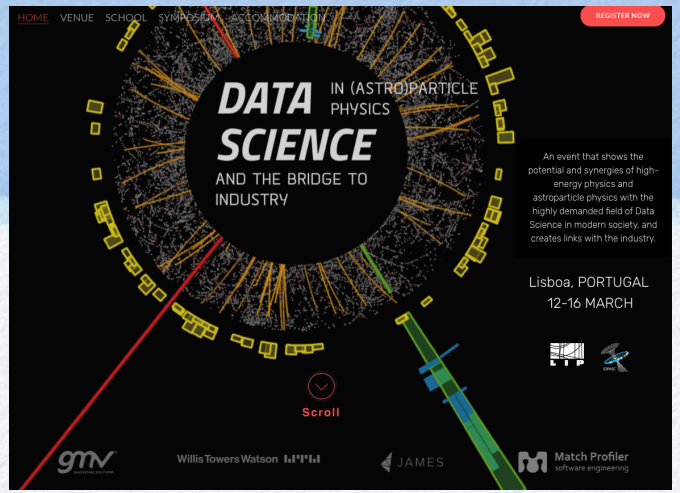
Machine Learning Software

ROOT/TMVA

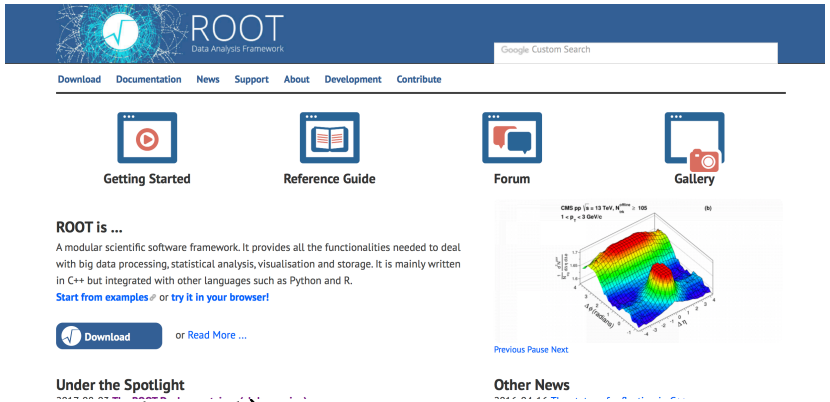


LIP Data Science School / 12-14 March 2018

ROOT

ROOT is a software toolkit which provides building blocks for:

- Data processing
- Data analysis
- Data visualisation
- Data storage



ROOT is written mainly in C++ (C++11 standard)

- Bindings for Python are provided.

http://root.cern.ch

Adopted in High Energy Physics and other sciences (but also industry)

- ~250 PetaBytes of data in ROOT format on the LHC Computing Grid
- Fits and parameters' estimations for discoveries (e.g. the Higgs)
- Thousands of ROOT plots in scientific publications

TMVA

- ROOT Machine Learning tools are provided in the package TMVA (Toolkit for MultiVariate Analysis)
- Provides a set of algorithms for standard HEP usage
- Used in LHC experiment production and in several analysis (e.g. Higgs studies)
- Easy interface for beginners, powerful for experts
- Several active contributors and several features added recently (e.g. deep learning)

TMVA

- TMVA is not only a collection of multi-variate methods. It is a
  - common interface to different methods
    - common interface for classification and regression
  - easy training and testing of different methods on the same dataset
    - consistent evaluation and comparison
    - same data pre-processing
  - several tools provided for pre-processing
  - embedded in ROOT
  - complete and understandable users guide

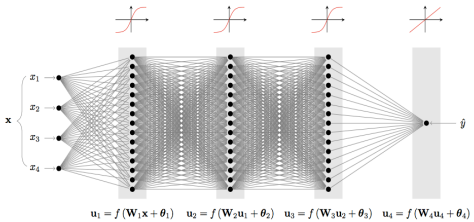
TMVA Methods

The available methods are:

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

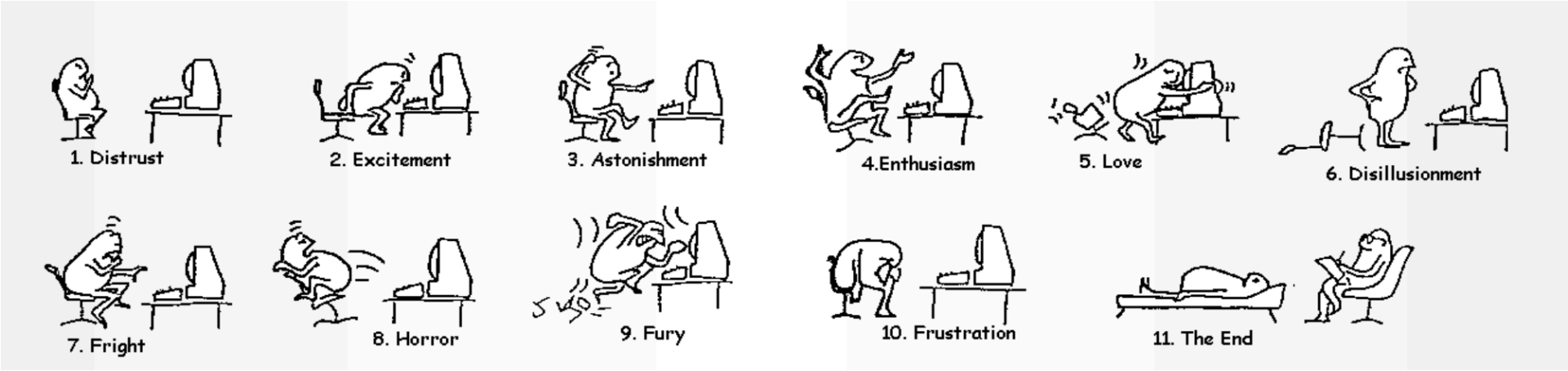
New Features

New features added since 2016:



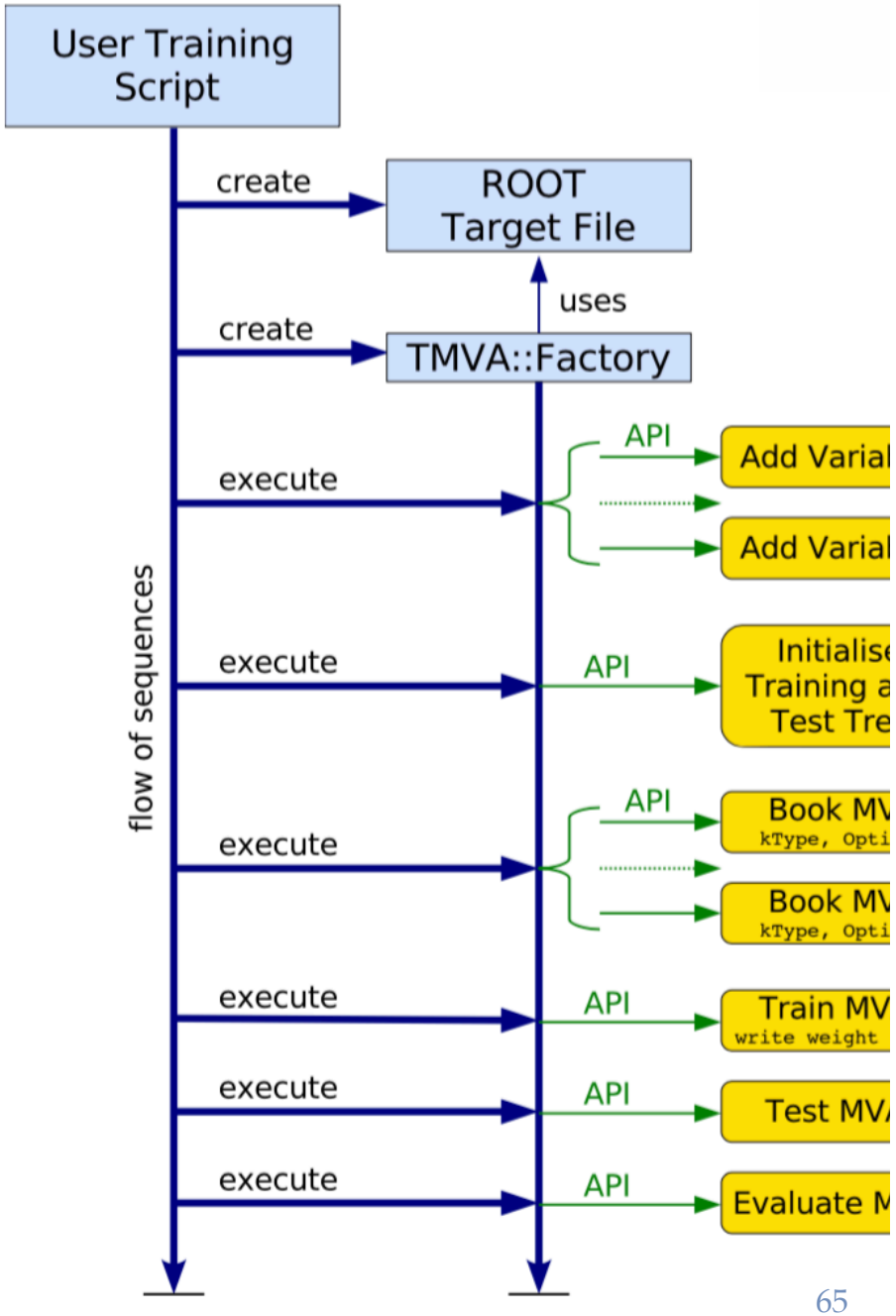
- Deep Learning
  - support for parallel training on CPU and GPU (with CUDA and OpenCL)
- Cross Validation and Hyper-parameter optimisation
- Improved loss functions for regression
- Interactive training and visualization for Jupyter notebooks
- new pre-processing features (variance threshold)

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

Workflow in TMVA

- 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



TMVA Custumizations and Features

TMVA supports:

- ROOT Tree input data (or ASCII, e.g. csv)
  - HSF support might come soon
- pre-selection cuts on input data
- event weights (negative weights for some methods)
- various method for splitting training/test samples
- k-fold cross-validation
- support variable importance
- hyper-parameter optimisations

TMVA Session

```
void TMVAnalysis( )
{
  TFile* outputFile = TFile::Open( "TMVA.root", "RECREATE" );

  TMVA::Factory *factory = new TMVA::Factory( "MVAnalysis", outputFile,"!V");

  TFile *input = TFile::Open("tmva_example.root");

  factory->AddVariable("var1+var2", 'F');
  factory->AddVariable("var1-var2", 'F');   //factory->AddTarget("tarval", 'F');

  factory->AddSignalTree ( (TTree*)input->Get("TreeS"), 1.0 );
  factory->AddBackgroundTree ( (TTree*)input->Get("TreeB"), 1.0 );
  //factory->AddRegressionTree ( (TTree*)input->Get("regTree"), 1.0 );
  factory->PrepareTrainingAndTestTree( "", "",
  "nTrain_Signal=200:nTrain_Background=200:nTest_Signal=200:nTest_Background=200:!V" );

  factory->BookMethod( TMVA::Types::kLikelihood, "Likelihood",
  "!V:!TransformOutput:Spline=2:NSmooth=5:NAvEvtPerBin=50" );
  factory->BookMethod( TMVA::Types::kMLP, "MLP",
  "!V:NCycles=200:HiddenLayers=N+1,N:TestRate=5" );

  factory->TrainAllMethods();  // factory->TrainAllMethodsForRegression();
  factory->TestAllMethods();
  factory->EvaluateAllMethods();
  outputFile->Close();
  delete factory;
}
```

Create Factory

Add variables/ targets

Initialize Trees

Book MVA methods

Train, test and evaluate

We will see better with a real example (e.g. TMVAClassification.C tutorial)

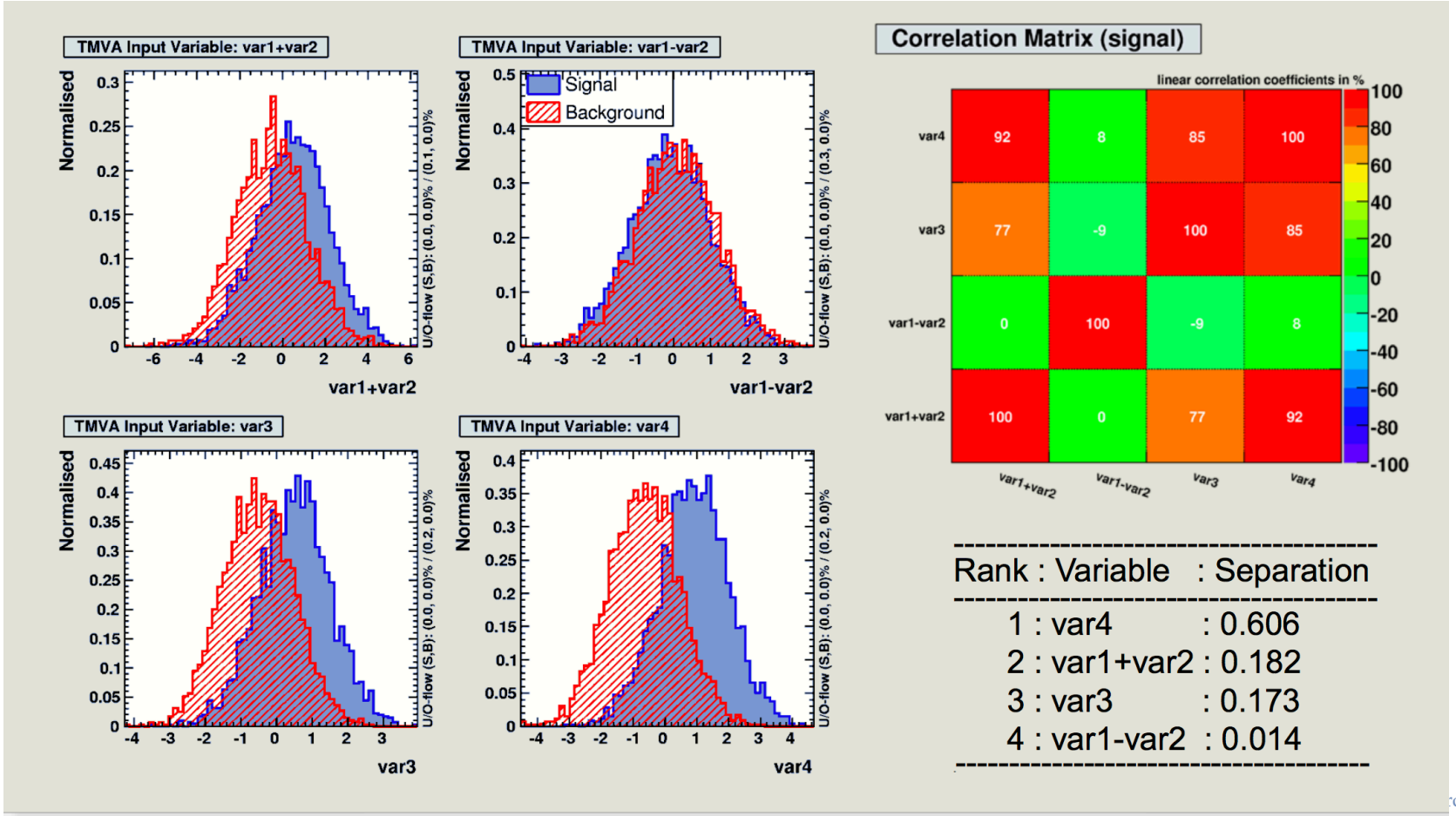
[E. v. Toerne]

TMVA Toy Example

4 Gaussian variable with linear correlations

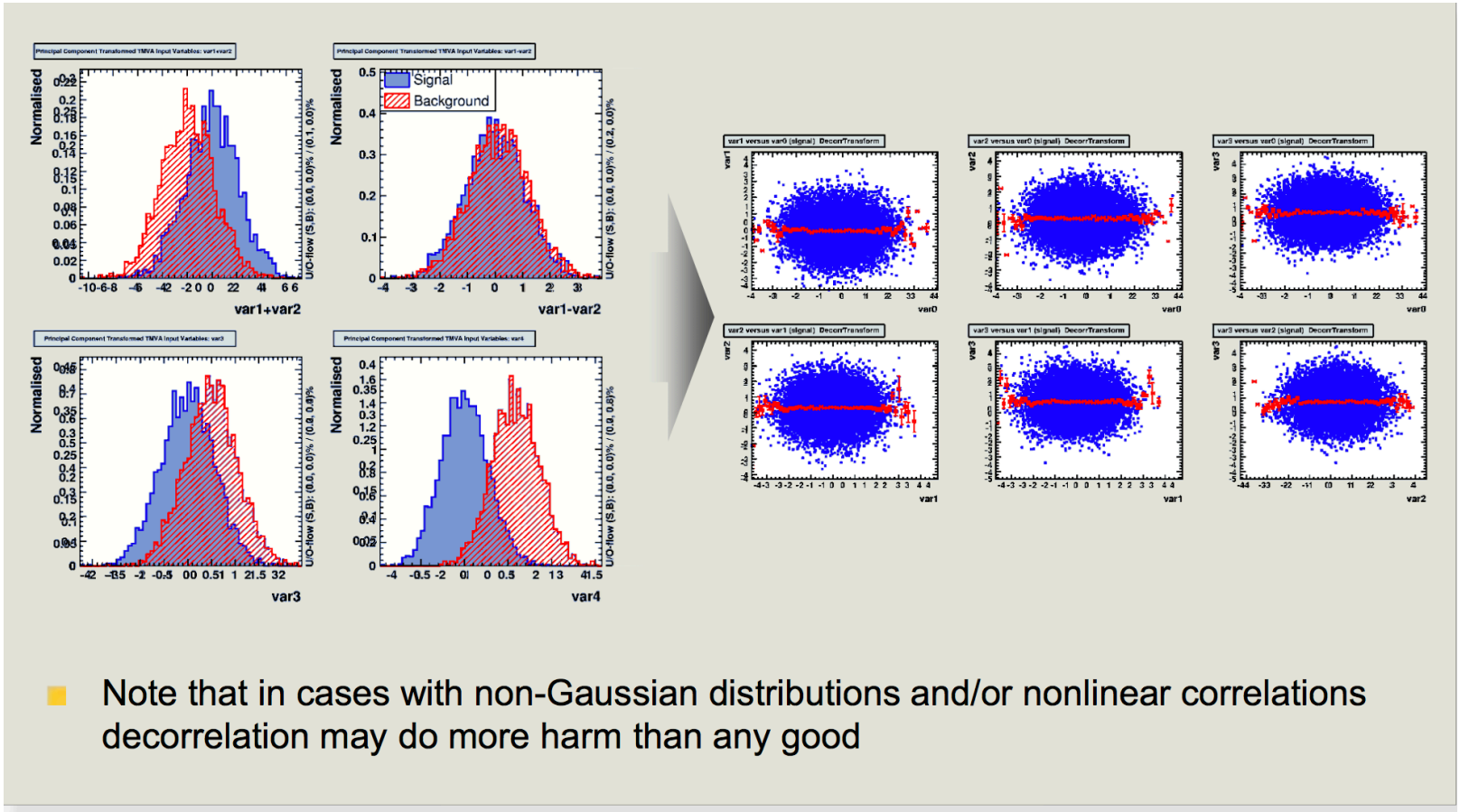
$$\{x_1 = v_1 + v_2,\ x_2 = v_1 - v_2,\ x_3 = v_3,\ x_4 = v_4\}$$

where $\{v_1, ..v_4\}$ are normal variables



Pre-processing of the Input Variables

- Example: decorrelation of variable before training can be useful



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

Several others pre-processing available (see Users Guide)

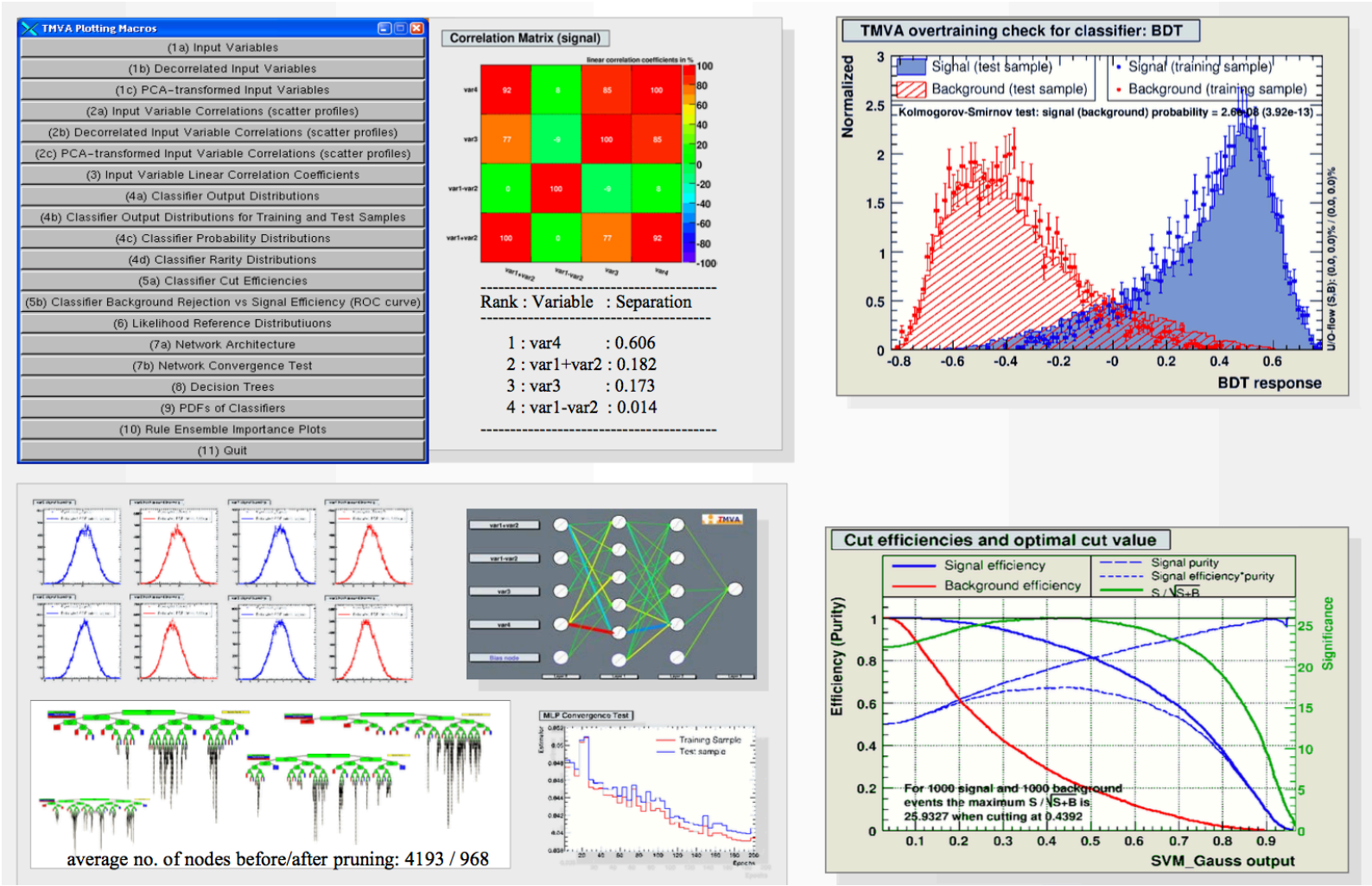
Available Preprocessing

This is the list of available pre-processing in TMVA

- Normalization
- Decorrelation (using Cholesky decomposition)
- Principal Component Analysis
- Uniformization
- Gaussianization

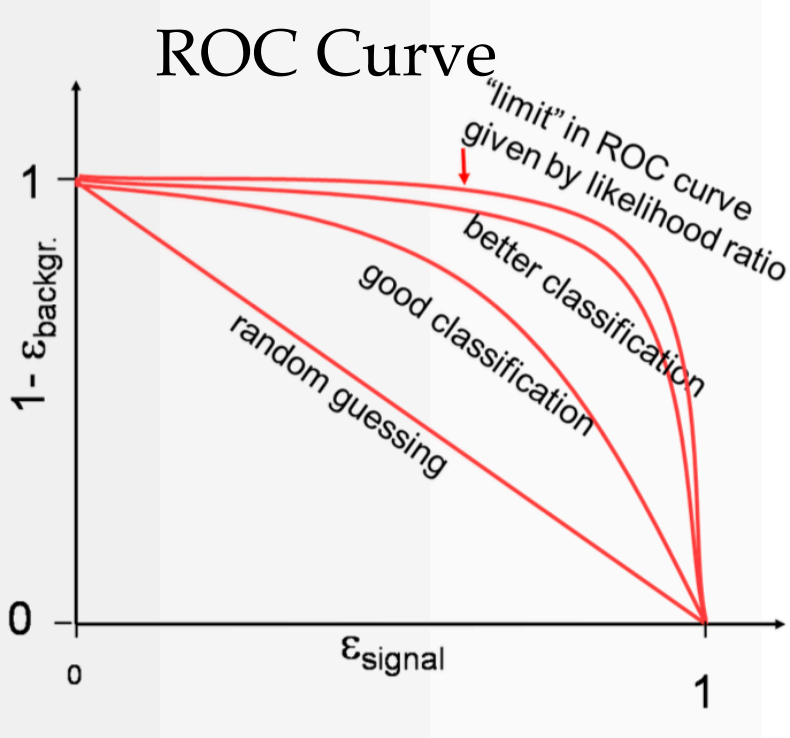
TMVA GUI

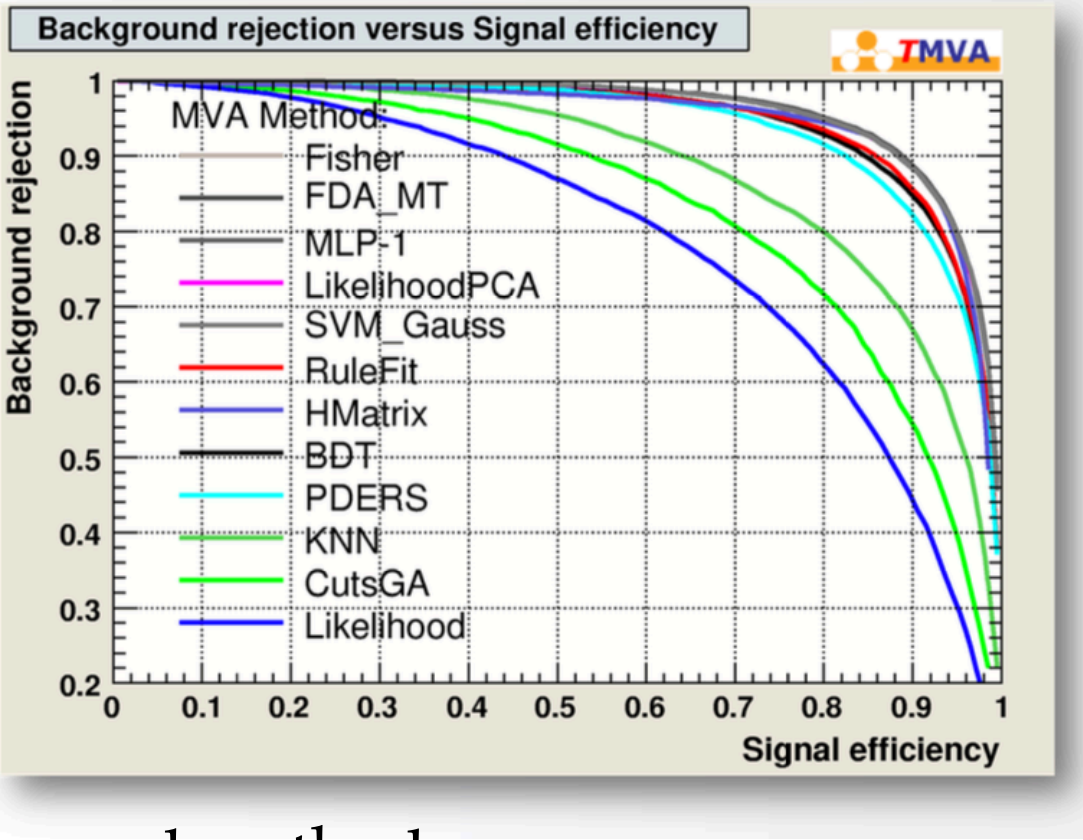
At the end of training + test phase TMVA produces an output file that can be examined with a special GUI (**TMVAGui**)



ROC Curve in TMVA

For example from GUI one can obtain a ROC curve for each method trained and tested on an independent data set

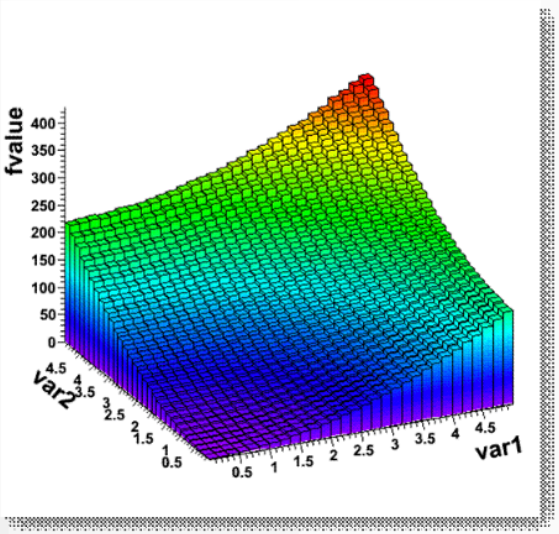


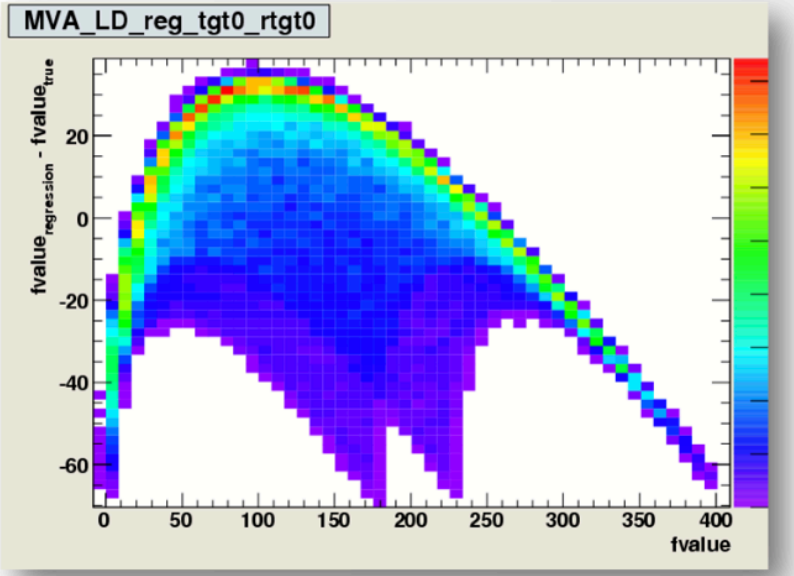


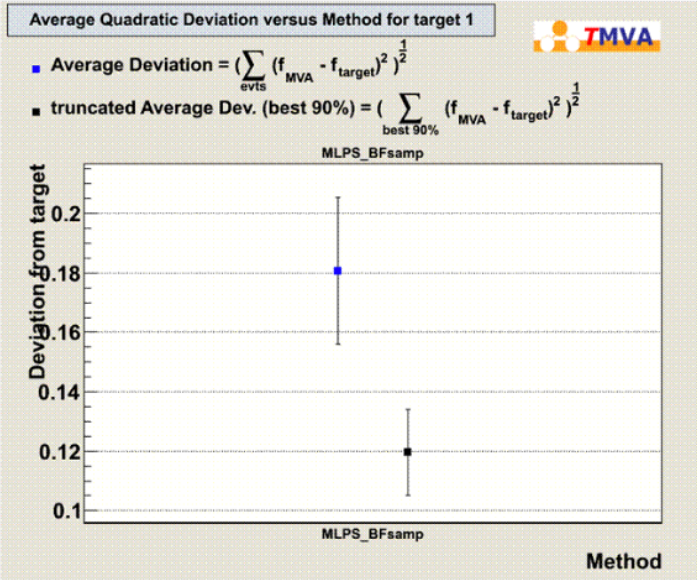
➜ Comparison of several methods

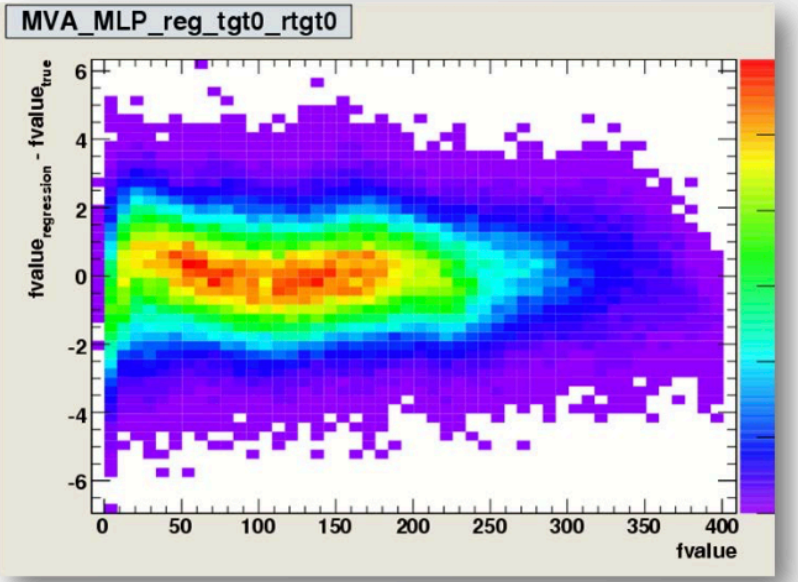
TMVA Regression GUI

A dedicated GUI exists for regression (**TMVARegGui**)



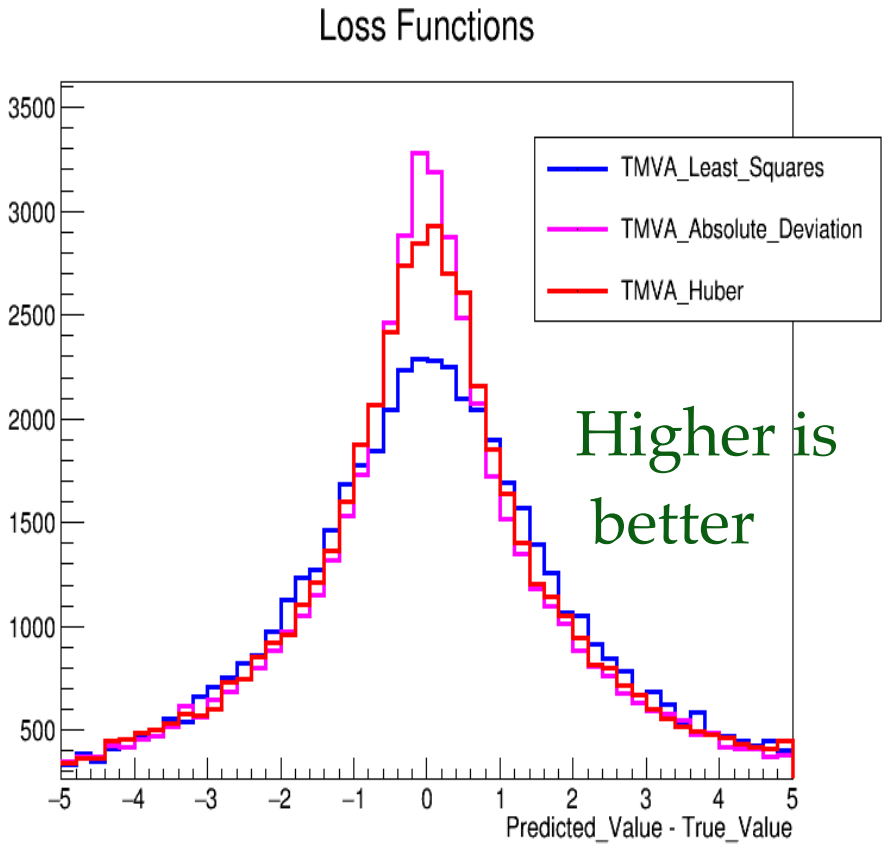






Regression in TMVA

- New Regression Features:
  - Loss function
    - Huber (default)
    - Least Squares
    - Absolute Deviation
    - Custom Function

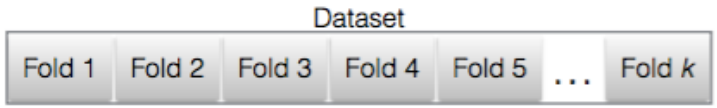


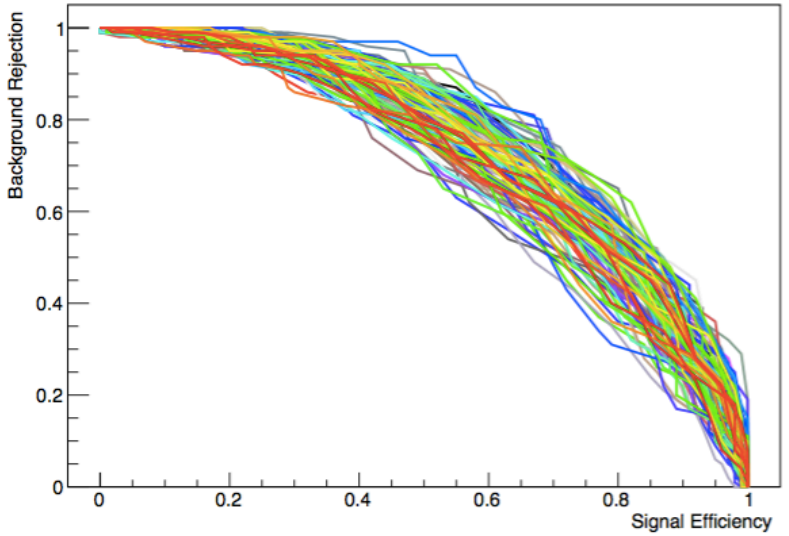
Important for regression performance

Cross Validation in TMVA

- TMVA supports k-fold cross-validation

k-fold cross-validation:



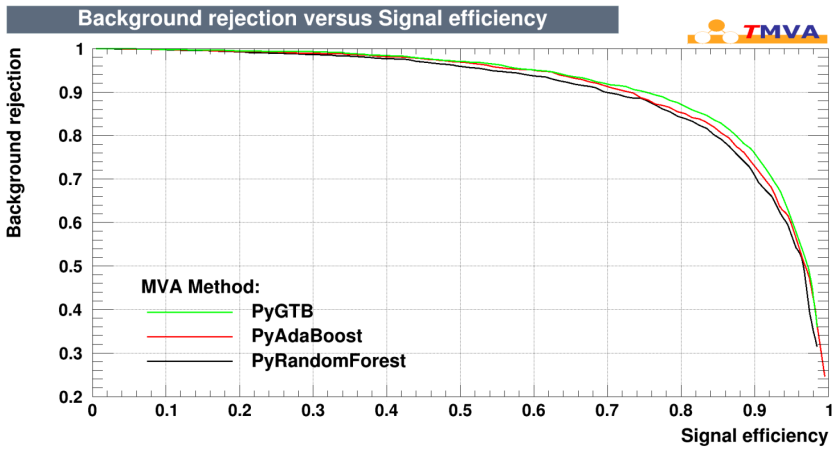


- Hyper-parameter tuning
  - find optimised parameters (BDT-SVM)
- Providing support for parallel execution
  - multi-process/multi-threads and on a cluster using Spark or MPI

TMVA Interfaces

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

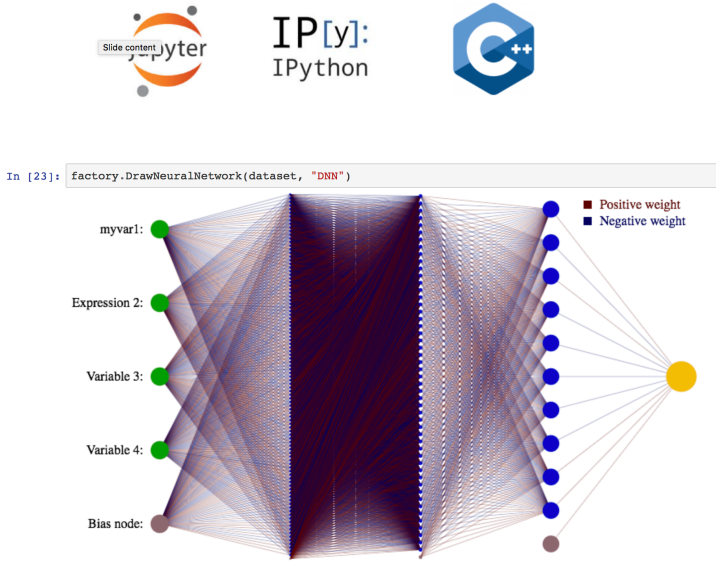
- RMVA: Interface to Machine Learning methods in R
  - c50, xgboost, RSNNS, e1071
    - see http://oproject.org/RMVA
- PYMVA: Python Interface
  - **skikit-learn** with RandomForest, Gradiend Tree Boost, Ada Boost)
    - see http://oproject.org/PYMVA
  - **Keras** (Theano + Tensorflow)
    - support model definition in Python
    - see https://indico.cern.ch/event/565647/contributions/2308668/attachments/1345527/2028480/29Sep2016_IML_keras.pdf
  - Input data are copied internally from TMVA to Numpy array

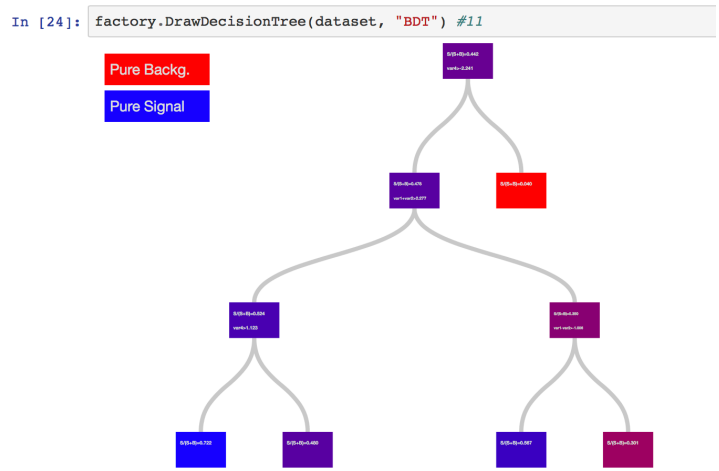


Jupyter Integration

New Python package for using TMVA in Jupyter notebook (jsmva)

- Improved Python API for TMVA functions
- Visualisation of BDT and DNN
- Enhanced output and plots (e.g. ROC plots)
- Improved interactivity (e.g. pause/resume/stop of training)
- see example in SWAN gallery
  https://swan.web.cern.ch/content/machine-learning





TMVA Tutorial

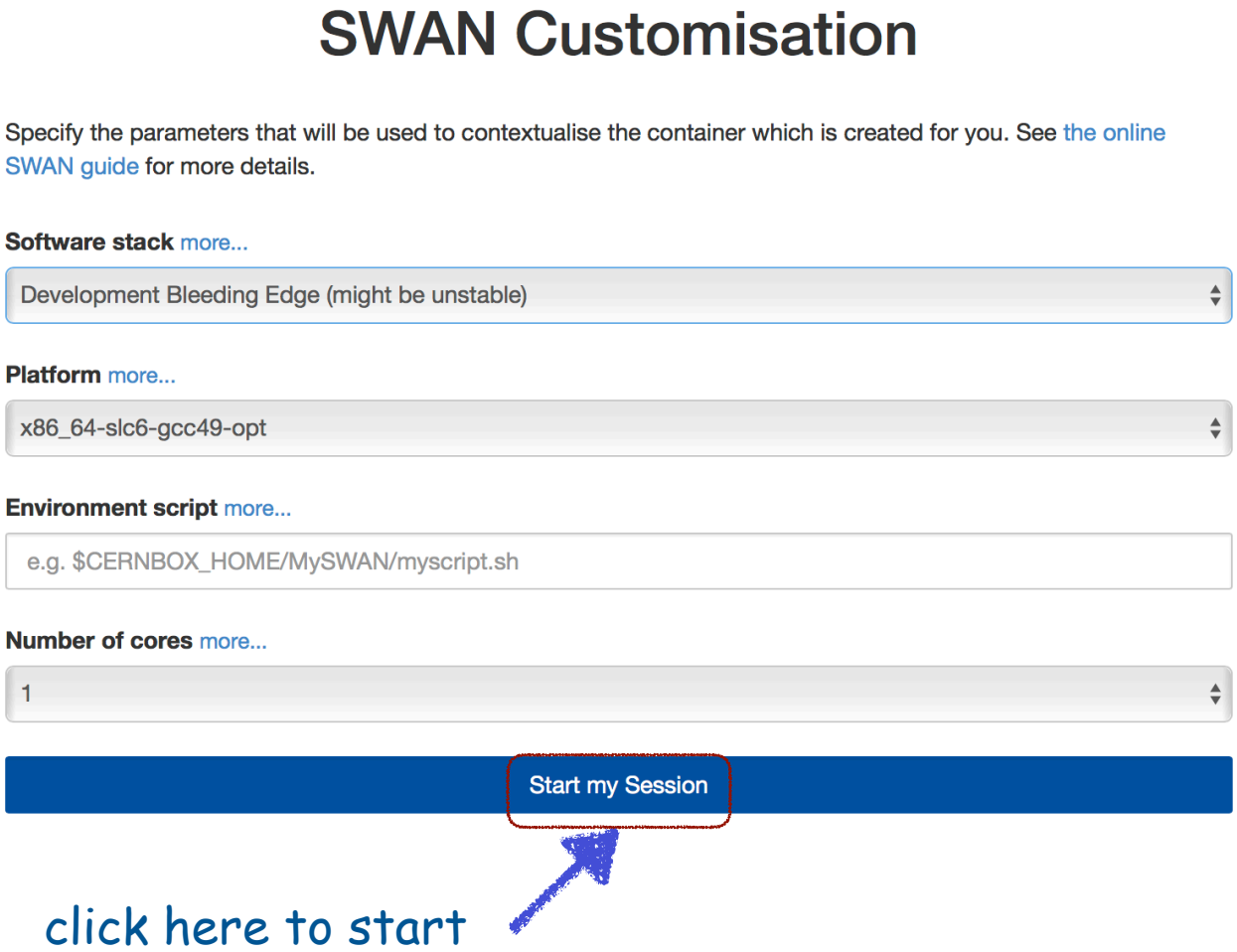
TMVA Tutorial

- **Run tutorial on notebook**
  - use **SWAN**
    - go to swan.cern.ch

  - **or running local notebooks**
    - root —notebook

If you don’t have CERN account for using SWAN please contact me
Some temporary account can be made available

Starting SWAN



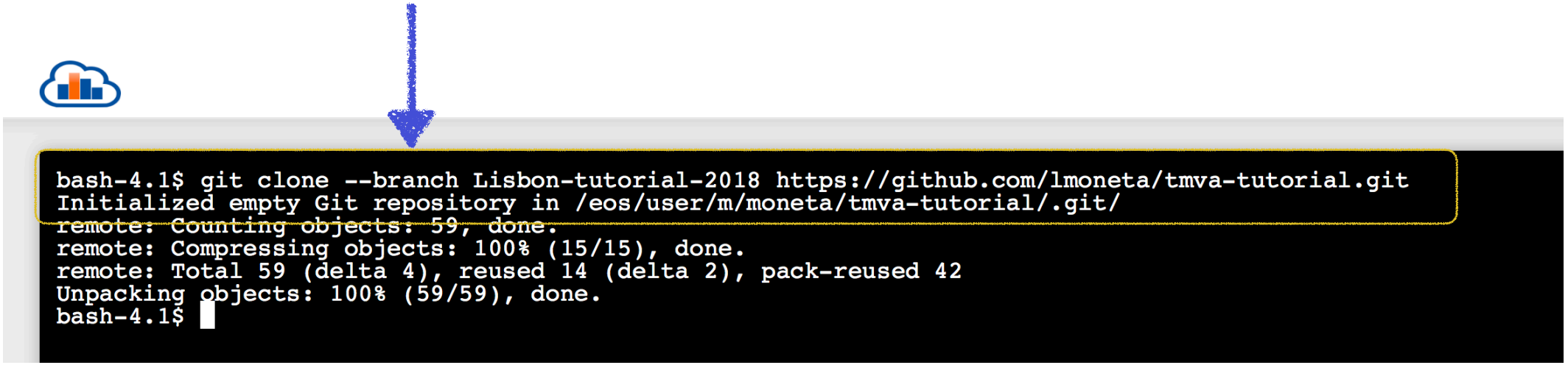
Starting a Terminal in SWAN

After login cernbox home directory will be visible

Start a terminal window

Getting the Notebooks

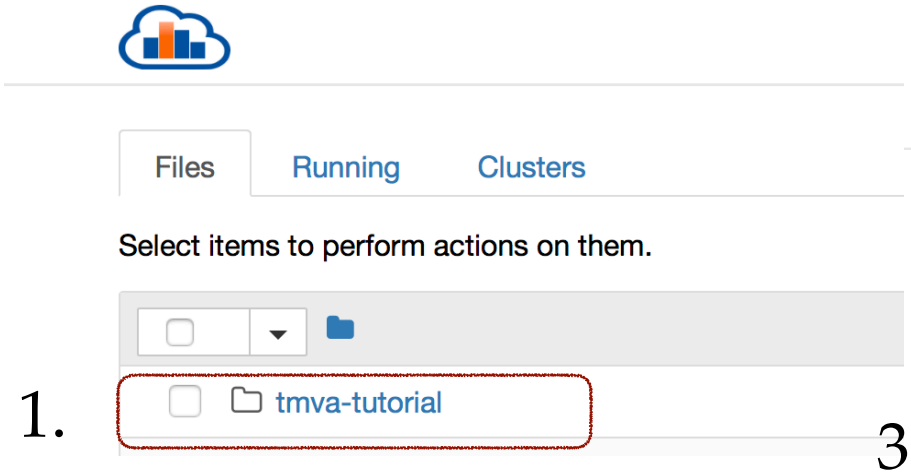
- **Clone the git repository**
  - clone directly **Lisbon-tutorial-2018** branch
  - **git clone —branch Lisbon-tutorial-2018** https://github.com/lmoneta/tmva-tutorial.git



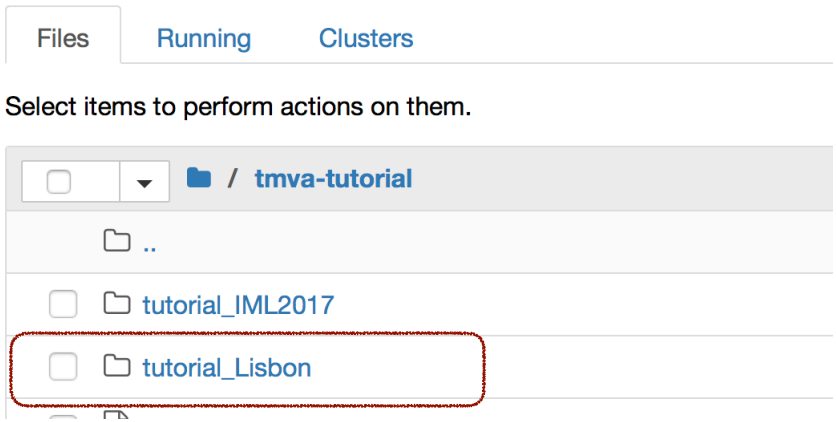
- Go back to Home page and select the directory tmva_tutorial/tutorial_Lisbon
  - Start using the notebooks

Notebooks

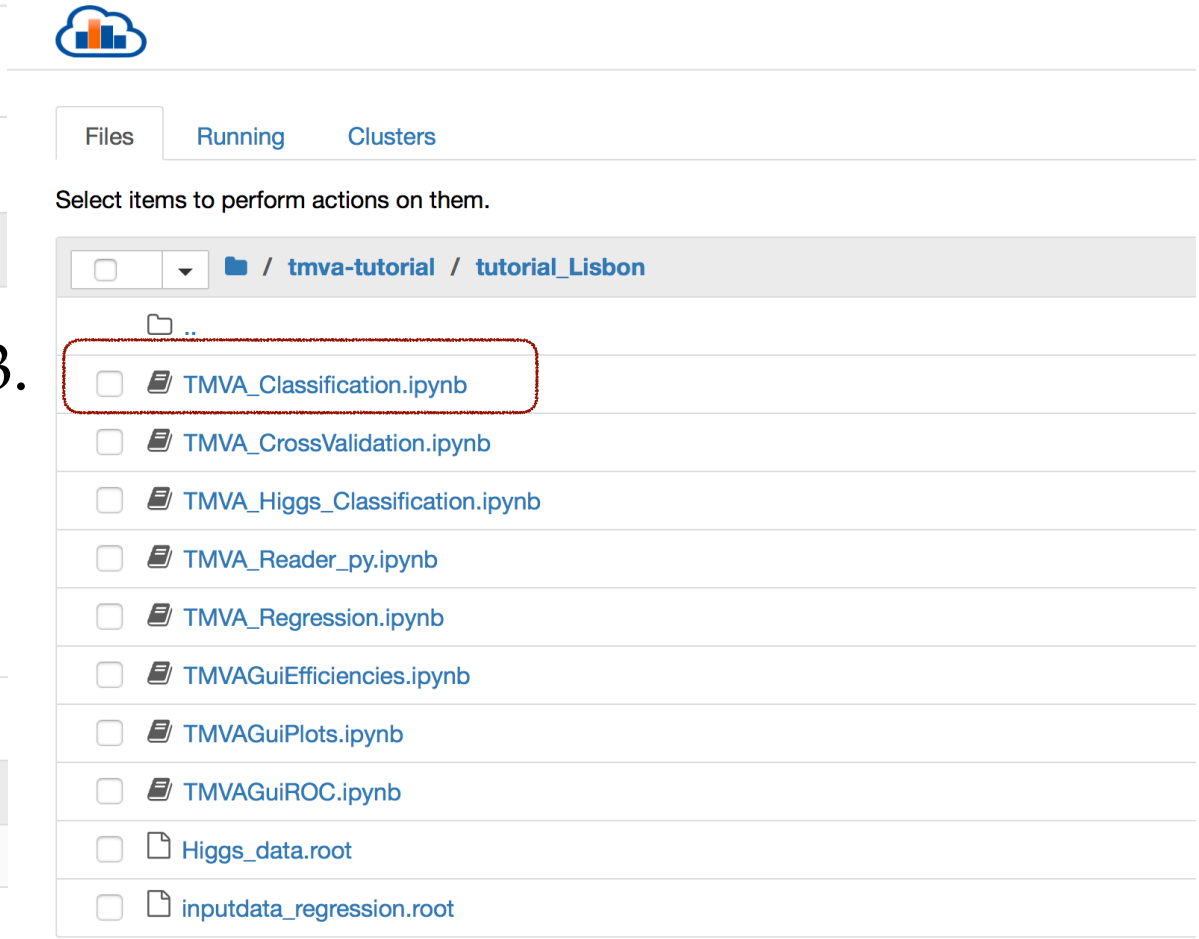
1.



2.



3.



TMVA Classification

TMVA_Classification (autosaved)

Control Panel | Logout

File Edit View Insert Cell Kernel Help | Not Trusted | ROOT C++

Markdown

TMVA Classification Example

Declare Factory

Create the Factory class. Later you can choose the methods whose performance you'd like to investigate.

The factory is the major TMVA object you have to interact with. Here is the list of parameters you need to pass

- The first argument is the base of the name of all the output weightfiles in the directory weight/ that will be created with the method parameters
- The second argument is the output file for the training results
- The third argument is a string option defining some general configuration for the TMVA session. For example all TMVA output can be suppressed by removing the "!" (not) in front of the "Silent" argument in the option string

In [1]:
```
TMVA::Tools::Instance();
```