# BENCHMARKING OF ROOT BOOSTED DECISION TREES AND XGBOOST

STUDENT: XANDRU MIFSUD

SUPERVISOR: DR. LORENZO MONETA

## 1. Problem Definition

ROOT is a data analysis framework used in many different experiments across CERN, and beyond. The TMVA library, which forms part of ROOT, is in particular a machine learning toolkit for multivariate analysis. In particular, amongst its various machine learning tools, it provides a robust *boosted decisions trees* (BDT) implementation. The primary task at hand was then to benchmark the CPU and memory performance of TMVA's BDT implementation, and see how it fairs.

Ideally, we would benchmark against some known implementation which is considered to be the 'gold standard', which in this case is *XGBoost*. Different machine learning libraries have different specifications on how data is to be formatted for input. To this extent, besides benchmarking, we also required a means by which we can convert training and testing data–sets prepared by TMVA into a format which is readable by XGBoost.

Lastly, we required that any written benchmarks should employ the use of *Google Benchmark* and be integrated into *rootbench*, a benchmarking library for ROOT.

## 2. Data–set Preparation

The problem of converting from TMVA prepared datasets to ones which are readable by XGBoost was considered before: a `pyROOT` script authored by [Wunsch, 2019], which reads the signal and background data from separate ROOT files, does exactly this. Since we wish to integrate our benchmarks into `rootbench`, ideally we move away from `pyROOT`. Hence a `C++` variant was implemented.

Amongst other responsibilities, a `TMVA::DataLoader` instance is responsible for splitting the input signal and background data–sets into respective training and testing subsets. These are maintained as `TMVA::Event` vectors in a `TMVA::DataSet` instance[1]. Also, the signal and background data are not separate in these vectors (i.e. the only 'distinction' made is between testing and training subsets). Hence we require a 'conversion' script that interfaces with `TMVA::DataSetInfo` instances (rather than `ROOT::RDataFrame`, as in the `pyROOT` script).

[1]Note that this is generally maintained by a `TMVA::DataSetInfo` instance, which in particular maintains further meta–data and provides a number of convenience functions.

In the end, two 'conversion' functions were implemented: one which interfaces with a `TMVA::DataSetInfo` instance, and another which interfaces with separate `ROOT::TTree` instances for the signal and background data–sets (similar to the existing `pyROOT` script). We maintain all the necessary XGBoost readable data in a custom–defined `struct`–type called `xgboost_data`.

```
xgboost_data* ROOTToXGBoost(const TMVA::DataSetInfo& dataset_info,
↪  TMVA::Types::ETreeType type);

xgboost_data* ROOTToXGBoost(TTree& signal_tree, TTree& background_tree,
↪  vector<string>& variables, const float* sig_weight = nullptr, const
↪  float* bgd_weight = nullptr);
```

Also implemented was a simple XGBoost trainer using the `C` API and our `xgboost_data struct`–type. Note that in XGBoost literature, the terms 'trees' and 'iterations' may be used interchangeably, since by default it uses one tree per iteration.

```
BoosterHandle xgboost_train(xgboost_data* data, xgbooster_opts* opts, UInt_t
↪  n_iter){
    // Initialise a BoosterHandle instance
    BoosterHandle booster;
    safe_xgboost(XGBoosterCreate(data->sb_dmats, 1, &booster))

    // For each specified option, set the booster parameters...
    for(auto opt: *opts){
        safe_xgboost(XGBoosterSetParam(booster, opt.first, opt.second))
    }

    // Then train the booster for the specified number of iterations
    for(UInt_t iter = 0; iter < n_iter; iter++){
        safe_xgboost(XGBoosterUpdateOneIter(booster, iter,
        ↪  (data->sb_dmats)[0]))
    }

    return booster;
}
```

2.1. **Example Usage.** The code snippet below shows an example usage of the conversion functions and the simple trainer, analogous to the script written by Wunsch.

```
const std::string filepath =
↪  "http://root.cern.ch/files/tmva_class_example.root";
const std::vector<std::string> variables = {"var1", "var2", "var3", "var4"};
```

```
3
4  // Open specified root file and load signal/background trees
5  auto data = TFile::Open(filepath.c_str());
6  auto signal = (TTree *)data->Get("TreeS");
7  auto background = (TTree *)data->Get("TreeB");
8
9  // Add variables and register the trees with the dataloader
10 auto dataloader = new TMVA::DataLoader("tmva003_BDT");
11 for (const auto &var : variables) {
12     dataloader->AddVariable(var);
13 }
14 dataloader->AddSignalTree(signal, 1.0);
15 dataloader->AddBackgroundTree(background, 1.0);
16 dataloader->PrepareTrainingAndTestTree("", "");
17
18 // Convert to an XGBoost readable format
19 xgboost_data* xg_train_data =
   ↪  ROOTToXGBoost(dataloader->GetDefaultDataSetInfo(),
   ↪  TMVA::Types::kTraining);
20
21 // Set the booster parameters
22 xgbooster_opts opts;
23 opts.push_back(kv_pair("max_depth", "3"));
24 opts.push_back(kv_pair("nthread", "1"));
25 opts.push_back(kv_pair("eta", "0.01")); // Sets XGBoost's learning rate
   ↪  similar to that of TMVA's BDT implementation i.e. the two are
   ↪  comparable
26
27 // And then finally, train...
28 BoosterHandle xgbooster = xgboost_train(xg_train_data, &opts, 500);
```

2.2. `TMVA::Event` **Vector Filtering.** The work above was somewhat expanded to a more general context, in order to allow the filtering of `TMVA::Event` vector instances held in a `TMVA::DataSet` instance. Filtering functions were implemented to extract subsets of the `Events` collection, filtering by whether we wish to keep signal or background events, and then the type of event (training, testing or both).

```
1  void GetSignalEventCollection(Types::ETreeType type,
   ↪  std::vector<TMVA::Event*>* filtered_events) const;
2  void GetBackgroundEventCollection(Types::ETreeType type,
   ↪  std::vector<TMVA::Event*>* filtered_events) const;
```

Further to this, utility functions for obtaining a 2–dimensional `float` matrix representation of these sub–collections were implemented. The aim was to provide

a common 'starting point' for conversion utilities to go from a TMVA representation to some other machine learning library representation, not necessarily just XGBoost.

```
Float_t** GetSignalMatrix(TMVA::Types::ETreeType type, Long64_t* dim1,
↪  Long64_t* dim2) const;
Float_t** GetBackgroundMatrix(TMVA::Types::ETreeType type, Long64_t* dim1,
↪  Long64_t* dim2) const;
```

Note that if the `TMVA::DataSet` instance has $m$ registered variables, then each `TMVA::Event` instance associated with the `TMVA::DataSet` instance has $m$ variables. If the filtered data set has $n$ `TMVA::Event` instances, then the 2–dimensional `float` matrix representation is an $n \times m$ matrix: each row corresponds to a `TMVA::Event` instance, while each column corresponds to a variable.

it is worth mentioning that similar filtering capabilities may be achieved using `RDataFrame` and `RTensor`. The aforementioned utility functions may be found at https://github.com/xmif1/root/tree/xmif1_event_filtering, along with the example script `tutorials/tmva/TMVAEventFiltering.C`.

## 3. Benchmarking

Recall that the end goal was to benchmark TMVA's BDT implementation against XGBoost's, with respect to CPU and memory usage. A pull–request was issued to extend `rootbench` with training and testing benchmarks for TMVA's BDT implementation. Note that in order to eliminate introducing XGBoost as a dependency for `rootbench`, the XGBoost training and testing benchmarks were implemented in a separate repository, containing the 'scaffolding' code of `rootbench` and having the `CMake` build system modified to include XGBoost.

The pull request extending `rootbench` may be found at https://github.com/root-project/rootbench/pull/231, while the repository extending the benchmarks to XGBoost may be found at https://github.com/xmif1/BDTBench.

Note that benchmarks are carried out for different numbers of trees, tree–depth, and threads. The benchmarks can easily be altered to either add further parameters to benchmark, or to change the parameter values tested for those already selected.

### 3.1. Limitations.

We suspect that due to the nature of Google Benchmark, only the first 'run' of a benchmark reports the correct memory use, i.e. we only report memory usage for one combination of the aforementioned parameters (in particular, one with the highest tree–depth from all tested combinations). We believe this is related to the memory management Google Benchmark carries out in–between different executions of a benchmark. Indeed, in any case, the reported memory benchmarks should be considered only as a guideline: this aspect of the project is still a work–in–progress.

Secondly, it was observed that as the number of threads was increased for XGBoost, the CPU usage performance grew worse. This should be investigated further

at later stages.

Lastly, it should be noted that further tuning of the parameters for XGBoost's trainer and predictor need to be carried out, in order to ensure that the two implementations (TMVA's BDT and XGBoost) are compared on an 'equal footing'.

3.2. **Preliminary Results.** The table below gives some preliminary CPU and memory benchmark results, which can be extracted from the benchmark tools developed. The final limitation mentioned in the previous sub–section should be taken into consideration when interpreting these results.

These benchmarks are all for data–sets with 500 events and 4 variables for both the signal and background data, and with the respective trainers having 2000 iterations, a tree depth of 10, and running in single–threaded mode.

| Benchmark | Time | CPU | Resident Memory |
|---|---|---|---|
| `BM_TMVA_BDTTraining` | 4225615895 ns | 3934167687 ns | 40k |
| `BM_XGBOOST_BDTTraining` | 565870131 ns | 562560264 ns | 24k |
| `BM_TMVA_BDTTesting` | 662881736 ns | 662755953 ns | 120k |
| `BM_XGBOOST_BDTesting` | 142666450 ns | 142665076 ns | (?) 0k |

## References

[Wunsch, 2019] Wunsch, S. (2019). tmva101_training.py. `https://root.cern/doc/master/tmva101__Training_8py.html`.

*Email address*: `xmif0001@um.edu.mt`