The RDF $t\bar{t}$-analysis implementation
Analysis Grand Challenge

IRIS-HEP Fellow: Andrii Falko,
Taras Shevchenko National University of Kyiv
Mentors: Enrico Guiraud, Alexander Held
Introduction

• Analysis Grand Challenge
  • Developing and testing workflows envisioned for the HL-LHC
  • Showing the performance and scalability
  • An environment where IRIS-HEP technologies and the adjacent ecosystem are connected

• Specification of a physics analysis
  • Public data
  • Represents HL-LHC requirements
  • agc.readthedocs.io

• IRIS-HEP’s reference implementation
  • based on the coffea framework
  • heavy use of libraries from the Scikit-HEP
  • iris-hep/analysis-grand-challenge

• RDataFrame & AGC project
  • root-project/analysis-grand-challenge
RDF & AGC project

• Implementation of Analysis Grand Challenge with ROOT’s modern analysis interface

• RDataFrame
  • flexibility to express virtually any HEP analysis
  • allows execution of any C++ code
  • seamless scaling out to computing clusters

• Project’s goals for summer
  • Update RDataFrame implementation to AGC v1
  • Update RDataFrame implementation to AGC v2
**t\bar{t}-analysis specification**

1. **Analysis task:** t\bar{t} cross-section measurement

2. **Input dataset:** 2015 CMS Open Data

3. **Event selection**
   - Semi-leptonic decay channel
   - 2 jets 2 b-jets 1 lepton

4. **Account for weights**

5. **Calculation of observables**

6. **Systematic uncertainties**
   - Jet kinematics variations
   - Event-weights variations

7. **Machine Learning Component**
   - Jet-parton assignment
   - BDT

---

**Reconstructed top mass**

![Reconstructed top mass histogram](image)
### Moving to the latest AGC version

<table>
<thead>
<tr>
<th>V</th>
<th>Data schema</th>
<th>Selection cuts</th>
<th>Machine learning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>jets</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>POET</td>
<td>( P_T &gt; 25 \text{ GeV} )</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>NanoAOD</td>
<td>( P_T &gt; 25 \text{ GeV} ) ( b\text{-tag} &gt; 0.5 )</td>
<td>None</td>
</tr>
<tr>
<td>2</td>
<td>NanoAOD</td>
<td>( P_T &gt; 30 \text{ GeV} ) (</td>
<td>\eta</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( P_T &gt; 30 \text{ GeV} ) (</td>
<td>\eta</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( P_T &gt; 25 \text{ GeV} ) (</td>
<td>\eta</td>
</tr>
</tbody>
</table>

- Switched POET new input data schema (NanoAOD)
  - [tag v1](#)
- Added new selection cuts of version 2
  - [root-project/analysis-grand-challenge](#)
- Added boosted decision tree inference for the jet-parton assignment
  - [andriiknu/agc-root/tree/add_inference](#)
Top mass reconstruction

1. Event filtering

Semi-leptonic decay channel schema

- 1 lepton
- 4 jets
- 2 b-tagged jets

2. Calc observable

The top mass is a total mass of three jets ($\bar{b}, \bar{q}, q$)

- Choose three jet combination which is the product of top decay
def = (  
    df.Define(  
        "Electron_mask",
        "Electron_pt > 30 && abs(Electron_eta) < 2.1 && Electron_sip3d < 4 && Electron_cutBased == 4",
    )
    .Define(  
        "Muon_mask",
        "Muon_pt > 30 && abs(Muon_eta) < 2.1 && Muon_sip3d < 4 && Muon_tightId && Muon_pfRelIso04_all < 0.15",
    )
    .Filter("Sum(Electron_mask) + Sum(Muon_mask) == 1")
    .Define("Jet_mask", "Jet_pt > 30 && abs(Jet_eta) < 2.4 && Jet_jetId == 6")
    .Filter("Sum(Jet_mask) >= 4")
)

def = (  
    .Define("Jet_btagCSVV2_cut", "Jet_btagCSVV2[Jet_mask]"
    .Define("Jet_eta_cut", "Jet_eta[Jet_mask]"
    .Define("Jet_phi_cut", "Jet_phi[Jet_mask]"
    .Define("Jet_mass_cut", "Jet_mass[Jet_mask]"
)
Trijet combination

Consider all trijet combinations

Assign to every combination some properties

- ML term: \textit{features}
- By these features, we \textit{infer} combination, which is more likely the top decay product
- Is there at least 1 b-tagged jet? (\textit{bool})
- Total transverse momentum

Set some criteria how to conclude from these properties (features) which trijet is the best candidate for being top decay product

- Choose the combination with the largest combined transverse momentum
How it does look in the code

```python
df = df.Define("Trijet_idx", "Combinations(Jet_ptcut, 3)")

df = df.Define(
    "Trijet_btag",
    ...
    auto J1_btagCSVV2 = Take(Jet_btagCSVV2_cut, Trijet_idx[0]);
    auto J2_btagCSVV2 = Take(Jet_btagCSVV2_cut, Trijet_idx[1]);
    auto J3_btagCSVV2 = Take(Jet_btagCSVV2_cut, Trijet_idx[2]);
    return J1_btagCSVV2 > 0.5 || J2_btagCSVV2 > 0.5 || J3_btagCSVV2 > 0.5;
    ".....",
)
```
# Assign four-momentums to each trijet combination

def = df.Define('Trijet_p4',
    "auto J1 = Take(Jet_p4, Trijet_idx[0]);
    auto J2 = Take(Jet_p4, Trijet_idx[1]);
    auto J3 = Take(Jet_p4, Trijet_idx[2]);
    return (J1+J2+J3)[Trijet_btag];"
)

# Get trijet transverse momentum values from four-momentum vectors

def = df.Define('Trijet_pt',
    "return Map(Trijet_p4, [](const ROOT::Math::PxPyPzE4Vector &v) { return v.Pt(); });"
)

# Evaluate mass of trijet with maximum pt and btag higher than threshold

def = df.Define('Trijet_mass', "Trijet_p4[ArgMax(Trijet_pt)].M()")
Histograms validation against reference

All of 122 histograms are in perfect bin-by-bin alignment
How our algorithm can be improved?

• The of choosing a trijet combination is **too rough**
  • Why only 2 properties *(features)*? (max $P_T$ and $b$-tag)
  • The criteria were not very reliable

• **Machine learning** allows us to create more complicated schemas to choose the best option

• We can construct many more properties for jet combinations to express more physics

• By using ML we delegate the work of making decisions to the computer
Boosted decision trees

- Decisions are made via decision trees
- A decision tree takes a set of input features and splits input data recursively based on those features
- Allows estimate probability to be a candidate of top decay by much more complicated logic: chain of decision
- Increasing the depth of the tree allows for expressing the logic of arbitrary complexity
- The magic occurs because we can:
  - create a forest of decision trees
  - delegate to the computer to draw the content of each tree by fitting the model with data (training)
• Imagine event with four jets
• Assign the label to each jet:
  • $b_{\text{top-lep}}$, $W$, or $b_{\text{top-had}}$
• Build all unique permutations of 4-jets

- Calculate 20 features for every permutation (not only 2)
• BDT inference gives the best candidate

see full details at official AGC documentation
Validation of ML output histograms

- 1211 out of a total of 1220 histograms are perfectly matched bin-by-bin!!
- 9 of them are within tolerance of 1%!
BDT inference implementation details

1. To perform inference we used retrained XGBoost models

2. RDataFrame itself is NOT a machine-learning library

3. TMVA – is it an option?
   ✓ The internal part of the ROOT Framework
   ✓ Supports BDT
   × Fails covert XGB to TMVA
     × ROOT.TMVA.Experimental.SaveXGBoost tries to pre-allocate storage for $2^{\text{depth}} + 1$ parameters
     × supports only trees where the splits are always closer to the root – this is not the case for our models

4. FastForest library
   ➢ the option we chose
   ➢ minimal library code to deploy XGBoost models in C++
   ➢ going to be integrated into ROOT
   ➢ integrated as an external library the project
     ▪ needs for FastForest installation to work
     ▪ needs to be linked to shared libraries
BDT inference implementation details

- ROOT can compile sources with external libraries
- `ml_helpers.cpp` source containing inference implementations
- `fastforest_path` – points to the location of headers and libs
- temporal solution

```python
include = os.path.join(fastforest_path, "include")
lib = os.path.join(fastforest_path, "lib")
ROOT.gSystem.AddIncludePath(f"-I{include}\")
ROOT.gSystem.AddLinkedLibs(f"-L{lib} -lfastforest")
ROOT.gSystem.Load(f"{lib}/libfastforest.so.1")
ROOT.gSystem.CompileMacro("ml_helpers.cpp", "kO")
```
BDT inference implementation details

```c++
ROOT::RVecF inference(const ROOT::VecOps::RVec<ROOT::RVecD> &features,
                        const fastforest::FastForest &forest) {

    size_t npermutations = features.at(0).size();
    size_t nfeatures = features.size();
    ROOT::RVecF res(npermutations);
    float input[nfeatures];

    for (int i = 0; i < npermutations; ++i) {
        for (int j = 0; j < nfeatures; ++j) {
            input[j] = features.at(j).at(i);
        }
        float score = forest(input, 0.0F);
        res[i] = 1.0/(1.0+std::exp(-score));
    }

    return res;
```
BDT inference implementation details

- returns a vector of probability scores with the size of the number of jet combinations
- applying model trained on odd events for inference on even events and vice versa
Benchmarks

- Total throughput for 64 cores 7.4M events/s
- Speedup up to 41x

Dataset of total size 1.78 TB / 940160174 events
The main achievements

• Switched ROOT’s RDataFrame implementation of AGC to version 1.
  • Used new input data schema (NanoAOD)
• Switched ROOT’s RDataFrame implementation of AGC to version 2.
  • Defined new selection cuts (slide 5)
  • Added calculation variables which are input features for machine learning inference
  • Added implementation of ML inference
  • ROOT’s RDataFrame integrated with the C++ FastForest library
• Validation
  • We are in good agreement with IRIS-HEP reference implementation
  • The performance benchmarks show good scalability
Thanks a lot for your attention!
Questions?
Backup
**Input data**

- **Input data struct**
  - Processes:
    - $t\bar{t}$
    - $s$-chan
    - $t$-chan
    - $W$-chan
    - $W$-jets
  - Variations:
    - Scale up
    - Scale down
    - ME_var
    - PS_var

**Datasamples**
- **2015 CMS Open Data**
- 9 subsets of root files produced in MC simulation
- 5 interaction channels $\rightarrow$ 5 processes involved
- 4 kinds of variations $\rightarrow$ given as 4 additional sets

**Data schemas**
- Physics Objects Extractor Tool (POET)
- NanoAOD
Observables

**Signal region** *(reconstructed top mass):*

>=4 jets, 2 b-tag

**Control region** *(sum of the $p_T$ of all jets in each event):*

>=4 jets, 1 b-tag

Top-quark mass peak plot

Scalar sum of transverse momenta plot
Comparison of ML output histograms

- built-in another way to generate permutations