Machine Learning with *T*MVA

A ROOT based Tool for Multivariate Data Analysis

Department of Physics and Astronomy

University of Pittsburgh, 11/16/2018

Many thanks to Lorenzo Moneta (CERN) for providing much of the information about recent developments

Today…

Intro to MVA and TMVA

Old and new features in TMVA

Jupyter and Swan

Tutorial

Multivariate Methods

Multivariate methods are designed to exploit large datasets in order to reduce complexity and find new features in data

Ideally suited for High Energy Physics to separate signal from background

- Particle Reconstruction
- Particle ID
- Event selection

Machine Learning is the applied research area studying multivariate methods on the theoretical and computational side

ML in HEP

- Until early 2000's used in HEP but met with large skepticism (black boxes)
  - *LEP and BABAR used NN, D0 used H-Matrix, MiniBoone used BDT, BABAR and BELLE used Fisher-D*
  - *Cut- and Likelihood-based approaches were the default*
- Explosion of application in the last 10 years

Now an integral part of the planning of the HEP community:

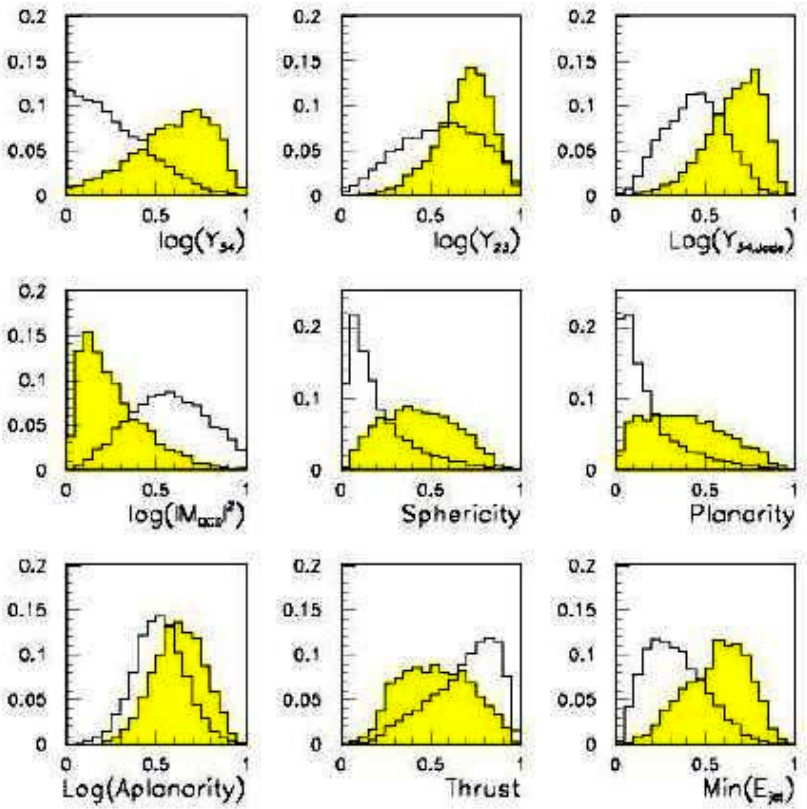
- Summer 2018: *Machine Learning in High Energy Physics Community White Paper (25p)*

https://arxiv.org/abs/1807.02876

Neural network example from LEP II

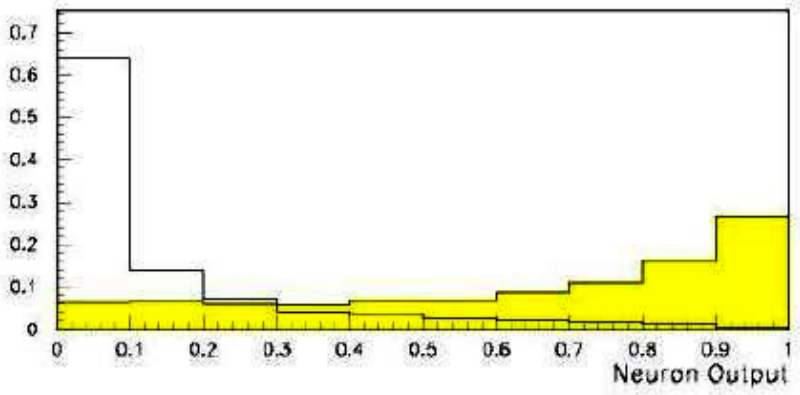
Signal: e+e−→ W+W− (often 4 well separated hadron jets)

Background: e+e−→ qqgg (4 less well separated hadron jets)



← input variables based on jet structure, event shape, ...
none by itself gives much separation.

Neural network output:

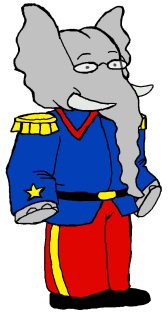


(Garrido, Juste and Martinez, ALEPH 96-144)

History of TMVA

In 2005: some usage of ML methods in HEP but lack of a systematic approach

From the BABAR experiment's (running 1999-2006) physics book:

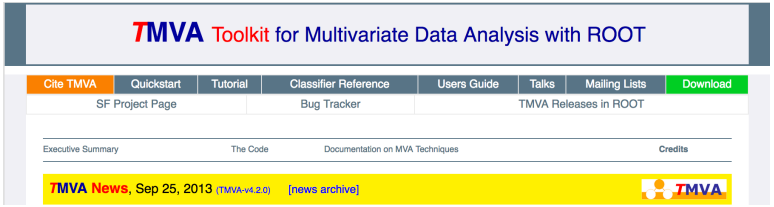


Considering the large choice of multivariate analysis methods and the need to compare their performances, it was considered desirable to provide common software for all of them. In order to achieve this goal, a general purpose package, named Cornelius* was developed.

*__C__ombined __O__ptimal __R__econstruction with __NE__ural network and __L__ikelihood for __I__dentification __Us__age

TMVA was the continuation of this in an experiment-independent way, using the root data format, which is the standard for HEP data

http://tmva.sourceforge.net/old_site/



- Started on sourceforge in 2005 and was merged into ROOT 5.11 in 2006
- Focus was on easy usability, many contributors added various methods
- Large variety of methods, also for educational purpose

Many kids on the block …

Large variety of packages available. To decide which one to use, consider

- Available methods and their performance
- Used data format
- Language
- Applicability (how to use the training result)
- User support and future

R is a commonly used language and environment for statistical analyses, including some MVA. Interactive mode as well as simple scripting. https://www.r-project.org/

Scikit-learn is a free software library (very extensive) for machine learning in python. https://scikit-learn.org

Keras, written in python, is an interface to several deep learning NN packages:

- TensorFlow originally from Google, supports easy GPU deployment (https://www.tensorflow.org/)
- Microsoft Cognitive Toolkit (CNTK) with a variety of DNN flavors https://docs.microsoft.com/en-us/cognitive-toolkit/
- Theano is a python library on top of Numpy, focus on code optimization on CPUs and GPUs https://pypi.org/project/Theano/

theano

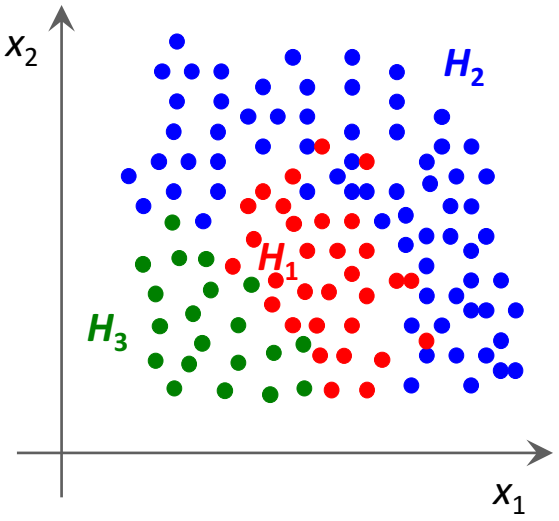
Theoretical background

Not possible to present a mathematical solid background today, there is a lot of statistics behind. Good material can be found:

- Ilya Narski's lectures on ML at the SLAC in 2006
  http://www-group.slac.stanford.edu/sluo/Lectures/Stat2006_Lectures.html
- Glen Cowan's statistics lectures at CERN
  https://indico.cern.ch/event/77830/
  and his seminar on *Recent progress in multivariate methods for particle physics*
  http://www.pp.rhul.ac.uk/~cowan/stat/cowan_weizmann10.pdf
- Free book: *An Introduction to Statistical Learning (with Applications in R)*
  http://www-bcf.usc.edu/~gareth/ISL/
- Another free book: *The Elements of Statistical Learning*
  https://web.stanford.edu/~hastie/ElemStatLearn/

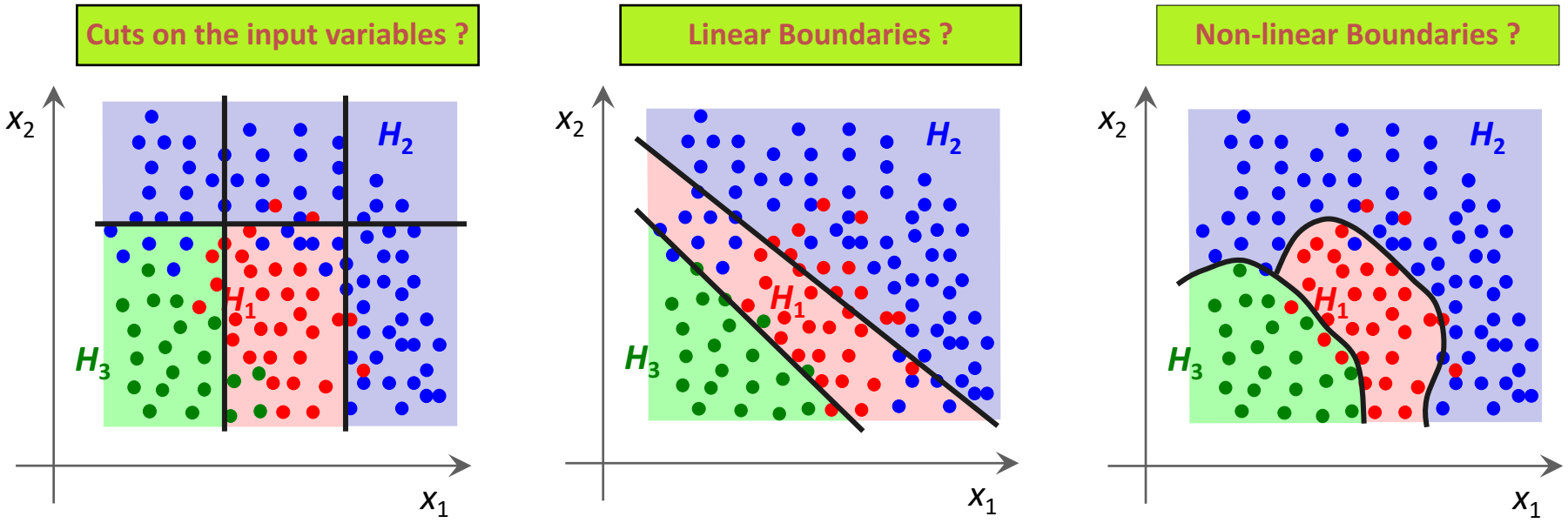
Classification problem

- Events described by k variables (that are found to be discriminating) → $(x_i) \in \mathfrak{R}^k$
- Events can be classified into n categories: $H_1 \ldots H_n$
- General classifier: $f: \mathfrak{R}^k \rightarrow \aleph$, $(x_i) \rightarrow \{1,\ldots,n\}$



General Event Classification Problem

The question: what 'decision boundary' should we use to accept/reject events as belonging to event types H_1, H_2 or H_3?



How can we do this in an optimal way, when we have many input variables ?

→ let the machine do the job, using multivariate techniques

A very brief overview on available early methods

Available methods in the old version (up-to 2015):

- Rectangular cut optimization
- Projective likelihood estimation (PDE approach)
- Multidimensional probability density estimation (PDE - range-search approach)
- Multidimensional k-nearest neighbor classifier
- Linear discriminant analysis (H-Matrix and Fisher discriminants)
- Function discriminant analysis (FDA)
- Predictive learning via rule ensembles (RuleFit)
- Support Vector Machine (SVM)
- Artificial neural networks (various implementations)
- Boosted/Bagged decision trees (BDT)

Will not go through them as nowadays mainly NN and BDT are used in HEP

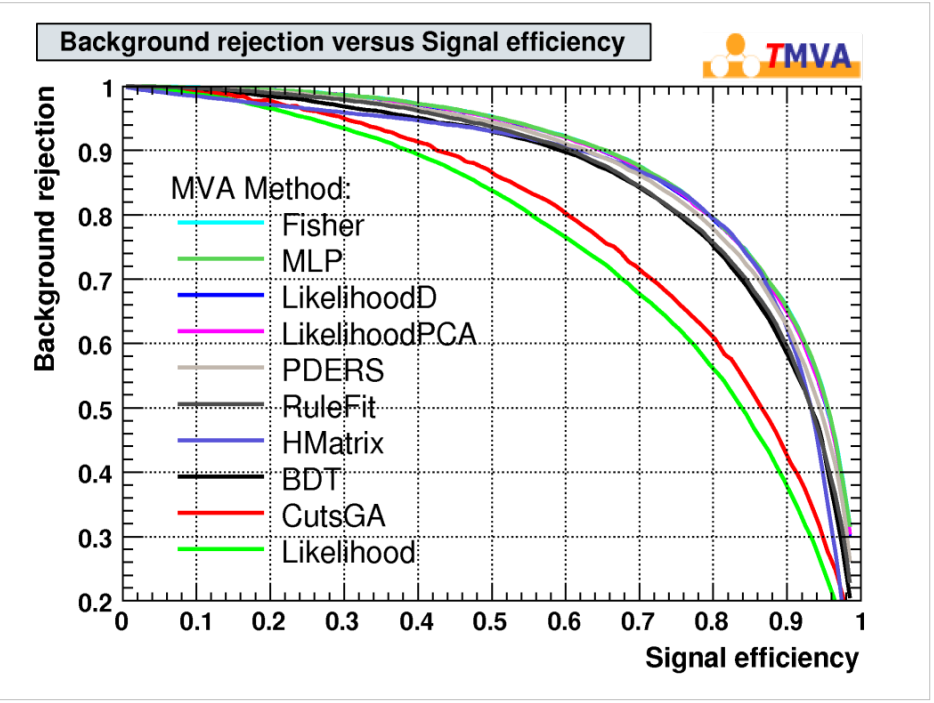
- Check out the old TMVA Users Guide for detailed information
  https://root.cern.ch/download/doc/tmva/TMVAUsersGuide.pdf

Performance comparison

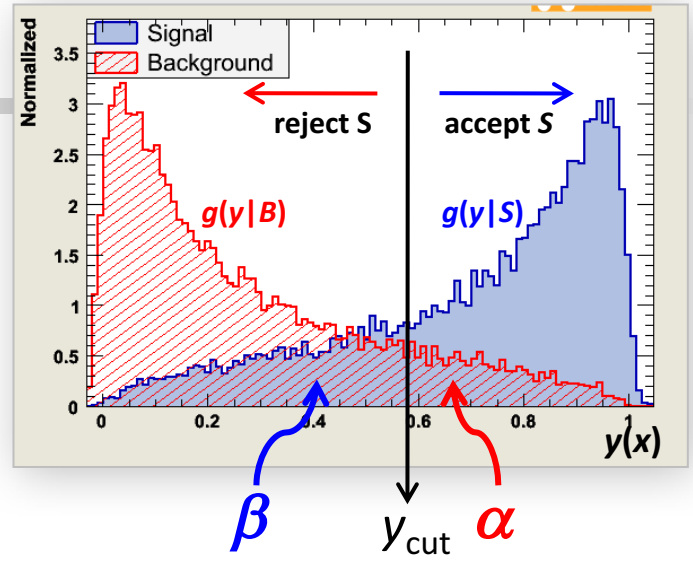
Receiver operating characteristic (ROC) curve plots false-negative rate vs true-positive rate of a system, e.g. a binary classifier.

- Term from Second World War, as a measure of the capability of a radar receiver operator to correctly identify radar signals as Japanes airplanes. Later adopted to signal detection theory. (Sometimes also false-positives vs true-positives)

Maximize background rejection for given signal efficiency. In general maximizes the area under the curve.



The Neyman-Pearson Lemma



Signal and background have true distributions, described by respective likelihood functions.

A test can be defined by a cut on y which separates signal an background.
Under background hypothesis:

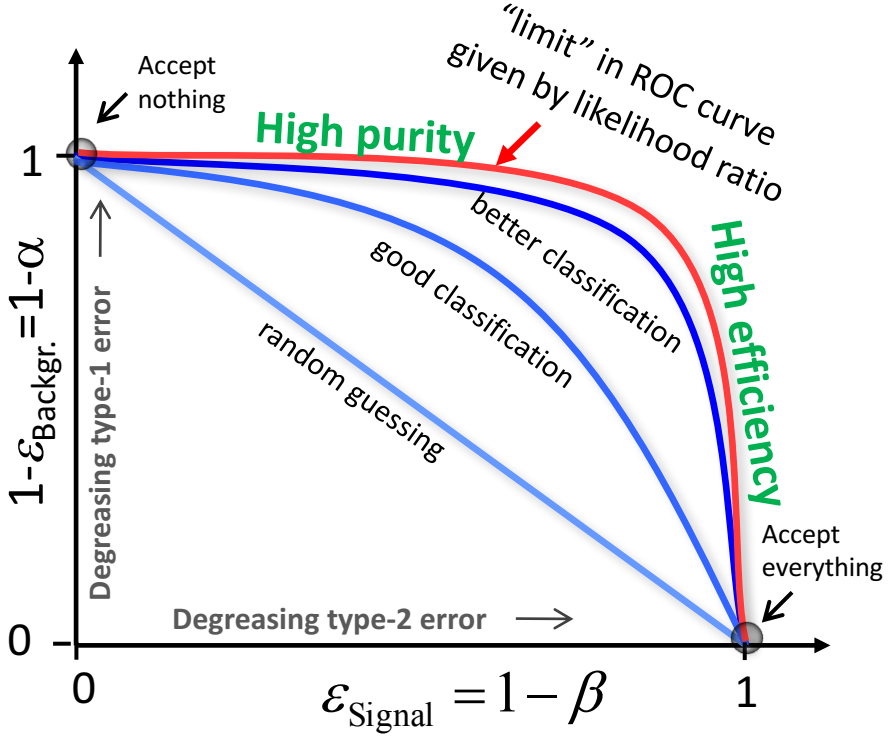
- Type-I error α : background identified as signal
- Type-II error β : signal identified as background

NP-Lemma:

"The likelihood-ratio test as selection criteria gives for each selection efficiency the best background rejection."

- It maximizes the area under the ROC-curve

There is a natural limit to how good a classifier can become



New features of TMVA

New major features added recently and available in the latest ROOT

Deep Learning

- support for parallel training on CPU and GPU (with CUDA)
- with fully connected (dense) , convolutional and recurrent layers

Improved BDT:

- new loss functions for regression
- improved performances with multi-thread parallelization

Cross Validation and Hyper-parameter optimization

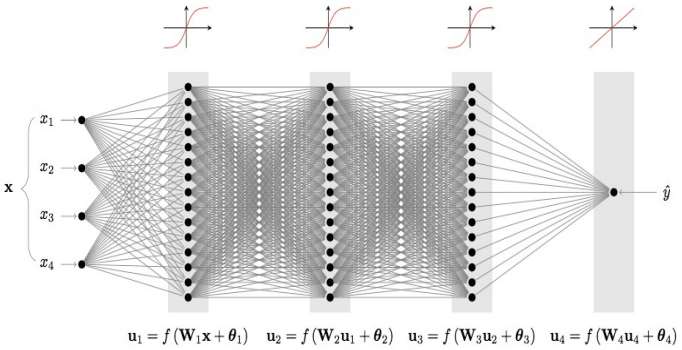
Interfaces to external ML library in R and Python (scikit-learn and Keras)

Integration with Jupyter

New Deep Learning Features

Deep neural networks (DNN) are artificial neural network with several hidden layers and a large number of neurons in each layer.

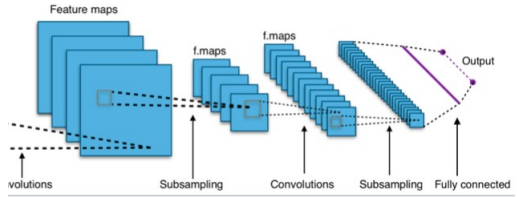
- Outperform ANN with only a couple of hidden layers
- Need sufficiently large datasets for training

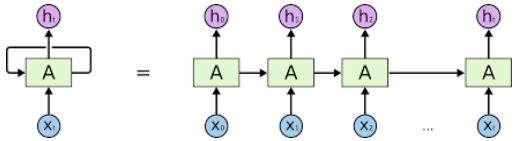


Available for dense layer since 2016 (ROOT 6.08)

Extended Deep Learning in ROOT 6.14 supporting

- Convolutional Layer
  - *powerful for image data sets*
- Recurrent Layer
  - *useful for time dependent data or varying number of input variables*





Optimized for parallel evaluation and training

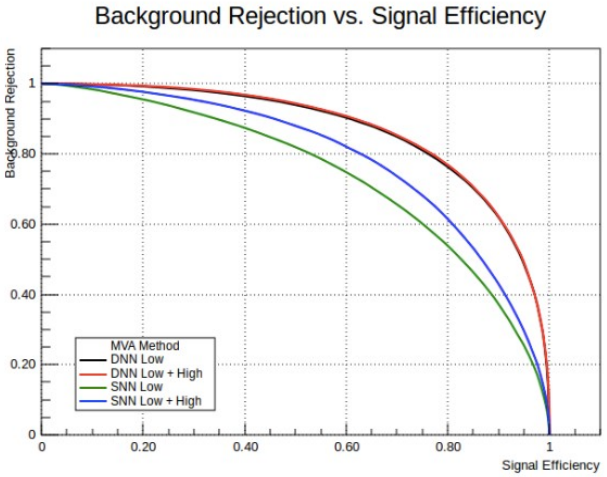
- CPU (with openBLAS + TBB) and GPU (CUDA)

Several optimizers available (e.g. SGD, ADAM, …)

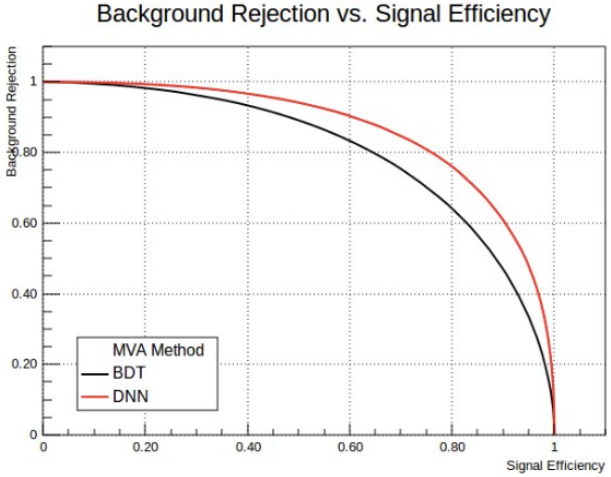
Deep Learning Performance

Using Higgs public dataset (from UCI) with 11M events

DNN vs Standard ANN

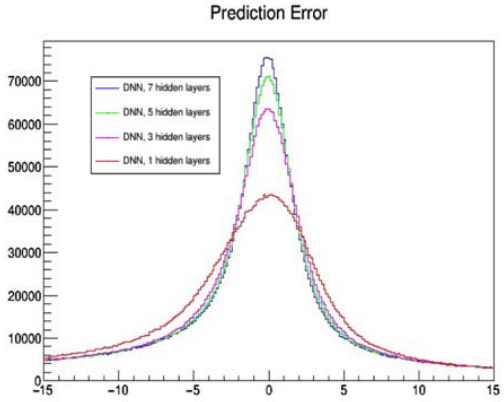


DNN vs BDT



Significant improvements compared to shallow networks and BDT

Performance grows significantly with more hidden layers

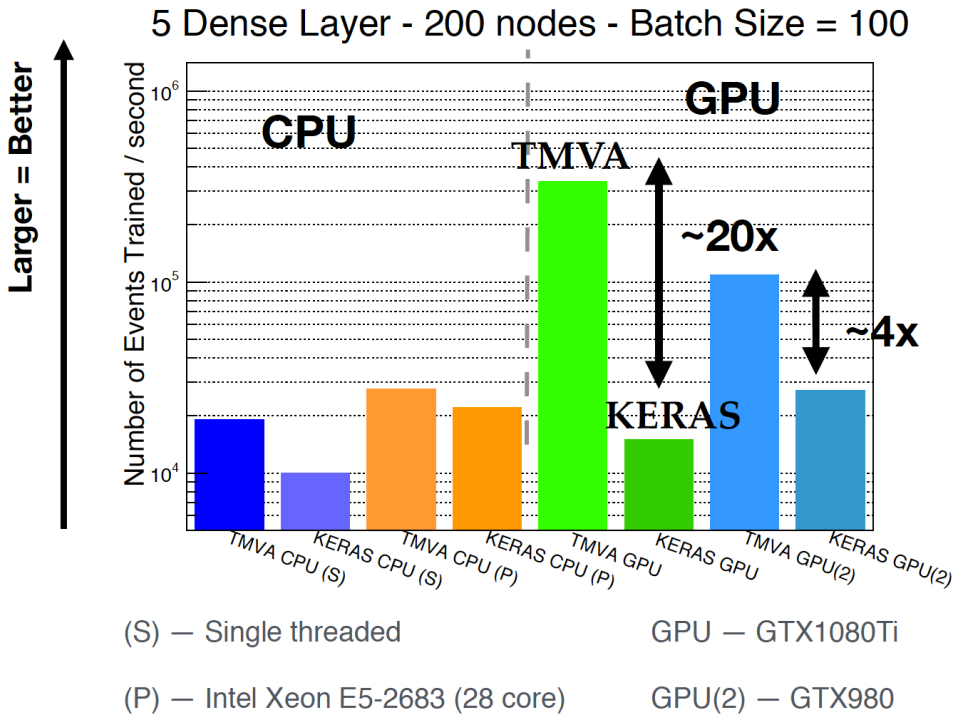


DNN Training Performance

Training time — Dense networks

- Higgs UCI dataset with 11M Events
- TMVA vs. Keras/Tensorflow
- "Out-of-the-box" performance

Excellent TMVA performance !



(S) — Single threaded

(P) — Intel Xeon E5-2683 (28 core)

GPU — GTX1080Ti

GPU(2) — GTX980

Convolutional Neural Network

CNN uses convolutional layers for task-specific feature extraction prior to dense layers for classification.

- Instead of being fully connected, the convolutional layer consists of small kernels to transform and image and extract features
- A nice read: https://developer.nvidia.com/discover/convolutional-neural-network

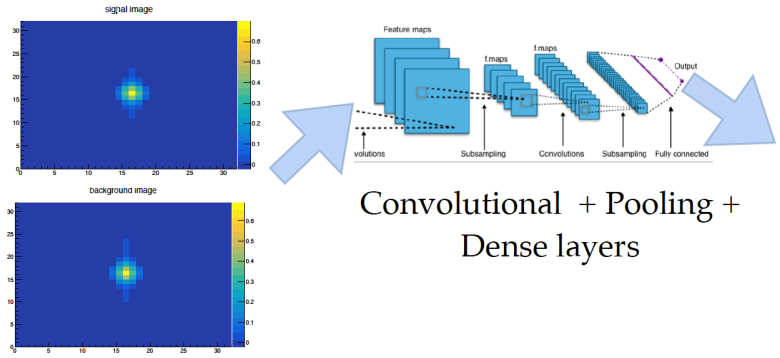
Available in latest ROOT version (6.14)

- Supports CPU parallelization, GPU is now also available
- Parallelization and code optimization is essential (excellent TMVA performances)

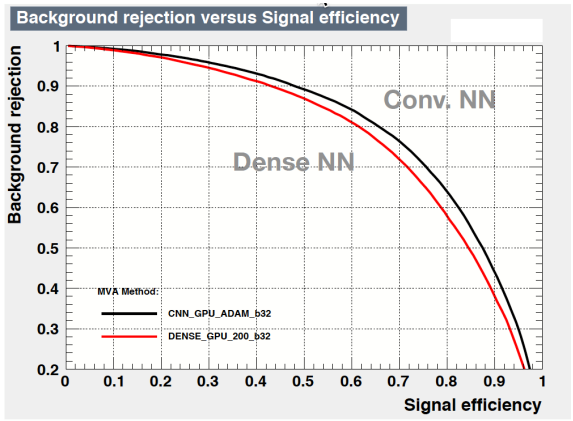
Example:

- Input: image dataset from simulated particle showers from an electromagnetic calorimeter
- Task: distinguish electron from photon showers

input 32x32 images



Convolutional + Pooling + Dense layers



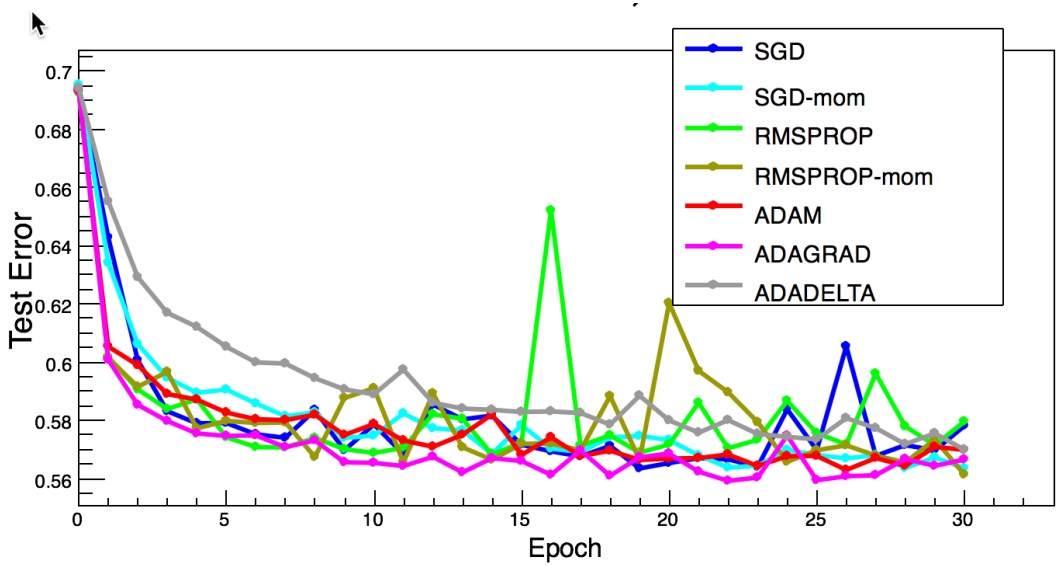
Convolution is probably the most important concept in deep learning right now with the advance of autonomous machines!

New Deep Learning Optimizers

Integrated in TMVA master new deep learning optimizers

In addition to SGD (Stochastic Gradient Descent) added these

- support acceleration using momentum
- ADAM (new default)
- ADADelta
- ADAGrad
- RMSProp

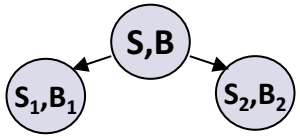


With these new optimizers need less epochs (iterations) to converge!

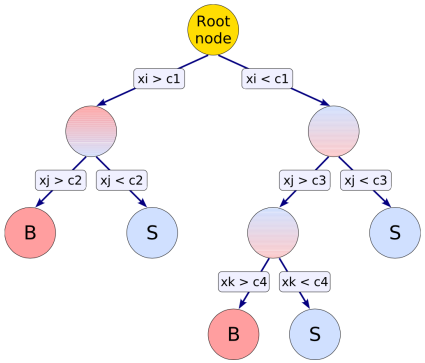
- More material, e.g.: *The Marginal Value of Adaptive Gradient Methods in Machine Learning* https://arxiv.org/abs/1705.08292

Decision Trees and Forests

Decision Trees (DT) are rather simple, they are grown by splitting the sample at each step by a cut optimizing



$$Gini = \frac{S_1 B_1}{S_1 + B_1} + \frac{S_2 B_2}{S_2 + B_2}$$



Power comes from growing an ensemble of trees in various ways. Final classifier is build by linearly combining the trees of the ensemble

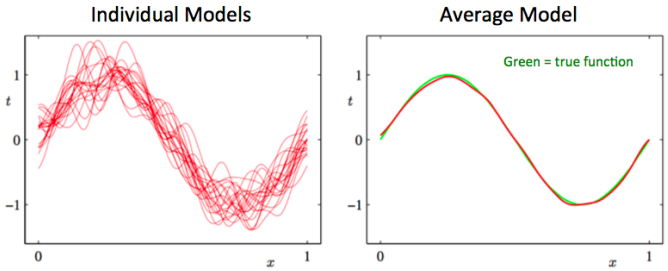
- Large coefficient for DT with small misclassification

Growing methods

- **Boosting (BDT)**: continuously increase the weight of incorrectly identified events and build a new DTs
  - *Often trees are pruned to avoid overtraining*
- **Randomly (Random Forest)**: build many independent trees from random subsets of the training sample. No pruning.

Features

- Excellent performance and easy to train (not much tuning needed)
- Robust to noisy data and rarely overtraining



Very popular method (before deep learning)

- ATLAS is moving from BDT to DL only now

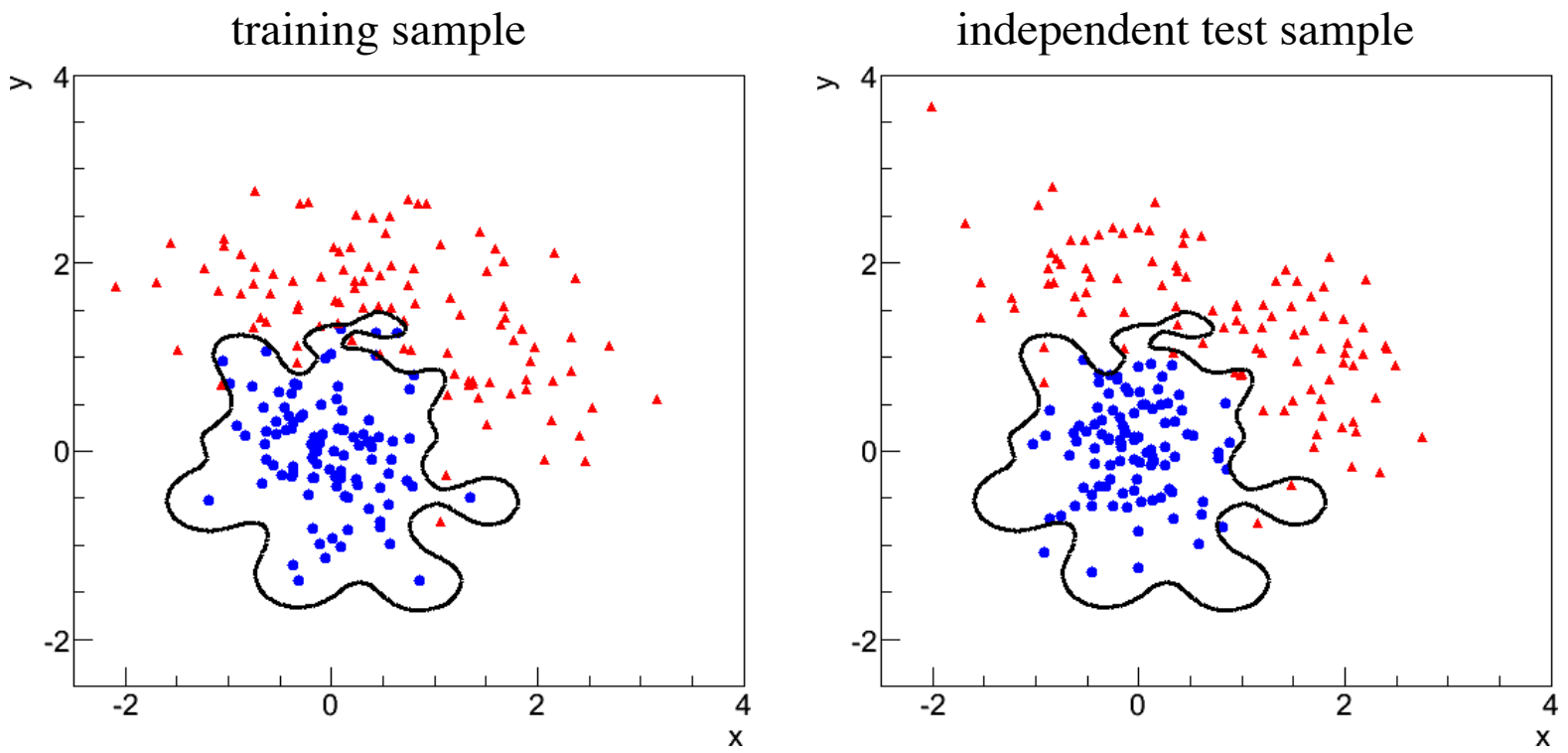
TMVA provides a very good implementation of BDT

Overtraining

If decision boundary is too flexible it will conform too closely to the training points → overtraining.

Monitor by applying classifier to independent test sample. Performance on the test sample should stay at maximum.

- TMVA has overtraining indicator for Neural Networks



Regression with BDT

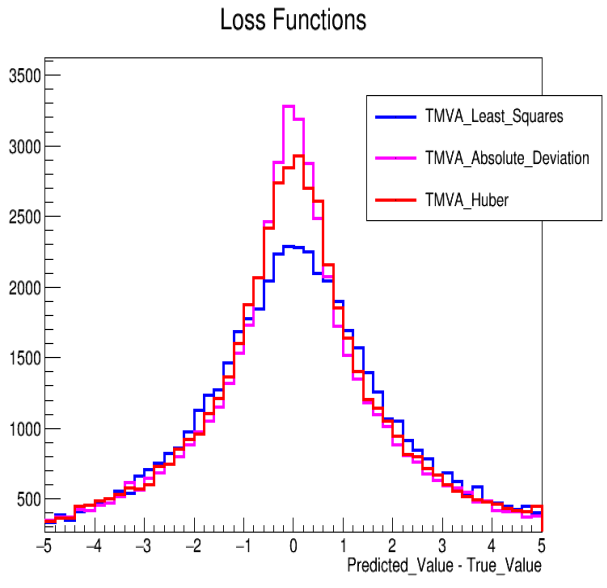
It is also possible to do regression training with TMVA

- Continuous target variable, e.g. cluster energy correction
- Not supported by all methods, used mostly with BDT

New loss functions for regression

- Huber (default)
- Least Squares
- Absolute Deviation
- Custom Function

Important for regression performance

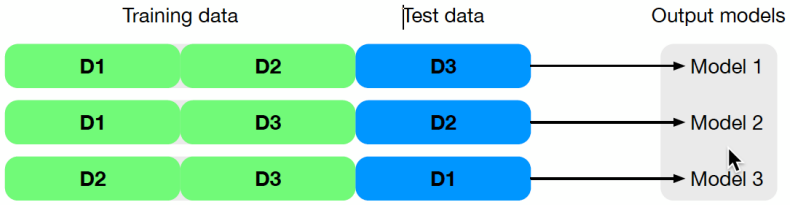


Cross validation in TMVA

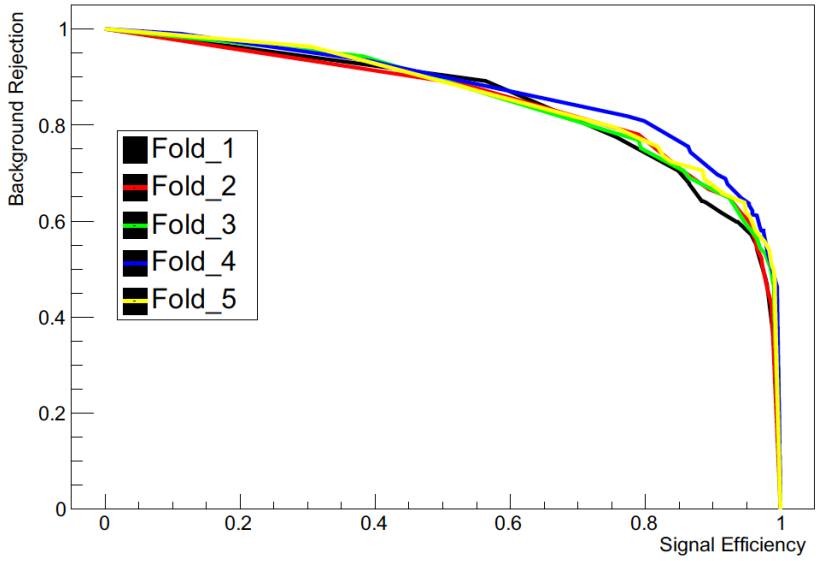
Cross-validation to assess how well a classifier can perform on a given dataset. In itself it does not result in a trained classifier.

TMVA supports k-fold cross-validation

- Dataset is partitioned into k subsets

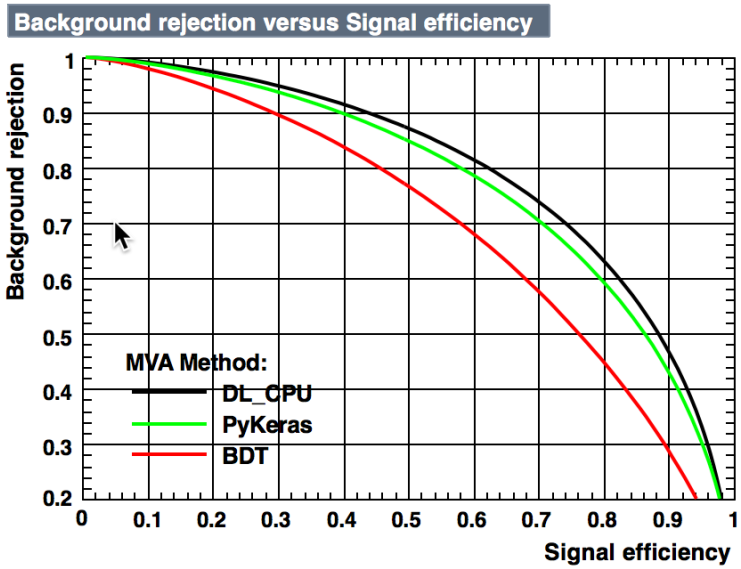


- k training rounds (or less), each time a different subset is excluded from the training and used for testing
- Average performance of each round
- Advantages:
  - *full dataset statistics is used for training*
  - *Little susceptibility to overtraining*



TMVA Interfaces to other packages

External tools are available as additional methods in TMVA and they can be trained and evaluated as any other internal ones!



RMVA: Interface to ML methods in R

- c50 (decision trees and rule-based models)
- xgboost eXtreme gradient boosting
- RSNNS (Stuttgart NN),
- SVM (e1071)

PYMVA: Interface to Python ML

- scikit-learn
  - *with RandomForest, Gradient Tree Boost, Ada Boost*
- Keras (Theano + Tensorflow)
  - *supports model definition in Python and then training and evaluation in TMVA*

Direct mapping from ROOT tree to Numpy arrays also available (TTree::AsMatrix) !!

Example PyMVA with Keras

1) define you Keras model in Python

Define model for Keras

In [5]:
```
# Define model
model = Sequential()
model.add(Dense(32, init='glorot_normal', activation='relu',
        input_dim=numVariables))
model.add(Dropout(0.5))
model.add(Dense(32, init='glorot_normal', activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(2, init='glorot_uniform', activation='softmax'))

# Set loss and optimizer
model.compile(loss='categorical_crossentropy', optimizer=Adam(),
        metrics=['categorical_accuracy',])

# Store model to file
model.save('model.h5')

# Print summary of model
model.summary()
```

2) book your method as usual in TMVA

- (information is exchange through file)

Book methods

Just run the cells that contain the classifiers you want to try.

In [6]:
```
# Keras interface with previously defined model
factory.BookMethod(dataloader, ROOT.TMVA.Types.kPyKeras, 'PyKeras',
        'H:!V:VarTransform=G:FilenameModel=model.h5:'+\
        'NumEpochs=10:BatchSize=32:'+\
        'TriesEarlyStopping=3')
```

Out[6]:
```
<ROOT.TMVA::MethodPyKeras object ("PyKeras") at 0x77e48b0>
```

3) train, test and evaluate inside TMVA (using TMVA::Factory)

Run training, testing and evaluation

In [8]:
```
factory.TrainAllMethods()
```

```
Factory                  : Train all methods
```

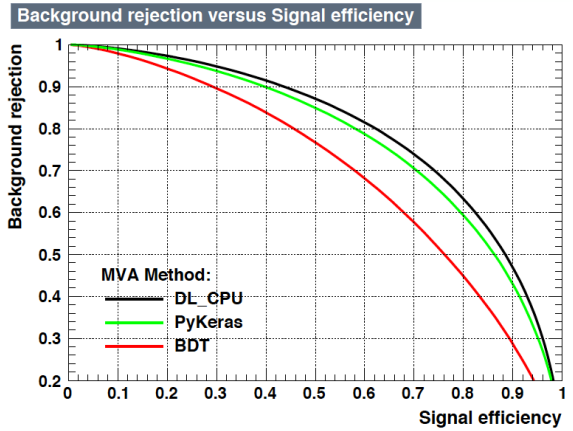
In [9]:
```
factory.TestAllMethods()
```

```
Factory                  : Test all methods
Factory                  : Test method: PyKeras
                         :
```

In [10]:
```
factory.EvaluateAllMethods()
```

```
Factory                  : Evaluate all methods
Factory                  : Evaluate classifier: PyKeras
                         :
```

4) examine the result using the GUI



Future Developments

The aim is to provide to the users community efficient physics workflows

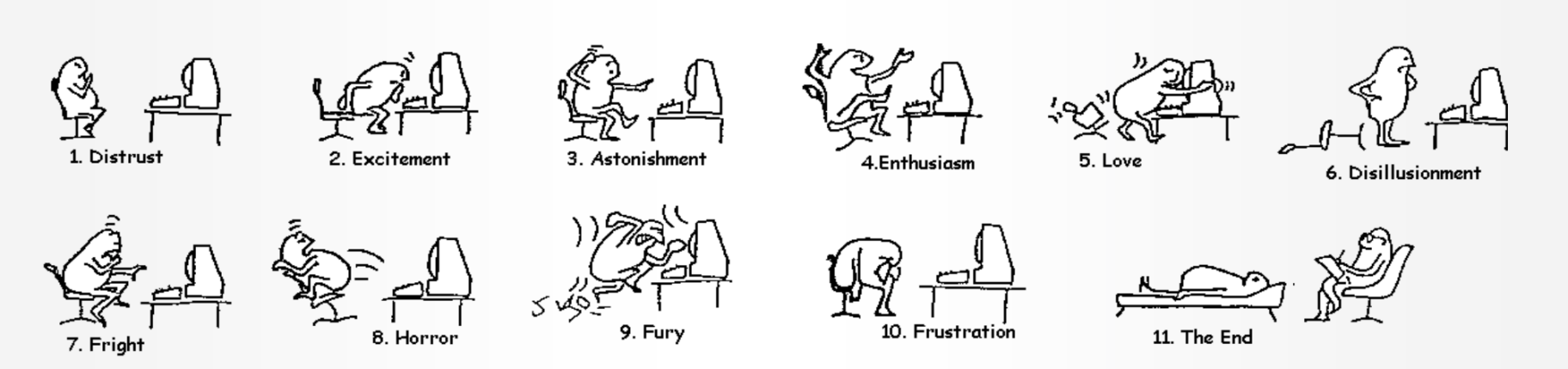
Includes tools for efficient

- data loading (using new RDataFrame for filtering the data)
- integration with external ML tools
- training of commonly used architectures
- deployment of trained models

TMVA efficiently connects input data to ML algorithms

- working on defining new user interfaces

Using TMVA


1. Distrust
2. Excitement
3. Astonishment
4.Enthusiasm
5. Love
6. Disillusionment
7. Fright
8. Horror
9. Fury
10. Frustration
11. The End

Using TMVA

TMVA workflow overview

Reading input data

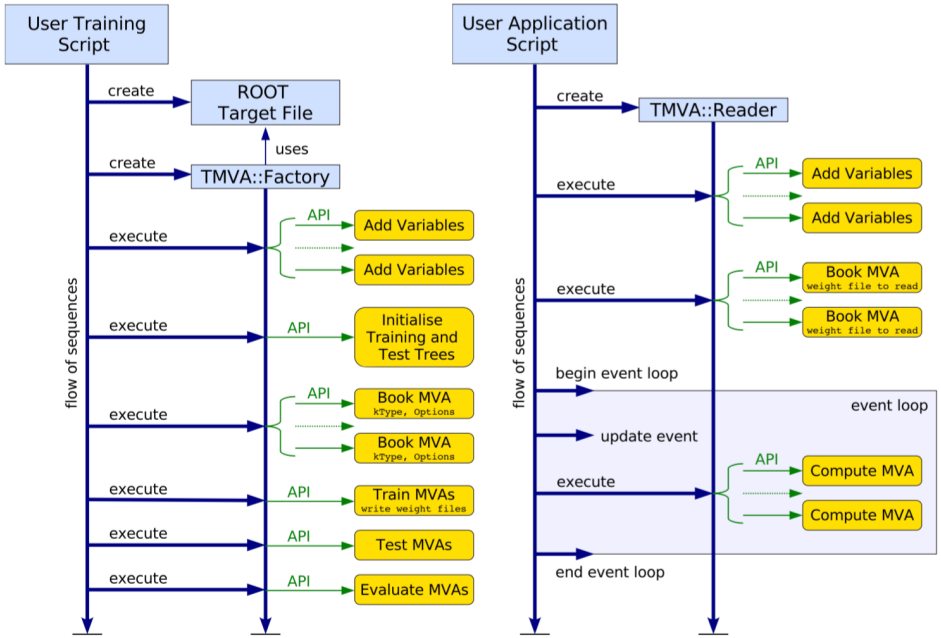
Select input features and preprocessing

Training

- find optimal classification or regression parameters using data with known labels (e.g. signal and background MC events)

Testing

- evaluate performance of the classifier in an independent test sample
- compare different methods



Application

- apply classifier/regressor to real data (where labels are not known)

Training result can be saved as text file, C++ class, and/or root file.

Training + Testing session

Instantiate TMVA::Factory and TMVA::DataLoader

- DataLoader is a new class that allows greater flexibility when working with datasets

Provide input data files

- input data from ROOT Trees or ASCII data (e.g. csv)
- pre-selection cuts on input data can be applied
- event weights (negative weights for some methods) can be given

Declare input and target variables using the DataLoader

Partition data (training/test split)

- various method for splitting training/test samples

Book MVA methods

```
factory->BookMethod( DataLoader *loader, Types::EMVA theMethod,
                     const char * methodTitle, const char *option = "" );
```

Train/Test/Evaluate using the method of the factory

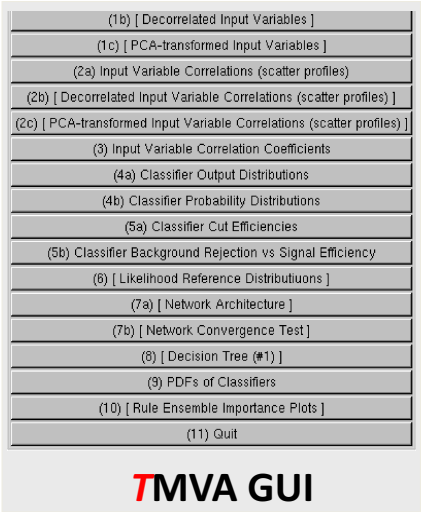
- GUI for output evaluation and analysis

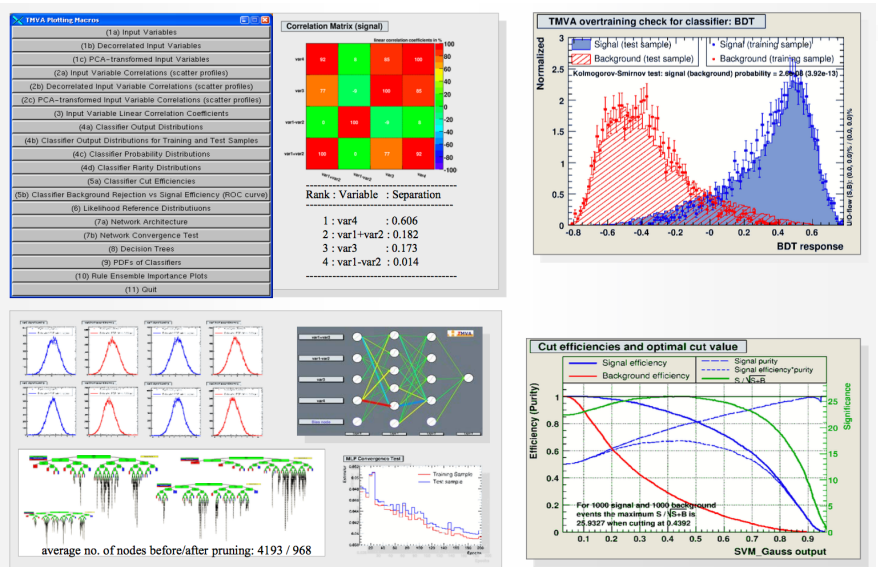
Make use of:

- k-fold cross-validation and hyper-parameter optimisation
- algorithm to identify importance of input variables

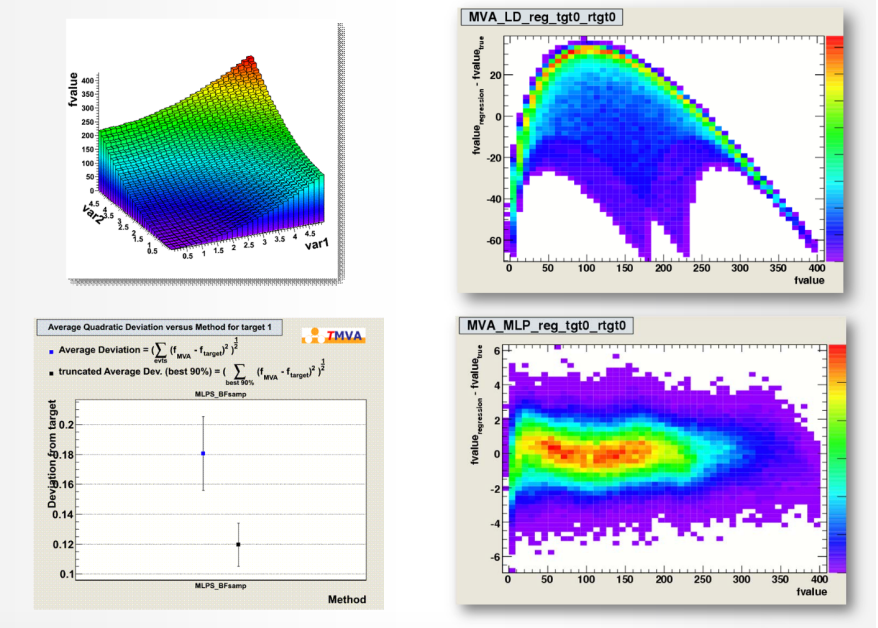
Evaluate

TMVA produces an output file that can be examined with a special GUI (TMVAGui)





For regression training a separate GUI exists (TMVARegGui)



FOR THE TUTORIAL

Jupyter for browser-based development

Interactive computing interface supporting multiple programming platforms and running in a browser

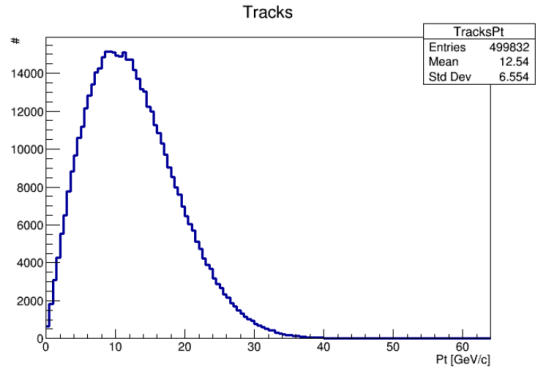
- Supported languages: Python 2 and 3, Haskell, Julia, R … One generally speaks about a "kernel" for a specific language
- Widely adopted outside HEP already

Access TTree in Python using PyROOT and fill a histogram

Loop over the TTree called "events" in a file located on the web. The tree is accessed with the dot operator. Same holds for the access to the branches: no need to set them up - they are just accessed by name, again with the dot operator.

In [1]:
```
import ROOT

f = ROOT.TFile.Open("http://indico.cern.ch/event/395198/material/0/0.root");
h = ROOT.TH1F("TracksPt","Tracks;Pt [GeV/c];#",128,0,64)
for event in f.events:
    for track in event.tracks:
        h.Fill(track.Pt())
c = ROOT.TCanvas()
h.Draw()
c.Draw()
```



Based on notebooks, which contain

- Information on which kernel (metadata)
- Source code
- Explanatory text and graphics
- Results

Appealing features:

- Sharing: scientists can share their results (code, plots, text) in the form of notebooks
- Teaching: runnable tutorials and exercises, combining code and explanations
- Reproducibility: a notebook contains results and the code that led to them

Using Root with Jupyter

Root kernel for Jupyter was written in 2016

Comes in two flavors, C++ (using the C++ interpreter) and python (using PyROOT)

Fit the function to the generated data.

In [3]:
```
f2.SetParameters(0.7, 1.5);  // set initial values for fit
f2.SetTitle("Fitted 2D function");
dte.Fit(&f2);
```

```
 FCN=517.445 FROM MIGRAD    STATUS=CONVERGED      38 CALLS          39 TOTAL
                     EDM=2.65702e-12    STRATEGY= 1      ERROR MATRIX ACCURATE
  EXT PARAMETER                                   STEP         FIRST
  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
   1  p0           6.81725e-01   4.37173e-01   2.40425e-05   8.08231e-04
   2  p1           1.46084e+00   9.36798e-01   5.15197e-05   3.78774e-04
```

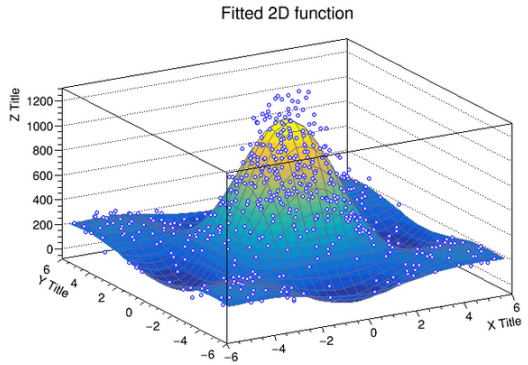
Configure the canvas for plotting the result.

In [4]:
```
TCanvas c1;
f2.SetLineWidth(1);
f2.SetLineColor(kBlue - 5);
f2.Draw("Surf1");

auto Xaxis = f2.GetXaxis(); Xaxis->SetTitle("X Title"); Xaxis->SetTitleOffset(1.5);
auto Yaxis = f2.GetYaxis(); Yaxis->SetTitle("Y Title"); Yaxis->SetTitleOffset(1.5);
auto Zaxis = f2.GetZaxis(); Zaxis->SetTitle("Z Title"); Zaxis->SetTitleOffset(1.5);
dte.Draw("P0 Same");
```

Display the 2D graph in the notebook.

In [5]:
```
c1.Draw();
```



Access TTree in Python using PyROOT and fill a histogram

First import the ROOT Python module.

In [1]:
```
import ROOT
%jsroot on
```

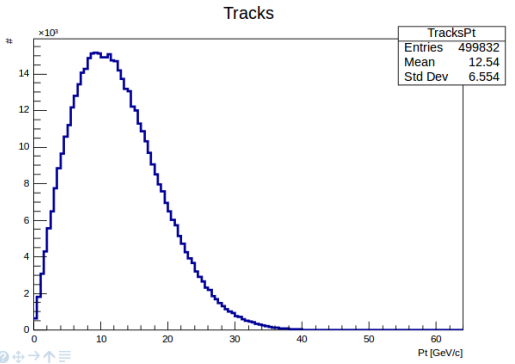
```
Welcome to JupyROOT 6.07/07
```

Open a file which is located on the web. No type is to be specified for "f".

In [3]:
```
f = ROOT.TFile.Open("http://indico.cern.ch/event/395198/material/0/0.root");
```

Loop over the TTree called "events" in the file. It is accessed with the dot operator. Same holds for the access to the branches: no need to set them up - they are just accessed by name, again with the dot operator.

In [4]:
```
h = ROOT.TH1F("TracksPt","Tracks;Pt [GeV/c];#",128,0,64)
for event in f.events:
    for track in event.tracks:
        h.Fill(track.Pt())
c = ROOT.TCanvas()
h.Draw()
c.Draw()
```



Using Root with Jupyter

If you have a preferred language, but want to include some colleagues' code snipped, C++ and Python can be mixed in the same Notebook

Interleave Python with C++: the %%cpp magic

In [1]:
```
import ROOT
```

```
Welcome to JupyROOT 6.07/03
```

Thanks to its interpreter and type system, entities such as functions, classes and variables, created in a C++ cell, can be accessed from within Python.

In [2]:
```
%%cpp
class A {
public:
    A() { cout << "Constructor of A!" << endl; }
};
```

In [3]:
```
a = ROOT.A()
```

```
Constructor of A!
```

%%python
also available in C++ notebooks

SWAN to do Analysis via the Web

Ready-to-use system (provided by CERN at https://swan.cern.ch) for performing data analysis with all the software on all the data we need using a web-browser

SWAN workflow

Analysis code

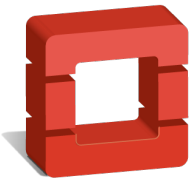
- Done using jupyter notebooks in the browser

Cloud will be used for

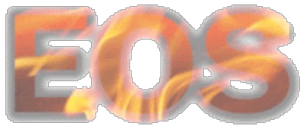
- Input data
- Computation
- Results

SWAN relies on CERN infrastructure

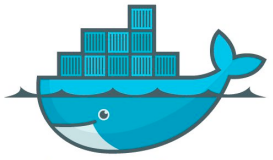
- You need a CERN account
- Machines in the OpenStack cloud
- Software distribution: CVMFS
- Workplace: EOS, CERNBox
  - *your new home directory when AFS goes away*
- User and experiments data available through the cloud



openstack™



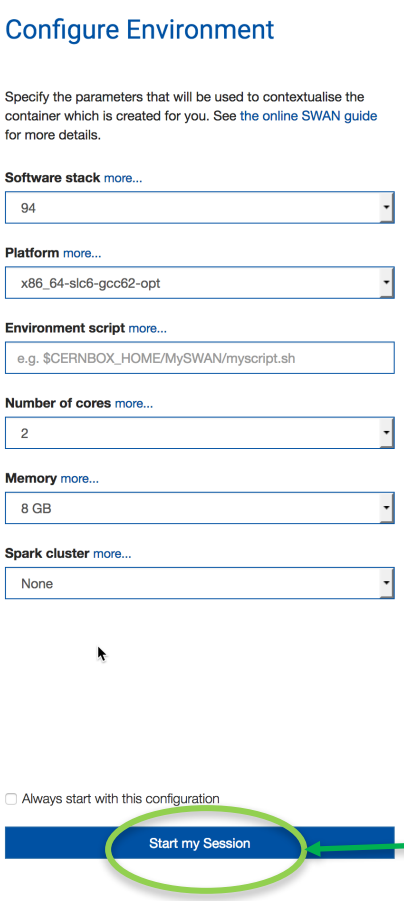
CERNBox



docker

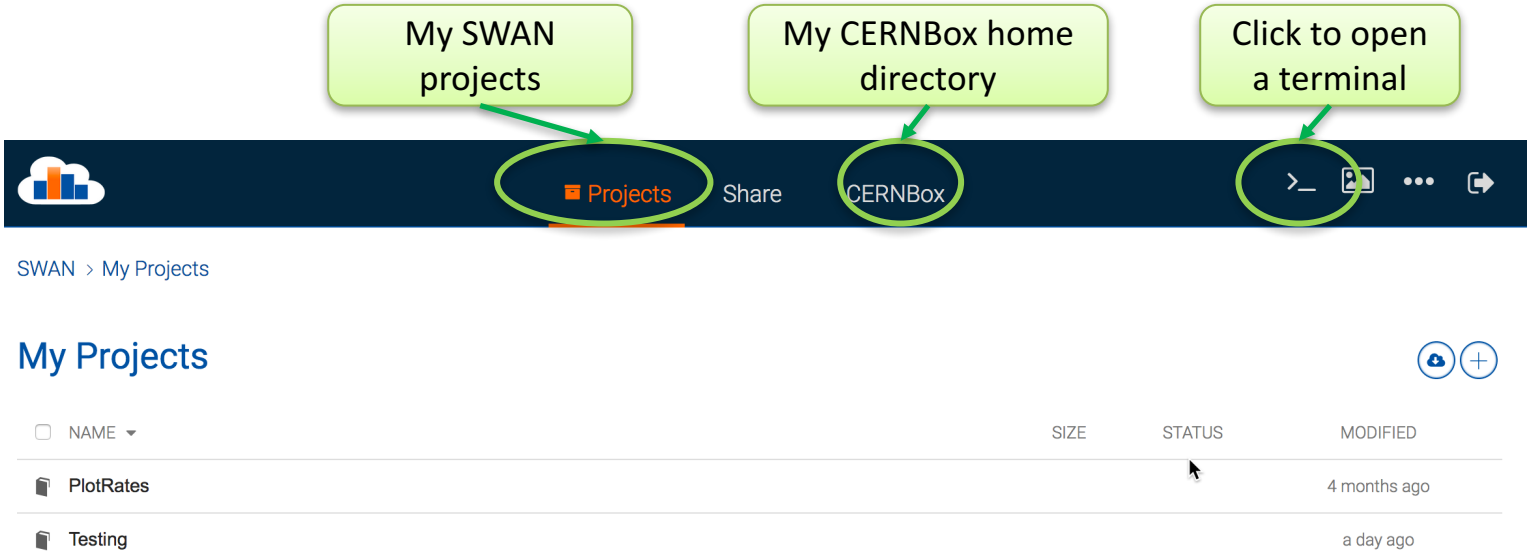
Let's get started – opening SWAN

In this tutorial I am using Jupyter and the Swan service: https://swan.cern.ch



Use default settings and start a session

Downloading the example

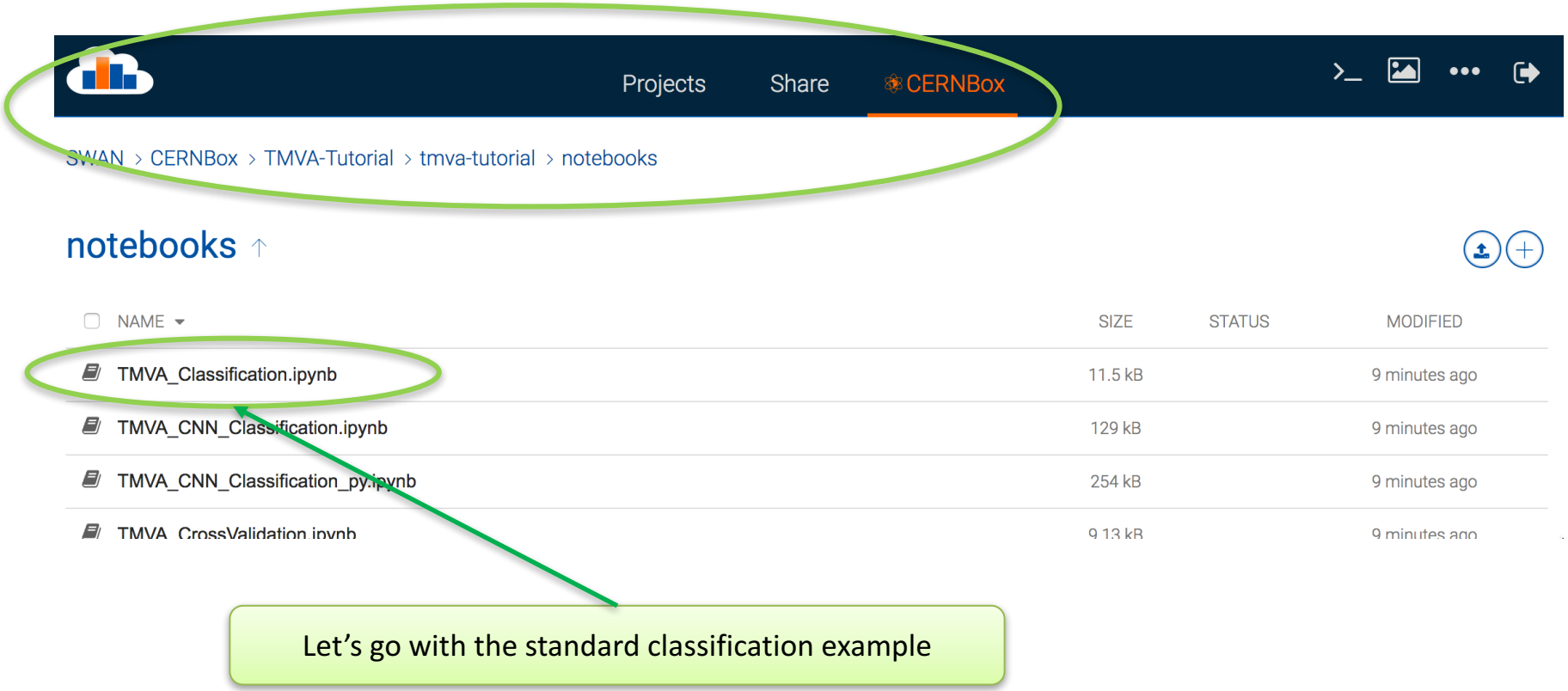


Let's create a directory and fetch the tutorial from git
git clone https://github.com/lmoneta/tmva-tutorial.git

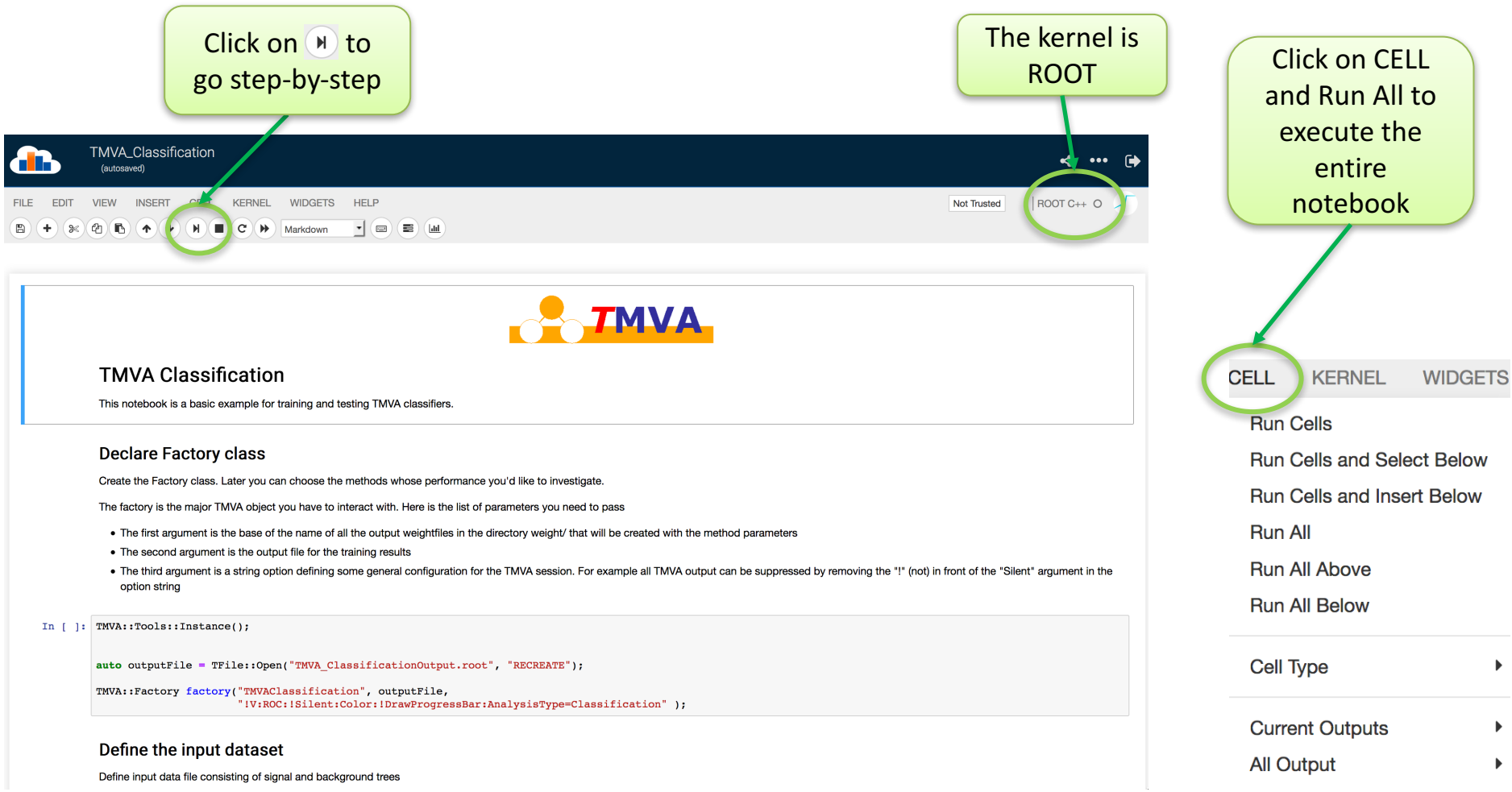
```
bash-4.1$ pwd
/eos/user/s/stelzer
bash-4.1$ mkdir TMVA-Tutorial
bash-4.1$ cd TMVA-Tutorial/
bash-4.1$ git clone https://github.com/lmoneta/tmva-tutorial.git
Initialized empty Git repository in /eos/home-s/stelzer/TMVA-Tutorial/tmva-tutorial/.git/
remote: Enumerating objects: 53, done.
remote: Counting objects: 100% (53/53), done.
remote: Compressing objects: 100% (38/38), done.
remote: Total 230 (delta 29), reused 36 (delta 14), pack-reused 177
Receiving objects: 100% (230/230), 24.15 MiB | 15.25 MiB/s, done.
Resolving deltas: 100% (122/122), done.
bash-4.1$ ls tmva-tutorial/notebooks/
Higgs_data.root             TMVA_CNN_Classification.ipynb     TMVAGuiROC.ipynb                  TMVA_Reader.ipynb                TMVA_VariableImportance.C
images_data.root            TMVA_CNN_Classification_py.ipynb  TMVA_Higgs_Classification.C       TMVA_Reader_ManyMethods.ipynb    TMVA_VariableImportance.ipynb
inputdata_regression.root   TMVA_CrossValidation.ipynb       TMVA_Higgs_Classification.ipynb   TMVA_Reader_ManyMethods_py.ipynb
TMVA_Classification.C       TMVAGuiPlots.ipynb                TMVA_Higgs_Classification_py.ipynb TMVA_Regression.C
TMVA_Classification.ipynb   TMVAGuiRegression.ipynb           tmva_logo.gif                     TMVA_Regression.ipynb
bash-4.1$
```

Now we start the example

The notebooks are located in your CERNBox under › TMVA-Tutorial › tmva-tutorial › notebooks



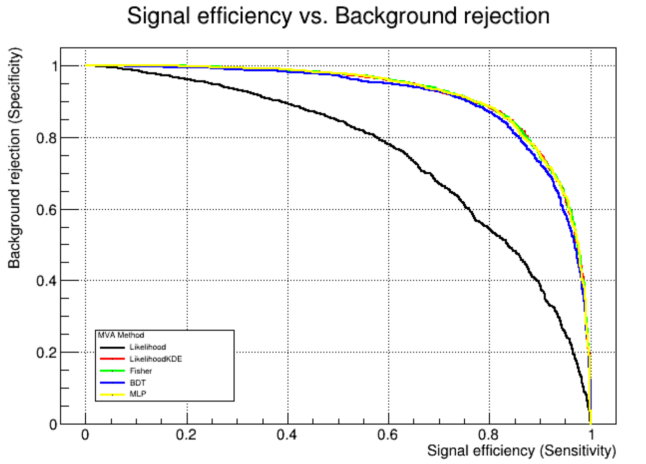
TMVA-Classification Example



Results

The result of our example should look like this

```
In [13]: auto c1 = factory.GetROCCurve(loader);
         c1->Draw();
```



Create were the following files

- TMVA_ClassificationOutput.root: contains some classifiers definition and the test results
- tmva-tutorial/notebooks/dataset/weights/: contains the classifier definitions

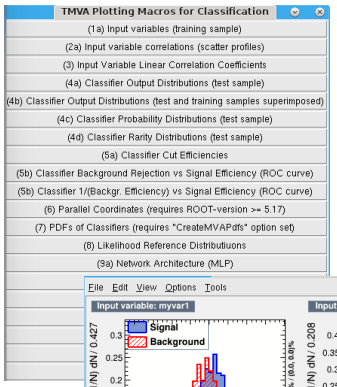
```
bash-4.1$ ls tmva-tutorial/notebooks/
dataset                      TMVA_Classification.ipynb          TMVAGuiPlots.ipynb                TMVA_Higgs_Classification_py.ipynb  TMVA_Regression.C
Higgs_data.root              TMVA_ClassificationOutput.root     TMVAGuiRegression.ipynb           tmva_logo.gif                       TMVA_Regression.ipynb
images_data.root             TMVA_CNN_Classification.ipynb      TMVAGuiROC.ipynb                  TMVA_Reader.ipynb                   TMVA_VariableImportance.C
inputdata_regression.root    TMVA_CNN_Classification_py.ipynb   TMVA_Higgs_Classification.C       TMVA_Reader_ManyMethods.ipynb       TMVA_VariableImportance.ipynb
TMVA_Classification.C        TMVA_CrossValidation.ipynb         TMVA_Higgs_Classification.ipynb   TMVA_Reader_ManyMethods_py.ipynb
bash-4.1$ ls tmva-tutorial/notebooks/dataset/weights/
TMVAClassification_BDT.class.C       TMVAClassification_Fisher.weights.xml     TMVAClassification_LikelihoodKDE.weights.xml  TMVAClassification_MLP.weights.xml
TMVAClassification_BDT.weights.xml   TMVAClassification_Likelihood.class.C     TMVAClassification_Likelihood.weights.xml
TMVAClassification_Fisher.class.C    TMVAClassification_LikelihoodKDE.class.C  TMVAClassification_MLP.class.C
bash-4.1$
```

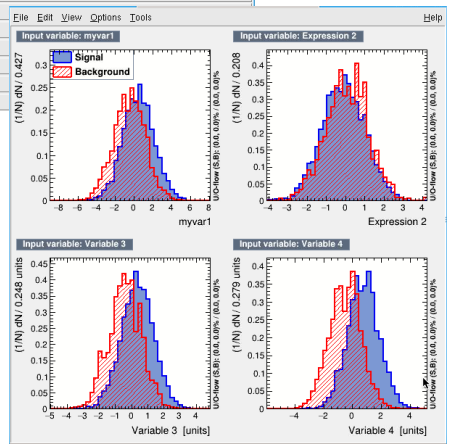
TMVA GUI

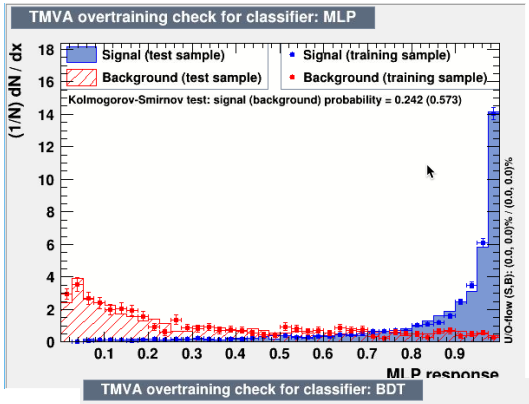
It is quite out of date now, but …

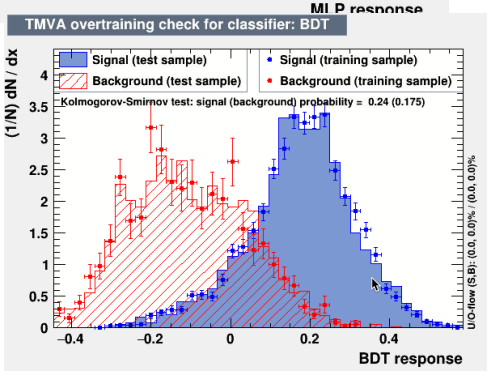
Log into lxplus (I use x2go and a private machine from OpenStack) and then

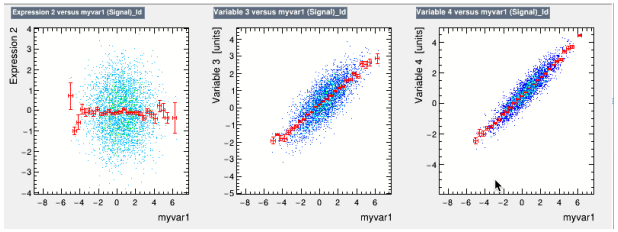
- cd /eos/home-s/stelzer/TMVA-Tutorial/tmva-tutorial/notebooks
- setupATLAS
- lsetup "root 6.14.04-x86_64-slc6-gcc62-opt"
- root -e 'TMVA::TMVAGUI g("TMVA_ClassificationOutput.root")'

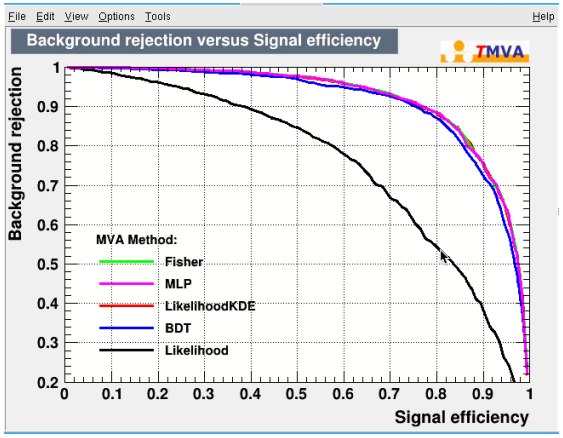












Summary

Very active development happening in TMVA

- several new features released recently
- and even more expected in a near future
- thanks to many student contributions (e.g. from Google Summer of Code)

Strong competition, but hopefully still good reasons for continuing using TMVA !

Feedback from users essential

- best way to contribute is with Pull Request in GitHub https://github.com/root-project/root
- ROOT Forum for user support with a category dedicated to TMVA https://root.cern/forum
- JIRA for reporting ROOT bugs: https://sft.its.cern.ch/jira
- or just contact TMVA developers directly for any questions or issues

Inter-experimental LHC Machine Learning working group

- Exchange of HEP-ML expertise and experience among LHC experiments
- ML Forum
- ML software development and maintenance
- Exchange between HEP and ML communities
- Education (Tutorials)
- Free for sign-up (http://iml.web.cern.ch/forum)

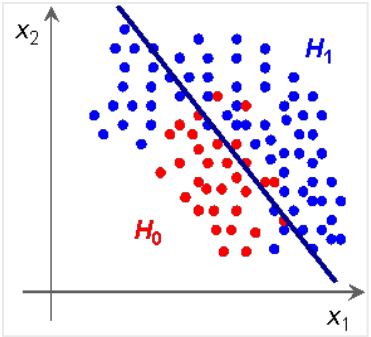
Thank you for your attention

BACKUP

Rectangular Cut Optimization

Intuitive and simple: rectangular volumes in variable space

$$y_{\text{cut}}(i) \in \{0,1\} = \bigcap_{v \in \{\text{variables}\}} \{x_v(i) \subset [x_{v,\min}, x_{v,\max}]\}$$



Technical challenge: cut optimization:

- *MINUIT fit*: (simplex) was found not to be reliable
- *Monte Carlo sampling*:
  - *random scanning of parameter space*
  - *inefficient for large number of input variables*
- *Genetic algorithm*: preferred method
  - *Samples of cut-sets (a population) are evaluated, the fittest individuals are cross-bred (including mutation) to create a new generation*
- The Genetic Algorithm can also be used as standalone optimizer, outside the TMVA framework
- *Simulated annealing*: still need to optimize its performance
  - *Simulated slow cooling of metal, introduce temperature dependent perturbation probability to recover from local minima*

Cuts usually benefit from prior decorrelation of cut variables

Projective Likelihood Estimator (PDE)

Probability density estimators for each input variable combined in likelihood estimator

$$y_{\mathrm{Lh}}(i) = \frac{L_S(i)}{L_S(i) + L_B(i)}, \quad L_{S/B}(i) = \prod_{v \in \{\text{variables}\}} p_{S/B,v}\left(x_v(i)\right)$$

Reference PDF's

Optimal MVA approach, **if** variables are uncorrelated

- In practice rarely the case, solution: **de-correlate input** or use **different method**

Reference PDFs are automatically generated from training data:

- Histograms (counting), splines (order 2,3,5), or unbinned kernel estimator

Output of likelihood estimator often strongly peaked at 0 and 1. To ease output parameterization TMVA applies inverse Fermi transformation.

$$y'_{\mathrm{lh}}(i) = -\tau^{-1} \ln\left(y_{\mathrm{lh}}^{-1}(i) - 1\right)$$

Estimating PDF Kernels

Technical challenge: how to estimate the PDF shapes

3 ways:

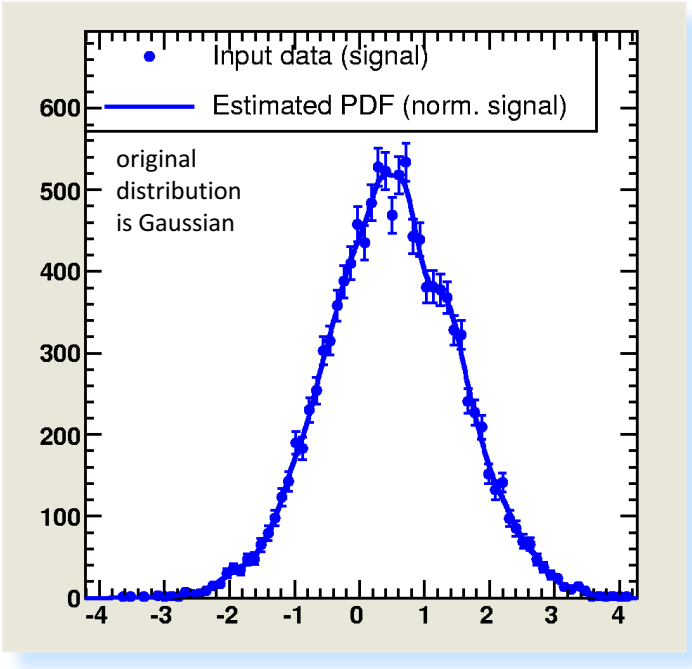
parametric fitting (function) — Difficult to automate for arbitrary PDFs

nonparametric fitting — Easy to automate, can create artefacts/suppress information

event counting — Automatic, unbiased, but suboptimal

We have chosen to implement nonparametric fitting in *T*MVA

- Binned shape interpolation using spline functions (orders: 1, 2, 3, 5)
- Unbinned kernel density estimation (KDE) with Gaussian smearing
- *T*MVA performs automatic validation of goodness-of-fit



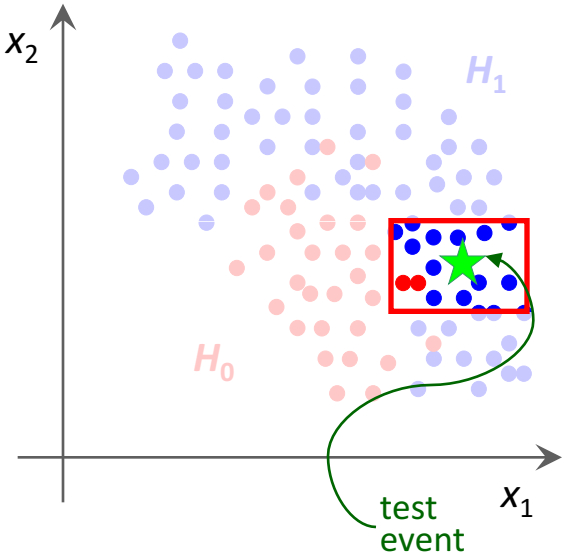
Multidimensional PDE

Extension of the one-dimensional PDE approach to *n* dimensions

- Counts signal and background reference events (training sample) in the vicinity V of the test event

Carli-Koblitz, NIM A501, 576 (2003)

$$y_{\mathrm{PDERS}}(i) = \frac{n_S(i)/N_S}{n_S(i)/N_S + n_B(i)/N_B}, \quad n_{S/B}(i) = \prod_{e \in \left\{ \begin{array}{l} \text{reference S/B} \\ \text{events in V(i)} \end{array} \right\}} w_e$$



Volume V definition:

- **Size**: fixed (defined by the data: % of Max-Min or RMS) or adaptive (define by number of events in search volume)
- **Shape**: box or ellipsoid

Improve y_{PDERS} estimate within V by using various *n*-D kernel estimators (function of the (normalized) distance between test- and reference events)

Practical challenges:

- Need very large training sample (**curse of dimensionality** of kernel based methods)
- No training, slow evaluation.
  - *Search speed improvement with kd-tree event sorting*

Fisher's Linear Discriminant Analysis

Well-known, simple and elegant MVA method

- Fisher analysis determines an axis in the input variable hyperspace ($F_1,\ldots,F_n$, such that a projection of events onto this axis separates signal and background as much as possible

Projection:

$$y_{\mathrm{Fi}}(i) = F_0 + \sum_{v\in\{\mathrm{variables}\}} F_v x_v(i)$$

W: sum of S and B covariance matrices

Fisher Coefficients:

$$F_v = \frac{\sqrt{N_S + N_B}}{N_S N_B} \sum_{l\in\{\mathrm{variables}\}} W_{vl}^{-1}\left(\bar{x}_{S,l} - \bar{x}_{B,l}\right), \quad \mathbf{W} = \mathbf{C}_S + \mathbf{C}_B$$

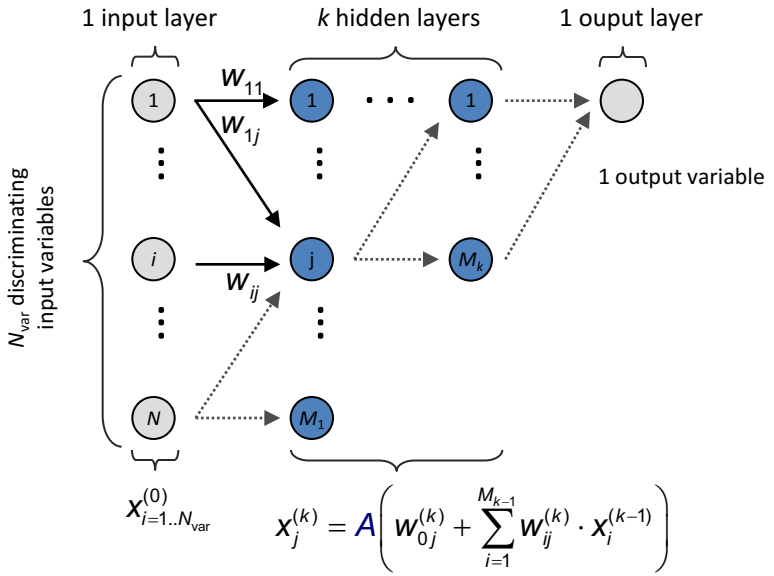
New classifier: **Function discriminant analysis (FDA)**

Fit any user-defined function of input variables requiring that signal events return →1 and background →0

- Parameter fitting: Genetics Alg., MINUIT, MC and combinations
- Easy reproduction of Fisher result, but can add nonlinearities
- Very transparent discriminator

Artificial Neural Network (ANN)

Multilayer perceptron: fully connected, feed forward, k hidden layers

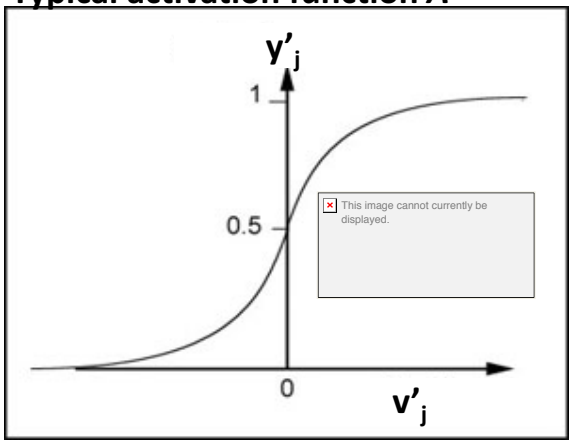


ANNs are non-linear discriminants

- Non linearity from activation function. (Fisher is an ANN with linear activation function)

Weierstrass theorem:
MLP can approximate every continuous function to arbitrary precision with just one layer and infinite number of nodes

Typical activation function *A*



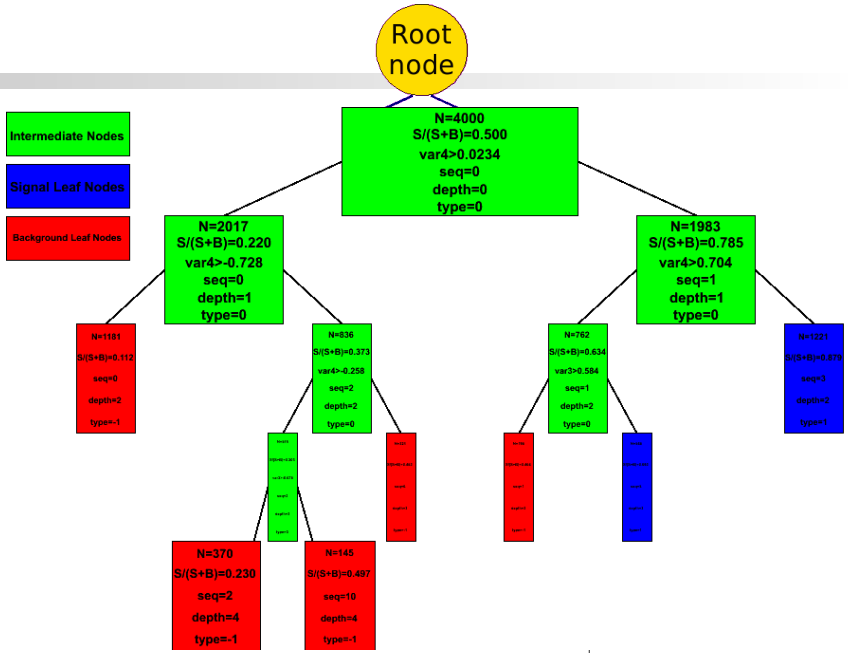
Training: back-propagation method

- Randomly feed signal and background events to MLP and compare the desired output {0,1} with the received output (0,1): ε = d - r
- Correct weights, depending on ε and learning rate η

Boosted Decision Trees (BDT)

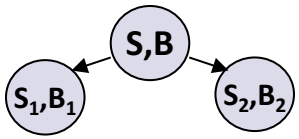
A DT is a series of cuts that split sample set into ever smaller sets, leafs are assigned either S or B status

- Classifies events by following a sequence of cuts depending on the events variable content until a S or B leaf



Growing

Each split try to maximizing gain in separation (Gini-index)



$$Gini = \frac{S_1 B_1}{S_1 + B_1} + \frac{S_2 B_2}{S_2 + B_2}$$

DT dimensionally robust and easy to understand but not powerful

1. Pruning

Bottom-up pruning of a decision tree

Protect from overtraining by removing statistically insignificant nodes

2. Boosting (Adaboost)

- Increase the weight of incorrectly identified events → build new DT
- Final classifier: 'forest' of DT's linearly combined
- Large coefficient for DT with small misclassification
- Improved performance and stability

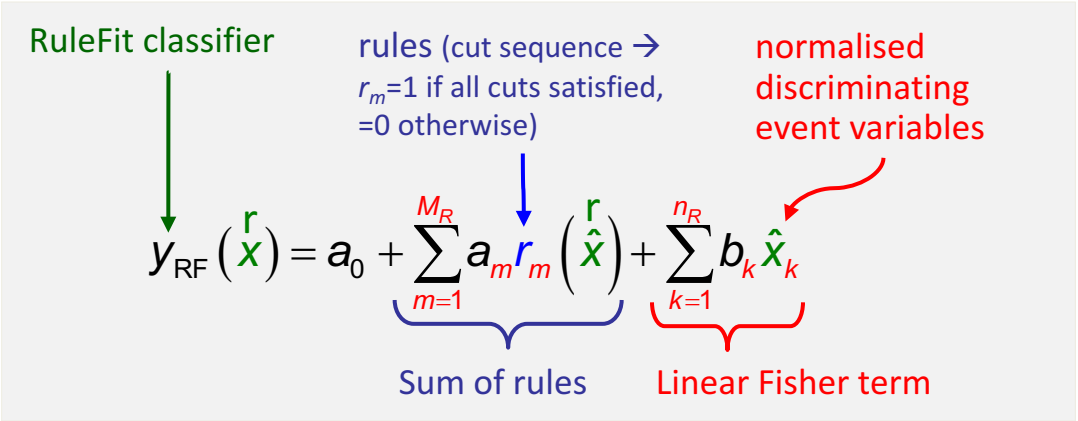
BDT requires only little tuning to achieve good performance

Predictive Learning via Rule Ensembles (RuleFit)

Following RuleFit approach by Friedman-Popescu

Friedman-Popescu, Tech Rep, Stat. Dpt, Stanford U., 2003

Model is linear combination of rules, where a rule is a sequence of cuts defining a region in the input parameter space



The problem to solve is

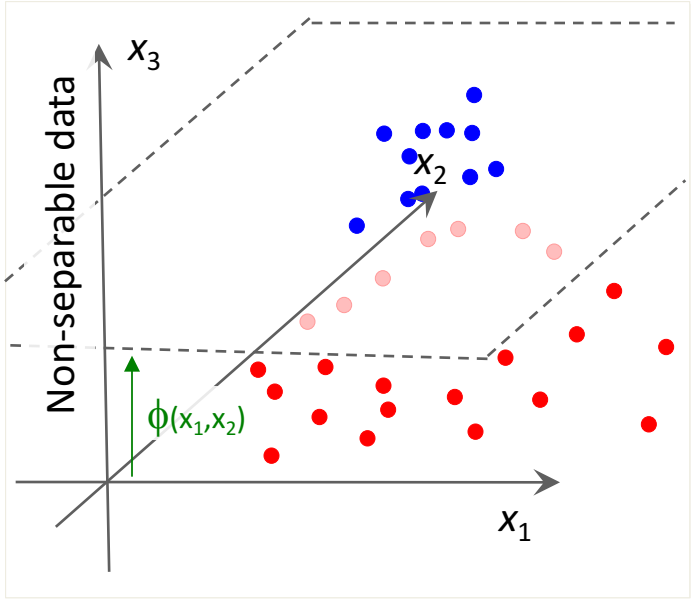
- Create rule ensemble: use forest of decision trees either from a BDT, or from a random forest generator (TMVA)
- Fit coefficients a_m, b_k, minimizing risk of misclassification (Friedman et al.)

Pruning removes topologically equal rules" (same variables in cut sequence)

Support Vector Machine

Find hyperplane that between linearly separable signal and background (1962)

- Best separation: maximum distance (margin) between closest events (*support*) to hyperplane
- Wrongly classified events add extra term to cost-function which is minimized



- Non-linear cases:
  - Transform variables into higher dimensional space where again a linear boundary (hyperplane) can separate the data (only mid-'90)
  - Explicit transformation form not required, cost function depends on scalar product between events: use Kernel Functions to approximate scalar products between transformed vectors in the higher dimensional space
  - Choose Kernel and fit the hyperplane using the linear techniques developed above
  - Available Kernels: Gaussian, Polynomial, Sigmoid

Chosing the WP

From the ROC of the classifier chose the working point

- need expectation for *S* and *B*

- Cross section measurement: maximum of $S/\sqrt{(S+B)}$ or equiv. $\sqrt{(\varepsilon \cdot p)}$
- Discovery of a signal: maximum of $S/\sqrt{(B)}$
- Precision measurement: high purity (*p*)
- Trigger selection: high efficiency (ε)