Latinos framework transition to RDF

Davide Valsecchi, Vincenzo Padulano

10/12/2020
88th ROOT PPP meeting

The Framework

Analysis framework built in CMS by HWW group and used also by many SMP analyses: used in 14 institutions with roughly **70 users**. Produced 15 public analyses and 10 are on-going on FullRun2 dataset

- 50% C++ / 50% Python
- Configuration: completely as **Python dictionaries** in files (details in backup)
- Submission: HTCondor@CERN, T2 in Bruxelles, locally in various T3
- Final results: skimmed ROOT files / TH1
- ***Large*** set of utils and plotting scripts built over time
- Main motivation to refactor the **core** code with RDataFrame:
  - The core of the framework (multidraw package by a former CMS member now working on Atlas) is an interface to TTrees working in a similar way to RDF: 1 event loop, multiple cuts and histograms
  - We would like to move to a centrally maintained package to improve stability and being able to add easily new features to our framework
  - Possibility to **speed-up and parallelize** even more our systematics variation workflow

Workflow

- **Preparation step:**
  - NanoAOD postprocessing to add branches and to build **variations** for derived quantities
  - We store friend trees with only the varied branches using a suffix in the branch name
    - For example for lepton scale variations: *fancy_variable* → *fancy_variable_lepPtUP*
  - 20 variations : ~100k jobs , ~1 week of processing, ~100k files, ~3 TB dataset
- **Analysis step:**
  - All **cuts**, **histograms** and **weights** built using string expressions and TTree functions like *Alt$*
  - Many variables are built on the fly using C++ classes (derived by TTreeFormula and used internally by the multidraw code)
    - Evaluating weights reading TGraph or TH1
    - Evaluating machine learning models
  - Cuts, weights and histograms run in parallel (1 event loop), but we have **1 loop for each systematic variation**

Systematic variations

2 types of systematic variations in the framework:

- **Weights based**: just fill the histograms with different weights to build the up and down variation. It is done already in parallel along with the events loop for the nominals
- **Shape variation**: The multidraw package implements internally a swap of the TTree leaves in the TTreeFormulas in order to switch automatically the branches for systematics variations (taken from friend trees). No input from the user apart from the mapping between variations and branches to swap.

```
# ######################
# # Theory nuisance
for sample in mc :
    nuisances['QCD_scale_'+sample] = {
        'name'   : 'QCDscale_'+sample,
        'kind'   : 'weight',
        'type'   : 'shape',
        'samples'   :  { sample: ["LHEScaleWeight[0]", "LHEScaleWeight[8]"] }
    }
```

Configurable by cut and sample (e.g. different backgrounds)

mapping of branches defined in another file

```
nuisances['muonpt']  = {
                        'name'   : 'CMS_scale_m_2018',
                        'kind'   : 'suffix',
                        'type'   : 'shape',
                        'mapUp': 'MupTup',
                        'mapDown': 'MupTdo',
                        'cuts': phase_spaces_tot_mu,
                        'samples': dict((skey, ['1.','1.']) for skey in mc if
                        'folderUp' : directory_mc+'_MupTup',
                        'folderDown' : directory_mc+'_MupTdo',
}
```

What would be nice to have in RDF

It would be great to include the systematic variations as part of a single cycle with RDataframe

- with more than 50 variation processes the processing time is becoming huge

In particular, here's a short list of requirements that comes to our mind:

- being able to define variations branches **on-the-fly without a preprocessing**:
  - already easy to do with RDataFrame *Define* method
- Maintain the systematic variation ***transparent*** to the user (avoid manually redefining all cuts and expressions for each variation)
- Run the varied analysis (all cuts, weights, and histograms) in **the same event loop** as the nominals

The main technical difficulty that I see is how to do the bookkeeping **to build varied variables, weights and histograms** without explicit inputs by the users. Bamboo package is addressing this in an interesting way but our user interface is much more declarative.

Backup material

Config example

```
aliases['gstarHigh'] = {
    'expr': 'Gen_ZGstar_mass <0 || Gen_ZGstar_mass > 4',
    'samples': 'VgS'
}

############################

aliases['veto_fatjet_180'] = {
            'class': 'VetoFatJetResolved',
            'args': (180.),
            'linesToAdd' : [
                'gSystem->Load("libLatinoAnalysisMultiDraw.so")',
                '.L {}/VBSjjlnu/macros/veto_fatjet_resolved.cc+'.format(configurations)
            ]
}
```

Aliases to build variables by root expression or by calling C++ classes

```
cuts["res_sig_ele"] = 'VBS_category==1 \
                       && abs(Lepton_pdgId[0])==11 \
                       && vjet_0_pt > 30 && vjet_1_pt > 30 \
                       && mjj_vjet > 65 && mjj_vjet < 105 \
                       && bVeto \
                       && w_had_pt < 200 \
                       && veto_fatjet_180 '
```

cuts dictionary to define categories where histograms need to be produced

```
samples['top'] = {    'name'   :   nanoGetSampleFiles(directory_bkg,'TTTo2L2Nu')
                                 + nanoGetSampleFiles(directory_bkg,'ST_s-channel_ext1')
                                 + nanoGetSampleFiles(directory_bkg,'ST_t-channel_antitop')
                                 + nanoGetSampleFiles(directory_bkg,'ST_t-channel_top')
                                 + nanoGetSampleFiles(directory_bkg,'ST_tW_antitop_ext1')
                                 + nanoGetSampleFiles(directory_bkg,'ST_tW_top_ext1')
                                 + nanoGetSampleFiles(directory_bkg,'TTToSemiLeptonic')
                                 + nanoGetSampleFiles(directory_bkg,'TTZjets')
                                 + nanoGetSampleFiles(directory_bkg,'TTWjets'),
                                # +  nanoGetSampleFiles(directory_bkg,'TTWJetsToLNu'), #also
                  'weight' : XSWeight+'*'+SFweight+'*'+GenLepMatch+'*'+METFilter_MC ,
                  'FilesPerJob' : 3,
                  'EventsPerJob' : 70000,
                  'suppressNegative' :['all'],
                  'suppressNegativeNuisances' :['all'],
                }
addSampleWeight(samples,'top','TTTo2L2Nu','Top_pTrw')
addSampleWeight(samples,'top','TTToSemiLeptonic','Top_pTrw')
```

Sample dictionary to defined input files and weights (they can be defined also by single files)

Variables dictionary to define the histogram to output

```
variables['DNNoutput_res_bins1'] = {
    'name': 'DNNoutput',
    'range': (15,0.,1),
    'xaxis': 'DNN output, resolved',
    'fold': 3 ,
    'cuts':  res_cuts,
    'blind': { c:[0.7,1] for c in cuts if "_sig_" in c},
}
```