# Mass-decorrelated Xbb tagger

Wei Ding, Tsinghua University

On behalf of Xbb tagging group

26 Sep 2019, **Flavour Tagging Algorithm meeting**

# Mass-decorrelated Xbb tagger

- Hbb tagging
  - H->bb with largest branching factor in Standard Model
  - Identification of jets containing pairs of b-hadrons
- Motivation
  - Tagger developed (reference) by Julian using Neural Network is good in performance but highly mass correlated
  - Develop a mass de-correlated Hbb tagger (XbbScore) using Neural Network also with good performance

# Sample processing

- Framework
  - https://github.com/computational-physcis-2011301020083/Mass-decorrelated-Xbb-tagger
  - Tools: Keras, TensorFlow, Numpy, H5py, Pandas
- Used samples:
  - https://gitlab.cern.ch/atlas-boosted-hbb/xbb-datasets/blob/master/p3870/h5-datasets-VRalg_0919-mc16a.txt
- Hbb samples:
  - GhostHBosonsCount>=1
- Top samples:
  - GhostTQuarksFinalCount>=1
- Dijets samples
  - Slices JZ0W to JZ12W

# Sample processing

| | Dijets | Hbb | Top | Total |
|---|---|---|---|---|
| Training | 4M | 2M | 2M | 8M |
| Validation | 1M | 0.5M | 0.5M | 2M |
| Testing | ~4.5M | ~0.8M | ~1M | ~6.3M |
| Total | ~9.5M (sub-sample) | ~3.3M | ~3.5M | ~16.3M |

# Training Features

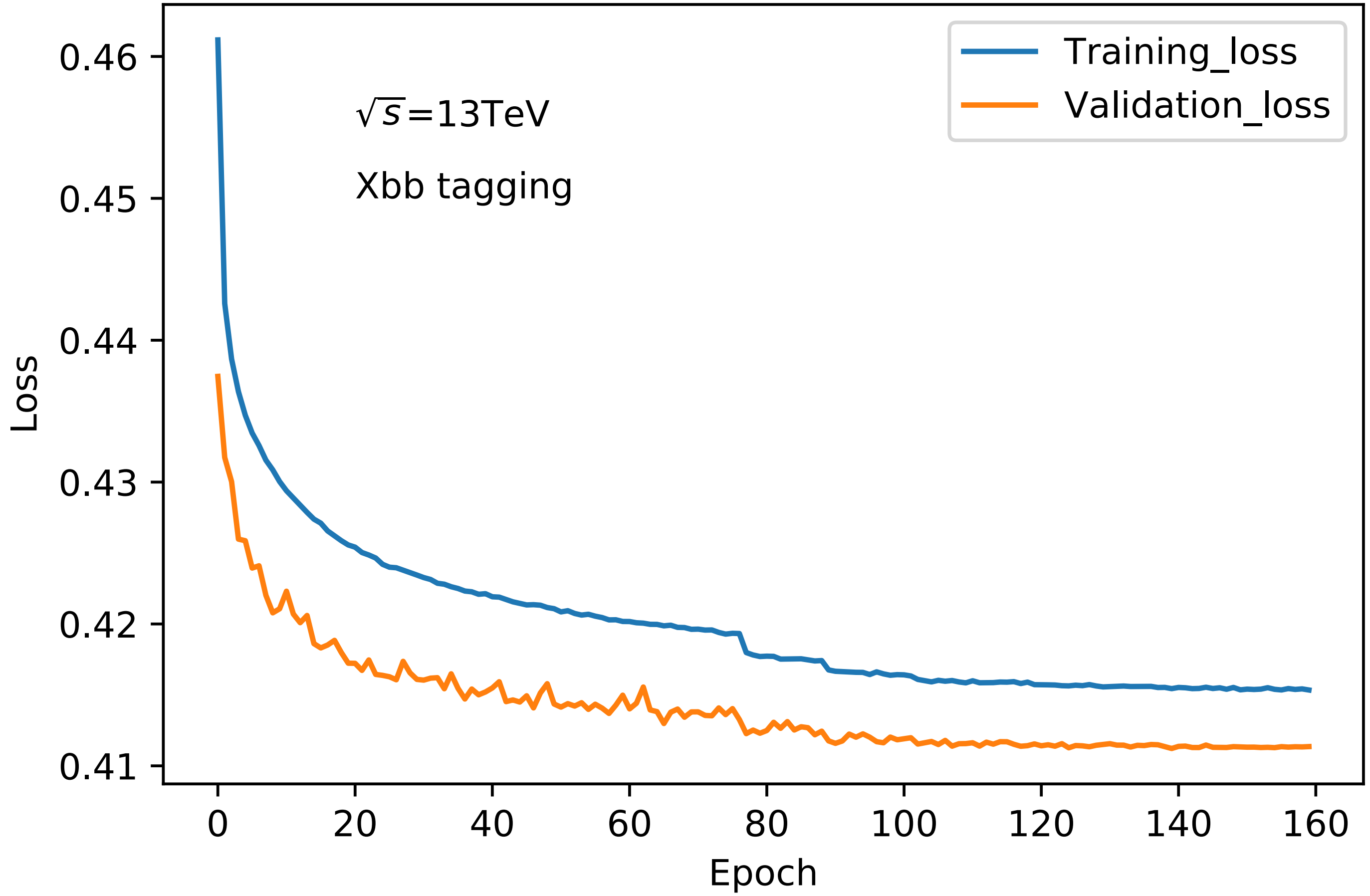
- Fat Jet:
  - Kinematics: ['pt','eta’]
- Subjet_VRGhostTag_{1,2,3}:
  - JetFitter: ['JetFitter_N2Tpair', 'JetFitter_dRFlightDir', 'JetFitter_deltaeta', 'JetFitter_deltaphi', 'JetFitter_energyFraction', 'JetFitter_mass', 'JetFitter_massUncorr', 'JetFitter_nSingleTracks', 'JetFitter_nTracksAtVtx', 'JetFitter_nVTX', 'JetFitter_significance3d’]
  - SV1: ['SV1_L3d', 'SV1_Lxy', 'SV1_N2Tpair', 'SV1_NGTinSvx', 'SV1_deltaR', 'SV1_dstToMatLay', 'SV1_efracsvx', 'SV1_masssvx', 'SV1_significance3d']
  - IPRNN: ['iprnn_pu', 'iprnn_pc', 'iprnn_pb’]
- Totally 71 training features without mass correlation

# Network Architecture

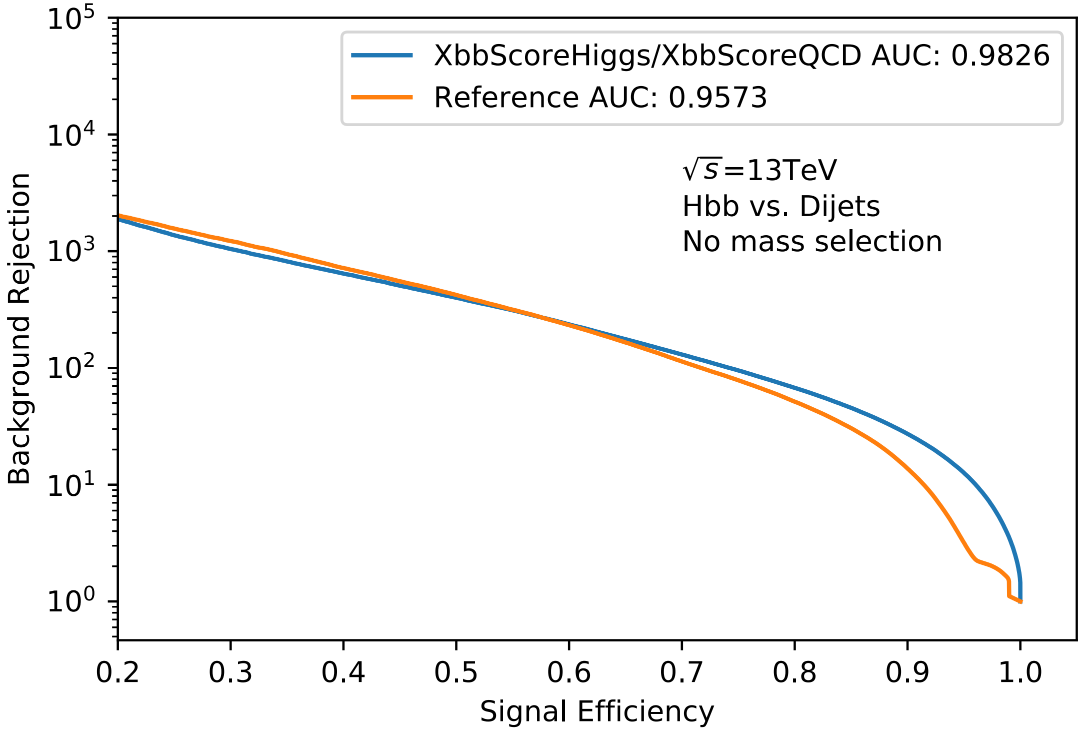
| Features | Training Features<br>( VRGhostTag ) |
|---|---|
| Optimizer | Adam |
| Number of hidden layers | 6 |
| Nodes per layer | 250 |
| activation | Relu |
| learning rate | 0.01 |
| decay | 0.00001 |
| epochs | 300 |
| loss | categorical crossentropy |

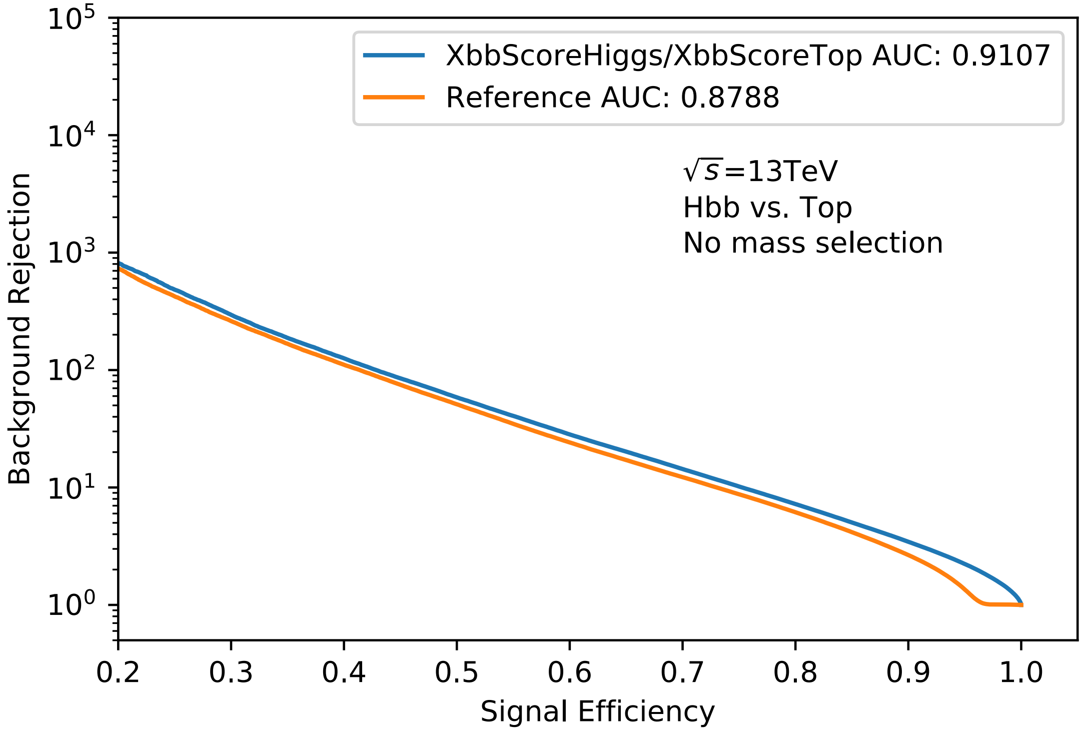
# Over-fitting Check

- Loss vs. Epoch

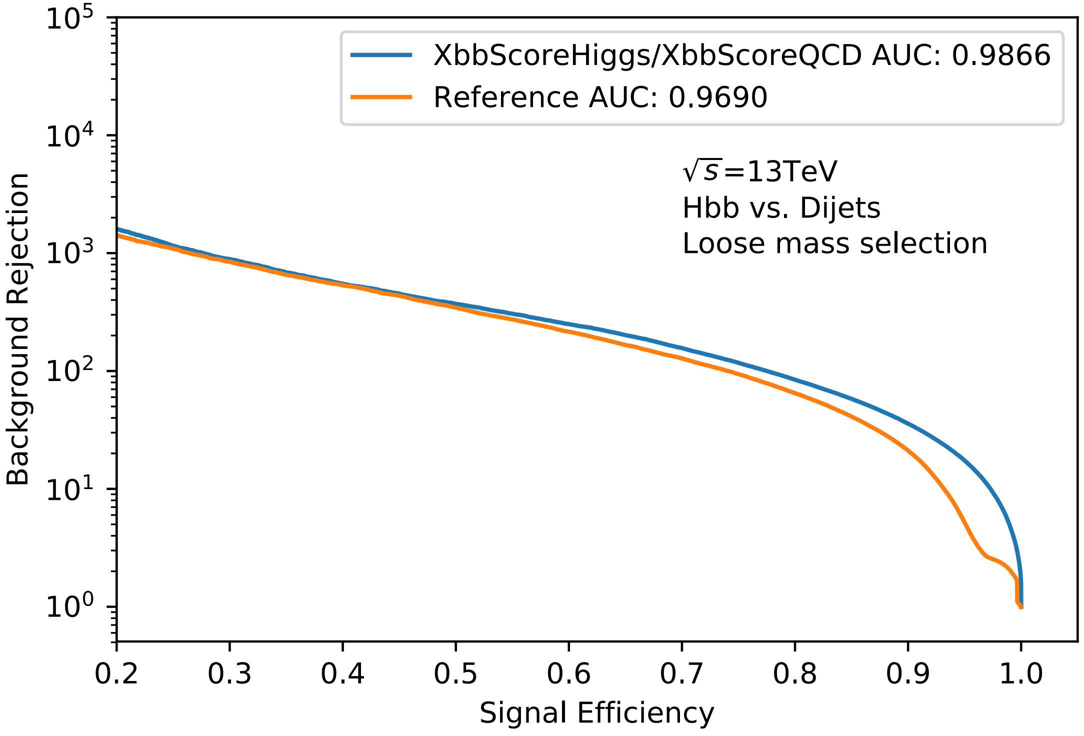


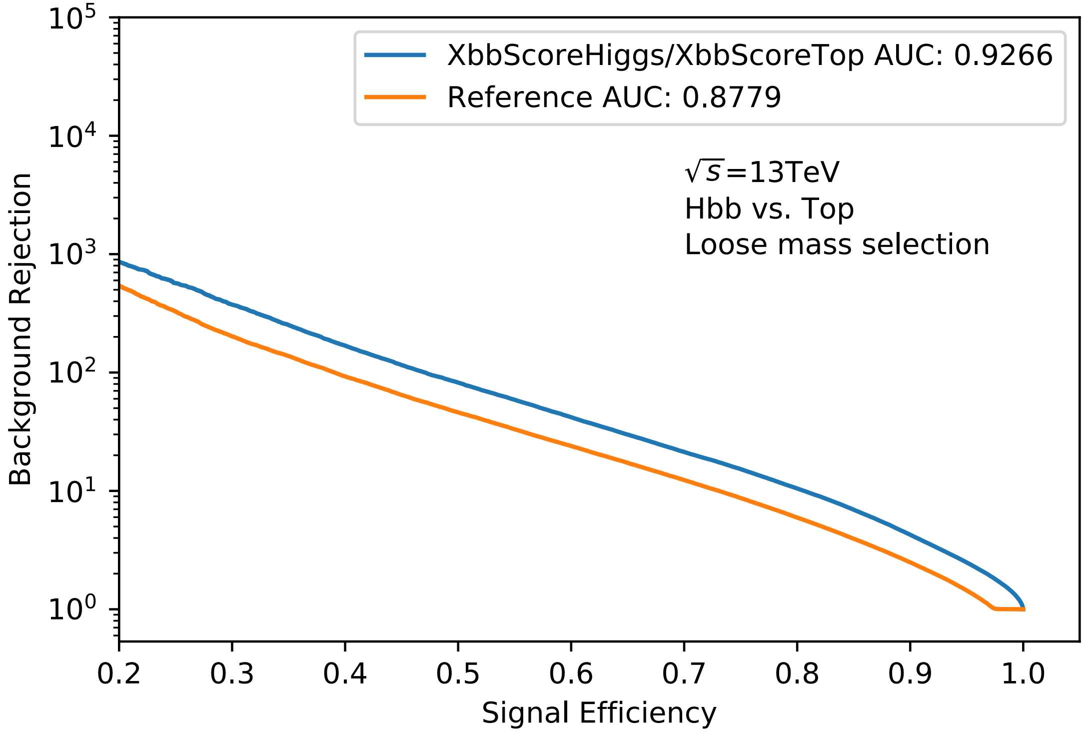
# ROC: Full range



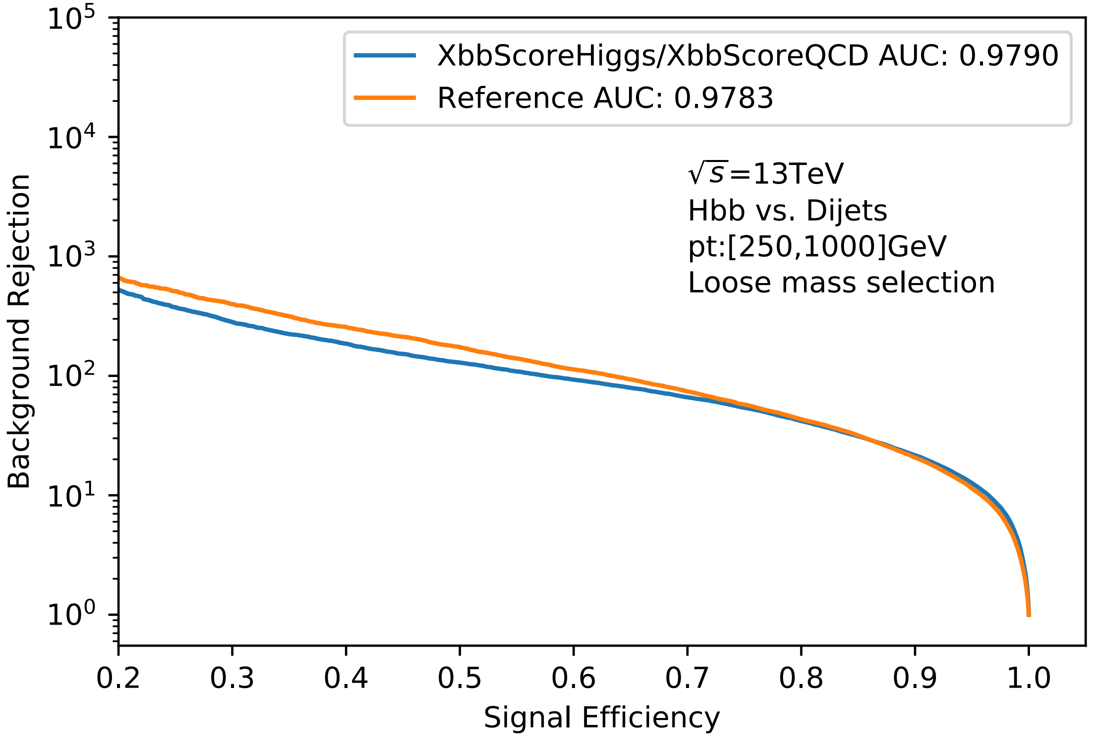



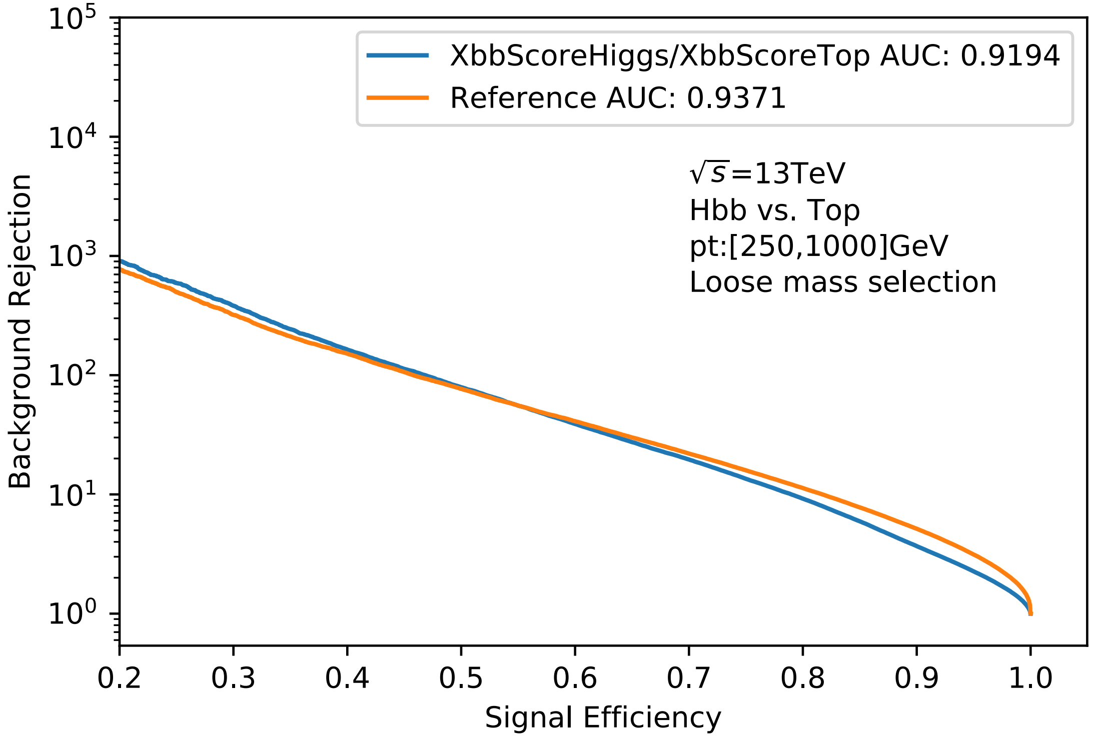
# ROC: Loose mass window [84,138]GeV contain 80% of Higgs jets



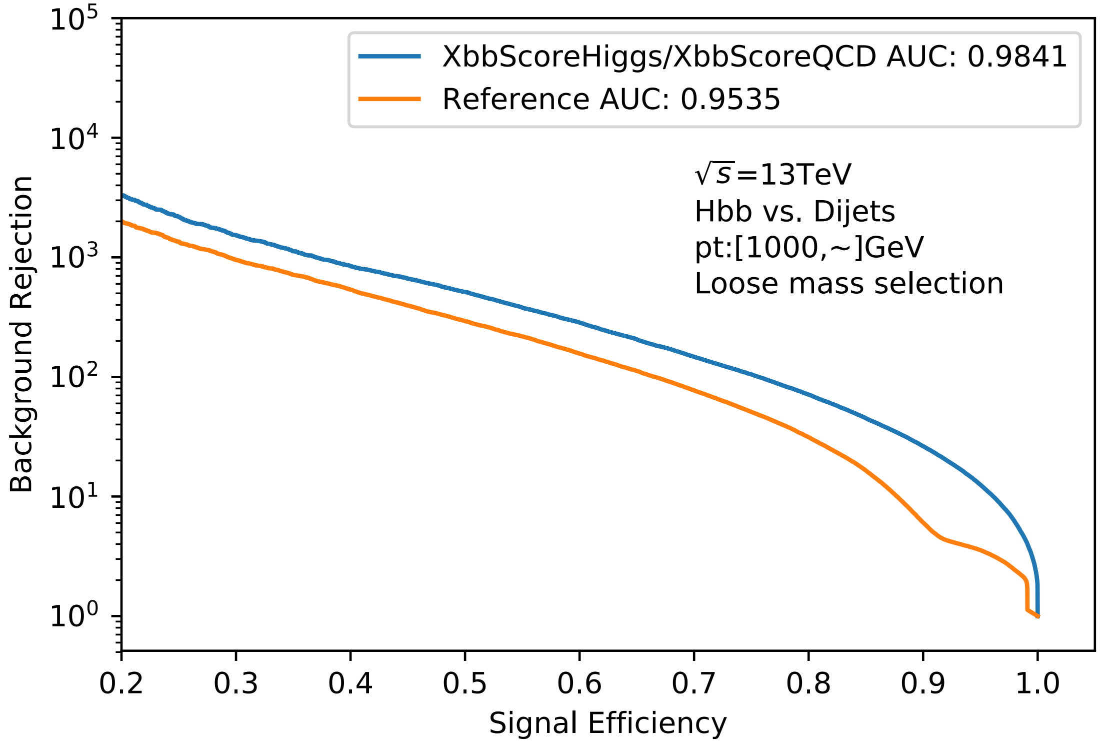



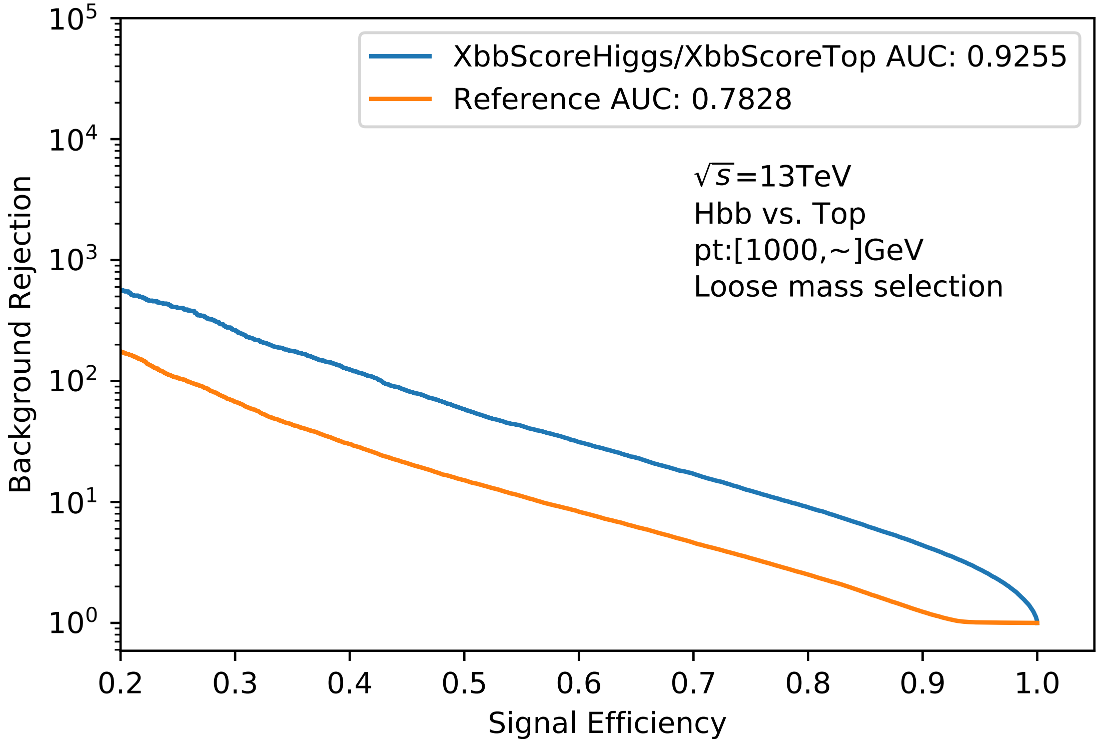
# ROC: Loose mass window in low pt range



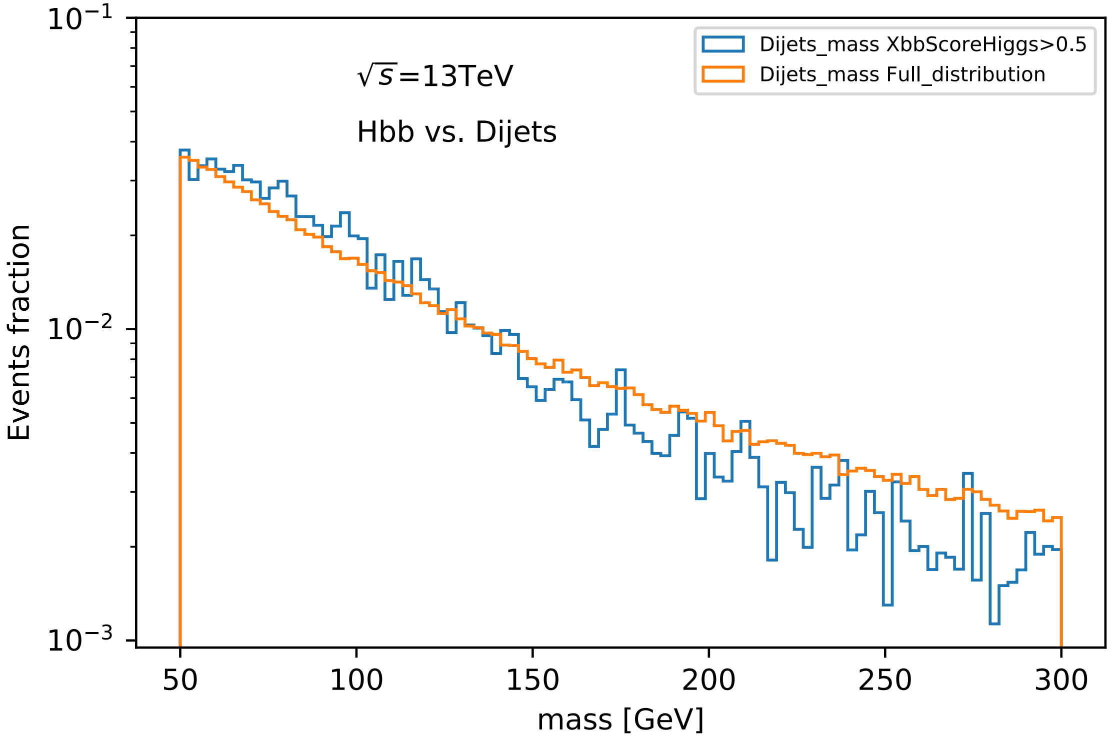



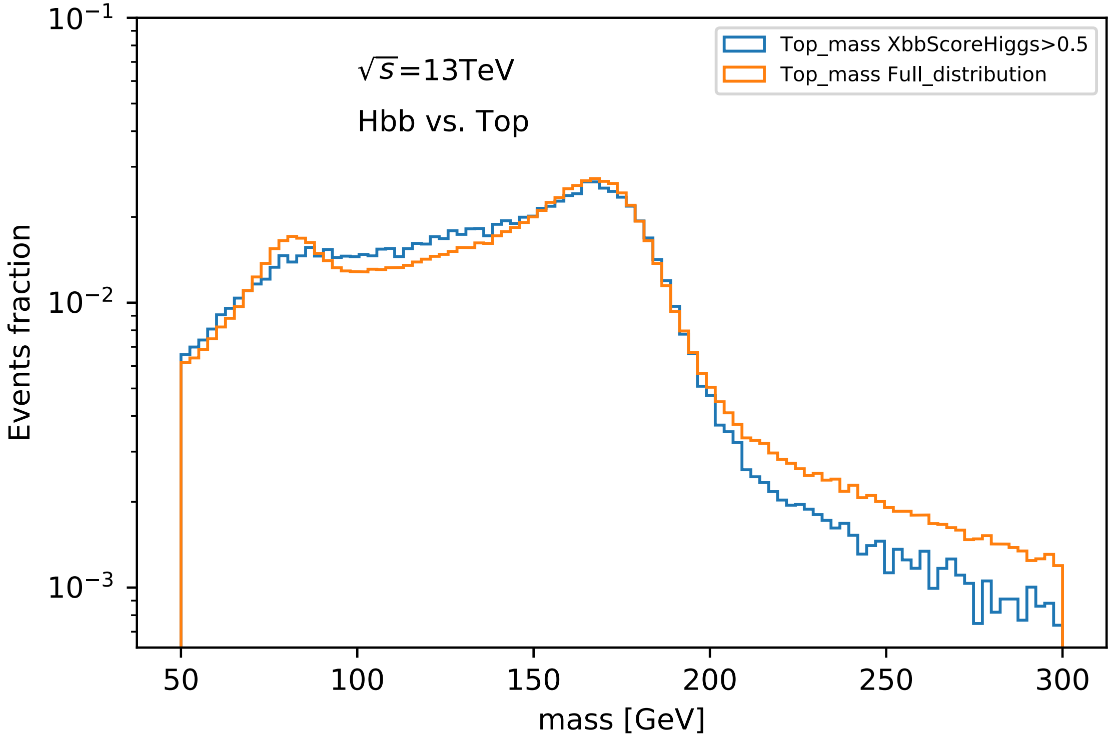
# ROC: Loose mass window in high pt range



# Jet mass distribution



# How to implement

- 1, Reduce sample size and select used features
  - In directory reweight/
- 2, Dijets sampling and split sample into training validation and testing
  - In directory process/
- 3, Sample scaling
  - In directory prepare/
- 4, Training
  - In directory train/
- 5, Performance study
  - In directory study/

# How to implement

Key steps involved in my implementation:

- 1, Dijets sampling
  - Dijets sample size is too large
- 2, Sample splitting to training validation and testing
  - Requested in training
- 3, Sample scaling
  - Some events with just one or two subjets and the features in the second or third jet are replaced by Numpy.nan
- 4, Training
  - Reweight strategy and optimizer

# 1, Dijets sampling

- Merge the Dijets samples by DSID first
  - In script process/mergeDijets/MergeDijetsDSID.py
    - Function Numpy.vstack()
- Randomly choose 0.035 of full samples in each DISD then
  - In script  process/mergeDijets/subsampleDijets.py
    - Function 1 select_index=Numpy.random.choice(N,int(0.035*N),replace=False)
    - Function 2 select_sample=Numpy.take(FullSample,select_index,axis=0)
- Merge these different DSID samples after sampling
  - In script process/mergeDijets/MergeDijets.py
    - Function Numpy.vstack()

# 2, Sample splitting to training validation and testing

- Merge one type of sample into one h5py file with used features
  - Merge hdf files in script reweight/MergeDatasets.py
    - Function Pandas.concat()
  - Convert hdf file into h5py file in script process/flatten.py
    - Function Pandas.DataFrame.values
  - Finally get three h5py files
    - MergedDijets.h5  MergedHbb.h5  MergedTop.h5

# 2, Sample splitting to training validation and testing

- Split these samples into training validation and testing
  - In script process/split.py (np == Numpy)
    - train_index=np.random.choice(N,2000000,replace=False)
    - remain_index=np.setdiff1d(np.arange(0,N),train_index)
    - valid_index=np.random.choice(remain_index,500000,replace=False)
    - test_index=np.setdiff1d(remain_index,valid_index)
    - {train,valid,test}_sample=np.take(FullSample,{train,valid,test}_index,axis=0)

# 2, Sample splitting to training validation and testing

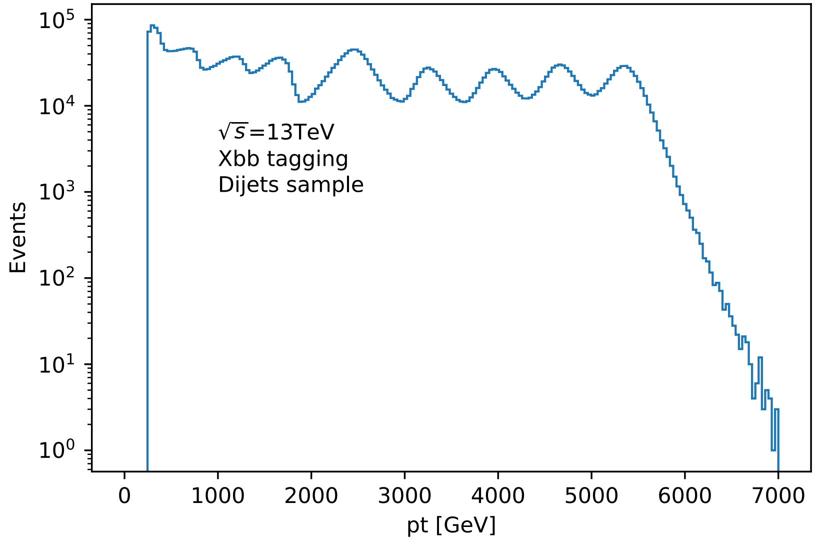
- Label signal and background samples and Merge them by {training, validation, testing}
  - In script prepare/prepare.py
    - label=np.full((N,),0,dtype=int)
    - label_Dijets=keras.utils.to_categorical(label, num_classes=3)
      - [1,0,0]==Dijets sample  [0,1,0]==Hbb sample [0,0,1]==Top sample
    - train_Dijets=np.hstack((label_Dijets,Dijets_sample))
    - train_data=np.vstack((train_Dijets,train_Hbb,train_Top))

# 3, Sample scaling

- Calculate mean and standard deviation
  - In script prepare/calculateMean.py
    - mean_vector = np.nanmean(training_sample, axis=0)
    - std_vector = np.nanstd(training_sample, axis=0)
- Scaling samples
  - In script prepare/scaling.py
    - training=(training-mean_vector)/std_vector
      - The np.nan is still np.nan after scaling
    - new_training=np.nan_to_num(training)
      - Convert these np.nan into number 0

# 4, Training

- Reweighting strategy
  - Three reweighting strategies are tried
    - Nominal weight which create a smooth falling pt distribution in Dijets sample
    - Reweighting to flat pt
    - Equal weight: all events are weighted to 1
  - Equal weight was chosen by comparison
  - All events are weighted to 1 in training in script train/train.py
    - train_w=np.full((8000000, ), 1)
  - Pt distribution with equal weight in Dijet sample →



# 4, Training

- Optimizer
  - Two optimizers SGD and Adam are tried
  - Adam was chosen by comparison
  - Adam is used in script train/train.py
    - from keras.optimizers import Adam
    - adm = Adam(lr=params['learning_rate'], decay=params['lr_decay'])

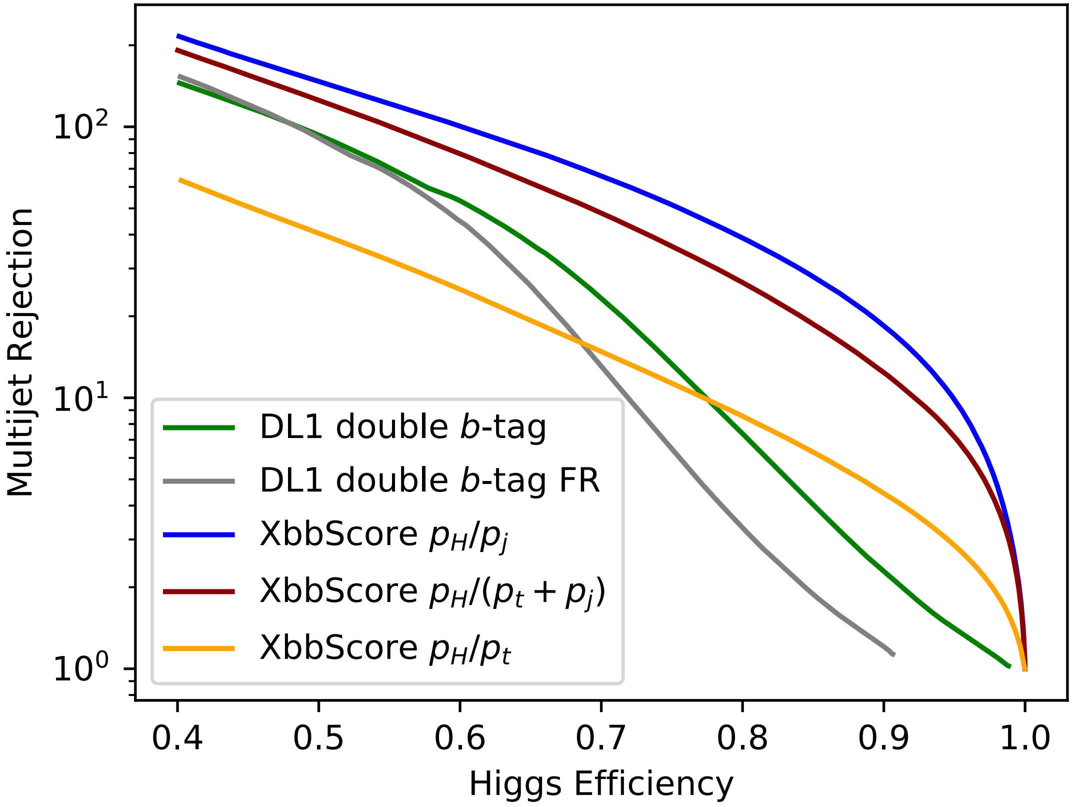
# Summary

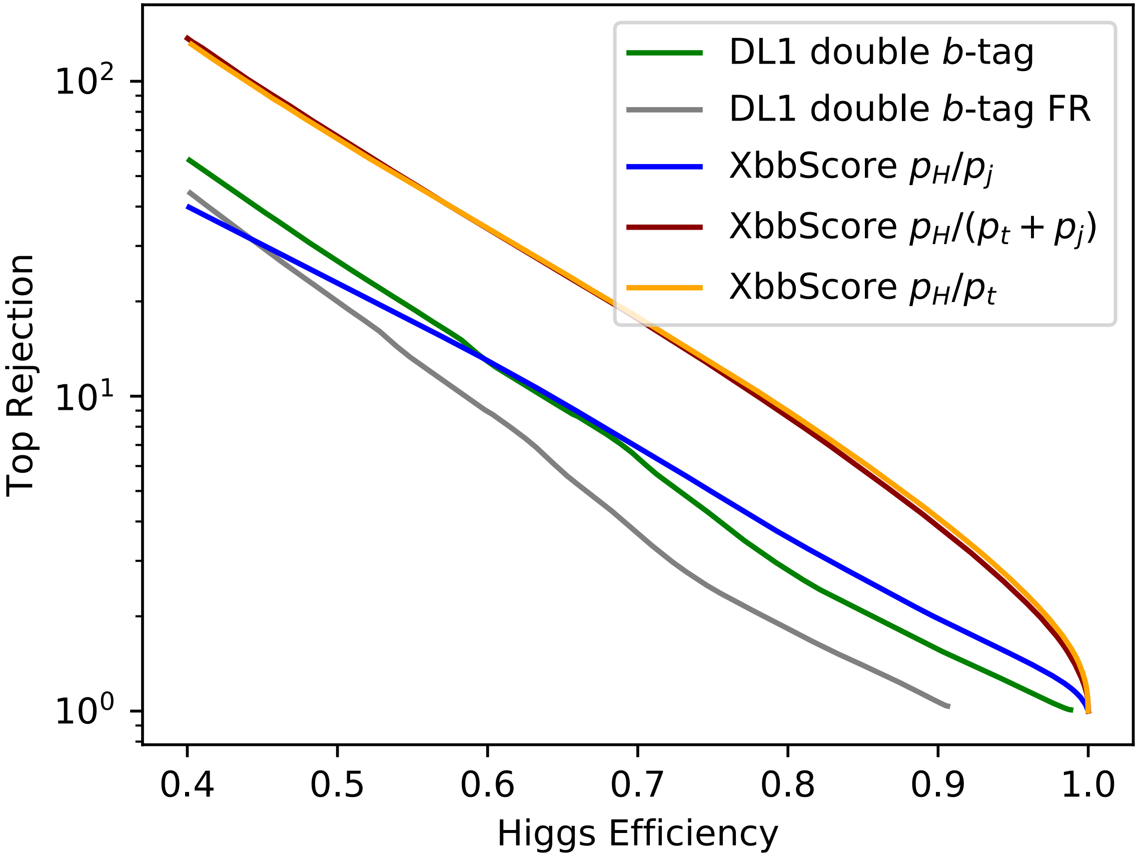
- Develop a mass-decorrelated Xbb tagger with good performance
- Tagger implementation and the keys technologies are showed:
  - Selecting training feature
  - Dijets sampling
  - Sample splitting
  - Sample scaling
  - Training strategy
- Gang is working on the implementation this to Athena

# Back up

# Validation from Daniel Williams

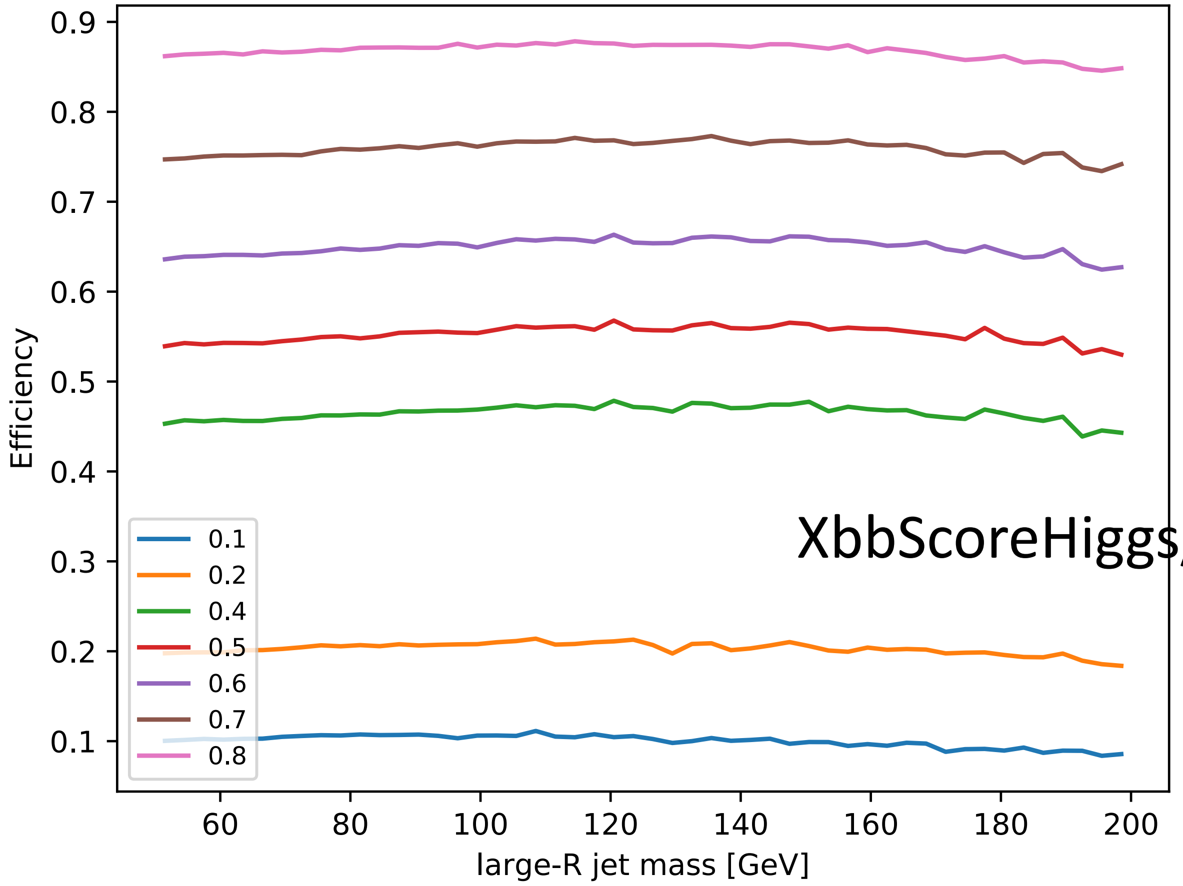
- ROC



# Validation from Daniel Williams

- Jet mass vs. Higgs efficiency




XbbScoreHiggs/(0.5*XbbScoreQCD+0.5*XbbScoreTop)