



Google Summer of Code Project

Jupyter and TMVA

Attila Bagoly (Eötvös Loránd University, Hungary)

Mentors:

- **Sergei V. Gleyzer**
- **Enric Tejedor Saavedra**



ROOT jupyter

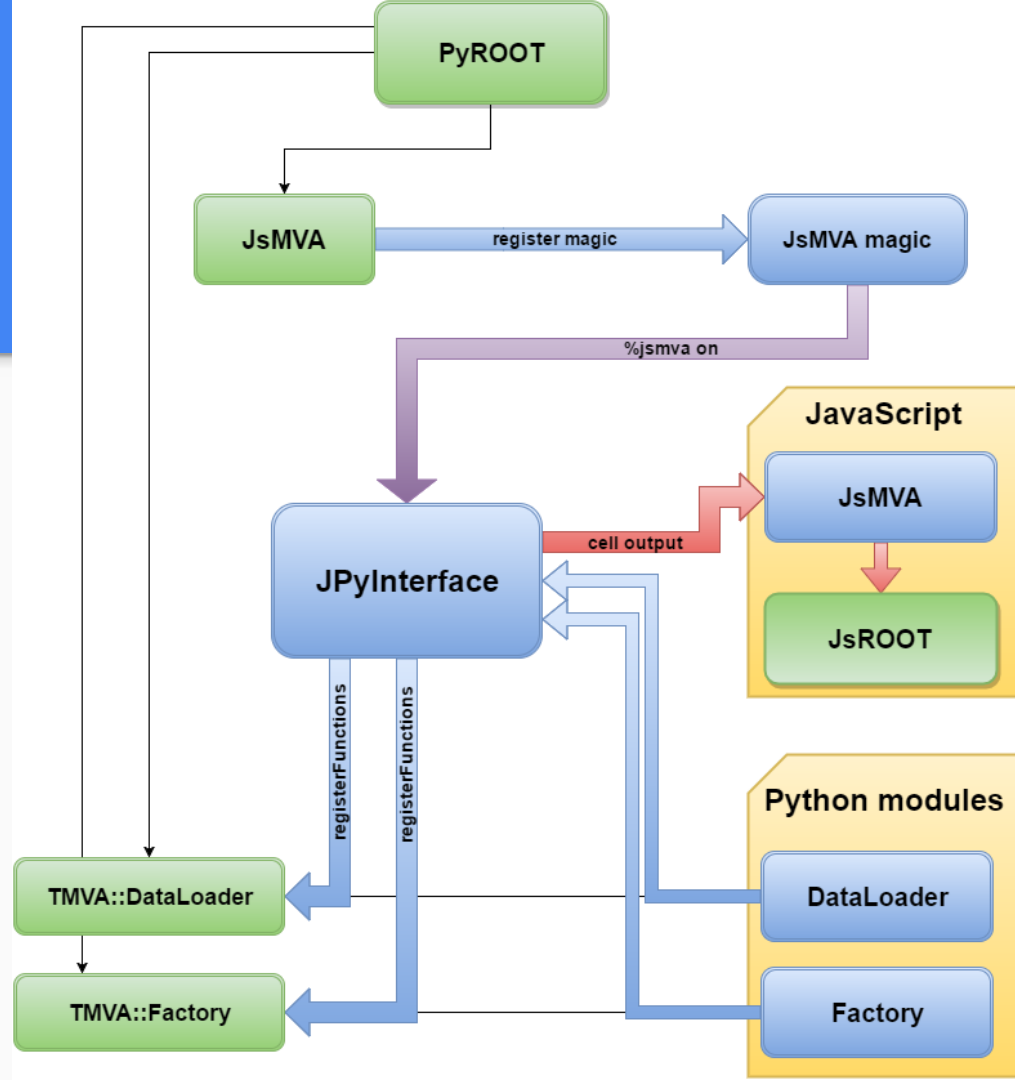
Data Analysis Framework



- Jupyter notebook:
 - Interactive coding
 - Document: HTML, Markdown support
 - Shareable: SWAN, nbviewer, binder
- Current status of TMVA:
 - We can't use TMVAGui
 - We can read back the classifier outputs and we can make visualizations.
 - **BUT users don't want to spend time with making visualizations**
- Integrating TMVA in Jupyter:
 - Support for TMVAGui in notebooks
 - New visualizations
 - Pythonic interface for a bunch of functions
 - Interaction: changes modify the state of TMVA
 - HTML formatted output

Code structure

- Importing ROOT will import **JsMVA**, this will register jsmva magic
- `%jsmva on`: JPyInterface inserts new methods to TMVA.DataLoader and TMVA.Factory, overloads some functions with a wrapper, register HTML transformer function
- New methods: inserting HTML to cell output, with JavaScript call for JsMVA.js
- JsMVA.js using **JsROOT** to create JavaScript plots



TMVAGui visualizations

```
loader.DrawCorrelationMatrix("Signal")
```

Correlation Matrix (Signal)



Visualizations related to input variables

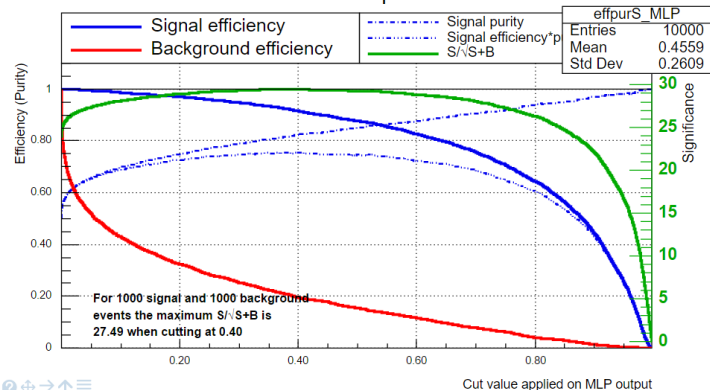
- Correlation matrix
- Input variables
- Transform input variables & show

Visualizations related to classifier outputs

- ROC curve
- Output distributions
- Cut efficiencies

```
factory.DrawCutEfficiencies(dataset, "MLP")
```

Cut efficiencies and optimal cut value



Classifier output: Neural networks, decision trees

Simple neural network

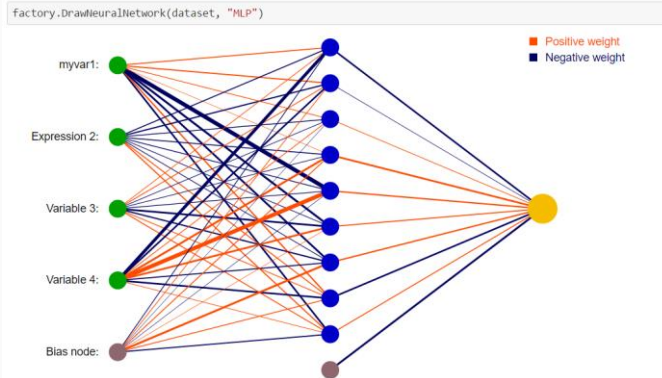
- Python function reads the network, converts to JSON; JS with d3js make the visualization from JSON
- Interactive: focusing connections, zooming, moving

Deep neural network

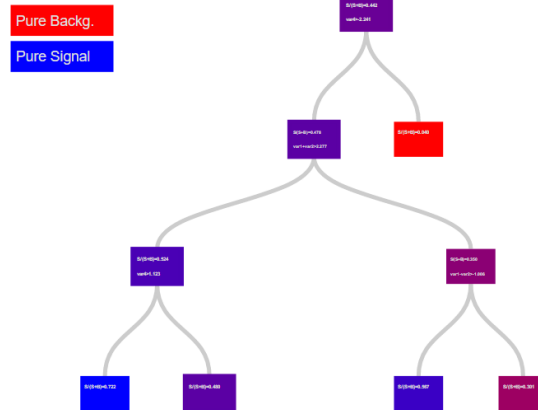
- HTML5 Canvas visualization (speed)
- Less interactive: zooming, moving

Decision trees

- Ipywidgets: input field for selecting the tree
- Visualization from JSON with D3js
- Interactive: closing subtree, showing the path, focusing, moving, zooming, reset



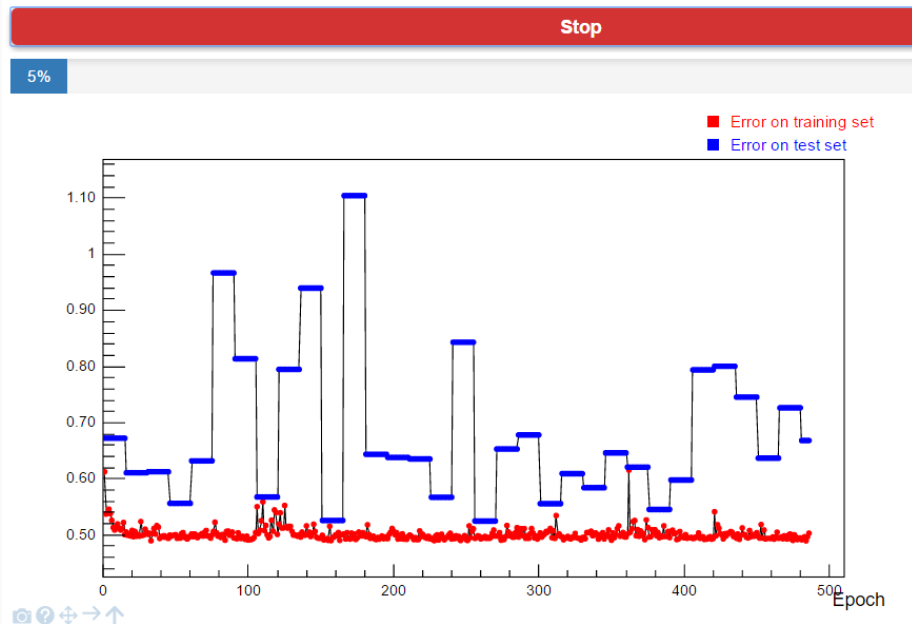
factory.DrawDecisionTree(dataset, "BDT") #11



Interactive training mode

- C++ interface for tracking/stopping the training
- New thread for training
- Main thread periodically refreshes the plot (inserts small JS script, which removes itself)
- Error plots supported for MLP, DNN, BDT methods
- Progress bar for a bunch of methods
- Stop button: by clicking on it the main thread will send stop message for training loop (just the loop, no interfere with saving the net, or other data)

Train method: DNN



Train method: Cuts



Pythonic user interface, tutorial notebooks

```
factory = TMVA.Factory(JobName="TMVAClassification", TargetFile=outputFile,  
                       V=False, Color=True, DrawProgressBar=True, Transformations=["I", "D", "P", "G", "D"],  
                       AnalysisType="Classification")
```

New interface

- Option strings not very nice, we can do better in python
- Bunch of functions use option string
- Wrapper functions for them, with jsmva magic these functions are replaced with corresponding wrapper function
- The settings can be passed by named arguments: `V=True, Transformations=["I", "D"]` will be translated to `!V:Transformations=I,D`

Arguments of constructor: The options string can contain the following options:

Keyword	Can be used as positional argument	Default	Predefined values	Description
JobName	yes, 1.	not optional	-	Name of job
TargetFile	yes, 2.	if not passed histograms won't be saved	-	File to write control and performance histograms histograms
V	no	False	-	Verbose flag
Color	no	True	-	Flag for colored output
Transformations	no	""	-	List of transformations to test. For example with "I;D;P;U;G" string identity, decorrelation, PCA, uniform and Gaussian transformations will be applied
Silent	no	False	-	Batch mode: boolean silent flag inhibiting any output from TMVA after the creation of the factory class object
DrawProgressBar	no	True	-	Draw progress bar to display training, testing and evaluation schedule (default: True)
AnalysisType	no	Auto	Classification, Regression, Multiclass, Auto	Set the analysis type

Tutorial:

<http://nbviewer.jupyter.org/github/qati/GSOC16/blob/master/notebooks/ROOTbooks-TMVA-JsMVA-UserInterface.ipynb>

Deep neural network builder

- Booking DNN confusing: lot of settings, everybody forgets the exact names
- Graphical interface: booking DNN with pleasure
- We can add different types of layers
- Specify the neuron number and training strategy for layer
- Connect the layers: building the network
- Save network: transform the graphical representation to option string and books the method

The screenshot displays a graphical user interface for building a deep neural network. At the top, the title bar reads "factory.BookDNN(loader)". Below it is a menu bar with options: "Global options", "Add layer", "Connect layers", "Scale colors", and "Save network".

The main workspace shows a network diagram with three layers:

- Input layer**: A green rounded rectangle.
- Options TANH**: A purple rounded rectangle connected to the input layer.
- Options ReLU**: A red rounded rectangle connected to the TANH layer.
- Output layer**: A yellow rounded rectangle connected to the ReLU layer, containing an "Options" sub-label.

Three configuration panels are overlaid on the interface:

- Global options**: Includes checkboxes for "Verbose", "Help messages", "CreateMVAPdfs", and "IgnoreNegWeightsInTraining". It also features dropdowns for "VerbosityLevel" (Default), "ErrorStrategy" (CROSSENTROPY), and "WeightInitialization" (XAVIER). Text input fields are present for "SignalWeightsSum" (1000) and "BackgroundWeightsSum" (1000).
- Add neurons**: Shows a "Number of neurons" input field set to 0 and a "Training Strategy" button.
- Training Strategy**: A table of parameters:

LearningRate	0.00001
Momentum	0.3
Repetitions	3
ConvergenceSteps	100
BatchSize	30
TestRepetitions	7
WeightDecay	0
Regularization	NONE
DropConfig	
DropRepetitions	3
Multithreading	<input checked="" type="checkbox"/>

HTML formatted output

- jsmva magic register output transformer function to JupyROOT
- also inserts CSS (for transformed output tables style) to notebook
- The output transformer class:
 1. Regular expressions for matching different output lines => logical units
 2. Based on matchings the output is transformed to table structure (style: CSS)
 3. Correlation matrix: table => histogram => JsROOT visualization in popup window (jquery)

Number of training and testing events		
Signal	training events	2400
	testing events	600
	training and testing events	3000
Background	training events	2400
	testing events	600
	training and testing events	3000

DataSetInfo	Correlation matrix (Signal)
DataSetInfo	Correlation matrix (Background)
DataSetFactory	Dataset: BDT_fold4
Factory	Train method: BDT for Classification
BDT	#events: (reweighted) sig: 2400 bkg: 2400
	#events: (unweighted) sig: 2400 bkg: 2400
	Training 2 Decision Trees ... patience please
	Elapsed time for training with 4800 events : 0.0133 sec
BDT	Dataset: BDT_fold4 Evaluation of BDT on training sample (4800 events)
	Elapsed time for evaluation of 4800 events : 0.00358 sec
	Creating xml weight file: BDT_fold4/weights/TMVAClassification_BDT.weights.xml
	Creating standalone class: BDT_fold4/weights/TMVAClassification_BDT.class.C

Everything on GitHub:

<https://github.com/qati/GSOC16>

Notebooks on nbviewer (static, rendered):

<http://nbviewer.jupyter.org/github/qati/GSOC16/blob/master/index.ipynb>

Notebooks on binder (interactive):

www.mybinder.org/repo/qati/GSOC16

Or you can download:

<https://github.com/qati/GSOC16/tree/master/notebooks>

Thank you for your
attention!