Machine Learning with ROOT/TMVA

Stefan Wunsch for the ROOT team

ROOT
Data Analysis Framework

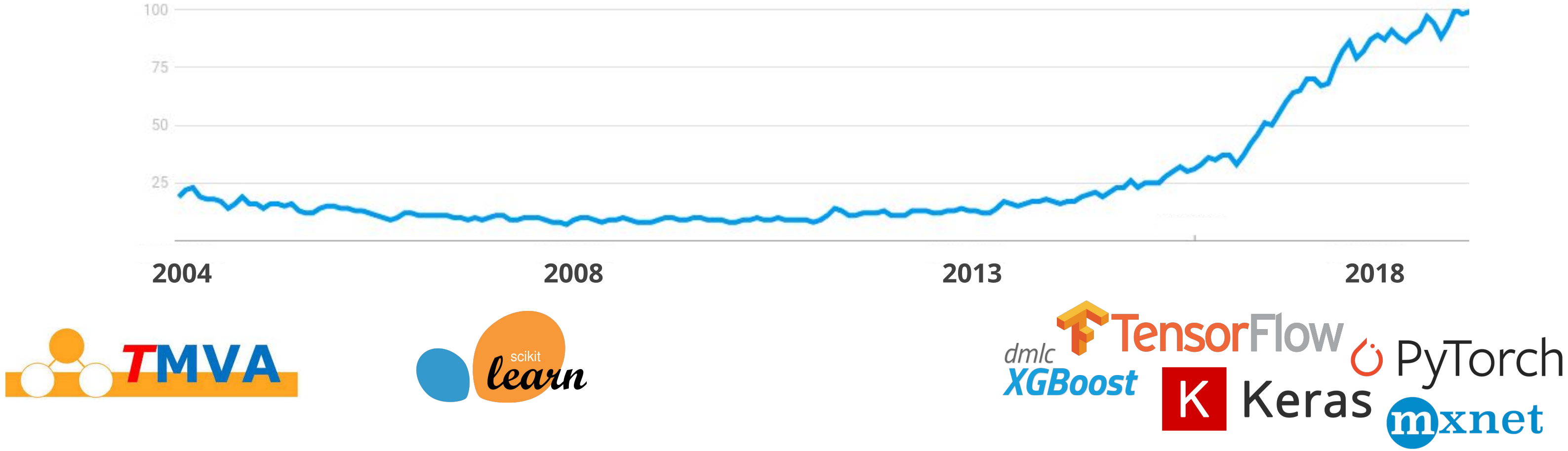
https://root.cern

TMVA in the current ML landscape

- TMVA provides implementations of a vast amount of ML methods collected over the last decade

- Todays developments from the industry shifts the scope of TMVA towards specialization on HEP specific requirements

Google Trends: machine learning



Interoperability with the ML ecosystem

ROOT 6.16

- **Crucial feature for ML**
  ML is Python centric
  → Crucial to move data from ROOT files to Python and vice versa

- Writing numpy arrays supported through `MakeNumpyDataFrame` feature

- Further information about the interoperability of ROOT with the scientific Python ecosystem in our poster:
  A new PyROOT: Modern, Interoperable and more Pythonic

```
# Heavy-lifting in C++ and remote access of data
df = ROOT.RDataFrame("Events", "http://file.root")
         .Filter("x1 > 0")
         .Define("x3", "x1 * x2")

# Read-out as numpy arrays
vars = ("x1", "x2", "x3")
cols = df.AsNumpy(vars)

# Create typical ML input data structure
x = numpy.stack([cols[v] for v in vars])

# Push data to scipy ecosystem
pdf = pandas.DataFrame(cols)
```

Modern interfaces

ROOT 6.20
(experimental)

- **Modern high-level interfaces**
  - Functional
  - Thread-safe
  - Support C++ STL container
  - Full C++ and Python support
  - Example tutorial available

- Introduce RTensor as replacement for missing container of multi-dimensional arrays in C++
  - Tutorials available ([1], [2])
  - Keep track of ML sub-group in Standard C++ Foundation

- RTensor allows for seamless integration with numpy arrays in Python
  - Interoperable with ML ecosystem

```
// Construct model
TMVA::RBDT bdt("myBDT", "model.root");

// Single-event inference
auto y = bdt.Compute({1.0, 2.0, ...});

// Batch inference
TMVA::RTensor<float> x(data, shape);
auto y2 = bdt.Compute(x);
```

C++ SG19, Machine Learning: Improve on C++'s ability to support [...] array, matrix, linear algebra, [...]

Integration with modern ROOT facilities

- Integration with ROOT’s implicit multi-threading paradigm
  - ROOT::EnableImplicitMT()
  - Correct sharing of resources
  - Already supported by TMVA::DNN and method BDT
- Tight integration with ROOT::RDataFrame
- Each method is standalone but follows a common interface
  - sklearn-like paradigm
  - Simple integration in modern C++

```
// Run workflow on multiple threads
ROOT::EnableImplicitMT();

// Construct model
TMVA::RBDT bdt("myBDT", "model.root");

// Process data in parallel using RDataFrame
ROOT::RDataFrame df("Events", "file.root");
auto df2 = df.Define("bdt_output",
                  TMVA::Compute<2, float>(bdt),
                  {"var1", "var2"});
```

Fast decision tree inference

- Inference engine taking model parameters from externally trained models

- **Features**
  - Simple to use from Python and C++
  - Thread-safe
  - Zero-copy
  - Fast for single event and batch inference

- **Coming soon**
  - Multi-threading support for batch inference
  - Additional converters for external frameworks

ROOT 6.20 (experimental)

External training and model conversion

```
xgb = xgboost.BDTClassifier(options)
xgb.fit(x, y)

ROOT.TMVA.SaveXGBoost(xgb, "myBDT", "model.root")
```

Python application

```
bdt = ROOT.TMVA.RBDT("myBDT", "model.root")
x = numpy.array(...)
y = bdt.Compute(x)
```

C++ application

```
TMVA::RBDT bdt("myBDT", "model.root");
auto y1 = bdt.Compute({1.0, ...});

auto x = TMVA::RTensor<float>(data, shape);
auto y2 = bdt.Compute(x);
```

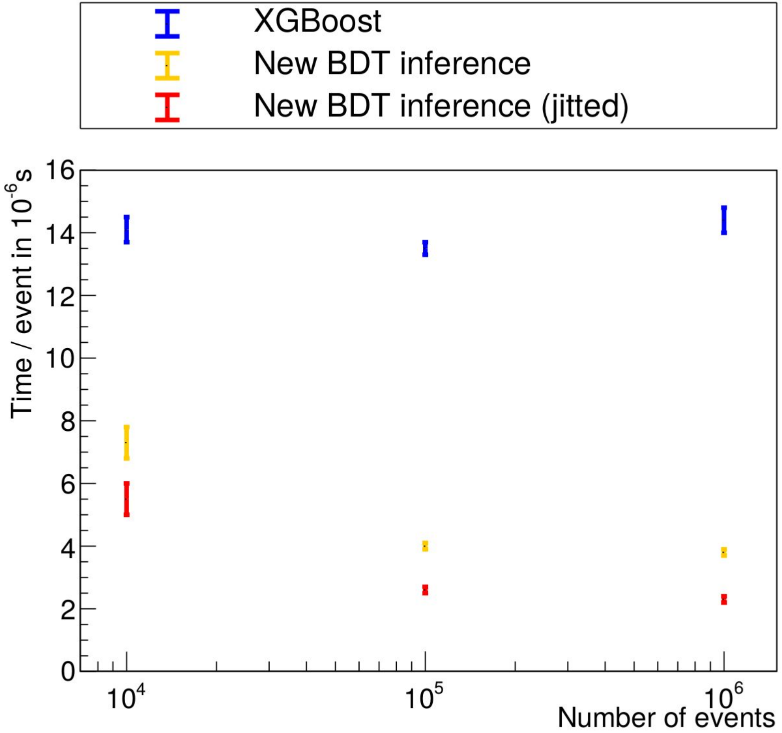
Fast BDT inference: Performance

- Performance measurement of a model with
  - 500 trees
  - 3 maximum depth
  - 10 input variables

- **Leverages successfully just-in-time compilation**
  - Using cling with optimization level 3
  - Optimize inference code at construction time based on model parameters

- Improved runtime performance in Python workflow compared to XGBoost
  - Batch evaluation on a single thread
  - 4x faster than XGBoost for 10^6 events
  - Jitting provides additional 40% speed-up improving to 6x faster inference

- **See our poster for the technical details**
  ROOT/TMVA in the evolving machine-learning landscape: Fast inference and modern interfaces



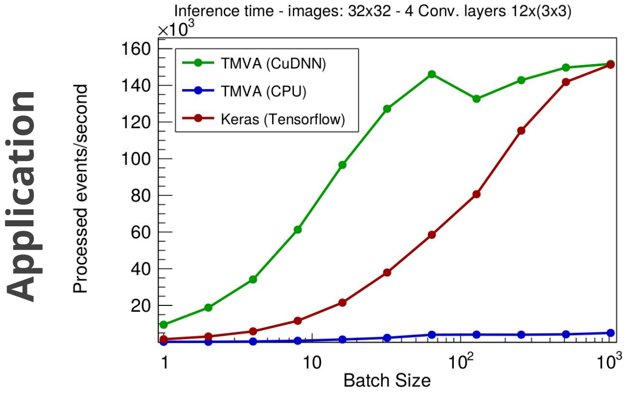
Fast neural networks

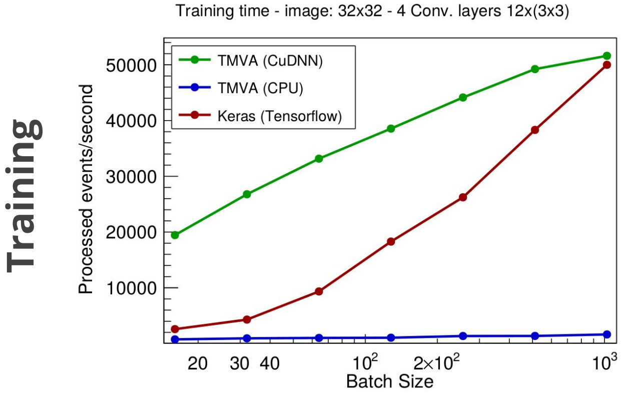
- Main focus of the industry tools
  - Large models
  - Batch inference
  - Fast training workflow
  - Accessible through Python ecosystem

- Focus of TMVA in upcoming developments
  - Minimal latency / fast single event inference
  - Seamless integration in Python and C++
  - Sustainability and reproducibility
  - **See our poster for more details**
    ROOT/TMVA in the evolving machine-learning landscape: Fast inference and modern interfaces

- New developments for neural networks
  - Integration of cuDNN
  - Support for LSTM and GRU layers

ROOT 6.20





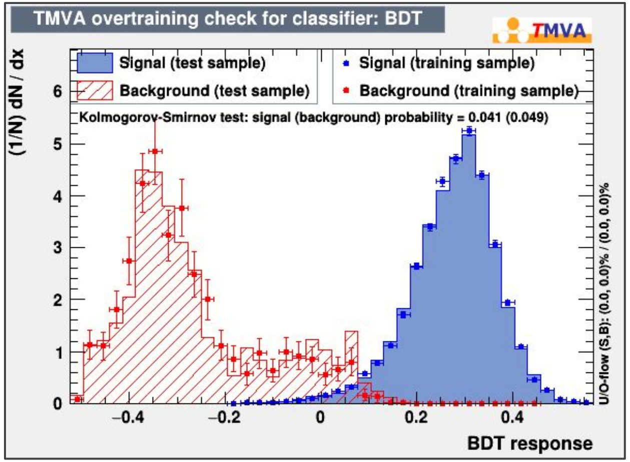
Outlook

- **Strategy of future TMVA developments**
    - Modularize
    - Interoperate with the ML ecosystem
    - Specialize on HEP peculiarities

Future developments

- **Example** Modernize TMVA GUI
    - Move from monolithic design to modular toolbox of visualization tools
    - Example HEP peculiarity:
      Statistical comparison of distributions
- **Example** Generic data-loader for ML workflows
    - Generator doing batching and shuffling from ROOT files on the fly
    - Allows for training on huge datasets

Typical TMVA GUI visualization



Example ML workflow loading batches

```
df = ROOT.RDataFrame("Events", "http://file.root")
generator = TMVA.BatchGenerator(df, cols, batchSize)
for step in gradientSteps:
    x = generator()
    model.fit(x)
```

Summary

- **New features**
  - Modern interfaces for inference
  - Integration with modern ROOT facilities
  - Fast inference for decision trees
  - Handling of multi-dimensional arrays in C++ and interoperability with Python
  - Facilitate integration with the ML ecosystem

- **Strategy of future TMVA developments**
  - Modularize
  - Interoperate with the ML ecosystem
  - Specialize on HEP peculiarities

- **Tutorials showing a full ML workflow using the new tools**
  - Data loading and preprocessing
  - External training and model conversion
  - Testing and application in Python
  - Application in C++