RooUnfold GSoC 2021 Task

Archit Agrawal
archit18221@iiitd.ac.in

March 19, 2021

Task 1: Compile under CMake

We will ensure thet ROOT correctly set up with all the environment variables. This can be achieved using ROOT's *thisroot.sh*.

```
archit@archit-pc:~/Downloads$ root -l RooUnfoldGSOC.cxx
root [0]
Processing RooUnfoldGSOC.cxx...
reco bins
8464, 70243, 10853, 2224,
prediciton
8464, 70243, 10853, 1389, root [1]
```

Figure 1: Running the code in ROOT interactive shell

Now, to compile the code with CMake

```
$ mkdir build && cd build
$ ctest ..
$ make
$ ./RooUnfoldGSOC
```

```
archit@archit-pc:~/RooUnfoldGSOC/build$ cmake ..
-- The C compiler identification is GNU 9.3.0
-- The CXX compiler identification is GNU 9.3.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/archit/RooUnfoldGSOC/build
archit@archit-pc:~/RooUnfoldGSOC/build$ make
Scanning dependencies of target tester
[ 25%] Building CXX object CMakeFiles/tester.dir/tester.cpp.o
[ 50%] Linking CXX executable tester
[ 50%] Built target tester
Scanning dependencies of target RooUnfoldGSOC
[ 75%] Building CXX object CMakeFiles/RooUnfoldGSOC.dir/RooUnfoldGSOC.cxx.o
[100%] Linking CXX executable RooUnfoldGSOC
[100%] Built target RooUnfoldGSOC
archit@archit-pc:~/RooUnfoldGSOC/build$ ./RooUnfoldGSOC
reco bins
8464, 70243, 10853, 2224,
prediciton
2.25044e+08, 3.05111e+09, 4.09698e+08, 5.76435e+06, archit@archit-pc:~/RooUnfoldGSOC/build$
```

Figure 2: Output

Task 2: Normalisation

The *response* is a 2-D histogram with a double per channel with x and y bin size as 4. Hence we can represent it in the form of

$$response = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{19} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

And the *truth* is a 1-D histogram with a double per channel with a bin size as 4. Hence it could be represented as

$$truth = \begin{bmatrix} b_1 \\ b_2 \\ a_3 \\ b_4 \end{bmatrix}$$

The mathematical representation of the operations performed in loop for calculation of *prediction* would be

$$prediction = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix} = \begin{bmatrix} a_{11}b_1 + a_{12}b_2 + a_{13}b_3 + a_{14}b_4 \\ a_{21}b_1 + a_{22}, b_2 + a_{23}b_3 + a_{24}b_4 \\ a_{31}b_1 + a_{32}b_2 + a_{33}b_3 + a_{34}b_4 \\ a_{11}b_1 + a_{42}b_2 + a_{43}b_3 + a_{44}b_4 \end{bmatrix}$$

Now after normalising the bins of the response to be the probability of reconstructing the event in each bin given the bin it truthfully came from. For that, if each column of *response* is represented as a vector like

$$response = \begin{bmatrix} w & x & y & z \end{bmatrix}$$

Then the normalised response would be

$$response_{norm} = \begin{bmatrix} \frac{w}{b_1} & \frac{x}{b_2} & \frac{y}{b_3} & \frac{z}{b_4} \end{bmatrix}$$

The mathematical representation of the operations performed in loop for calculation of *prediction* would be

$$\begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix} = \begin{bmatrix} \frac{a_{11}b_1}{b_1} + \frac{a_{12}b_2}{b_2} + \frac{a_{13}b_3}{b_3} + \frac{a_{14}b_1}{b_1} \\ \frac{a_{21}b_1}{b_1} + \frac{a_{22}b_2}{b_2} + \frac{a_{23}b_3}{b_3} + \frac{a_{24}b_3}{b_4} \\ \frac{a_{31}b_1}{b_1} + \frac{a_{32}b_2}{b_2} + \frac{d_{33}b_3}{b_3} + \frac{a_{34}b}{b_4} \\ \frac{a_{41}b_1}{b_1} + \frac{a_4b_2}{b_2} + \frac{a_{43}b_3}{b_3} + \frac{a_{44}b_4}{b_4} \end{bmatrix}$$

Hence the value of truth gets cancelled for each term.

Figure 3: Old Code(Without Normalisation)

```
TH1D prediction = TH1D("prediction","prediction",4,-10.,10.);
for (Int_t i= 0; i<reco.GetNbinsX(); i++){
  for (Int_t j= 0; j<truth.GetNbinsX(); j++){
      auto current = prediction.GetBinContent(i+1);
      auto contribution = truth.GetBinContent(j+1)*response.GetBinContent(i+1,j+1);
      prediction.SetBinContent(i+1,current+contribution);
  }
}
```

Figure 4: New Code(With Normalisation)

```
TH1D prediction = TH1D("prediction","prediction",4,-10.,10.);
for (Int_t i= 0; i<reco.GetNbinsX(); i++){
  for (Int_t j= 0; j<truth.GetNbinsX(); j++){
      auto current = prediction.GetBinContent(i+1);
      auto contribution = response.GetBinContent(i+1,j+1);
      prediction.SetBinContent(i+1,current+contribution);
  }
}
```

Figure 5: Output

```
archit@archit-pc:~/RooUnfoldGSOC/build$ ./RooUnfoldGSOC
==================================== TEST ============================
================================ PREDICT (With Normalization) ========
reco bins
8464, 70243, 10853, 2224,
prediciton
8464, 70243, 10853, 1389,
archit@archit-pc:~/RooUnfoldGSOC/build$
```

Task 3: Matrix Operation

Now to replace the loop with the matrix operation, we would be first converting the *response* and *truth* from histograms to a ROOT TMatrix object. *response* would be a 4×4 matrix and *truth* would be a ROOT vector object with 4 rows.

```
TMatrix normalisedResponse = TMatrix(4,4);
TVector truth_mat = TVector(4);
for (Int_t i= 0; i<4; i++) {
  truth_mat[i] = truth.GetBinContent(i+1);;
}

for (Int_t i= 0; i<reco.GetNbinsX(); i++){
  for (Int_t j= 0; j<truth.GetNbinsX(); j++){
      normalisedResponse[i][j] = response.GetBinContent(i+1,j+1)/truth.GetBinContent(j+1);
  }
}
```

After conversion, we would be performing dot product of *response* and *truth*. Hence

$$prediction = response \cdot truth$$

When written using the same notation as used above would be

$$prediction = \begin{bmatrix} \frac{w}{b_1} & \frac{x}{b_2} & \frac{y}{b_3} & \frac{z}{b_4} \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

Which if expanded, we would obtain

$$prediction = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix} = \begin{bmatrix} \frac{a_{11}b_1}{b_1} + \frac{a_{12}b_2}{b_2} + \frac{a_{13}b_3}{b_3} + \frac{a_{14}b_1}{b_1} \\ \frac{a_{21}b_1}{b_1} + \frac{a_{22}b_2}{b_2} + \frac{a_{23}b_3}{b_3} + \frac{a_{24}b_3}{b_4} \\ \frac{a_{31}b_1}{b_1} + \frac{a_{32}b_2}{b_2} + \frac{d_{33}b_3}{b_3} + \frac{a_{34}b}{b_4} \\ \frac{a_{41}b_1}{b_1} + \frac{a_4b_2}{b_2} + \frac{a_{43}b_3}{b_3} + \frac{a_{44}b_4}{b_4} \end{bmatrix}$$

Hence, it is same as the operation performed by the loop.

```
archit@archit-pc:~/RooUnfoldGSOC/build$ ./RooUnfoldGSOC
==================================== TEST ====================================
================================ PREDICT (With Normalization) ================
reco bins
8464, 70243, 10853, 2224,
prediciton
8464, 70243, 10853, 1389,
==================== PREDICT (With Normalization and Matrix Operation) =======
reco bins
8464, 70243, 10853, 2224,
prediciton
8464, 70243, 10853, 1389,
```

Figure 6: Output

Task 4: Developing CTests

To confirm the similarity of results reported by loop and that of matrix, I developed a tester.cpp to run the compiled file and compare the reported output.
To run test

```
$ cd build
$ ctest ..
$ make
$ ctest
```

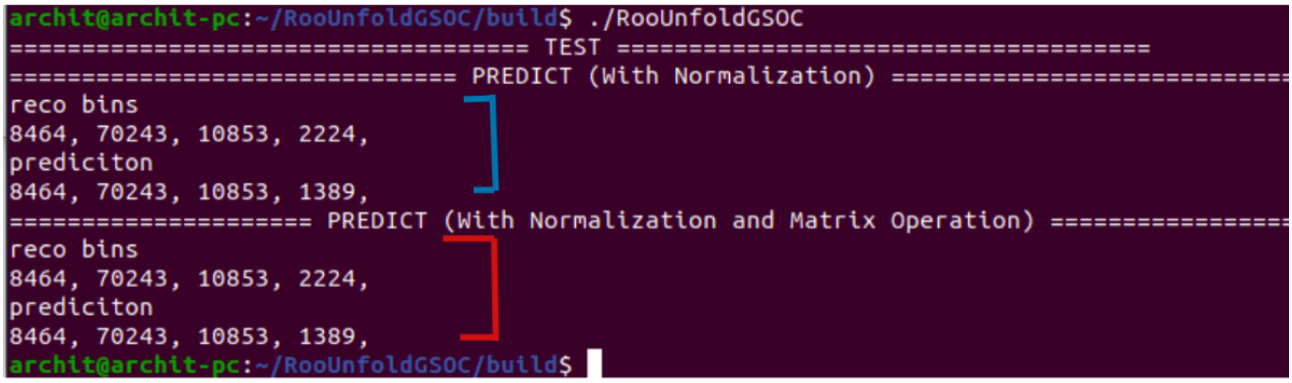

Figure 7: tester.cpp compares the blue highlighted part with red highlighted part

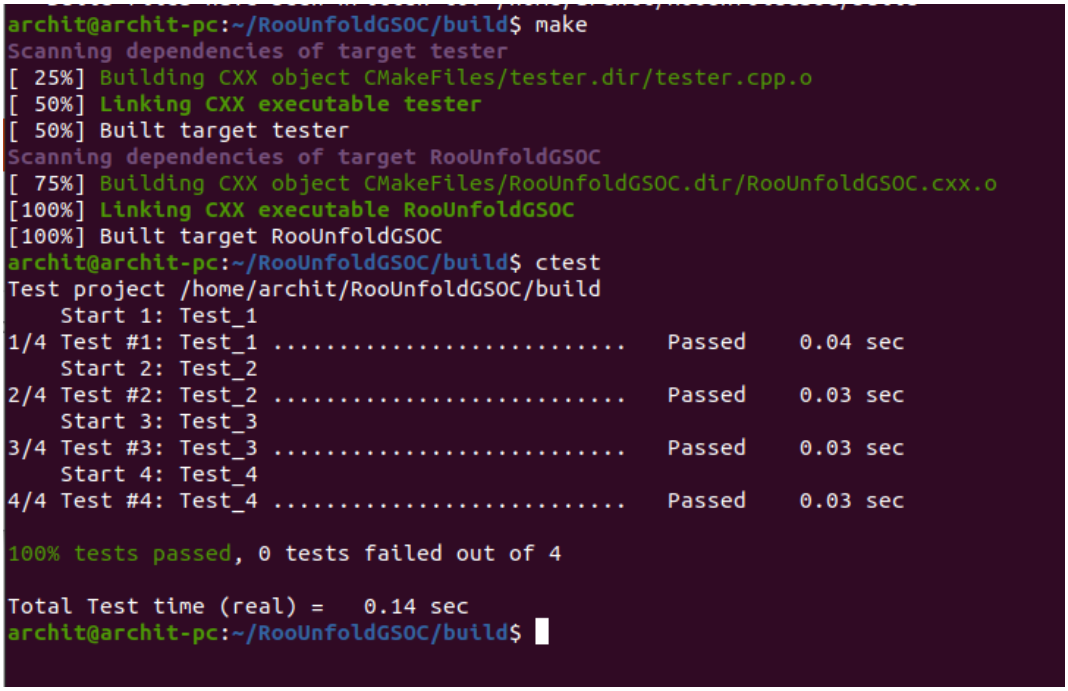

Figure 8: Total four test were set up

Reference

ROOT Documentation(https://root.cern/primer/)
RooUnfold Readme (https://gitlab.cern.ch/RooUnfold/RooUnfold)
ROOT Forms(https://root-forum.cern.ch) "Not all heroes wear capes"
And ofcourse StackOverflow (Hail creators of StackOverflow!!)
YouTube videos on CTest