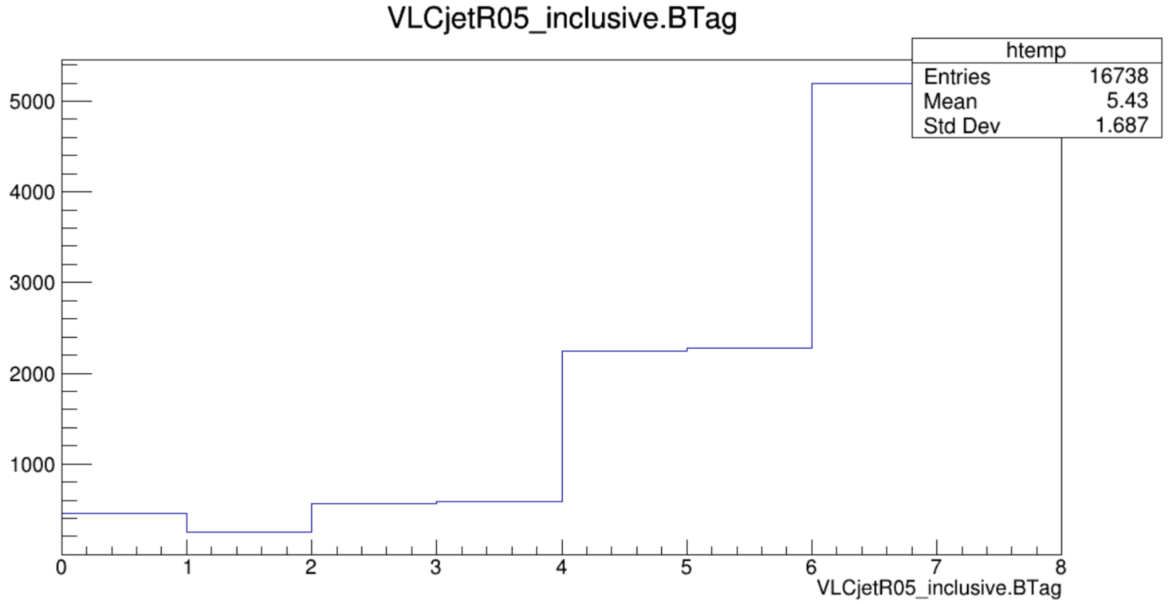
Feasibility Study of Measuring the Higgs Self-coupling Using the Muon Collider

Kenny Jia

Aug 26th, 2021

B-tagging for $k_T/anti\ k_T$

0-7?



VLCjetR05_inclusive.BTag
htemp
Entries 16738
Mean 5.43
Std Dev 1.687
5000
4000
3000
2000
1000
0
1
2
3
4
5
6
7
8
VLCjetR05_inclusive.BTag

Btagging in Delphes

BTag value	7	6	5	4	3	2	1	0
BTag binary	111	110	101	100	011	010	001	000
(bit 0) & BTag	T	F	T	F	T	F	T	F
(bit 1) & BTag	T	T	F	F	T	T	F	F
(bit 2) & BTag	T	T	T	T	F	F	F	F

B-tagging is implemented in the card with 3 working points (WP), which tag 50%, 70%, and 90% of all b-quark-initiated jets, respectively. Delphes uses different bits to store the b-tagging information in the following way, where bit 0 is the tight WP with 50% tagging efficiency, bit 1 is the medium WP with 70% efficiency, and bit 2 is the loose WP with 90% b-tagging efficiency.

Transfer B-tagging algo from VLC to $anti - k_T$

cards/MuonCollider/MuonColliderDet_BTagging.tcl

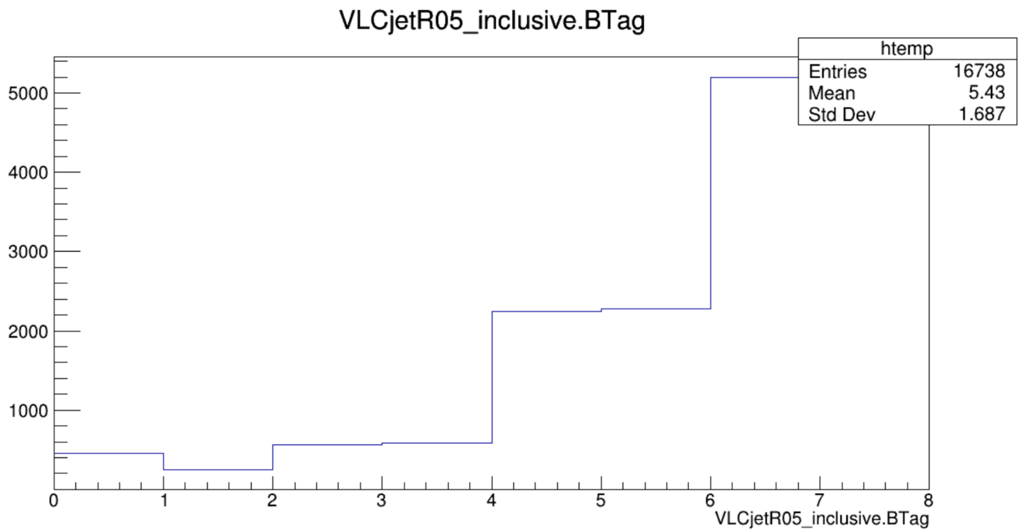
```
#################################
# B-Tagging module for anti-kT
#################################

module BTagging AKT_BTagging_WP50_R05_inclusive {
 set JetInputArray FastJetFinderAKt/AKTjets
 set BitNumber 0
 source MuonCollider/MuonColliderDet_BTag_50.tcl
 }
module BTagging AKT_BTagging_WP70_R05_inclusive {
 set JetInputArray FastJetFinderAKt/AKTjets
 set BitNumber 1
 source MuonCollider/MuonColliderDet_BTag_70.tcl
}
module BTagging AKT_BTagging_WP90_R05_inclusive {
 set JetInputArray FastJetFinderAKt/AKTjets
 set BitNumber 2
 source MuonCollider/MuonColliderDet_BTag_90.tcl
}
```

Add new modules to Execution process in the main card

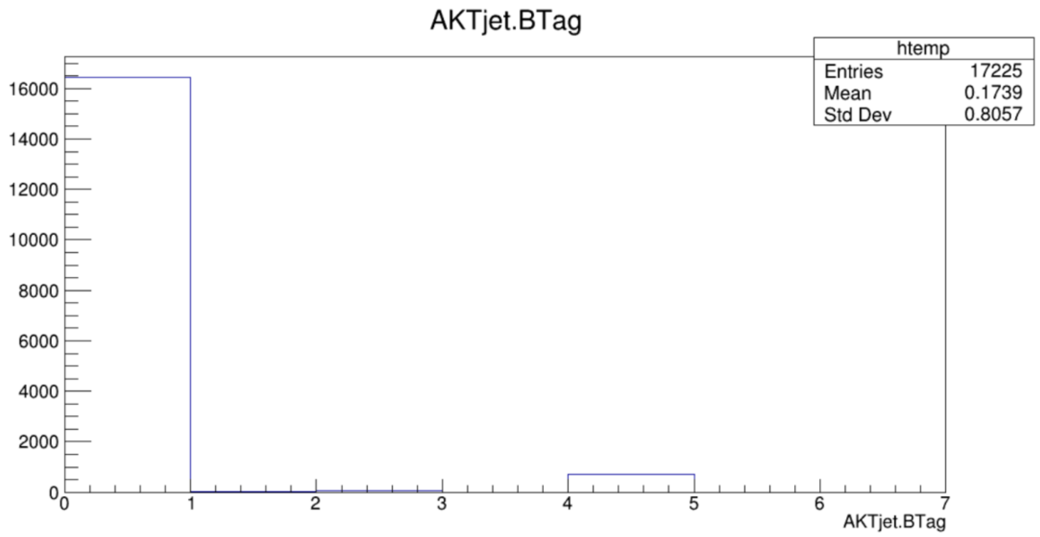
```
AKT_BTagging_WP50_R05_inclusive
AKT_BTagging_WP70_R05_inclusive
AKT_BTagging_WP90_R05_inclusive

BTagging_WP50_R02N2
BTagging_WP70_R02N2
BTagging_WP90_R02N2
BTagging_WP50_R02N3
BTagging_WP70_R02N3
BTagging_WP90_R02N3
BTagging_WP50_R02N4
BTagging_WP70_R02N4
BTagging_WP90_R02N4
BTagging_WP50_R02N5
BTagging_WP70_R02N5
BTagging_WP90_R02N5
```

Comparison with original algo for 2v4b(10k)

VLC R05_inclusive BTag

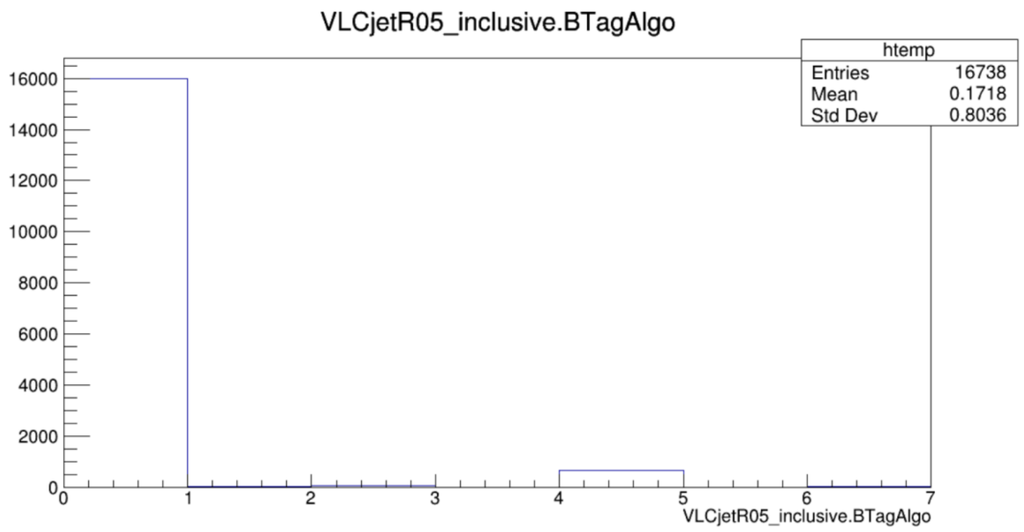


Anti kT R05 BTag

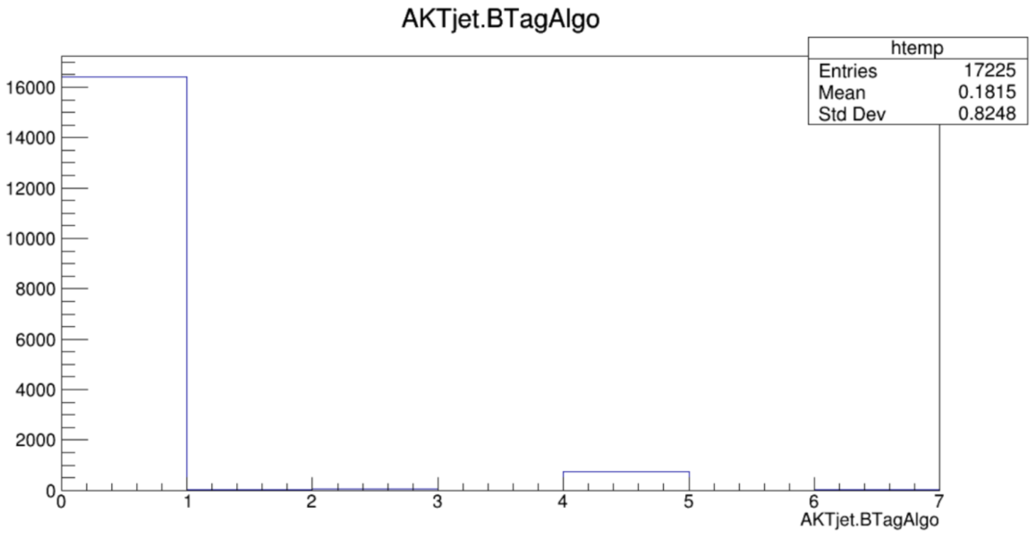


Comparison with original algo for 2v4b(10k)

VLC R05_inclusive BTagAlgo

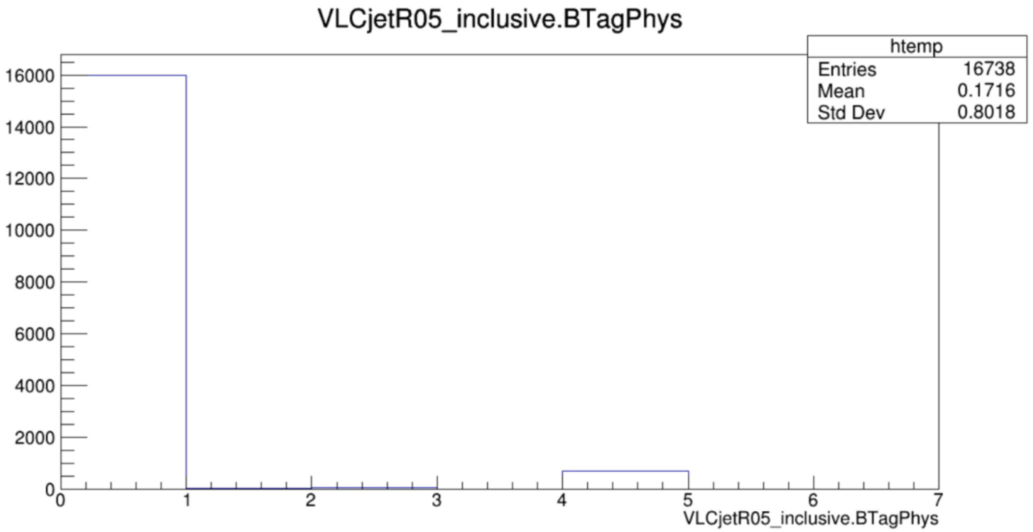


Anti kT R05 BTagAlgo

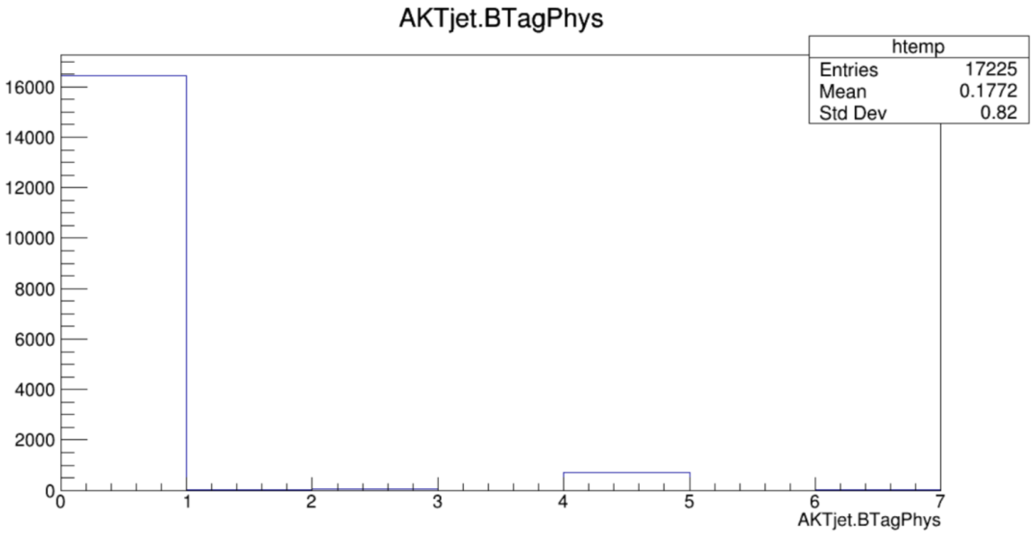


Comparison with original algo for 2v4b(10k)

VLC R05_inclusive BTagPhys



Anti kT R05 BTagPhys



Three different Btag?

VLCjetR05_inclusive.Flavor
VLCjetR05_inclusive.FlavorAlgo
VLCjetR05_inclusive.FlavorPhys
VLCjetR05_inclusive.BTag
VLCjetR05_inclusive.BTagAlgo
VLCjetR05_inclusive.BTagPhys
VLCjetR05_inclusive.TauTag
VLCjetR05_inclusive.TauWeight
VLCjetR05_inclusive.Charge

Flavor	jet flavor
FlavorAlgo	jet flavor
FlavorPhys	jet flavor
BTag	0 or 1 for a jet that has been tagged as containing a heavy quark
BTagAlgo	0 or 1 for a jet that has been tagged as containing a heavy quark
BTagPhys	0 or 1 for a jet that has been tagged as containing a heavy quark

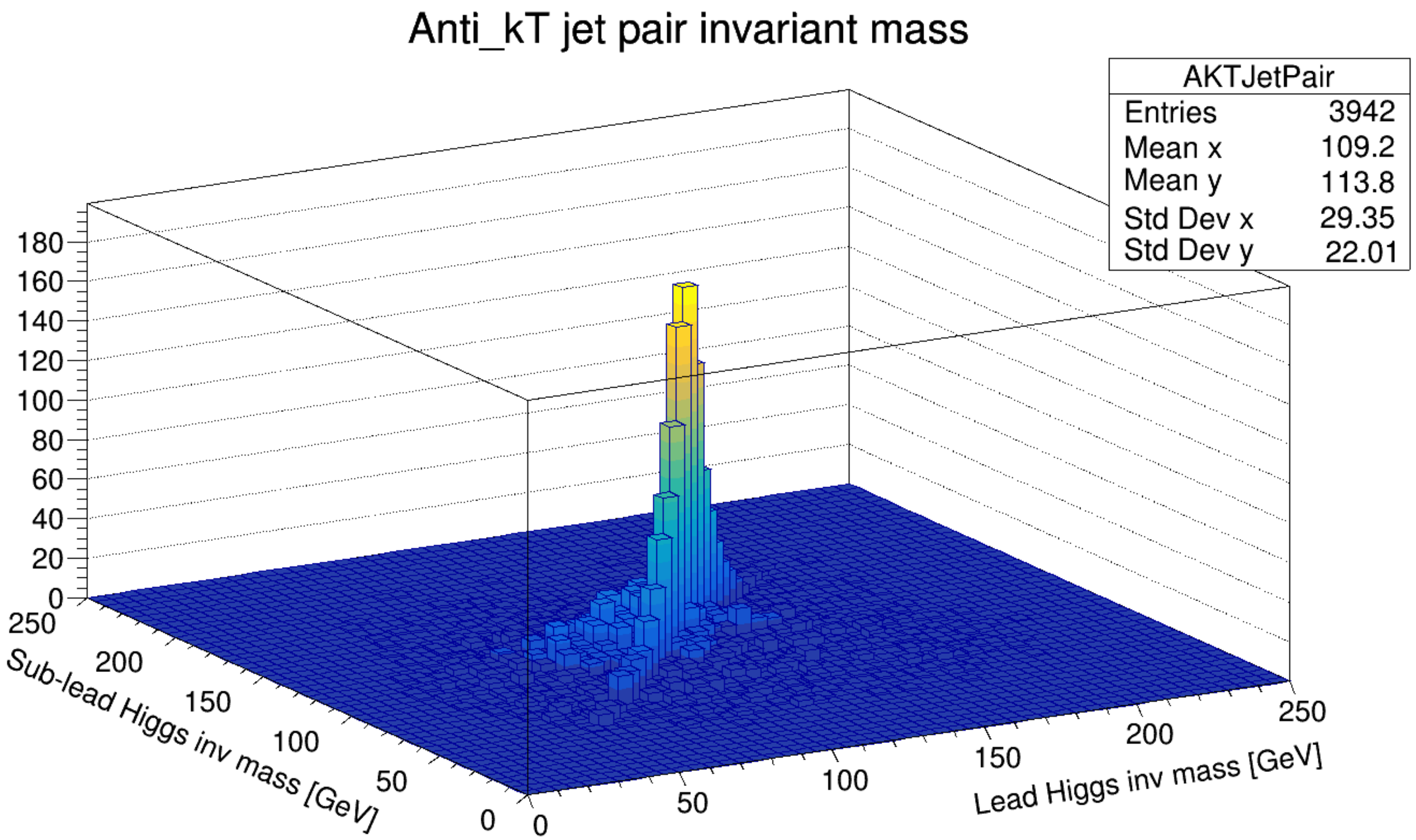
No explaination on Delphes workbook

In /module/Btagging.cc

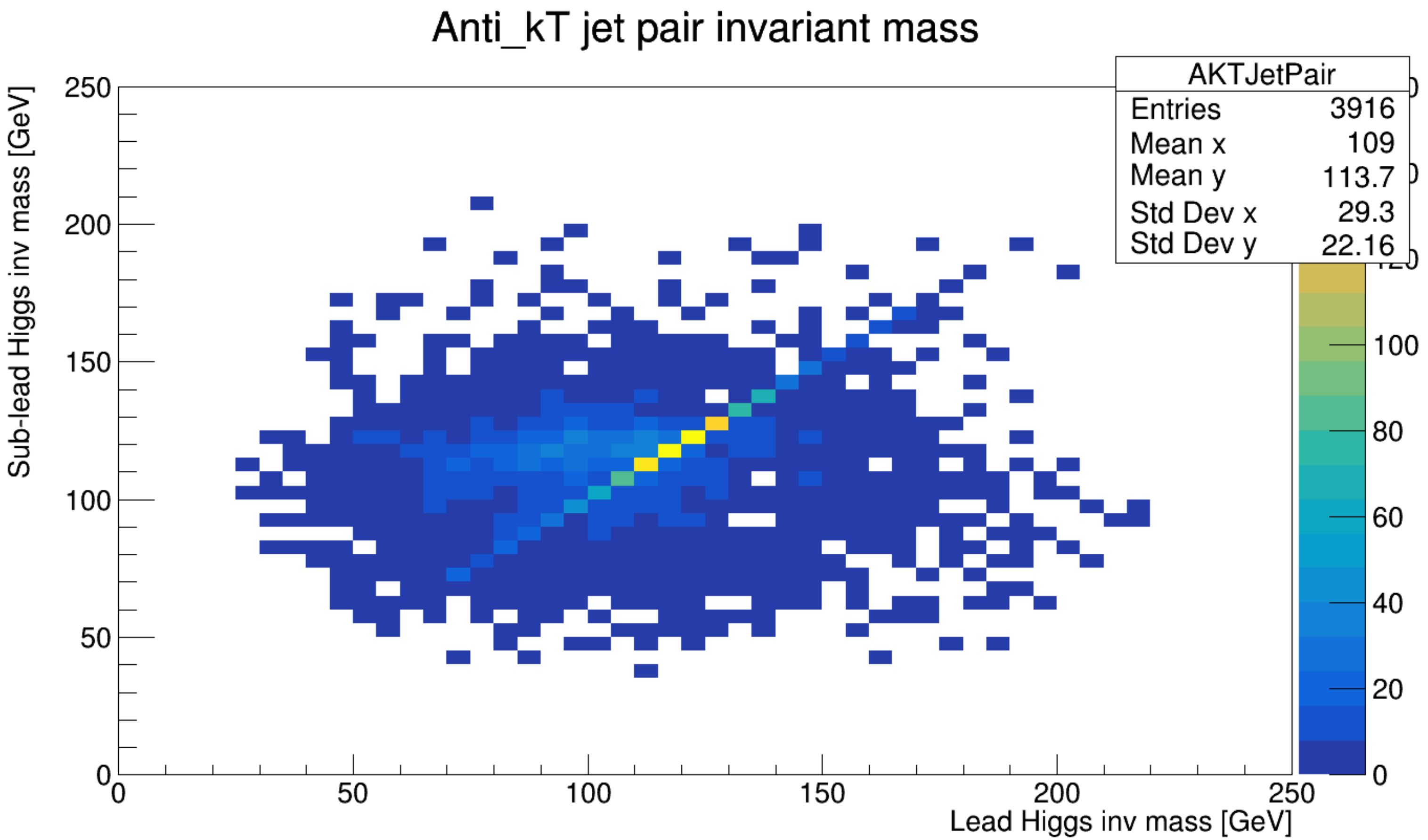
```
    // find an efficiency formula
    itEfficiencyMap = fEfficiencyMap.find(jet->Flavor);
    if(itEfficiencyMap == fEfficiencyMap.end())
    {
      itEfficiencyMap = fEfficiencyMap.find(0);
    }
    formula = itEfficiencyMap->second;

    // apply an efficiency formula
    jet->BTag |= (gRandom->Uniform() <= formula->Eval(pt, eta, phi, e)) << fBitNumber;

    // find an efficiency formula for algo flavor definition
    itEfficiencyMap = fEfficiencyMap.find(jet->FlavorAlgo);
    if(itEfficiencyMap == fEfficiencyMap.end())
    {
      itEfficiencyMap = fEfficiencyMap.find(0);
    }
    formula = itEfficiencyMap->second;

    // apply an efficiency formula
    jet->BTagAlgo |= (gRandom->Uniform() <= formula->Eval(pt, eta, phi, e)) << fBitNumber;

    // find an efficiency formula for phys flavor definition
    itEfficiencyMap = fEfficiencyMap.find(jet->FlavorPhys);
    if(itEfficiencyMap == fEfficiencyMap.end())
    {
      itEfficiencyMap = fEfficiencyMap.find(0);
    }
    formula = itEfficiencyMap->second;

    // apply an efficiency formula
    jet->BTagPhys |= (gRandom->Uniform() <= formula->Eval(pt, eta, phi, e)) << fBitNumber;
  }
}
```

Reconstruction

2D histogram for reconstructed higgs inv mass



2D histogram for reconstructed higgs inv mass

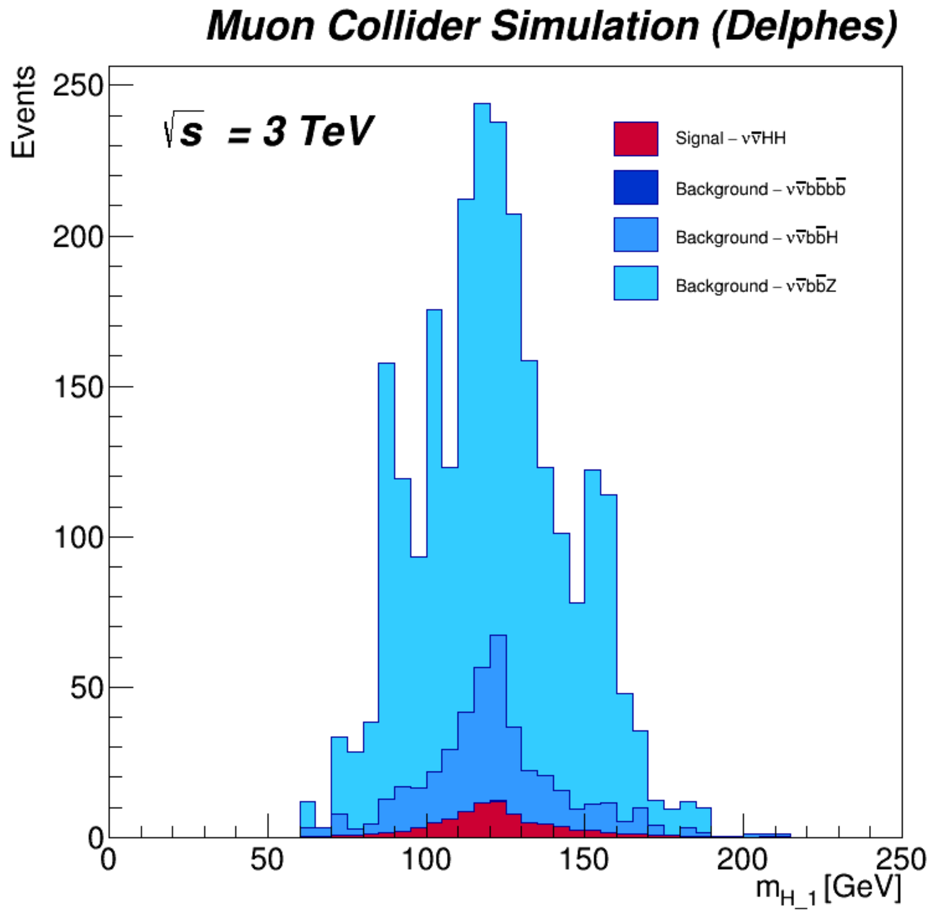


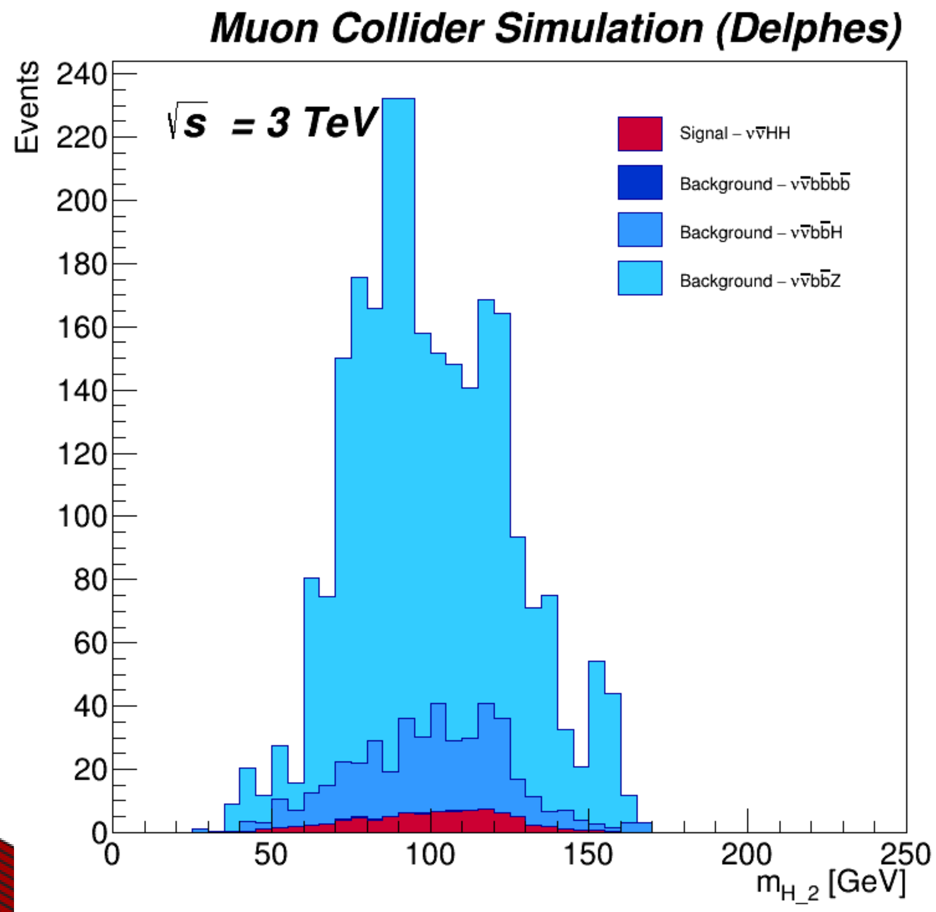
MVA for discriminating sig and bkg

- Signal: $\mu^- + \mu^+ \to v_\mu + \bar{v}_\mu + H + H$ $(0.0008201 \pm 2.2\text{e}-6\ \text{pb})$
- Background:
  - $\mu^- + \mu^+ \to v_\mu + \bar{v}_\mu + b + \bar{b} + Z$ $(0.03168 \pm 0.00027\ \text{pb})$
  - $\mu^- + \mu^+ \to v_\mu + \bar{v}_\mu + b + \bar{b} + H$ $(0.003743 \pm 2.7\text{e}-5\ \text{pb})$
  - $\mu^- + \mu^+ \to v_\mu + \bar{v}_\mu + b + \bar{b} + b + \bar{b}$ $(0.0009227 \pm 7.02\text{e}-6\ \text{pb})$

10k for each, pythia8 bug fixed!

Normalized by cross section





MVA to discriminate vvHH with dominant bkg vvbbZ

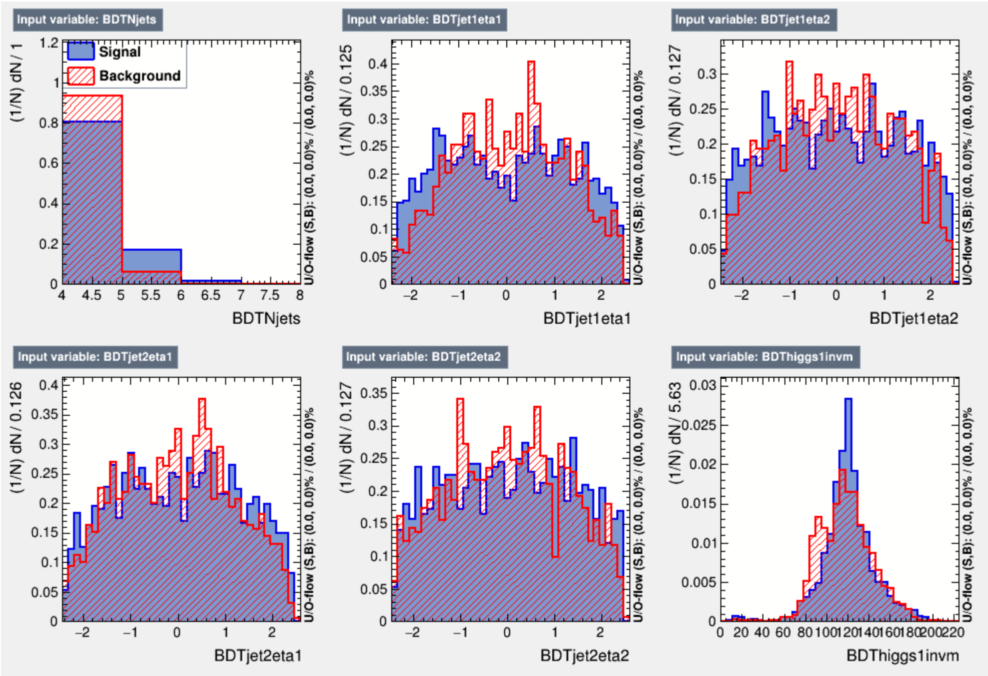
- With TMVA, a built-in package in ROOT
  - Support different ML approach:
    - Boosting Decision Tree
    - Multilayer perceptron (a type of feed-forward Artificial Neural Network)
    - Deep Neural Network
  - External Machine learning framework supported:
    - XGBoost (I used in CMSDAS but not through TMVA), keras
- Transfer to python, use other external framework like TensorFLow or PyTorch
  - Need to switch Root data to panda or numpy
  - Need installation

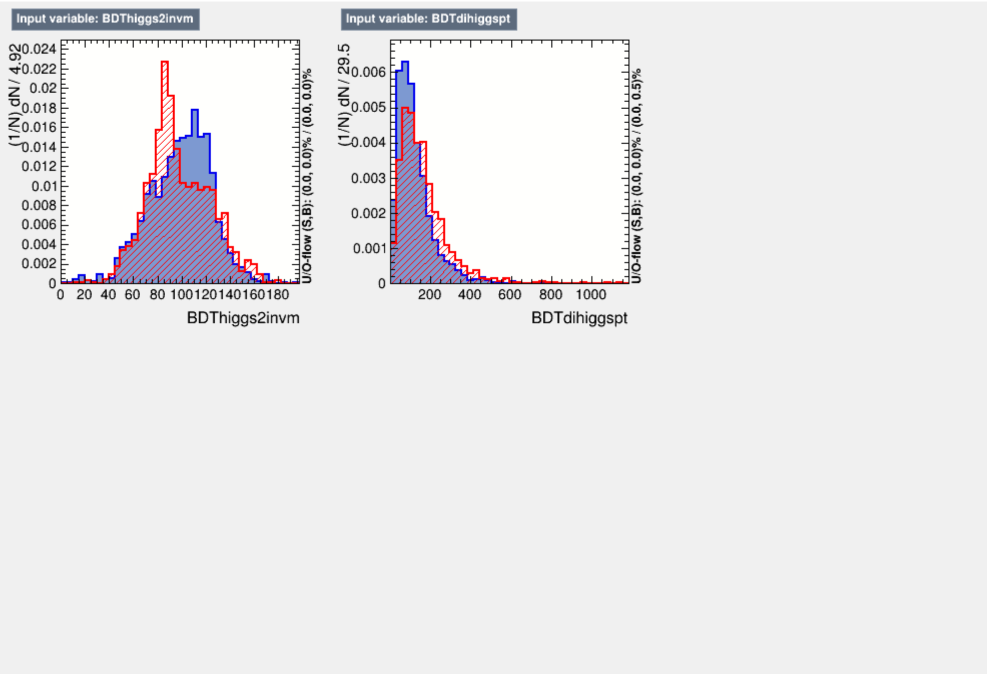
Trained variables

- Number of jets in the event
- p_T for all four jets, higgs, and di-Higgs system
- η and ϕ for all four jets, higgs, and di-Higgs system
- Invariant mass for Higgs and di-Higgs system
- (B-Tagging for all four jets, MET, deltaEta for higgs, deltaPhi for higgs …..)

Trained variables

- For most of the variables, sig is almost identical with bkg. Select following for training:



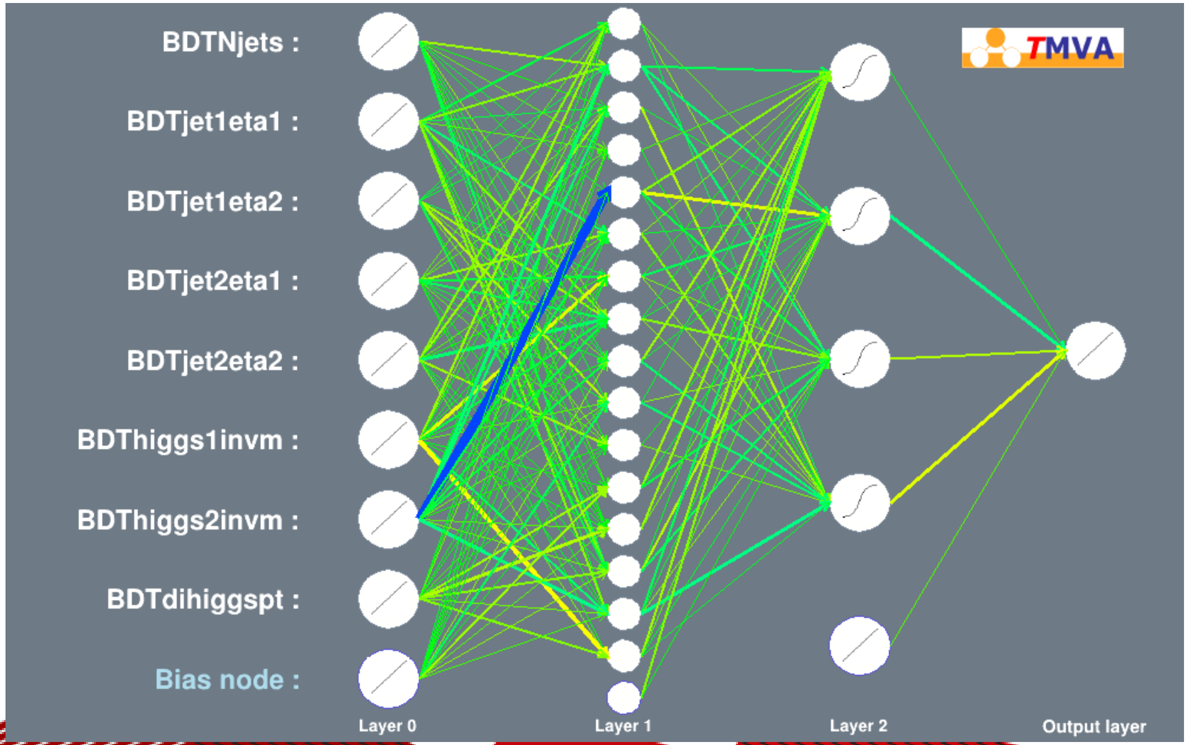


Boosting Dicision Trees

- 5000 trees, MaxDepth =6
- BoostTypr = RealAdaBoost,
- Separation Type: Gini Index
- No pruning since we don’t worry about overtraining for our case

Multilayer Perceptron (MLP_ANN)

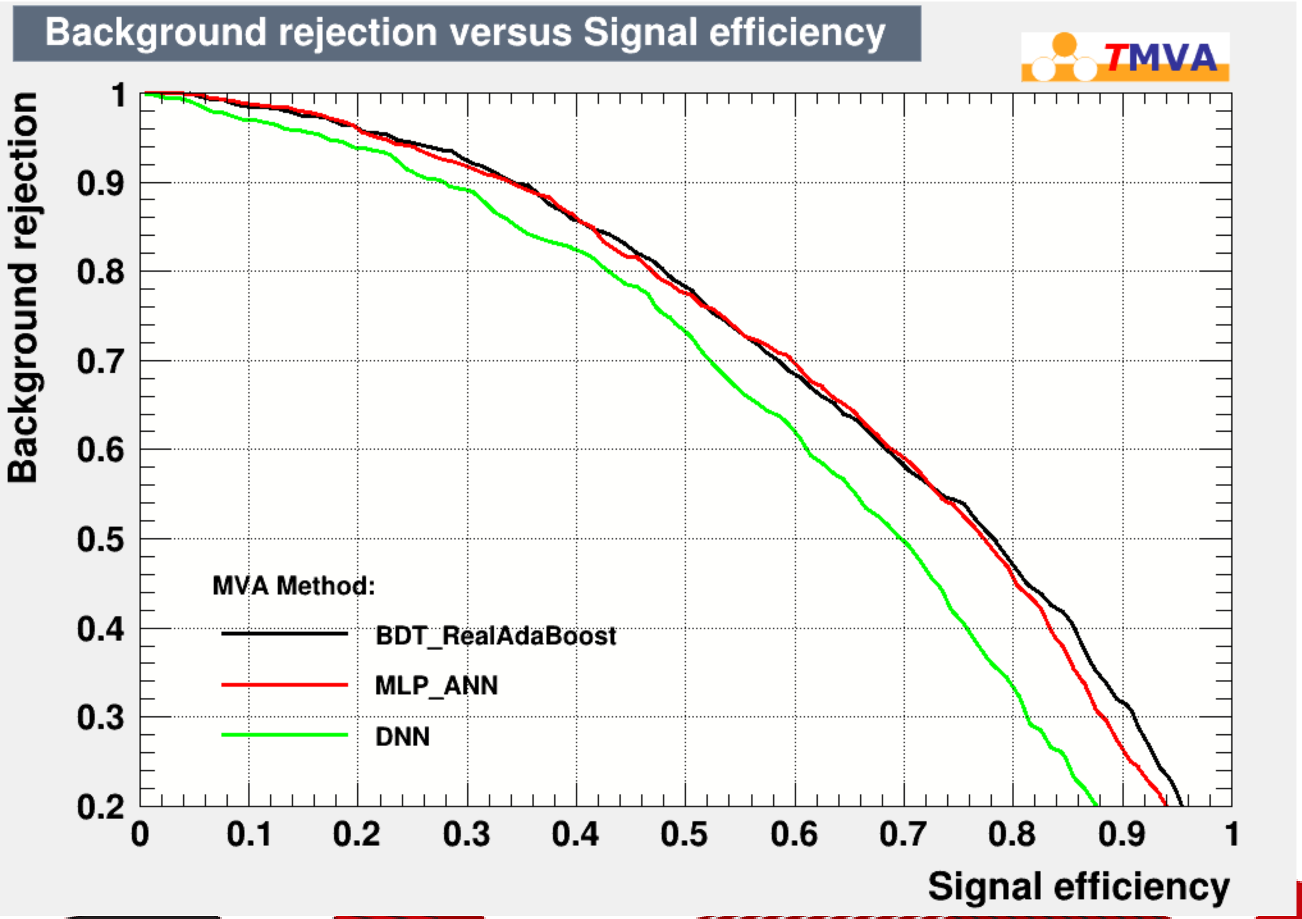
- 2 layer of sigmoid node, with back propagation
- Learning rate = 0.05



Deep Neural Network (DNN)

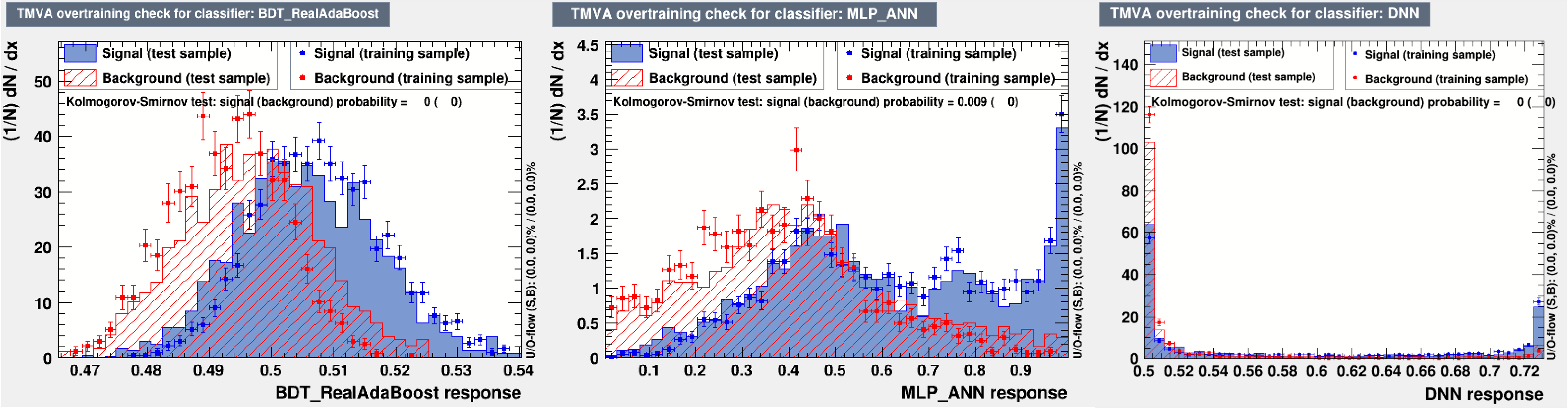
- Current two hidden layer of sigmoid nodes, first layer 1024 nodes, second later 4 nodes.
- Learning rate = 2e-4
- Deep NN with more than 3 layer all diverge, need to check

Performance for different methods



- BDT with RealAdaBoost perform the best, AUC=0.712586

response for each methods



Next Week

- Add JetFlavorAssociation to anti-kt
- Try more training variables
- Try different structure of DNN
- Hyperparameter tuning for BDT
- Application of the model