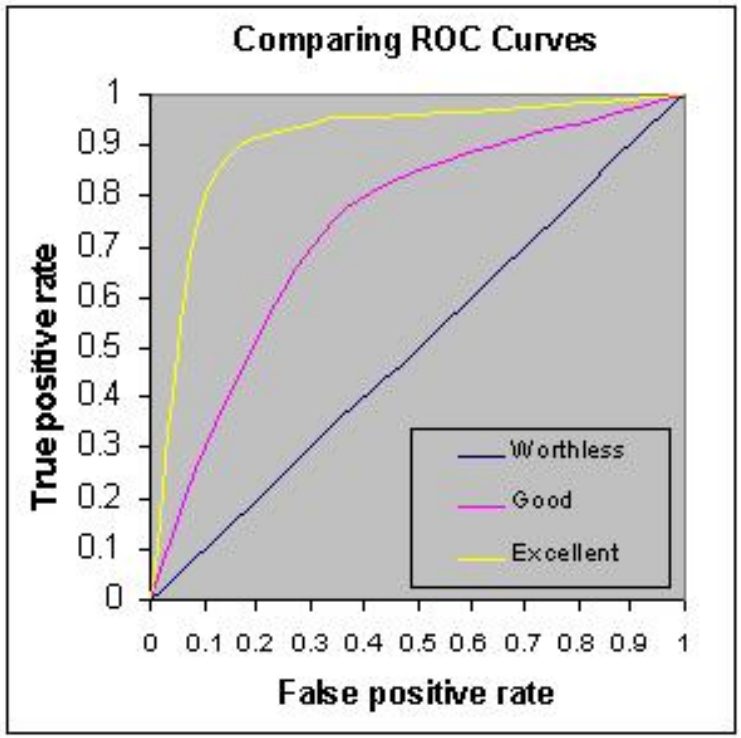
BDT vs ANN

	BDT	ANN
Classification Power	✓	✓
Power in High Dimensions	✓	✓
Response Time	$\sim$	✓
Irrelevant Inputs	✓	X*
Interpretability	$\sim$	X

*ANNs can be improved here by first pruning away the irrelevant inputs. So, on their own they struggle but if during training another algorithm is employed to remove these inputs then they're fine. Many packages do this automatically so you wouldn't even notice.

ROC

Performance is often visualized using a Receiver Operating Characteristic (ROC) Curve. ROCs were originally developed during WWII and used with radar systems.



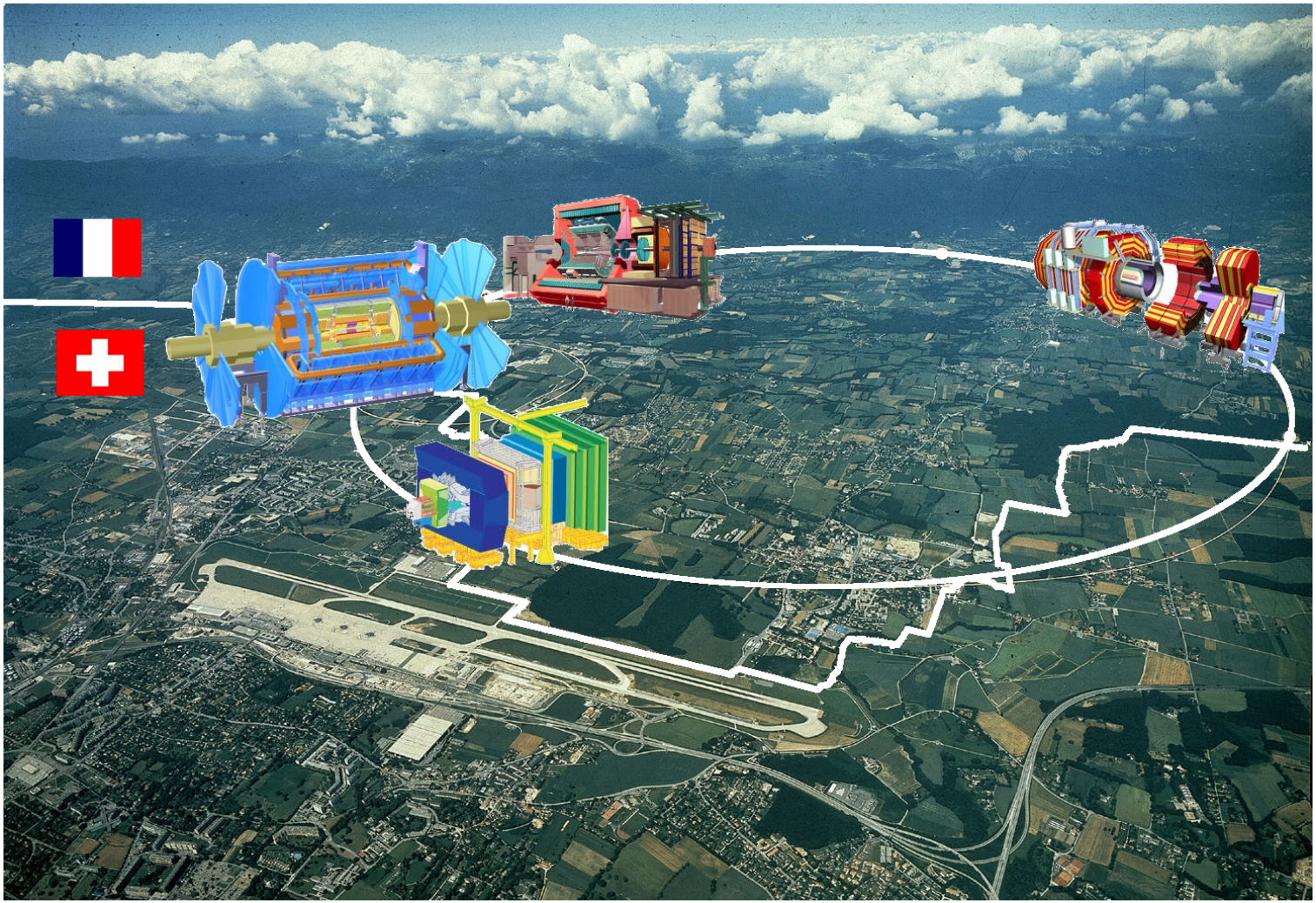
However, you really care about some FOM (*e.g.*, $S/\sqrt{S+B}$), so plot your FOM vs response instead (and optimize to it too)!

Now Some Real-World Examples from LHCb

(use of LHCb data is pure laziness; BDTs used in all HEP experiments)

LHCb @ CERN

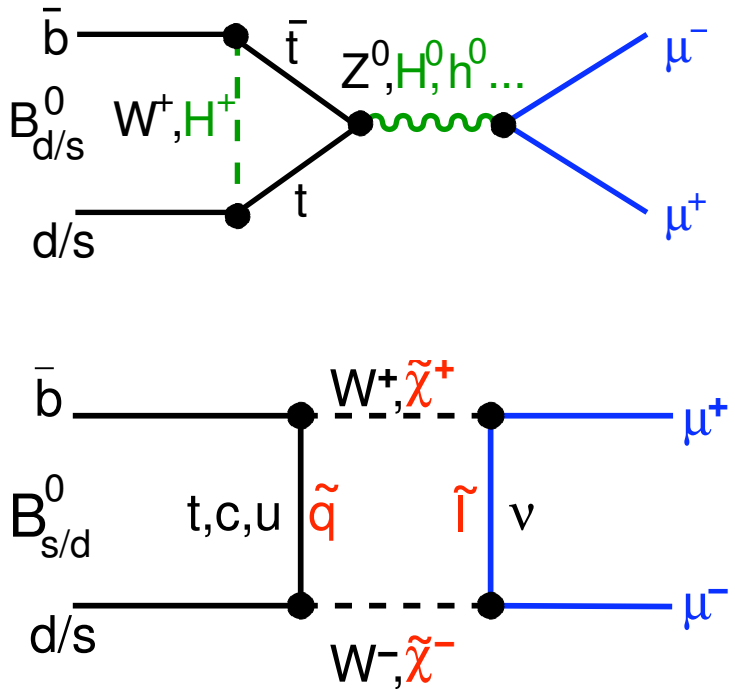
LHC(2012): $\sqrt{s} = 8$ TeV, $f_{\text{int}}^{\text{max}} \sim 20$ MHz, $\mathcal{L}^{\text{max}} \sim 7 \cdot 10^{33}$ cm^{-2}s^{-1}



LHCb is a FWD spectrometer built to study heavy-quark physics.

$B_{d,s} \to \mu^+\mu^-$

The SM predicts $\mathcal{B}(B_s \to \mu^+\mu^-) = (3.5 \pm 0.3) \times 10^{-9}$, but SUSY (or many other BSM physics) can greatly enhance this ($B_{d,s} = \bar{b}(d,s)$).

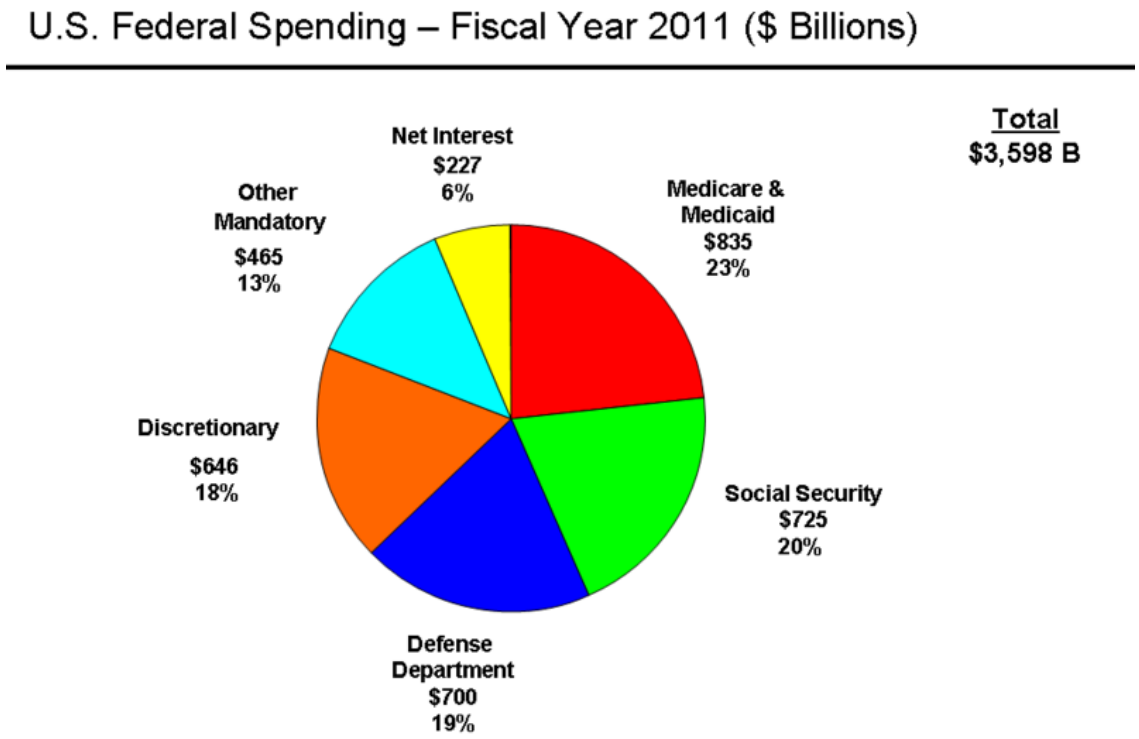


This is a **rare** process in a *dirty* environment; even when this decay happens, most of the energy and tracks in the event have nothing to do with it. Advanced analysis techniques are required!

$B_{d,s} \to \mu^+\mu^-$

$$\mathcal{B}(B_s \to \mu\mu) = (3.5 \pm 0.3) \times 10^{-9} \implies \text{1 in 1.6 trillion } pp \text{ in LHCb}$$

$\Rightarrow$

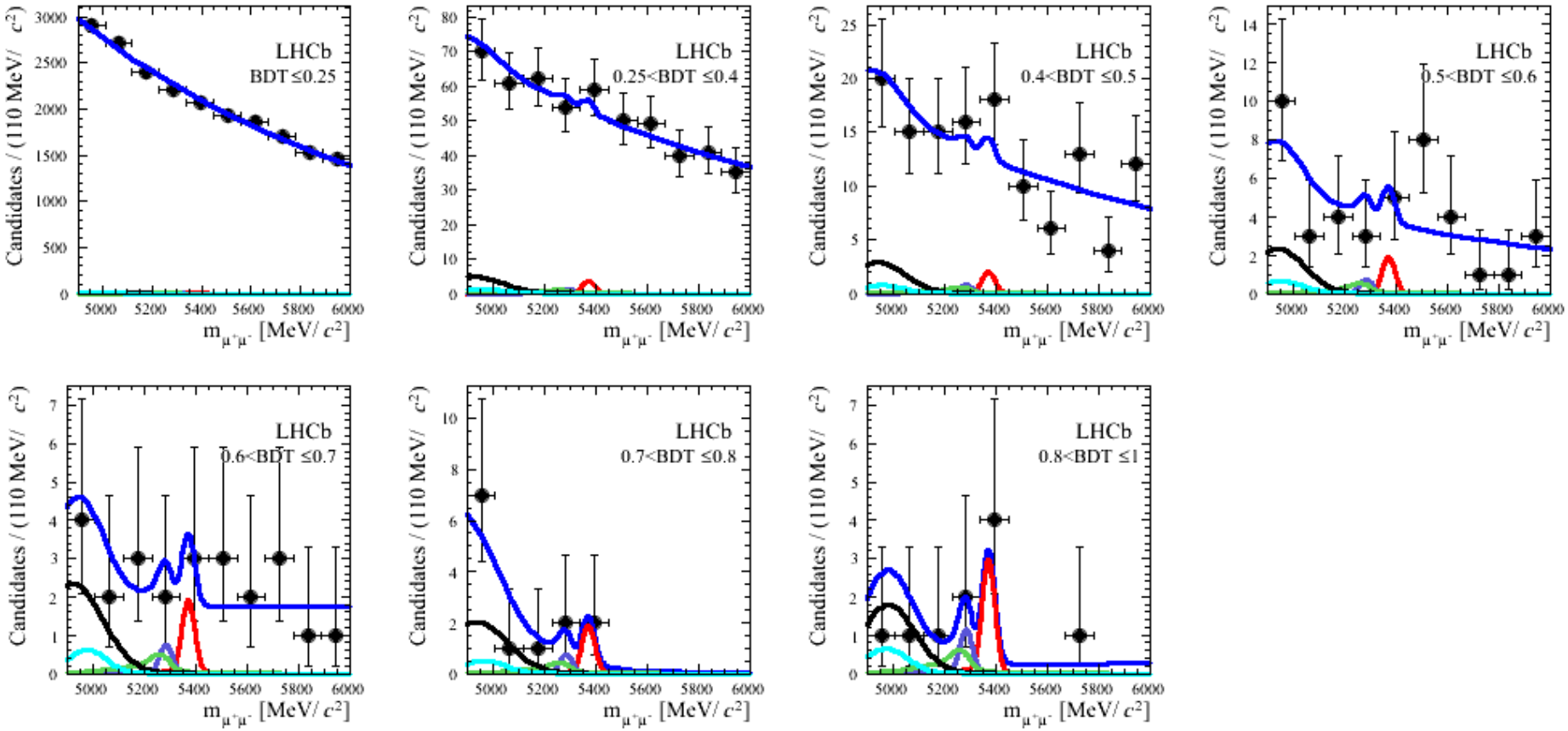


Cutting won't work here any better than with the US budget. So, a BDT selection and many data-driven constraints are used in the LHCb analysis.

$B_{d,s} \to \mu^+\mu^-$ (First Evidence!)

LHCb, TBP in PRL [arXiv:1211.2674]

About 1/2 of 2012 data shown (2011 data also used but not shown here):

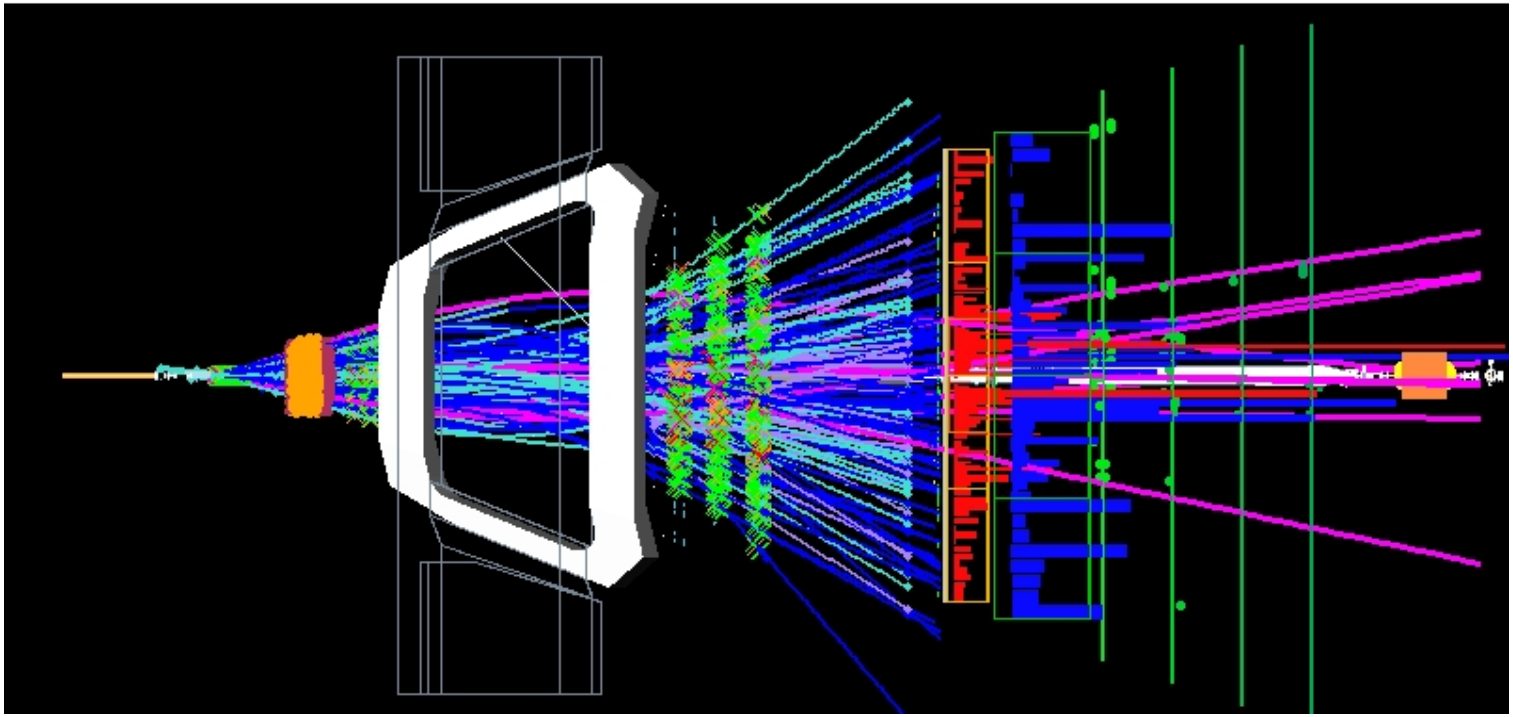


$\mathcal{B}(B_d \to \mu\mu) < 9.4 \times 10^{-10}$

$\mathcal{B}(B_s \to \mu\mu) = (3.2^{+1.5}_{-1.2}) \times 10^{-9}$

Triggering @ LHCb

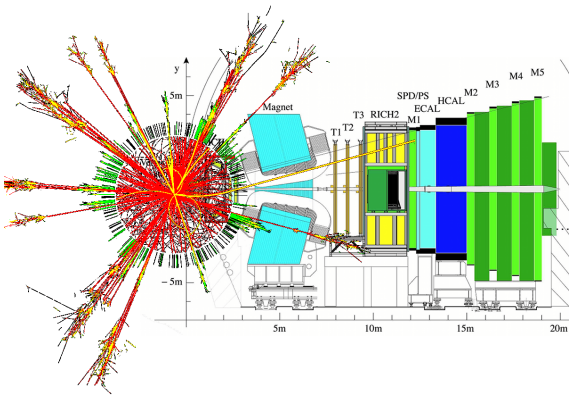
$f_{\rm int}^{\rm LHC} \sim 20$ MHz, $\sigma(pp \to b\bar{b}X)/\sigma(pp_{\rm inel}) \sim 0.4\%$



LHC*b* Event Display

We can only keep 2 kHz (1 in 10,000 events) of data. How do we decide?

LHCb Trigger Strategy



20 MHz →

1 MHz →

2 kHz →

We can only read out our detector at 1 MHz so a hardware trigger is required (hopefully removed for upgrade) to reduce the rate to 1 MHz.

LHCb's HLT (software) runs 26k PROCs (giving it just 20 ms/event) to reduce the rate by another factor of 400. Cut-based selections aren't able to provide adequate signal efficiency and background rejection. Can we use a BDT online?

Trigger BDT: The Bonsai BDT

There are three major concerns which need to be addressed prior to using a BDT in a high-level trigger:

- If the leaves are small relative to the resolution or stability, the signal could oscillate in and out of the signal regions. This would result in a lower efficiency and one that is very difficult to understand.
- Many signals of interest have unknown PDFs or aren't known to be of interest prior to collecting the data. An *inclusive* trigger should select classes of signal types rather than one specific signal whose PDF is known at training time.
- Any HLT algorithm must run in the online environment; thus, it must be extremely fast.

There is a single simple solution to all three of these issues . . .

The Bonsai BDT (BBDT)

The simplest solution is to discretize all of the variables used in the BDT. This limits where the splits of the data can be made permitting the analyst to control and shape its growth; thus, we call it a bonsai BDT (BBDT).

Math

Taking $\vec{x} \rightarrow x_{\text{discrete}}$ enforces $\Delta x_{\text{min}} > \delta_x \ \forall \ x$ on all leaves, where $\delta_x = \text{MIN}\{|x_i - x_j| : x_i, x_j \in x_{\text{discrete}}\}$.

Words

Remove the information you don't want the BDT to use prior to training so you know it won't use it (*e.g.*, tails of PDFs with low stats or regions highly sensitive to misalignments). Also allows converting response into a 1-D array lookup (super fast).

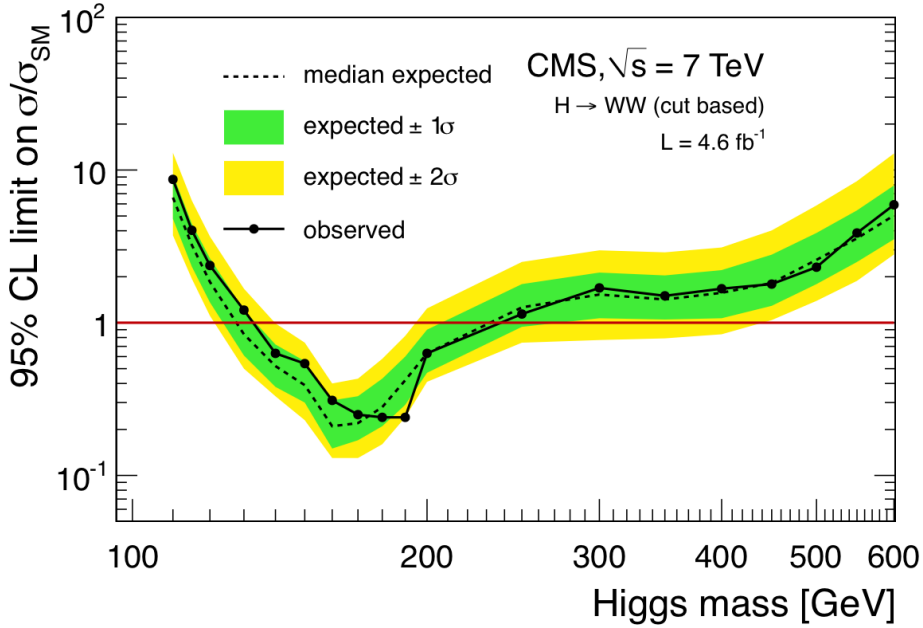
See V.Gligorov and M.Williams, TBP in JINST [arXiv:1210.6861].

Performance

Performance is better than was thought possible. The $b\bar{b}$ purity is $\sim 95\%$.

$H \to WW$ @ CMS

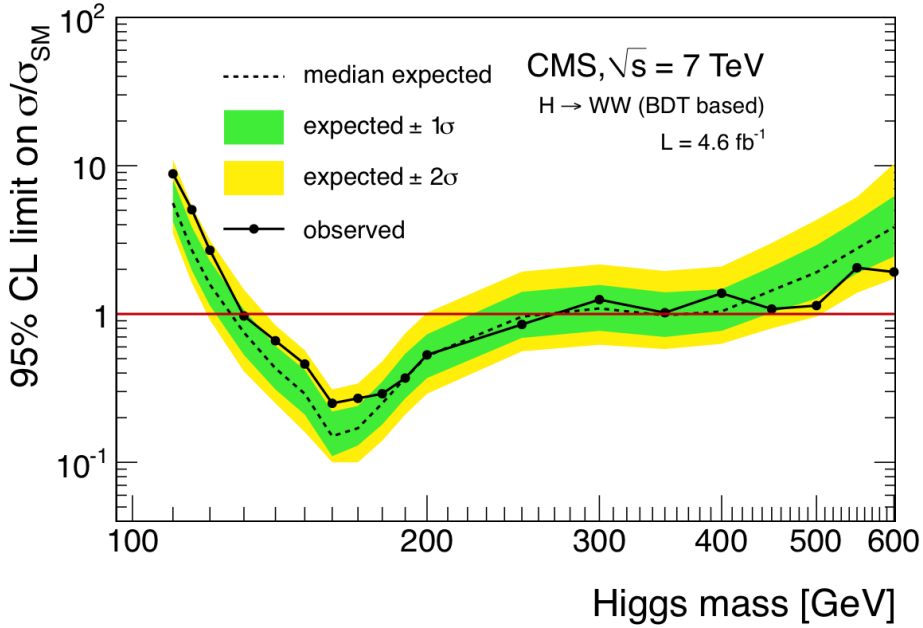
In 2011, CMS published $H \to WW$ results using both cut-based and BDT-based analyses. The BDT performance is (not surprisingly) better.



The cut-based exclusion was (132-238) GeV; the BDT-based exclusion was (129-270) GeV [arXiv:1202.1489].

$H \to WW$ @ CMS

In 2011, CMS published $H \to WW$ results using both cut-based and BDT-based analyses. The BDT performance is (not surprisingly) better.



The cut-based exclusion was (132-238) GeV; the BDT-based exclusion was (129-270) GeV [arXiv:1202.1489].

Advice

Machine learning methods are powerful but "with great power comes great responsibility" so make sure you follow this advice:

- adjust free parameters so that training error and validation error are as similar as possible.
- it's not paranoia if the classifiers **are** out to get you . . . always assume it has overtrained and convince yourself/referees/your boss otherwise;
- boost, bag, *etc.*;
- choose your cut (if you're going to make one) based on a FOM that is close to what you want to measure;
- it's a good idea to check the variable "ranking" and ask yourself "does this make sense?"

N.b., you can *reverse engineer* a BDT/ANN for other purposes. *I.e.*, you can train a BDT and then study its performance *vs* variates to see what it has learned about the data. This information may then be used elsewhere (*e.g.*, detector design) without the BDT.

Summary

Machine learning methods are more powerful than making cuts. Sure, training takes a while but not as long as designing cuts "by hand".

Ph.D. students best excuse for slacking off: Training their ANN/BDT.

Just be sure to train/validate "properly" and to have a reliable way to determine the efficiency (if needed). Have fun training!