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)
- Main ML Tool until few years ago (e.g. 2013)
- Easy interface for beginners, powerful for experts
- Several active contributors and several features added recently (e.g. deep learning)
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
The available methods are (up-to 2015 version):

- 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 major features added since 2016 and available in the ROOT version 6.12:

- 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)
Most Recent Features

Features available in ROOT master and/or being released for the ROOT version 6.14:

- **Deep Learning Module with**
  - Convolutional Layer
  - Recurrent Layer
- **Improved Cross Validation**
- **Improved BDT performance using multi-thread parallelisation**
Deep Learning in TMVA

- Deep Learning library in ROOT/TMVA
  - parallel evaluation on CPU
    - implementation using OpenBLAS and TBB
  - GPU support
    - CUDA
    - OpenCL
  - Excellent performance and high numerical throughput

For more information see

Deep Learning Performance

DNN vs Standard ANN

- DNN: 5 hidden layers with 256 neurons
- SNN: 1 hidden layer

Using Higgs public dataset with 11M events

Significant improvements compared to shallow networks and BDT

DNN vs BDT

<table>
<thead>
<tr>
<th>Method</th>
<th>Training Time [h]</th>
<th>Area under ROC Curve</th>
</tr>
</thead>
<tbody>
<tr>
<td>BDT</td>
<td>4.78 h</td>
<td>0.806</td>
</tr>
<tr>
<td>DNN</td>
<td>1.46 h</td>
<td>0.876</td>
</tr>
</tbody>
</table>
Deep Learning Developments in TMVA

- Extend existing Deep Neural Network classes by adding:
  - **Convolutional Neural Network**
    - very powerful for image data sets
  - **Recurrent Neural Network**
    - useful for time-dependent data
  - **Deep Auto Encoder**
    - useful for dimensionality reduction (pre-processing tool)
    - can be used as unsupervised tool (e.g. for anomaly detection)
Convolutional Neural Network

- Integrated in ROOT master, for next ROOT release (6.14)
- Supporting now CPU parallelization, GPU support will come in the summer
- parallelisation and code optimisation is essential

Convolutional + Pooling + Dense layers

Background rejection versus Signal efficiency

input 32x32 images
Cross Validation in TMVA

- TMVA supports k-fold cross-validation

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

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

Important for regression performance

Higher is better
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://o проект.орг/RMVA](http://o проект.орг/RMVA)

- **PYMVA**: Python Interface
  - skikit-learn with RandomForest, Gradiend Tree Boost, Ada Boost)
  - see [http://o проект.орг/PYMVA](http://o проект.орг/PYMVA)

- **Keras** (Theano + Tensorflow)
  - support model definition in Python
  - see [https://indico.cern.ch/event/565647/contributions/2308668/attachments/1345527/2028480/29Sep2016_2ML_keras.pdf](https://indico.cern.ch/event/565647/contributions/2308668/attachments/1345527/2028480/29Sep2016_2ML_keras.pdf)

- Input data are copied internally from TMVA to Numpy array
Example PyMVA with Keras

Define model for Keras

```python
# 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 methods

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

```python
# Keras interface with previously defined model
factory.BookMethod(dataloader, ROOT.TMVA.Types.kPyKeras, 'PyKeras',
                   'H:IV: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

Background rejection versus Signal efficiency

MVA Method:
- DL_CPU
- PyKeras
- BDT
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 Customizations and Features

TMVA supports:

- ROOT Tree input data (or ASCII, 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
- 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:NAvePerBin=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;
}

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

4 Gaussian variable with linear correlations

\[ \begin{align*}
    x_1 &= v_1 + v_2, \\
    x_2 &= v_1 - v_2, \\
    x_3 &= v_3, \\
    x_4 &= v_4
\end{align*} \]

where \( \{v_1, ..v_4\} \) are normal variables
Pre-processing of the Input Variables

- Example: decorrelation of variable before training can be useful

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)
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)
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
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
Starting SWAN

SWAN Customisation

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-sl6-gcc62-opt

Environment script more...
- e.g. $CERNBOX_HOME/MySWAN/myscript.sh

Number of cores more...
- 2

Memory more...
- 8 GB

Select to use new Deep Learning

click here to start

Start my Session
Starting a Terminal in SWAN

After login cernbox home directory will be visible

Start a terminal window
Getting the Notebooks

- Clone the git repository of the tutorials
  https://github.com/lmoneta/tmva-tutorial.git
- get directly the **IML-tutorial-2018** branch
- `git clone --branch IML-tutorial-2018 https://github.com/lmoneta/tmva-tutorial.git`

If directory already exists delete it before or update its git repository doing:
`git fetch; git checkout -b IML-tutorial-2018 origin/IML-tutorial-2018`

- Go back to SWAN Home page and select the directory `tmva-tutorial/tutorial_IML2018`
- Start using the notebooks
Notebooks

1. Select items to perform actions on them.
   - tmva-tutorial

2. Select items to perform actions on them.
   - tutorial_IML2018

3. Select items to perform actions on them.
   - TMVA_Classification.ipynb
   - TMVA_CrossValidation.ipynb
   - TMVA_Higgs_Classification.ipynb
   - TMVA_Reader_py.ipynb
   - TMVA_Regression.ipynb
   - TMVAGuiPlots.ipynb
   - TMVAGuiROC.ipynb
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 weight files 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();
```
Outlook for TMVA

- Very active development happening in TMVA (thanks to contribution from doctoral students, summer students and Google Summer of Code students)
  - e.g. New Deep Learning tools
- Important to have a rich set of modern ML tools in ROOT and at the same time provide interfaces to popular external libraries
- New planned developments (GSOC projects for 2018)
  - Complete Deep Learning module (add GPU support)
  - GAN for fast simulation
  - Direct interface to Tensorflow
- Improve big data (I/O) handling for ML
  - avoid un-needed extra data copies and optimise memory usage
  - better interface to external tools
Conclusions

- Very active development in TMVA
  - expect several new features in next release
- Feedback from users is essential
  - we are defining ROOT plan of work for next year, requests and feedback from experiments will be taken into account
- Users contributions are extremely important
  - best way to contribute is opening a Pull Request in GitHub (https://github.com/root-project/root)
- For support use ROOT Forum: https://root.cern.ch/phpBB3/
  - with categories dedicated on TMVA
- For reporting ROOT bugs: https://sft.its.cern.ch/jira
- or just contact us directly
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
- Peter Speckmeyer: Deep Learning CPU
- Simon Pfreundschuh: Deep Learning CPU and GPU
- Vladimir Ilievski: New Deep Learning module, CNN layers
- Saurav Shekhar: New Deep Learning module and Recurrent layer
- Akshay Vashistha: Deep Auto Encoder development
- Mark Huwiler: Deep Auto Encoder and Deep Learning integration tests
- Mammad Hagili: Parallelisation of Cross-Validation
- Adrian Bevan, Tom Stevenson: SVMs, Cross-Validation, Hyperparameter Tuning
- Attila Bagoly: Jupyter Integration, Visualization, Output
- Paul Seyfert: Performance optimization
- Andrew Carnes: Regression, Loss Functions, BDT Parallelization

- Continued invaluable contributions from Andreas Hoecker, Helge Voss, Eckhard v. Thorne, Jörg Stelzer, and key support from CERN EP-SFT Group