TMVA Tutorial

L. Moneta (CERN EP-SFT)

1st Terascale School of Machine Learning

22-26 October 2018, Desy

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 analyses
  - several publications produced using TMVA
- Development done in collaboration with CERN experiments and HEP community
- HEP Software Foundation (HSF) community:
  - Machine Learning white paper
  - importance of providing internal machine learning software tools for HEP

Key Features of TMVA

- Facilitates HEP research, from detector to analysis
  - best suitable for HEP analysis with direct connection to ROOT I/O
  - written in C++
- Good performance (makes use of GPU and CPU parallelisation)
- Stability of interfaces
- Easy to use
- Long term support
- **Challenge in integrating new algorithms**
  - Machine learning world evolves very fast
- **Several features added recently** (e.g. deep learning)
- **Interfaces to integrate external tools** easily (from Python and R)

TMVA

TMVA Methods

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

- 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)
- Predictive learning via rule ensembles (RuleFit)
- Support Vector Machine (SVM)
- Artificial neural networks (various implementations)
- Boosted/Bagged decision trees

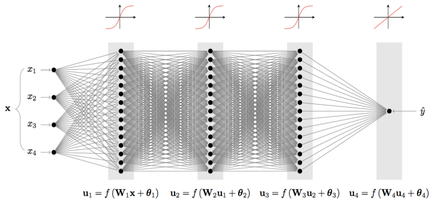
New Features

New major features added recently and available in the latest ROOT:

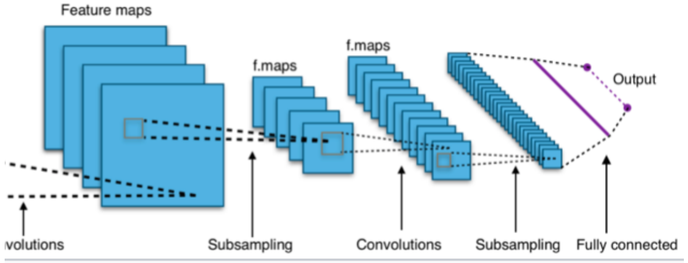
- Deep Learning
  - support for parallel training on CPU and GPU (with CUDA)
  - with fully connected (dense) , convolutional and recurrent layers
- Improved BDT:
  - new loss functions for regression
  - improved performances with multi-thread parallelisations
- Cross Validation and Hyper-parameter optimisation
- Interfaces to external ML library in R and Python (scikit-learn and Keras)

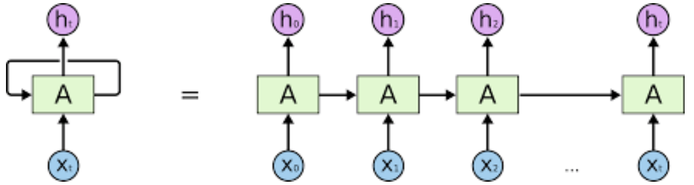
New Deep Learning Features

- Available for dense layer since 2016 (ROOT version 6.08)



- Extended Deep Learning in ROOT 6.14 supporting
  - Convolutional Layer
    - powerful for image data sets
  - Recurrent Layer
    - useful for time dependent data

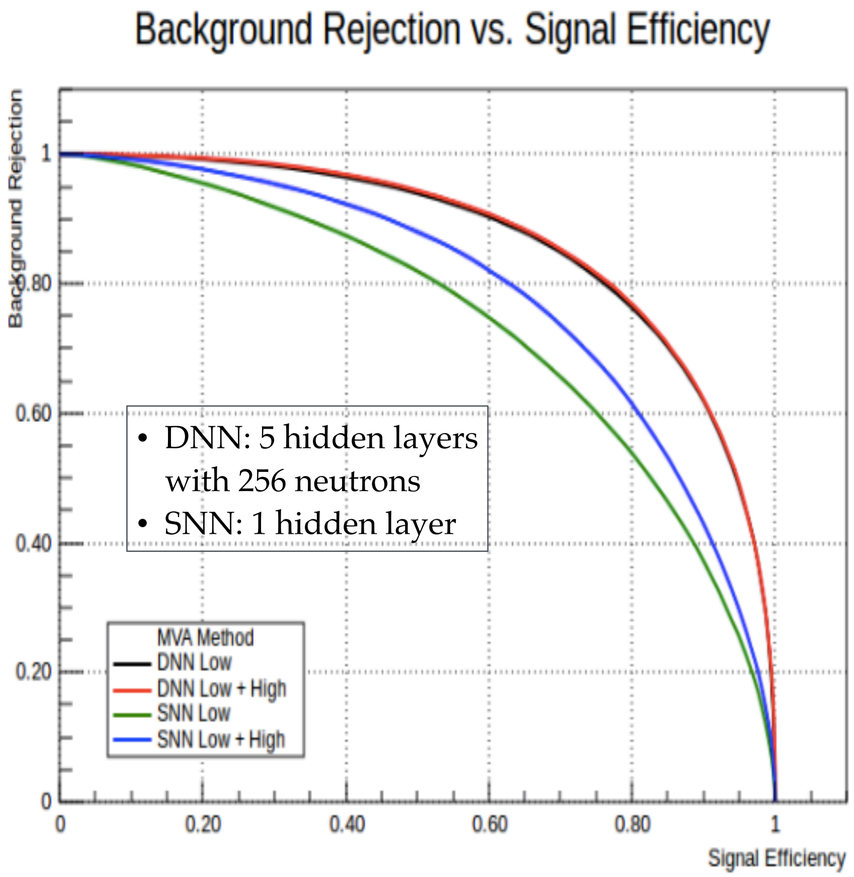




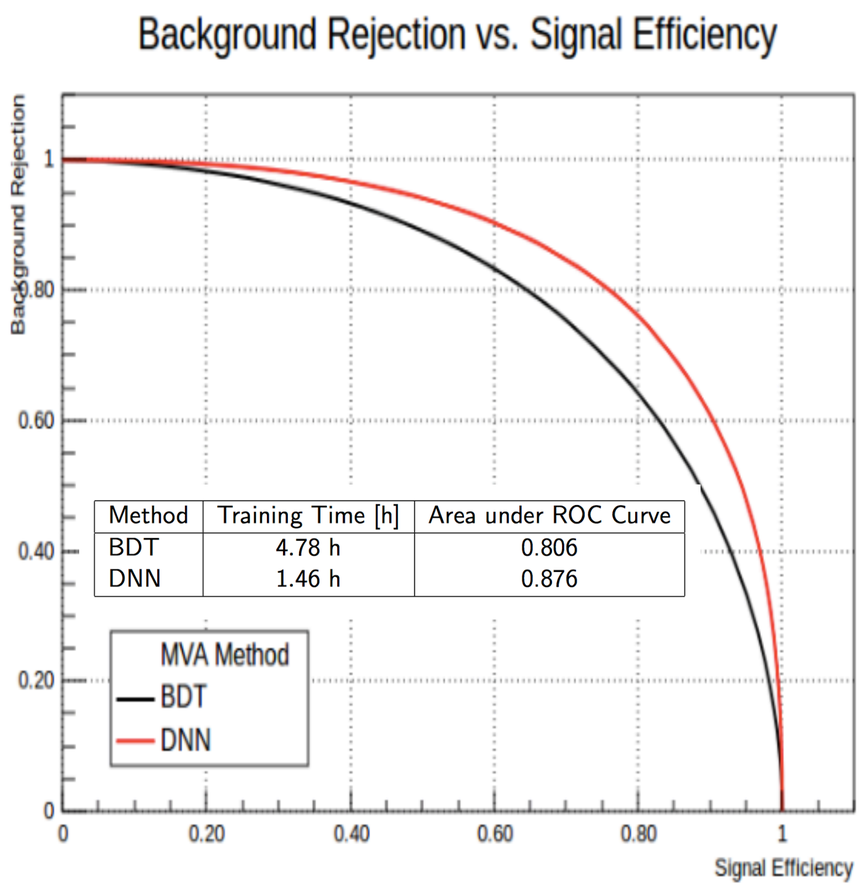
- Optimized for parallel evaluation and training
  - CPU (with openBLAS + TBB) and GPU (CUDA)
  - Several optimisers available (e.g. SGD, ADAM, …)

Deep Learning Performance

DNN vs Standard ANN



DNN vs BDT



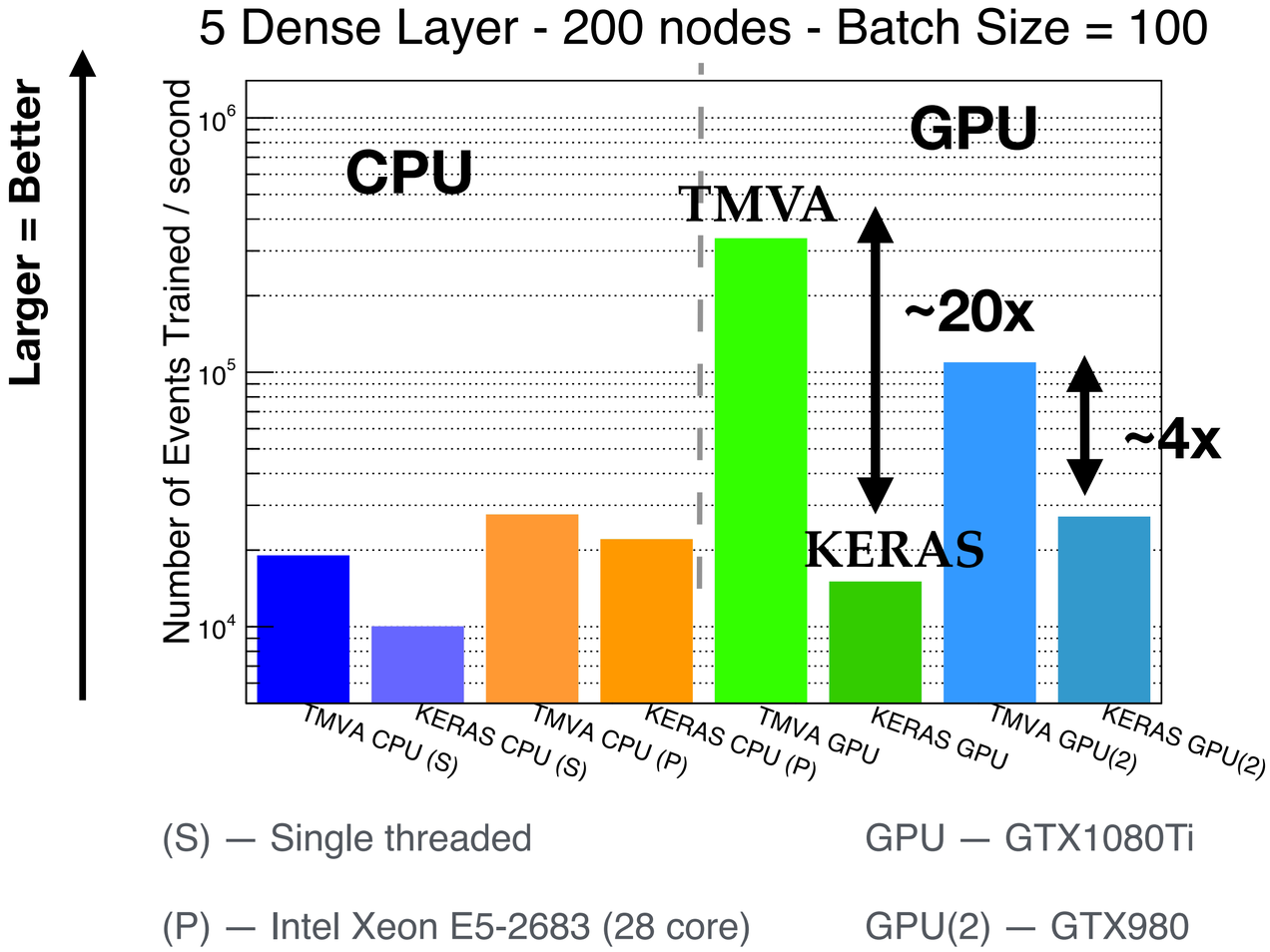
- Using Higgs public dataset (from UCI) with 11M events
- Significant improvements compared to shallow networks and BDT

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)

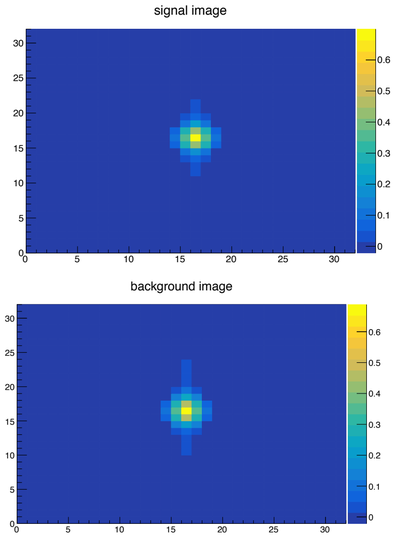
GPU — GTX1080Ti

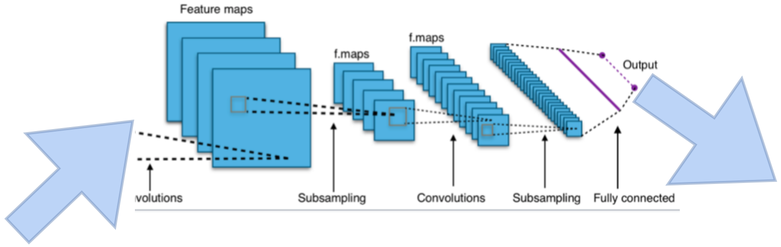
GPU(2) — GTX980

Convolutional Neural Network

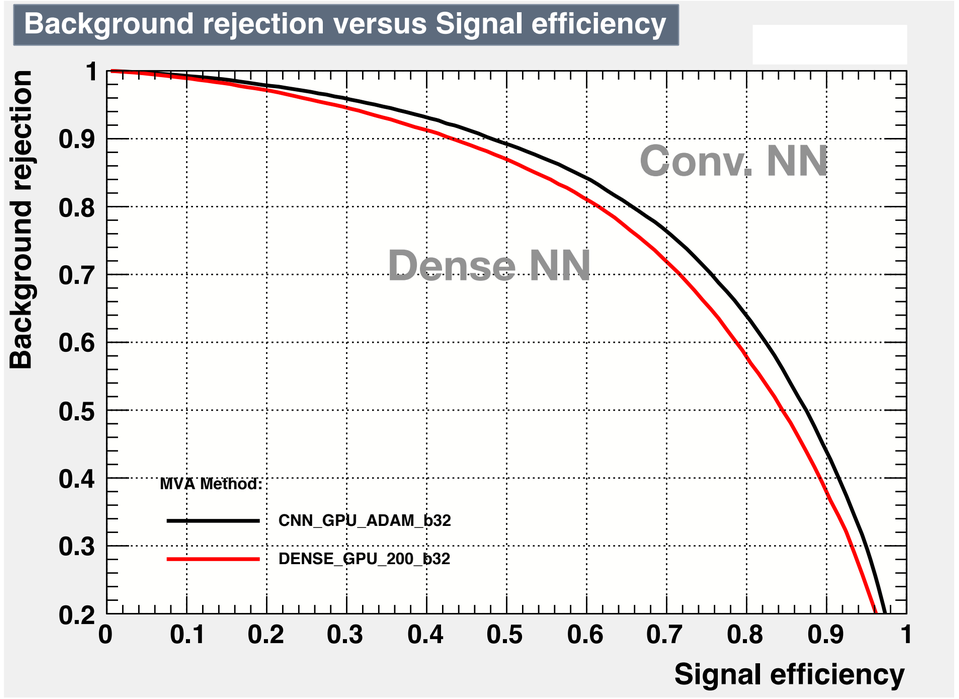
- Available in latest ROOT version (6.14)
- Supporting CPU parallelization, GPU is now also available
  - parallelisation and code optimisation is essential (excellent TMVA performances)
- Image dataset from simulated particle showers from an electromagnetic calorimeter
  - distinguish electron from photon showers

input 32x32 images



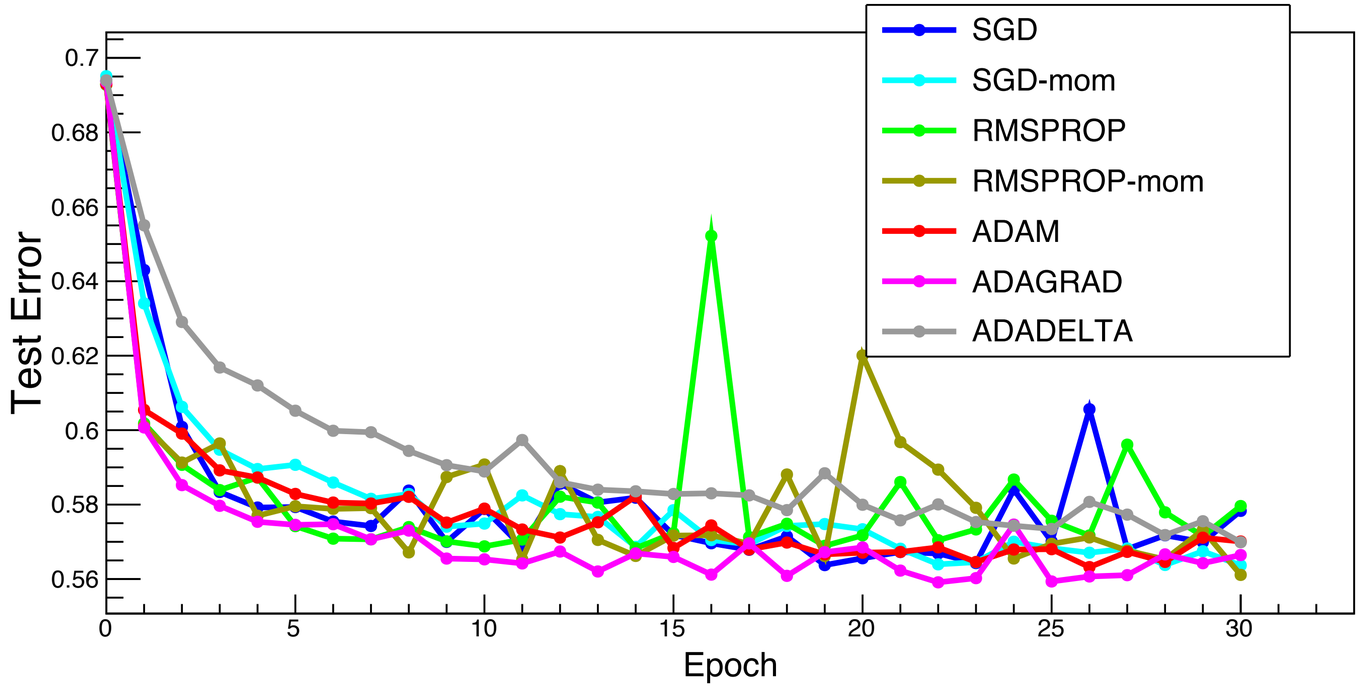


Convolutional + Pooling + Dense layers



New Deep Learning Optimizers

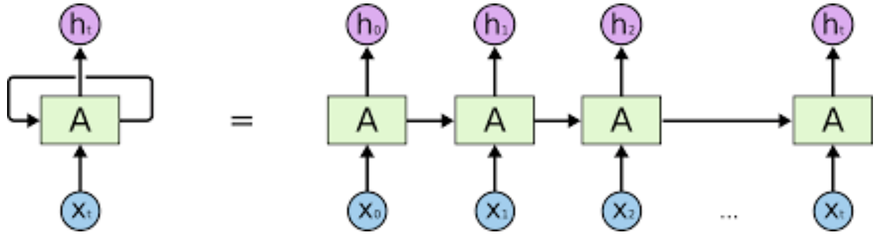
- Integrated in TMVA master new deep learning optimisers
  - In addition to SGD (Stochastic Gradient Descent) added
    - support acceleration using momentum
    - ADAM (new default)
    - ADADelta
    - ADAGrad
    - RMSProp



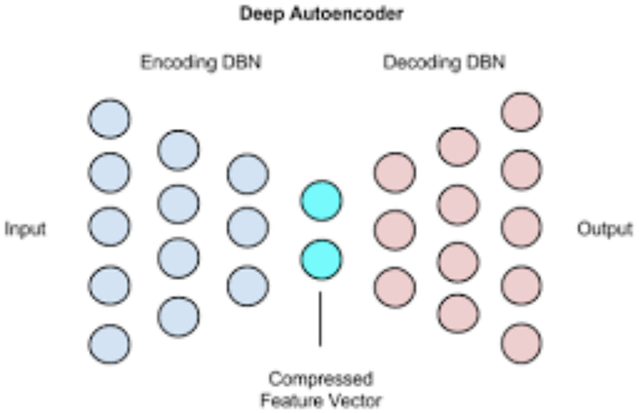
With these new optimisers need less epochs (iterations) to converge !

Other Deep Learning Developments

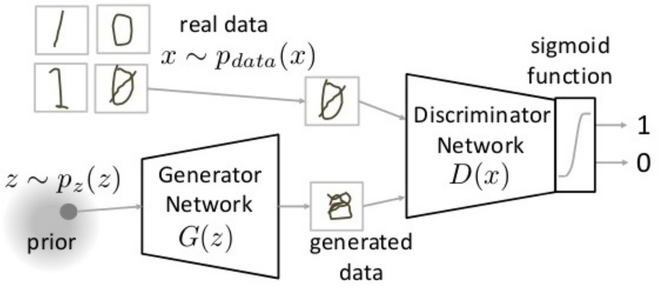
- **Recurrent Neural Network**
  - useful for time-dependent data
  - first version available in 6.14
  - working on extending the support for LSTM layer



- **Deep Auto Encoder**
  - dimensionality reduction (pre-processing tool)
  - unsupervised tool (e.g. for anomaly detection)
  - usable also for generating models,
    Variational Auto Encoder (VAE)



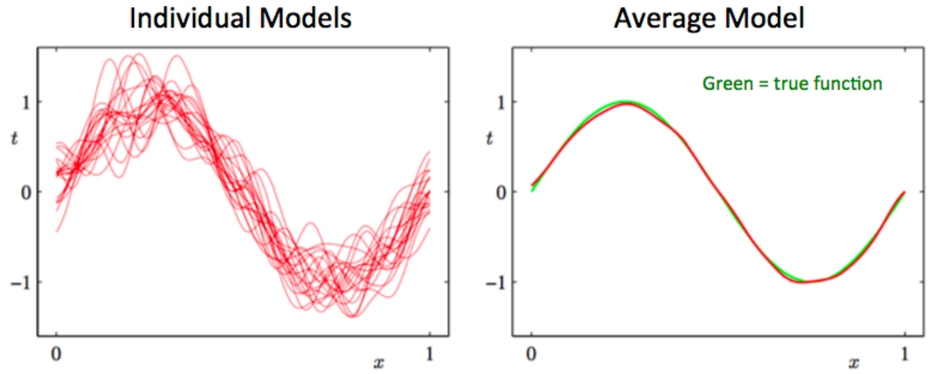
- **Generative Adversarial Network (GAN)**
  - model generation tool (fast simulation)



GSOC projects during this summer for these new developments

Decision Trees

- Method based on ensembles of decision trees
  - enhance considerably performances combining several trees
  - several variants exists (Random Forest, ADABoost, Gradient Boost)
  - Rarely overfitting
  - Robust to noisy data
  - Easy and fast to train them

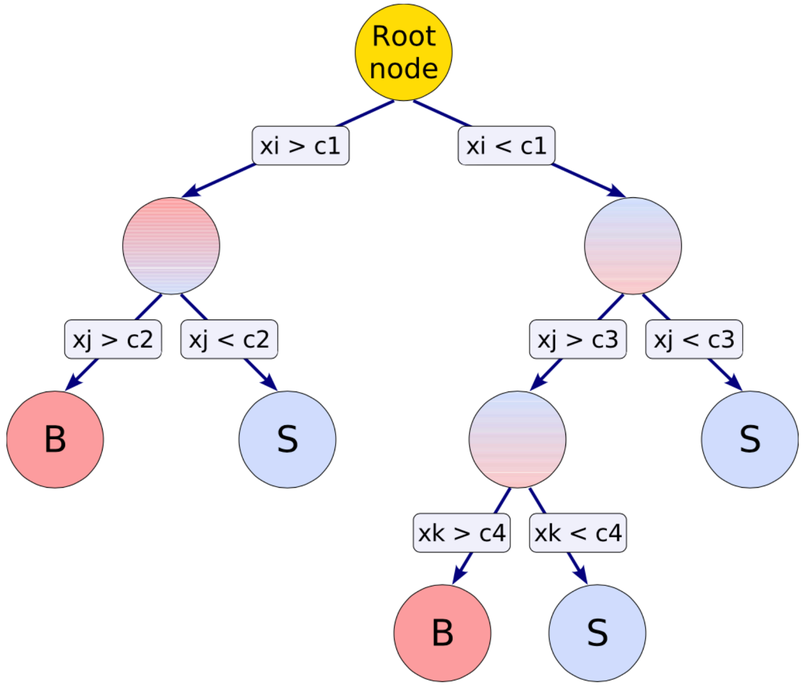


[Bishop]

- Very popular method (before deep learning)
  - Example: using 179 classifier on 121 public data sets:
    - Random Forest found best classifier

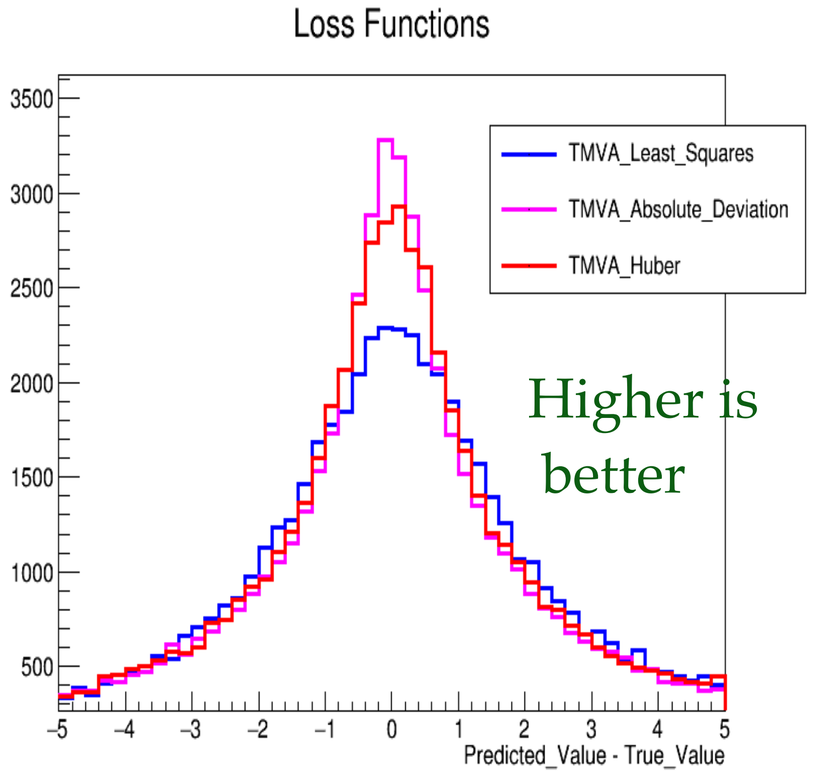
Decision Trees in TMVA

- TMVA provides a very good implementation of Decision Trees (BDT)
  - ADABoost with 3 different variants
  - Gradient boosting
  - Bagging
  - Random Forest
    (bagging and randomised trees)
- Several configuration options available
  (See TMVA Users Guide)



Regression with BDT

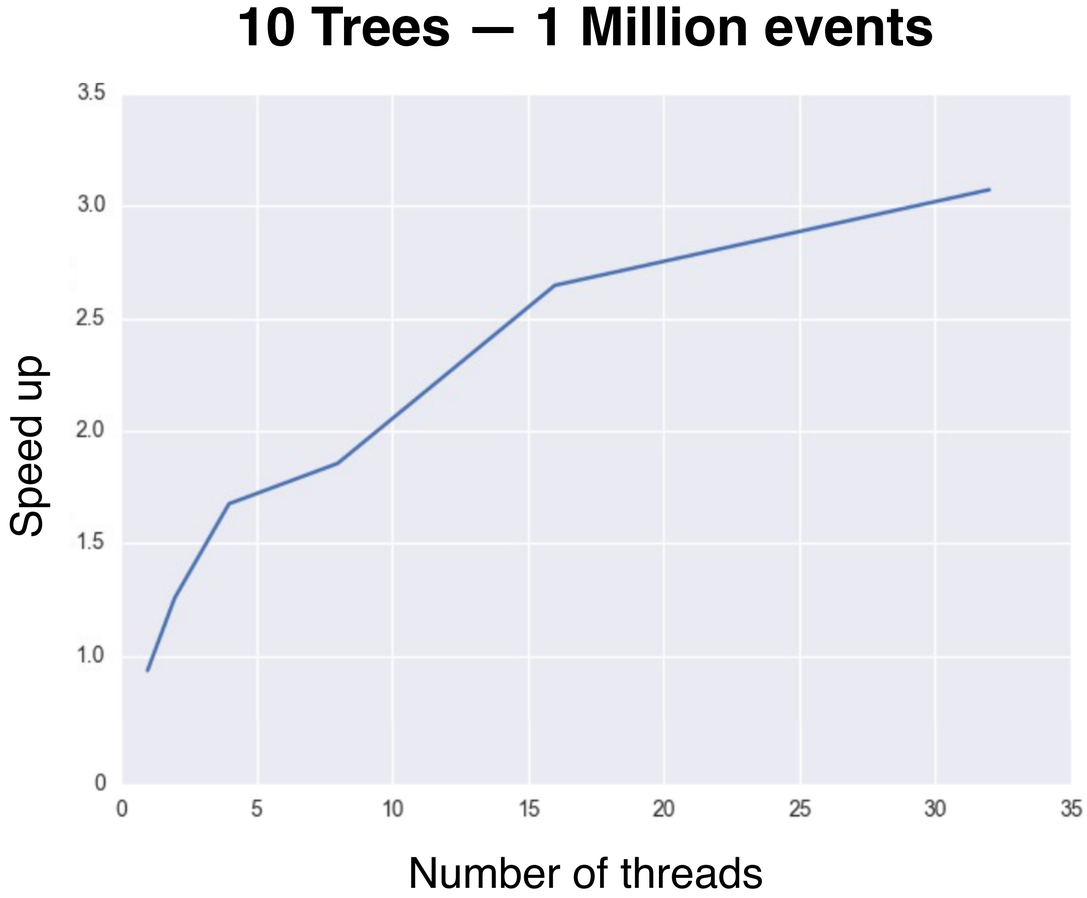
- **New regression Features**:
  - Loss function
    - Huber (default)
    - Least Squares
    - Absolute Deviation
    - Custom Function



Important for regression performance

Parallelisation of BDT's

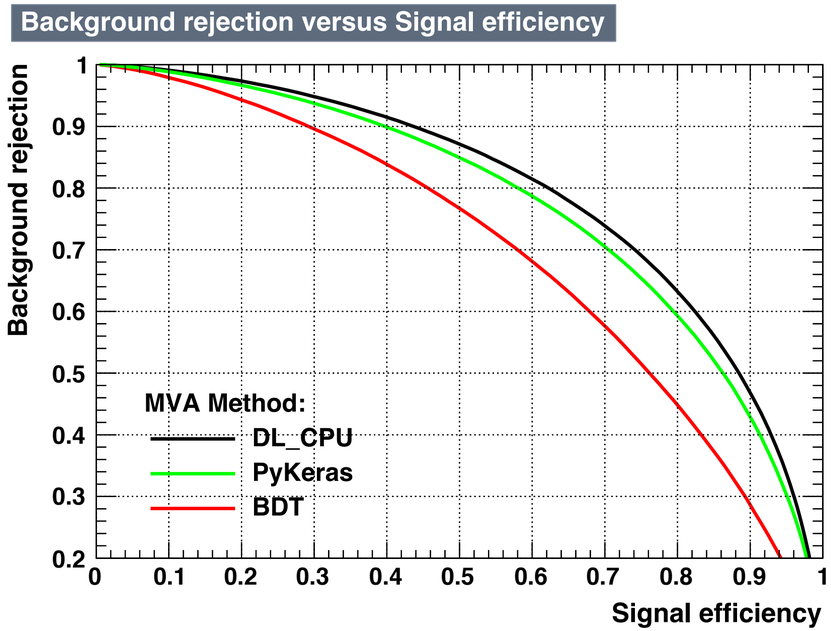
- Boosting is serial ⟶ Can't construct all trees in parallel
- Training time speed up ~1.6x with 4 threads approaching ~3x asymptotically
- To use, just add
  ROOT::EnableImplicitMT()
  to your code



Original slide by Andrew Carnes

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 ML methods in R**
  - c50, xgboost, RSNNS, e1071
- **PYMVA**: **Interface to Python ML**
  - **scikit-learn**
    - with RandomForest,
      Gradiend Tree Boost, Ada Boost
  - **Keras** (Theano + Tensorflow)
    - support 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

Define the 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()
```

Book the method as any others of TMVA

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')
```

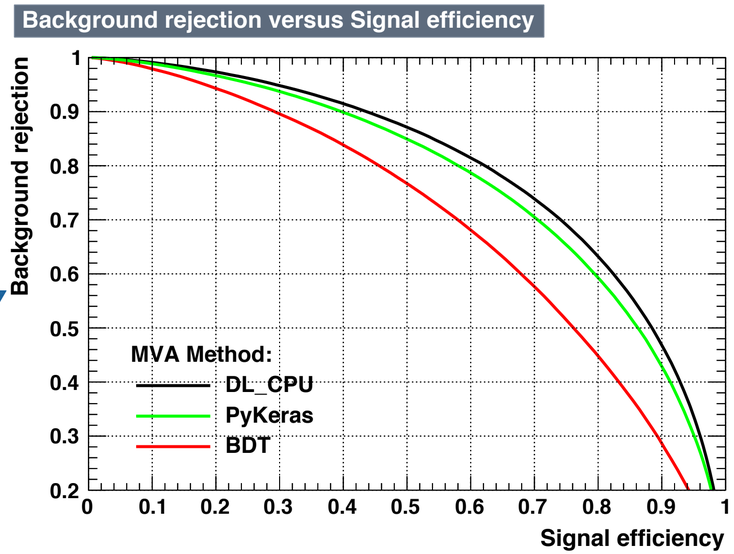
Out[6]:
```
<ROOT.TMVA::MethodPyKeras object ("PyKeras") at 0x77e48b0>
```

PyMVA with Keras

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
:
```



Examine result with TMVA GUI

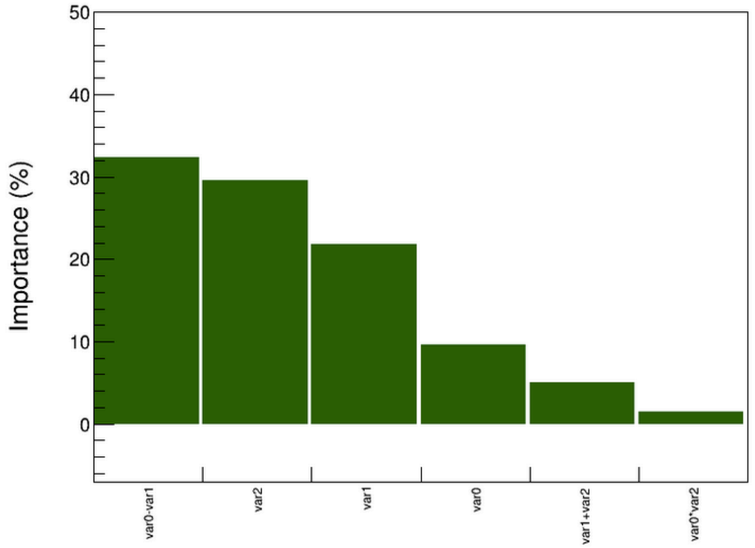
Feature Importance

- Ranks the importance of features based on contribution to classifier performance
  - A stochastic algorithm independent of classifier choice

$$FI(X_i) = \sum_{S \subseteq V : X_i \in S} F(S) \times W_{X_i}(S)$$

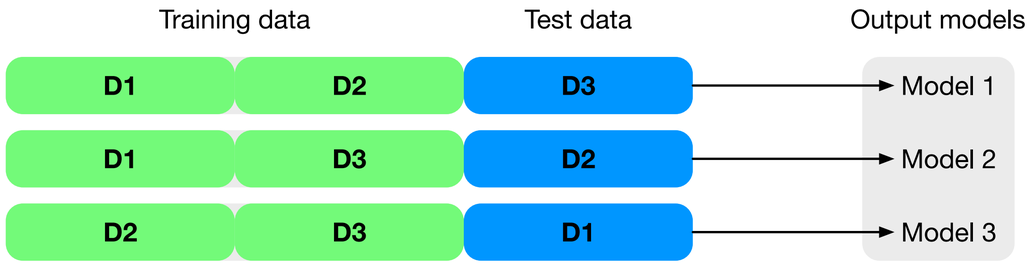
$$W_{X_i}(S) \equiv 1 - \frac{F(S - \{X_i\})}{F(S)}$$

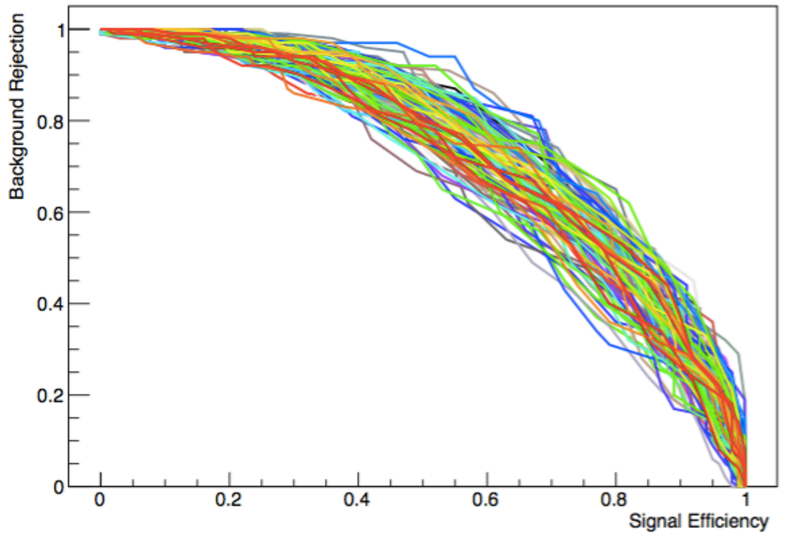
- Feature set {V}
- Feature subset {S}
- Classifier Performance F(S)



Cross Validation in TMVA

- TMVA supports k-fold cross-validation

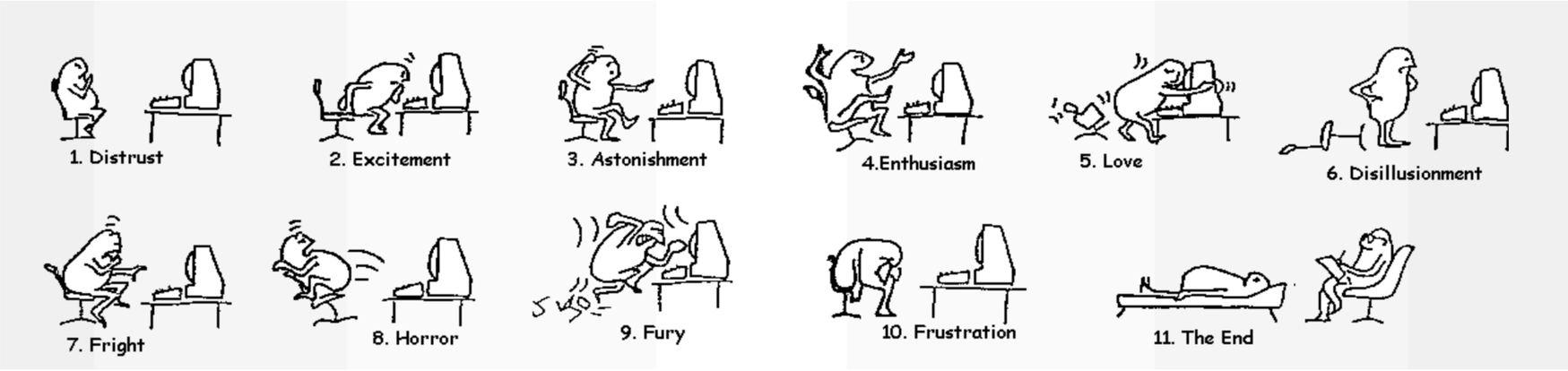




  - Integration with TMVA analysis tools (e.g. GUI)
- Hyper-parameter tuning
  - find optimised parameters (BDT-SVM)
- Parallel execution of folds in CV
  - using multi-processes execution in on a single node
  - foreseen to provide parallelisation in a cluster using Spark or MPI

Future Developments

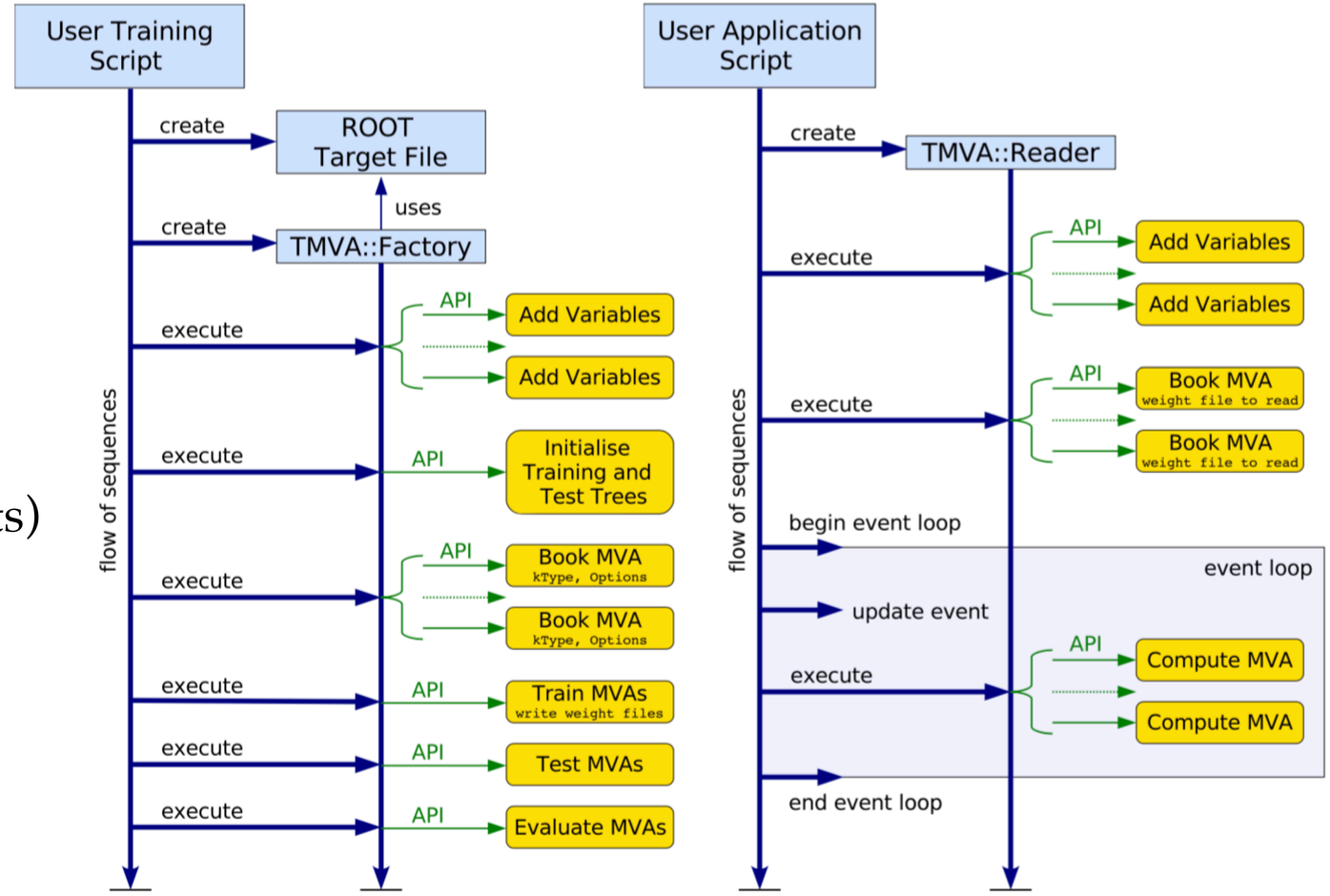
- Our aim is to provide to the users community efficient physics workflows
  - tools for efficient
    - data loading (using new RDataFrame for filtering the data)
    - integration with external ML tools
    - training of commonly used architectures
    - deployment and inference of trained models
- TMVA efficiently connects input data to ML algorithms
  - working on defining new user interfaces


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

- 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 Workflow Features

TMVA supports:

- input data from ROOT Trees or ASCII data (e.g. csv)
- pre-selection cuts on input data
- event weights (negative weights for some methods)
- various method for splitting training/test samples
- k-fold cross-validation and hyper-parameter optimisation
- algorithm to identify importance of input variables
- GUI for output evaluation and analysis

TMVA Session

- Create Factory
- Create DataLoader class
- Provide input data and add variables/target using the DataLoader
- Prepare data (training/test split)
- Book MVA methods
- Train/Test/Evaluate using
- Save output and train methods

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

TMVA DataLoader

- DataLoader is a new class that allows greater flexibility when working with datasets. It is an interface to
  - load the datasets
    - root files (TTrees) but can be extended to other types
  - add variables

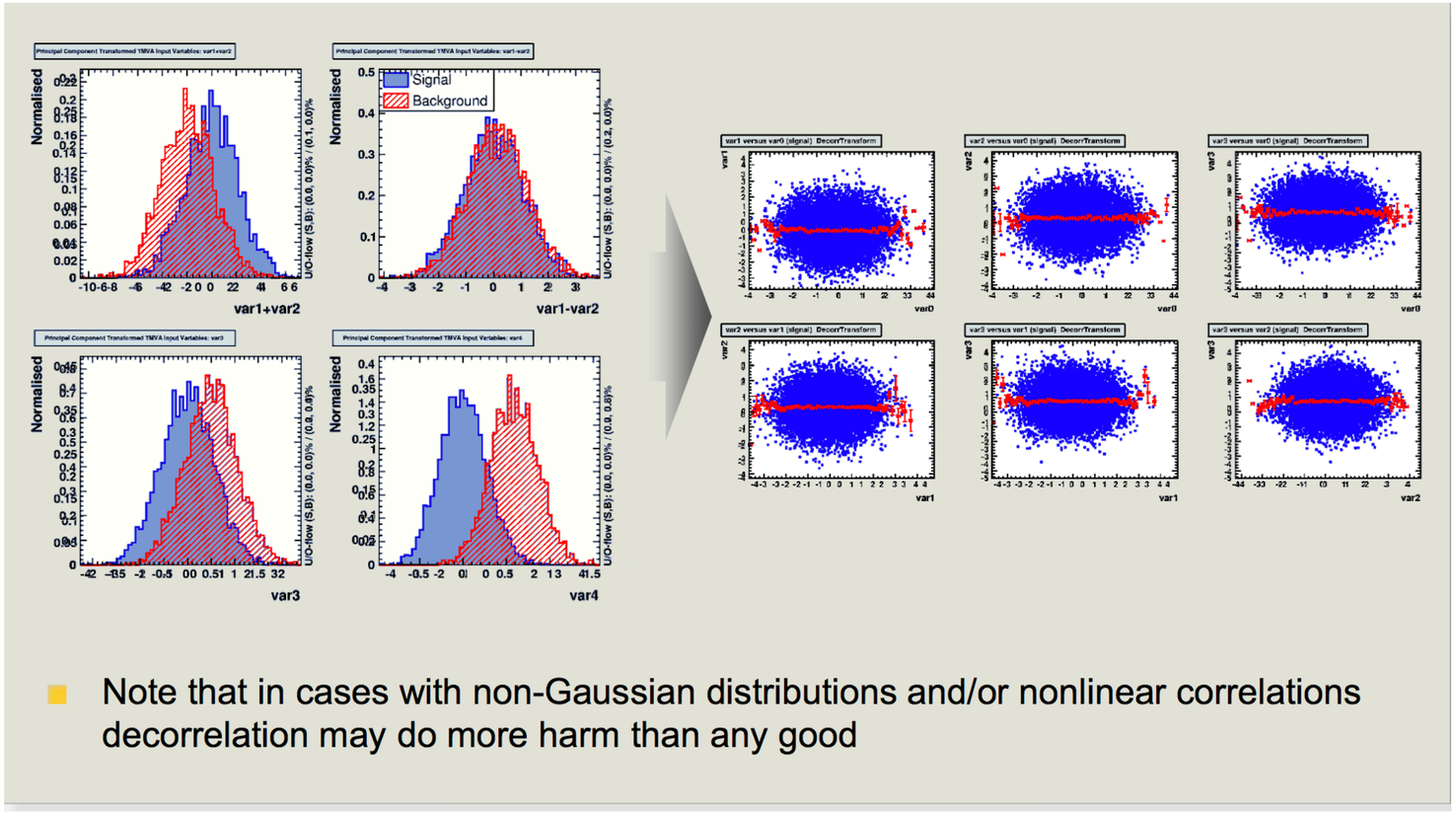
- TMVA Factory links DataLoader with a specific MVA method when booking

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

- Obtained desired flexibility in de-coupling methods/dataset/variables

Pre-processing of the Input

- 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)

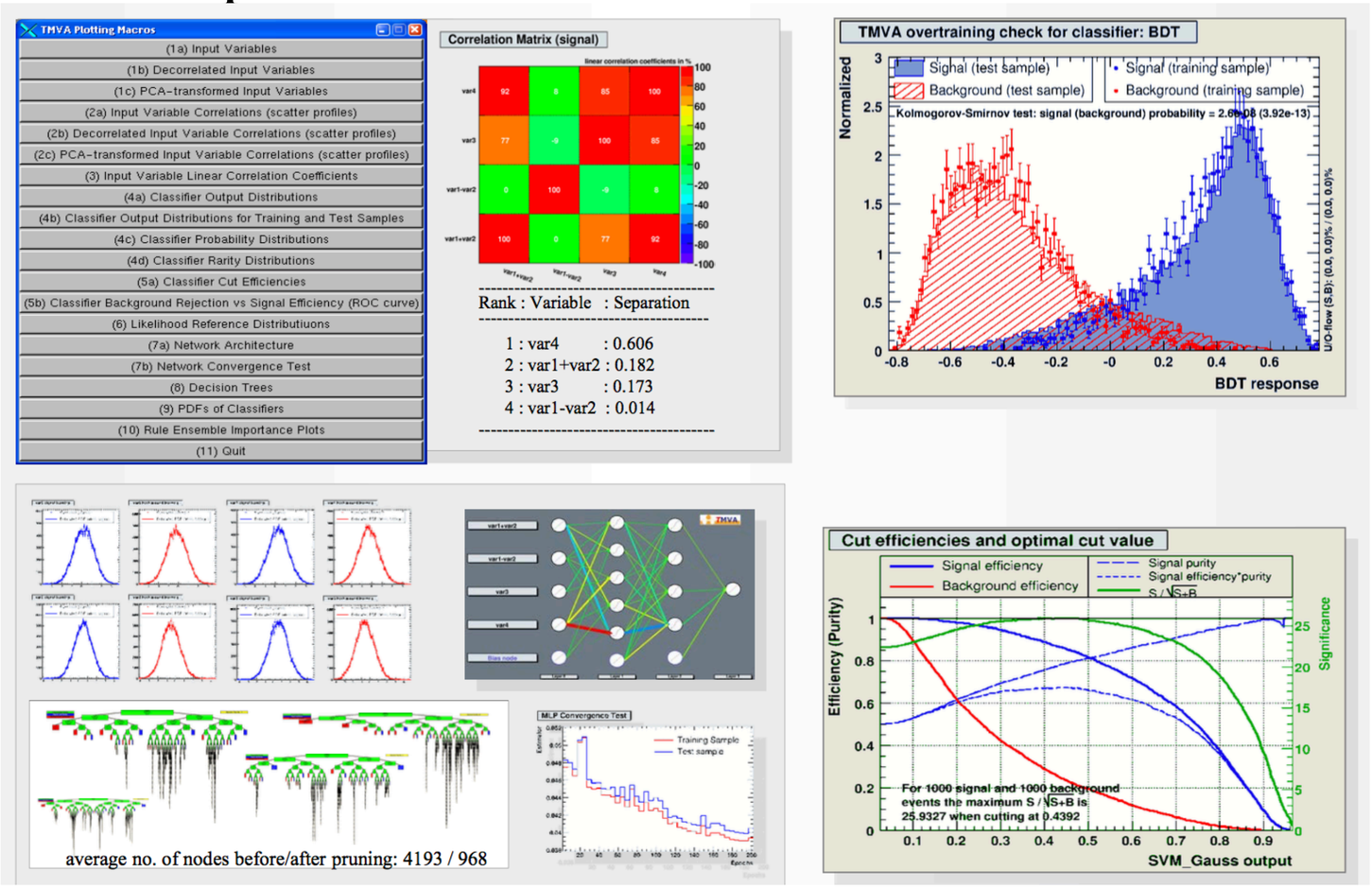
Available Preprocessing

- List of available pre-processing in TMVA
  - Normalization
  - Decorrelation (using Cholesky decomposition)
  - Principal Component Analysis
  - Uniformization
  - Gaussianization

- Can be selected individually for each single method (when booking)

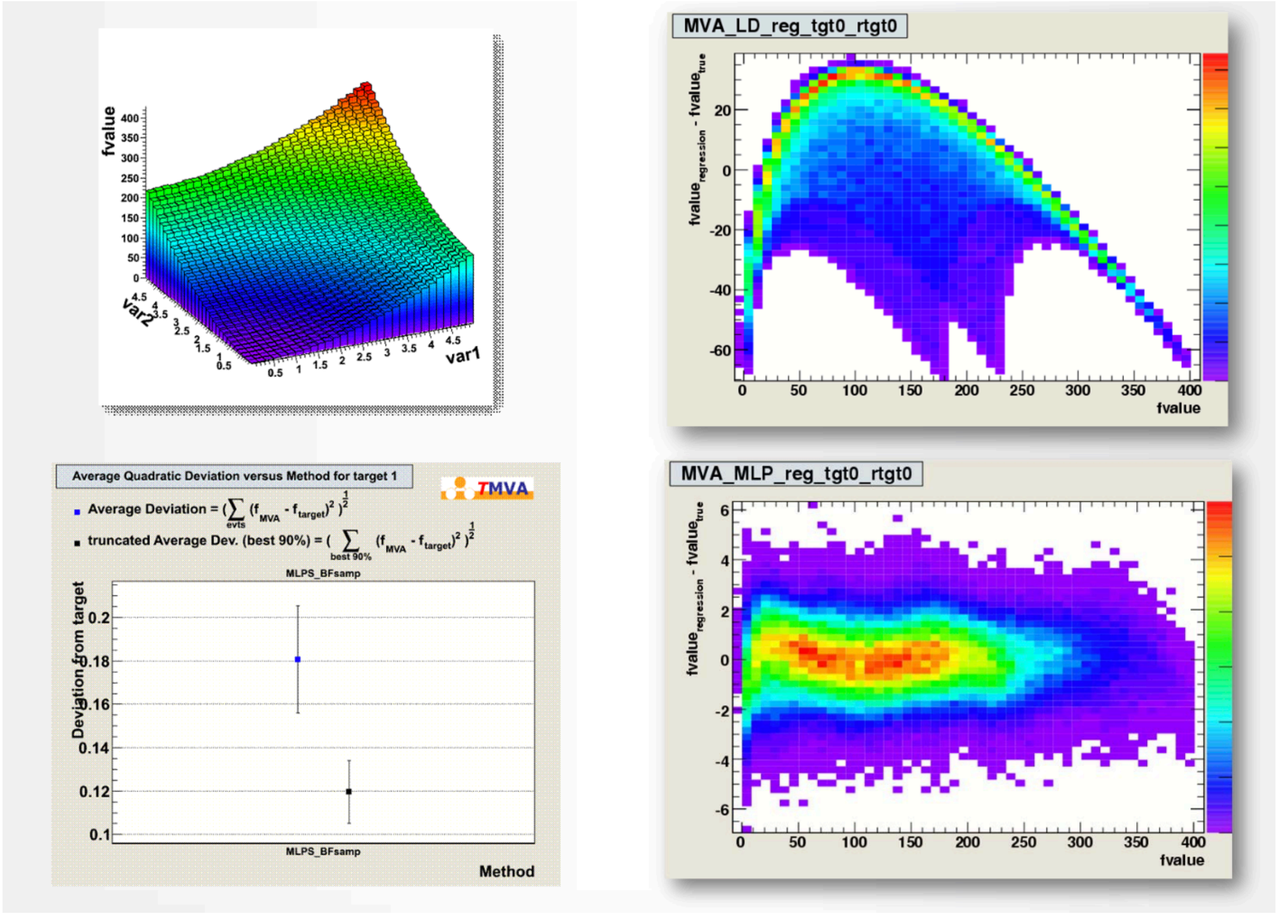
TMVA GUI

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



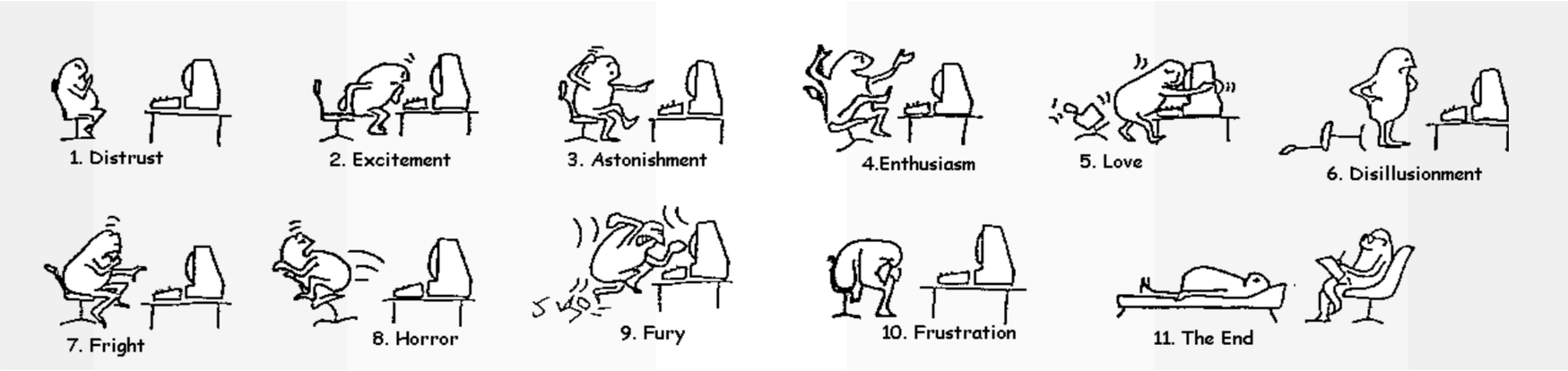
TMVA Regression GUI

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



SWAN: Data Analysis as a Service

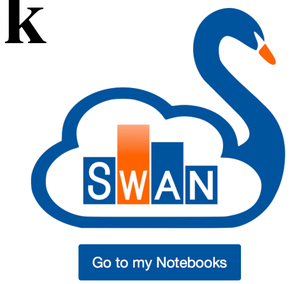
- *Interface:*
  - Jupyter Notebooks
- *Goals:*
  - Analysis only with a web browser
    - Platform independent ROOT-based data analysis
    - Calculations, input and results “in the Cloud”
  - Easy sharing of scientific results: plots, data, code
    - Storage is crucial: mass & synchronised
  - Integration with other analysis ecosystems: R, Python, ...



Let's start using TMVA

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
But before please feel the online form available **here**

- For running ROOT macros you can also use **DESY accounts**
  - setup environment (for ROOT master version) by typing after login

. /cvmfs/sft.cern.ch/lcg/views/dev3/latest/x86_64-centos7-gcc62-opt/setup.sh

Starting SWAN

Configure Environment

Specify the parameters that will be used to contextualise the container which is created for you. See the online SWAN guide for more details.

Software stack more...

Development Bleeding Edge (might be unstable)

Platform more...

x86_64-slc6-gcc62-opt

Environment script more...

e.g. $CERNBOX_HOME/MySWAN/myscript.sh

Number of cores more...

2

Memory more...

8 GB

Spark cluster more...

None

Select to use new Deep Learning

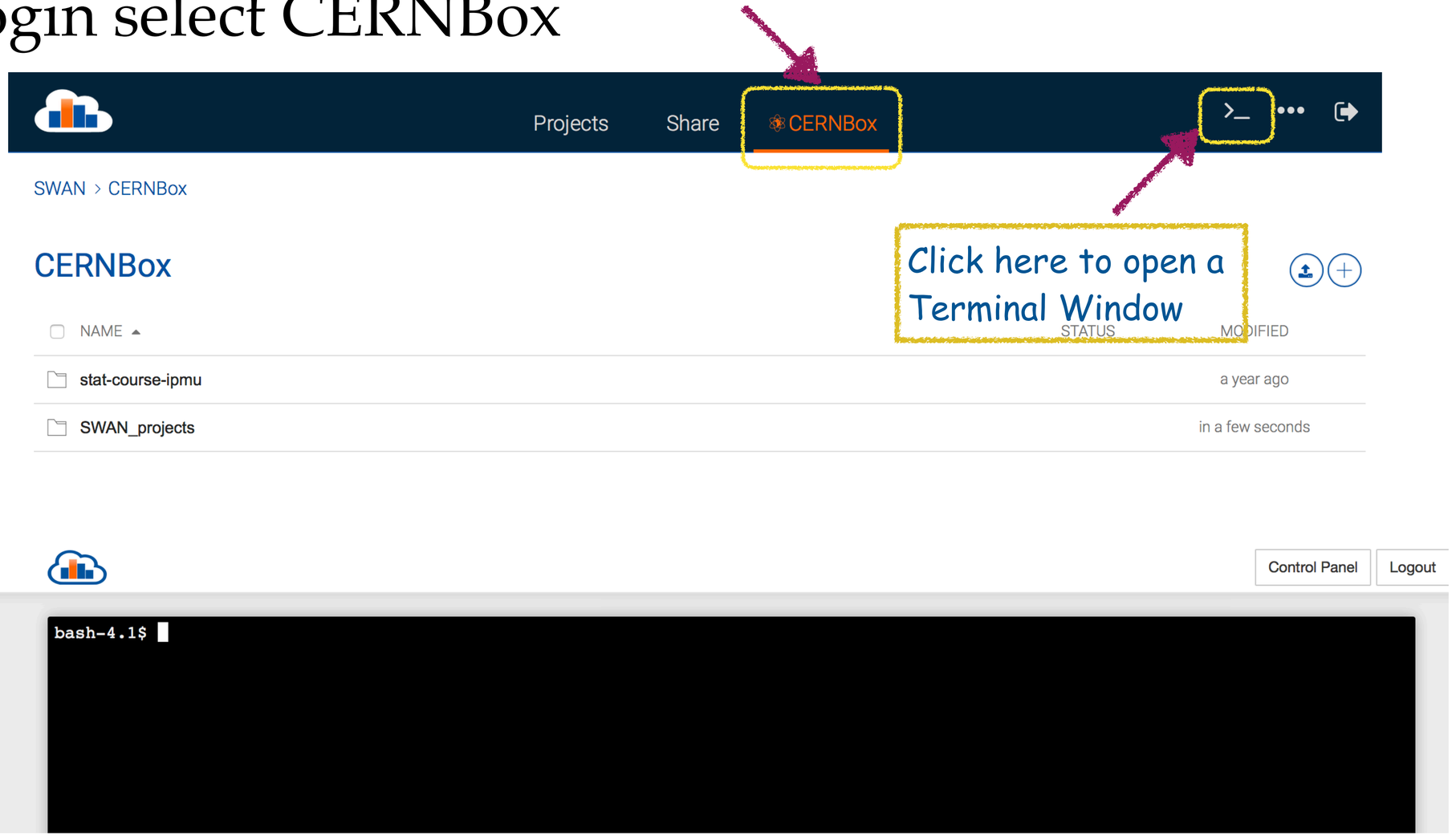
Always start with this configuration

Start my Session

Click here to start

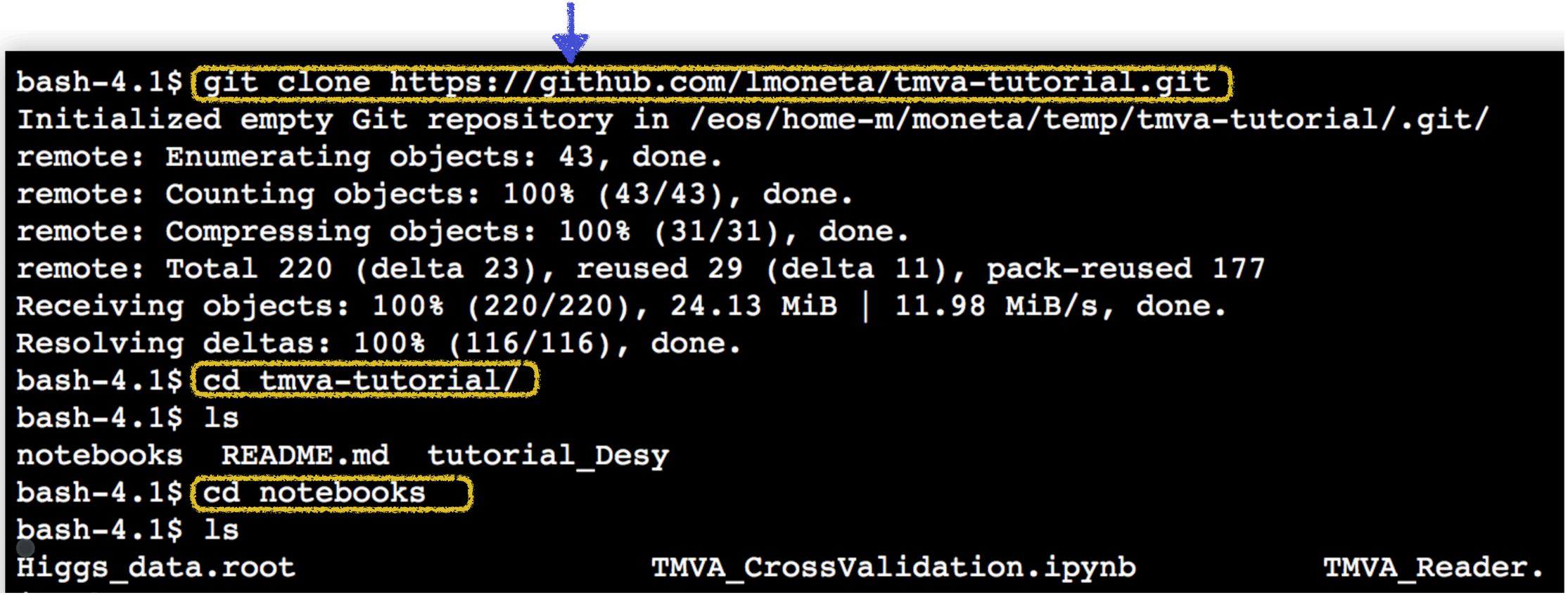
Starting a Terminal in SWAN

After login select CERNBox



Getting the Notebooks

- Clone the git repository of the tutorials
  https://github.com/lmoneta/tmva-tutorial.git
  - git clone https://github.com/lmoneta/tmva-tutorial.git

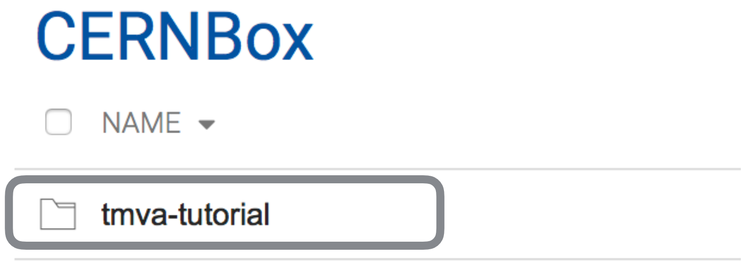


```
bash-4.1$ git clone https://github.com/lmoneta/tmva-tutorial.git
Initialized empty Git repository in /eos/home-m/moneta/temp/tmva-tutorial/.git/
remote: Enumerating objects: 43, done.
remote: Counting objects: 100% (43/43), done.
remote: Compressing objects: 100% (31/31), done.
remote: Total 220 (delta 23), reused 29 (delta 11), pack-reused 177
Receiving objects: 100% (220/220), 24.13 MiB | 11.98 MiB/s, done.
Resolving deltas: 100% (116/116), done.
bash-4.1$ cd tmva-tutorial/
bash-4.1$ ls
notebooks  README.md  tutorial_Desy
bash-4.1$ cd notebooks
bash-4.1$ ls
Higgs_data.root                    TMVA_CrossValidation.ipynb          TMVA_Reader.
```

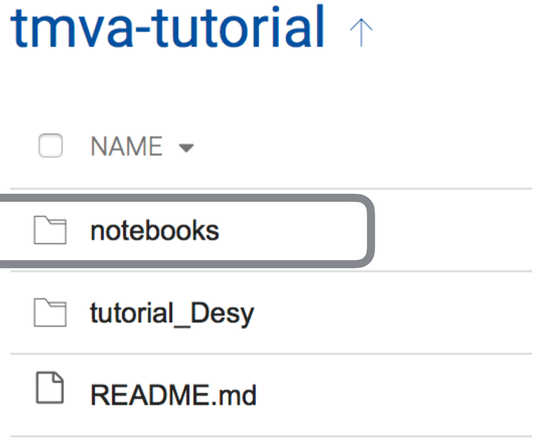
Notebooks

- The notebooks are located in the directory **`tmva-tutorial/notebooks`** in your CERNBOX

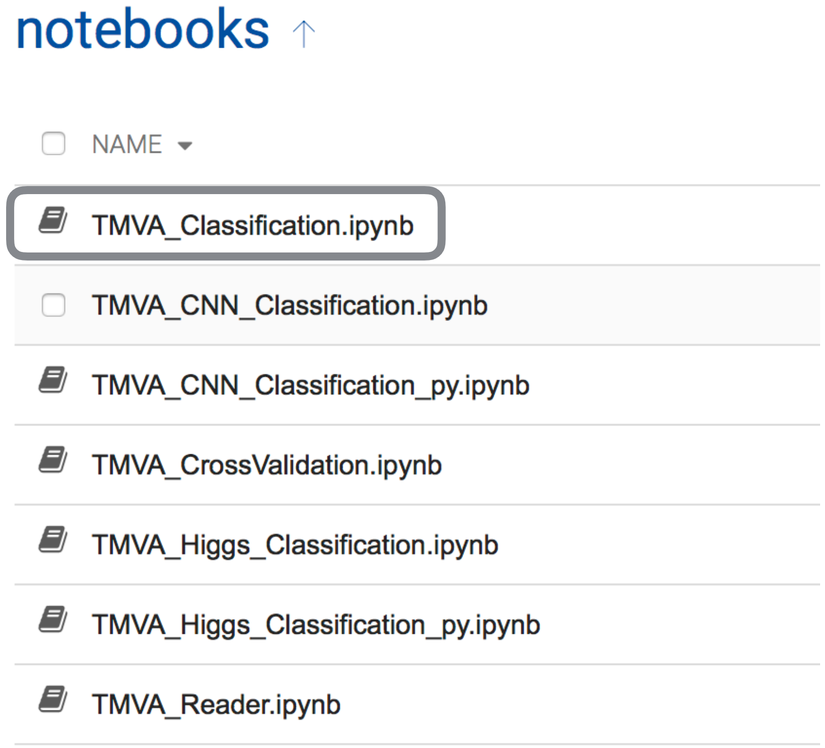
1.



2.



3.



TMVA Classification

TMVA_Classification
(autosaved)

FILE EDIT VIEW INSERT CELL KERNEL HELP

Not Trusted | ROOT C++

Markdown

TMVA Classification Example

Declare Factory

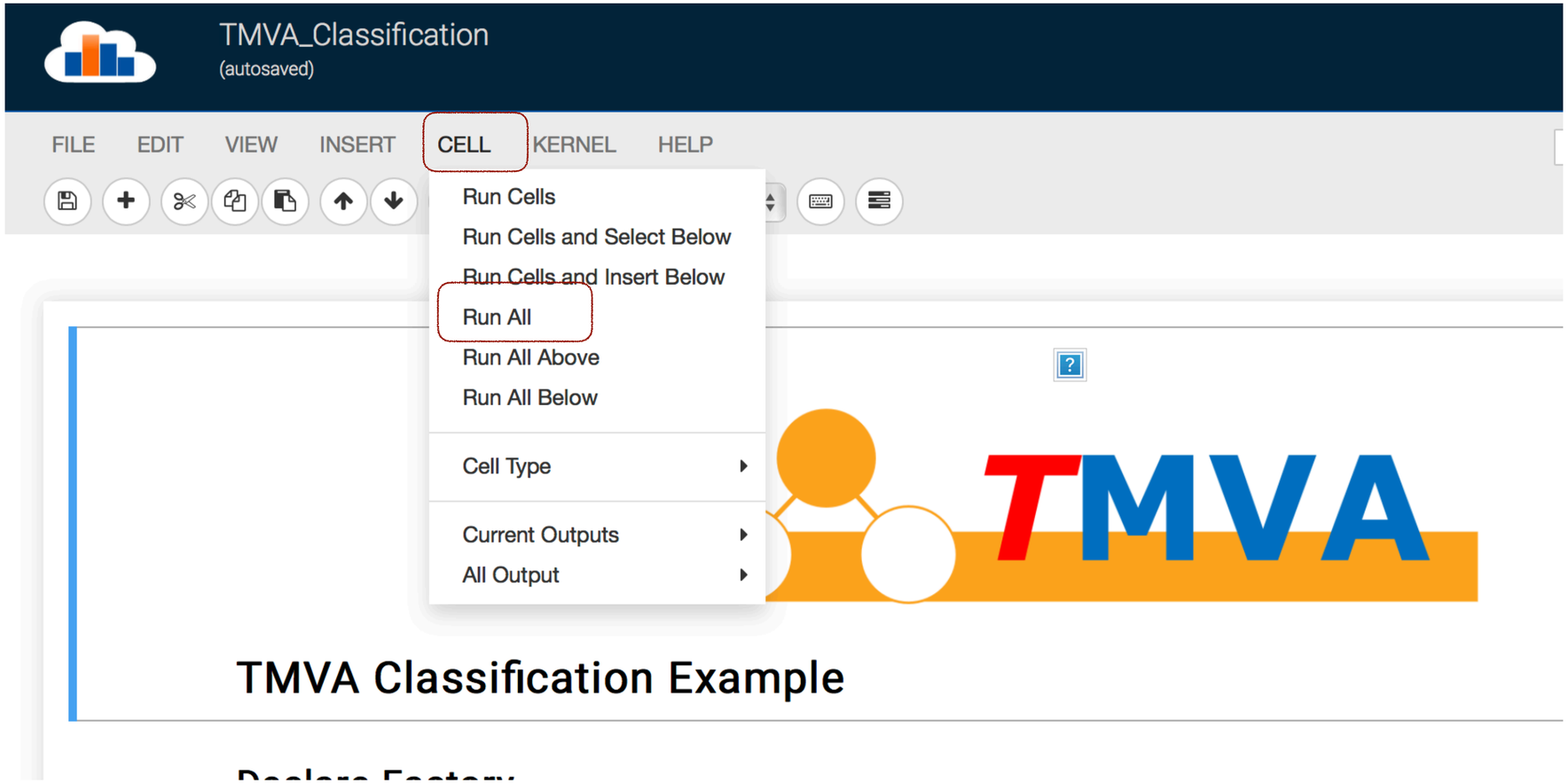
Create the Factory class. Later you can choose the methods whose performance you'd like to investigate.

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

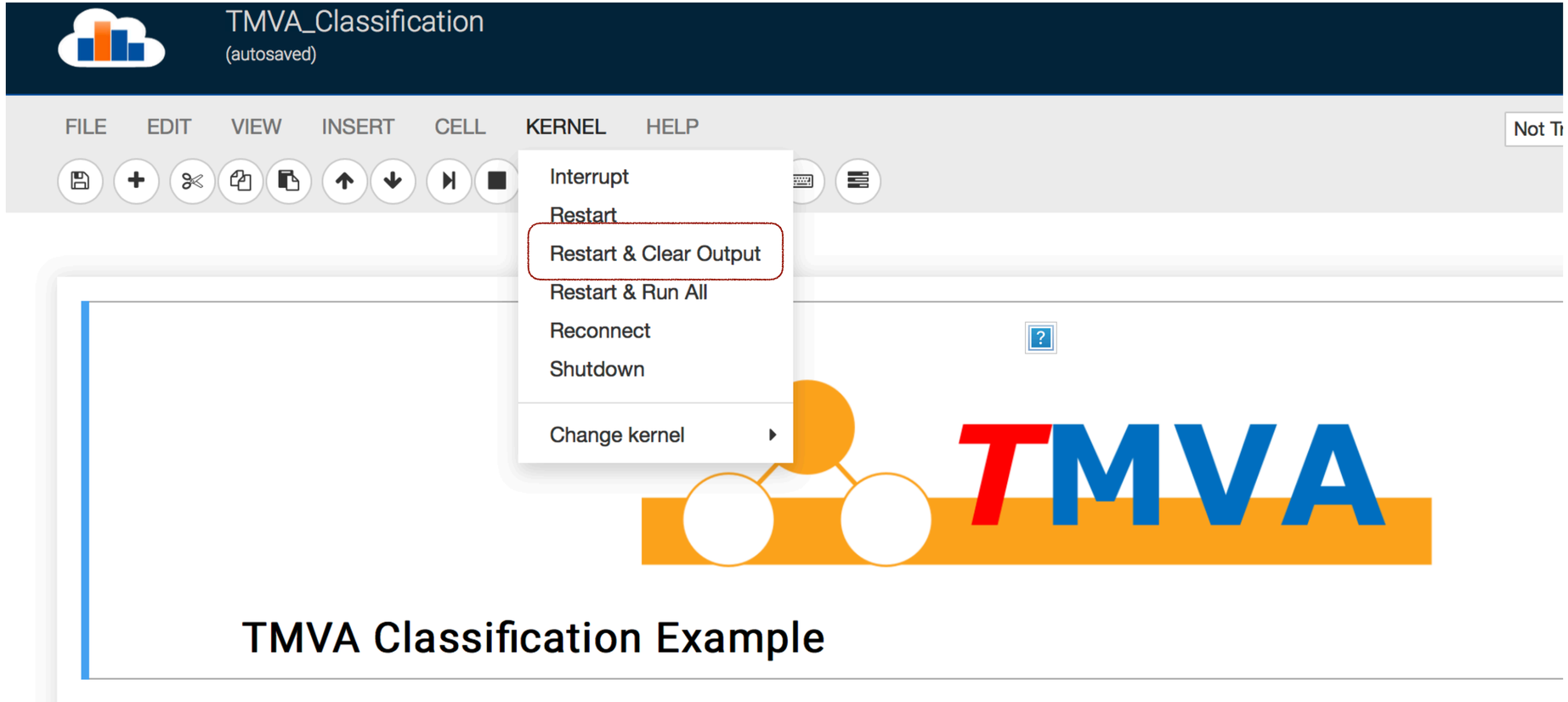
In [1]:
```
TMVA::Tools::Instance();
```

SWAN Commands: Run Cells



For running all cells

SWAN Commands: Kernel



For restarting Kernel in case of errors

Conclusions

- 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 us (TMVA developers) directly for any questions or issues

TMVA Contributors

Lorenzo Moneta	Algorithm development, Integration and support
Sergei Gleyzer	Analyzer Tools, Algorithm Development
Omar Zapata Mesa	PyMVA, RMVA, Modularity, Parallelization and Integration
Kim Albertsson	Multi-class for BDT, cross validation/evaluation and support
Stefan Wunsch	Keras Interface, integration, improved data handling
Peter Speckmeyer	Deep Learning CPU
Simon Pfreundschuh (GSOC 2016)	Deep Learning CPU and GPU
Vladimir Ilievski (GSOC 2017)	New Deep Learning module, Convolutional layers
Saurav Shekkar (GSOC 2017)	New Deep Learning module and Recurrent layers
Manos Stergiadis (GSOC 2018)	GPU support for CNN
Ravi Selvam (GSOC 2018)	New optimisers for deep learning
Adrian Bevan, Tom Stevenson	SVMs, Cross-Validation, Hyperparameter Tuning
Attila Bagoly (GSOC 2016)	Jupyter Integration, Visualization, Output
Paul Seyfert	Performance optimization
Andrew Carnes	Regression, Loss Functions, BDT Parallelization
Anurshee Rankawat, Siddhartha Rao, Harsit Prasad	GSOC 2008 projects: GAN, VAE and LSTM

And with continued invaluable contributions from Andreas Hoecker, Helge Voss, Eckhard v.Thorne, Jörg Stelzer