MuXboostedBTagging user guide

Keith Pedersen (kpeders1@hawk.iit.edu)

November 18, 2015

Abstract

This user guide explains the function of the MuXboostedBTagging module for DELPHES. This guide assumes that the reader has already read the μ_x boosted-b tag paper [1], and understands the basics of the μ_x tag.

1 MuXboostedBTagging basics

The MuXboostedBTagging module (henceforth frequently abbreviated as MuX) is a relatively straightforward extension to the standard DELPHES functionality. Like DELPHES' native BTagging module, MuX draws from JetInputArray, a list of fully reconstructed/calibrated jets (e.g. from JetEnergyScale), and only alters their Candidate::BTag bit-flag. Additionally, to allow MuX to be used in conjunction with BTagging, MuX allows the user to define *which bit* in BTag is set when a jet is tagged (the MuX default is bit-3, versus the BTagging default of bit-0).

Because the μ_x boosted-b tag only works for jets undergoing semi-leptonic decay, the jets it tags systematically lack neutrino energy. This means that MuXboostedBTagging should perform an additional jet energy correction, in the form of **neutrino estimation**. Currently, only the simplest estimation is implemented: the neutrino's associated muon is used as an exact proxy for the neutrino ($p_{\nu_\mu} = p_\mu$), since most of the momentum of both comes from their shared boost. *A more sophisticated estimate will improve the energy resolution.*

In order to accommodate this jet energy correction in a *sensible* way, JetInputArray is cloned into JetOutputArray, and only clone jets which are *tagged* by μ_x have their neutrino(s) estimated. Thus, both input/output jets contain the same jets, in the same order, with all the same fields (including BTag) — *except* when they're tagged by μ_x, in which case the version in JetOutputArray will have a larger Candidate::Momentum.

These two lists of jets can be iterated through in parallel during post-detector analysis code, giving the user the power to determine whether they want the original or corrected jet. For example, consider a dijet analysis which only requires a single b tag. *IFF* only one jet is μ_x-tagged, then the event $\not{E}_T$ could be used for neutrino estimation (instead of $p_{\nu_\mu} = p_\mu$). However, since a general process could possess other sources of $\not{E}_T$, this decision should only only be made in post-analysis, not inside DELPHES.

2 Using MuXboostedBTagging

MuX will only tag jets *which already contain muons*! **There is no input array for muons!** This is a deliberate choice, as these semi-leptonic muons are part of the jet, and should participate in jet clustering (allowing hard muons to seed anti-k_T jets). Hence, the list of detected muons *must* be merged into the input for jet clustering, otherwise MuX won't be able to tag anything.

Neutrino corrected jet clones (MuX/JetOutputArray) must be explicitly written out in TreeWriter. Alternatively, the user can manually correct the original jets during post-detector analysis (looping through the constituents of jets tagged by MuX, finding muons passing the p_T cut, and estimating their ν_μ).

Since MuXboostedBTagging can be used in conjunction with BTagging, the BTag field can contain multiple flagged bits. Hence, (BTag > 0) only indicates a jet was tagged by *some* module. To isolate a μ_x tag, one can use bit-wise-AND.

```
#include "DelphesClasses.h" // "Jet" class is defined here
... // A bunch of ROOT includes
const UInt_t muXbitNumber = 3;

const UInt_t bTaggingFlag = 1;
const UInt_t muXFlag = 1 << muXbitNumber;

TClonesArray* jetArray = 0;
TIterator* itJet = 0;
Jet* jet = 0;

// Read TTree, link &jetArray, loop over entries, GetEntry() (fill jetArray)
  // Initialize itJet, then use while to loop over all jets
  while((jet = static_cast<Jet*>(itJet->Next())))
  {
    if((jet->BTag & bTaggingFlag) > 0) {//taggged by BTagging}
    if((jet->BTag & muXFlag) > 0){//taggged by MuXboostedBTagging}
  }
```

This DELPHES card sets each MuX parameter to its default value.

```
module MuXboostedBTagging MuXboostedBTagging {
    set JetInputArray JetEnergyScale/jets
    set JetOutputArray jets

# Which bit to activate when a jet is tagged
    set BitNumber            3
# The main cuts in the tag (x_max and f_subjet^min)
    set MaxX                 3.0
    set MinCorePtRatio       0.5
# Basic kinematic cuts (to define taggable jets and taggable muons)
    set MinJetPt             300.
    set MinMuonPt            10.
# The core reclustering pT cut on towers
    set MinTowerPtRatio      0.05
# The core reclustering radius
    set CoreAntiKtR          0.04
# The minimum boost of a candidate core (low improves light-jet fake rate)
    set BCoreMinBoost        1.0
# The mass hypotheses
    set CoreMassHypothesis   2.0
    set SubjetMassHypothesis 5.3
# Maximum reconstructed subjet mass
    set MaxSubjetMass        12.
}
```

For a detailed understanding of `MuX`'s internal algorithm, please refer to the source code, which has been extensively commented.

3 Appendix

3.1 ECal pointing and standalone muons

An important feature of the μ_x paper [1] was its attempt to sever its μ_x tagging predictions from the tracking performance of anything but "standalone" muons (muons which only require hits in the muon chamber). To obtain the necessary angular resolution for the core reconstruction, a procedure called "ECal pointing" was implemented (see the paragraph directly before Sec. III-A in [1]).

The DELPHES code to implement ECal pointing, ATLAS's standalone muon efficiency, and the `MuXboostedBTagging` can be found in "./MuX.tcl".

3.2 `AllParticlePropagator`

Part of the accurate simulation of the light jet efficiency was the simulation of in-flight muon production inside the tracking volume (e.g. from π^+/K^+ decay). Since the μ_x tag is effectively an angular tag, it was important to simulate the bending of the muon's mother. However, DELPHES' `ParticlePropagator` only propagates/bends "final-state" particles in the magnetic field; their mothers (at least from PYTHIA) are propagated to their decay vertices in a straight line.

Getting these muons required the activation of in-flight π^+/K^+ decay (off by default in PYTHIA), and the development of the `AllParticlePropagator` module to properly handle such decays (which also required custom handling of pileup). Since this module required alteration of the `Candidate` class (to store each particle's random, proper lifetime) it is not currently distributed with `MuXboostedBTagging`. This means that `MuX` will mildly *over*-estimate the light jet fake rate (once in-flight π^+/K^+ decays are turned on), as muons from in-flight decays will arrive at slightly tighter angles to the reconstructed cores of their jets.

While `AllParticlePropagator` is available on GitHub [2], it is not designed for public use (i.e. no user guides, just the comments). It *may* be officially released in the future, but this is not guaranteed.

3.3 Robust subjet mass

The subjet of a B hadron decay is defined as

$$p_{\text{subjet}} = p_{\text{core}} + p_\mu + p_{\nu_\mu}, \tag{1}$$

where the "core" is ostensibly a charm hadron. Since the neutrino is not observable, we estimate it using the simplest choice ($p_{\nu_\mu} = p_\mu$, since most of the momentum comes from their shared boost). This gives us an approximate subjet

$$p_{\text{subjet}} \approx p_{\text{core}} + 2\,p_\mu. \tag{2}$$

The muon is detected in the muon chamber, and core is found by reclustering the jet using a much smaller radius parameter. This produces a list of *candidate* cores, of which only one is the "correct" core (the one which, using only the hardest muon in the jet, produces $\sqrt{p^2_{\text{subjet}}}$ closest to $m_B = 5.3$ GeV, the mass of a B hadron admixture). However, the mass of these core candidates is poorly measured (due to the the granularity of the Calorimeter), which produces an inaccurate value of $\sqrt{p^2_{\text{subjet}}}$. To alleviate this problem, m_{core} is simply constrained to its expected mass, given the B hadron decay hypothesis ($m_{\text{core}} =$ 2 GeV, the approximate mass of a charm hadron).

The correct core is found by finding the minimum value of $\left|\sqrt{p^2_{\text{subjet}}} - m_B\right|$. In order to complete this search quickly, we don't want to correct the mass of each core candidate before adding the muon. Instead, we can use our large cut on muon p_T to treat the muon as massless, then calculate the mass of the subjet analytically

$$\begin{aligned} p^2_{\text{subjet}} &= m^2_{\text{core}} + 4\, p_{\text{core}} \cdot p_\mu \\ &= m^2_{\text{core}} + 4\, E_\mu E_{\text{core}}(1 - \cos(\xi)\sqrt{1-g}), \end{aligned} \tag{3}$$

where ξ is the angle between the muon and the core and $g \equiv \gamma^{-2}_{\text{core}}$. Using $y \equiv \tan^2(\xi) = \left(\frac{\vec{p}_{\text{core}} \times \vec{p}_\mu}{\vec{p}_{\text{core}} \cdot \vec{p}_\mu}\right)^2$ and $\cos(\xi) = \frac{1}{\sqrt{1+y}}$ (since $\xi < \pi/2$), this becomes

$$p^2_{\text{subjet}} = m^2_{\text{core}} + 4\, E_\mu E_{\text{core}}(1 - \frac{\sqrt{1-g}}{\sqrt{1+y}}). \tag{4}$$

This expression can be optimized to minimize floating point cancellation, (which should be minuscule, but mitigating cancellation is a good habit)

$$p^2_{\text{subjet}} = m^2_{\text{core}} + 4\, E_\mu E_{\text{core}} \frac{g+y}{1+y+\sqrt{1-((g-y)+g\,y)}}. \tag{5}$$

References

[1] K. Pedersen and Z. Sullivan, [arXiv:1511.05990 [hep-ph]]

[2] `https://github.com/keith-pedersen/delphes/tree/myDelphes`