# Research Summary

Vector Boson Scattering Analysis in ZZ + 2 jets Production
with Toolkit for Multivariate Analysis

Zunran Guo[1]

Supervised by: Prof. Bin Zhou[2], Dr. Yusheng Wu[2,3]

[1]University of Rochester
[2]University of Michigan
[3]Institute of Physics, Academia Sinica

December 2016

# Contents

# 1 Introduction

## 1.1 Physics Background and Purpose of Measuring

Vector-boson-scattering (VBS) two-boson production gives a unique opportunity to examine the nature of electroweak symmetry breaking (EWSB) in the Standard Model (SM).

The scattering of gauge bosons violates unitarity at the TeV scale if there is no Higgs boson. The unitarity can be restored by including the Higgs bosons which leads to a delicate cancellation of divergence at high energy. This mechanism can be tested via measuring the VBS production cross sections. Any anomalies will bring up questions into whether the Higgs boson is as predicted in the SM; and whether the EWSB mechanism is as predicted in the SM.

Previously, the luminosity at the Large Hadron Collider (LHC) is not high enough to detect the scattering of gauge bosons. It is until recent that we are able to collect possible valid signals from the detectors at $\sqrt{s} = 13$ TeV. This upgrade offers the feasibility to identify the events involved in the scattering process and this research is thus dedicated to the techniques to help identify such events in the scattering process.

All the analysis in this research is based on the truth events generated from the theory-level simulation program VBFNLO. In the previous studies, the classification in comparison was made based on direct threshold cuts on those discriminant variables, such as mass of jets, change of pseudorapidity, transverse momentum of bosons, etc. For example, all the events with the mass of 2 jets ($m_{jj}$) greater than 500 GeV and the change of pseudorapidity of 2 jets ($|\Delta\eta_{jj}|$) greater than 3 could be classified as the signal events and all the other events that do not pass this criteria are classified as the background events.

This research, however, will focus on identifying the signal events and the backgrounds events we are interested in through a trained classifier based on selected discriminant variables. While we try to improve the traditional classification technique through a machine learning approach. 1 million Monte Carlo simulated signal events and background events are first generated respectively as the training data to train the classifier. Once the classification result is proved to be reliable, this well-trained classifier can be tested on the detector-level simulation or extended and adapted for other data and processes in the future. After the detector-level simulation, the real data collected from the LHC could then be reconstructed to measure the final state particles in the collisions and hopefully we will find out whether certain reaction processes really happened in those collision events. [1]

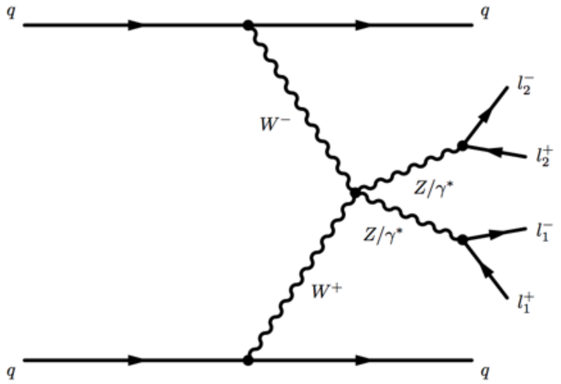
One potential problem in this research is that the simulations do not describe the data perfectly and we usually need to compare different simulations for consistency check.

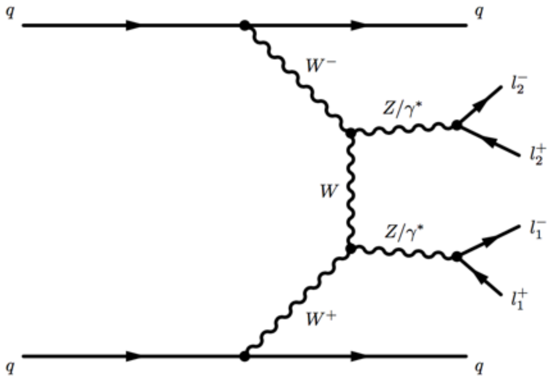
## 1.2 Signal: VBS

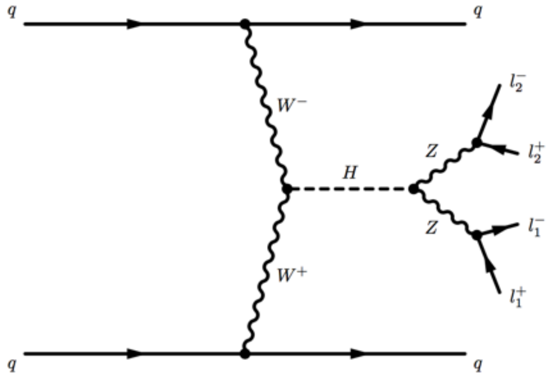
The signal is generated by Z pair + 2 jets production in vector boson fusion with decay into charged leptons:

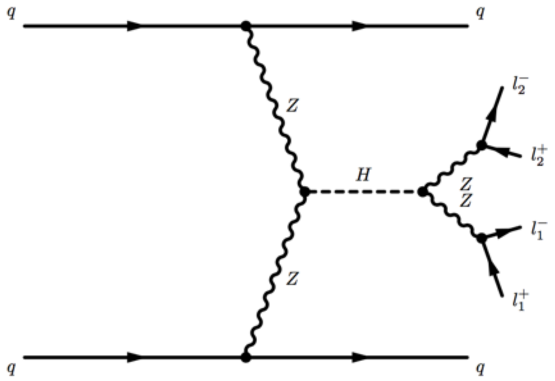
p p → Z Z $jj \rightarrow l^+ \ l^- \ l^+ \ l^- \ jj$

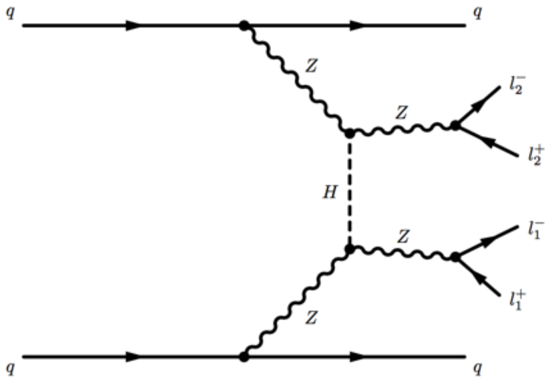
The following Feynman diagrams give some examples of this process:









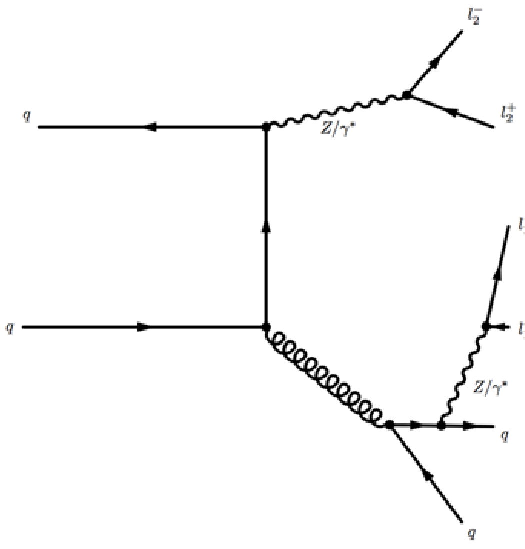


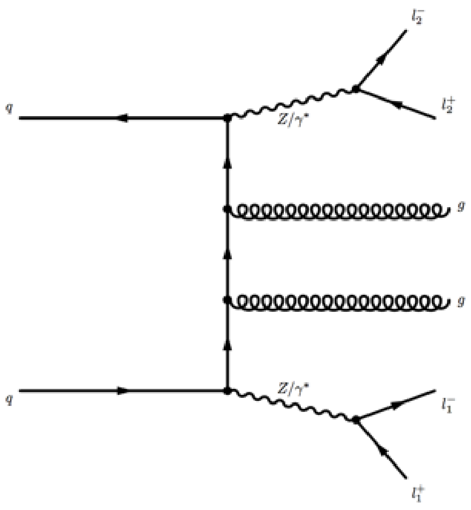
## 1.3 Background: QCD

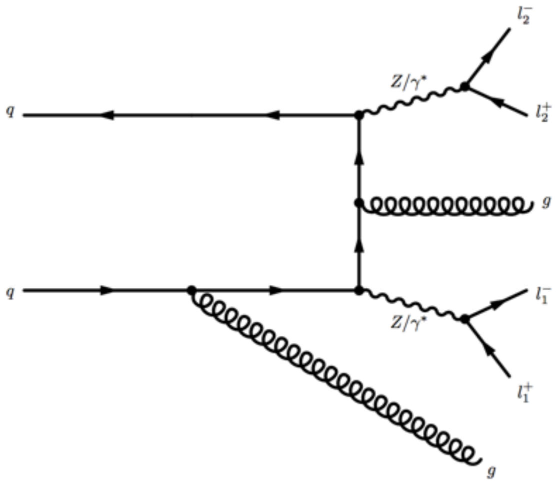
The major background is generated by QCD induced Z Z + 2 jets production with fully leptonic decay:

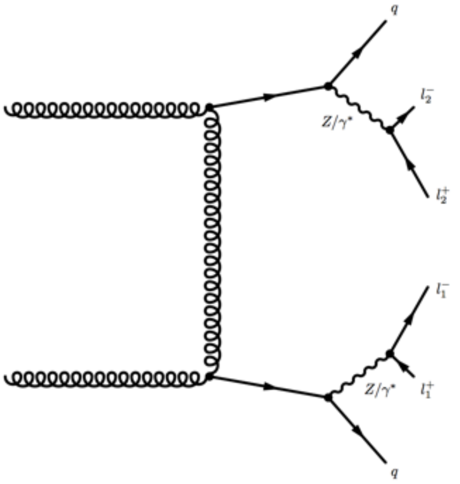
p p → Z Z $jj \rightarrow l^+ \; l^- \; l^+ \; l^- \; jj$

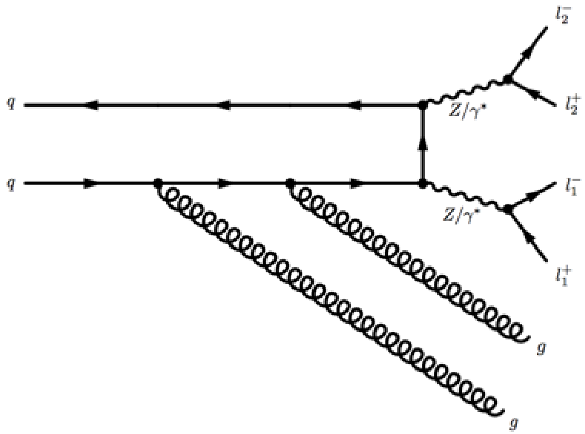
The following Feynman diagrams give some examples of this process:

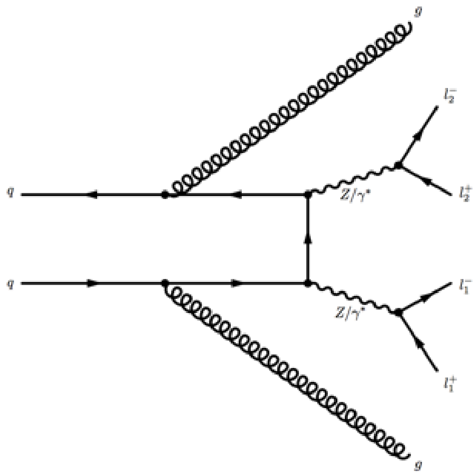

$l_2^-$
$q$
$Z/\gamma^*$
$l_2^+$
$l_1^-$
$q$
$l_1^+$
$Z/\gamma^*$
$q$
$q$

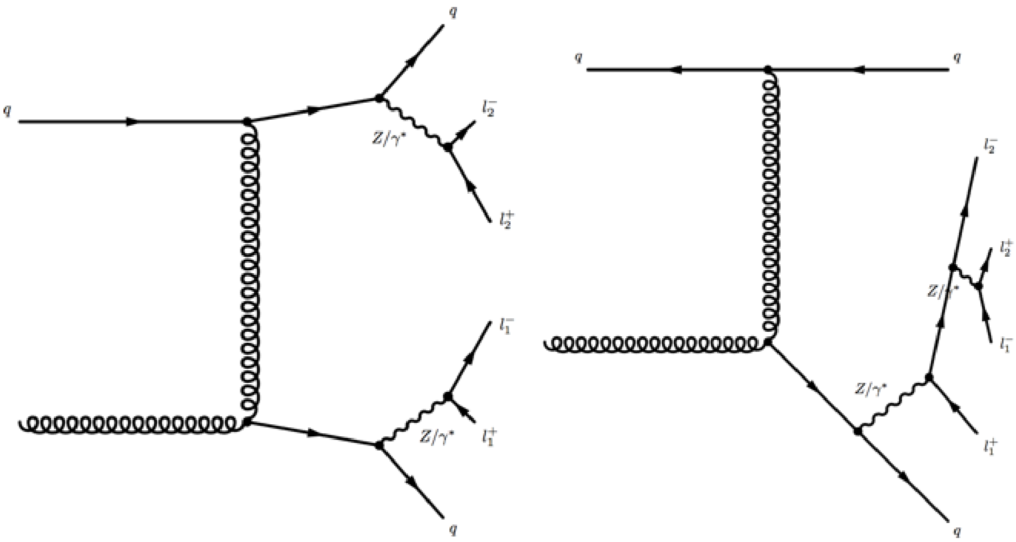

$l_2^-$
$q$
$Z/\gamma^*$
$l_2^+$
$g$
$g$
$q$
$Z/\gamma^*$
$l_1^-$
$l_1^+$


$l_2^-$
$Z/\gamma^*$
$q$
$l_2^+$
$g$
$q$
$l_1^-$
$Z/\gamma^*$
$l_1^+$
$g$


$q$
$l_2^-$
$Z/\gamma^*$
$l_2^+$
$l_1^-$
$Z/\gamma^*$
$l_1^+$
$q$


$l_2^-$
$q$
$Z/\gamma^*$
$l_2^+$
$q$
$l_1^-$
$Z/\gamma^*$
$l_1^+$
$g$
$g$


$g$
$l_2^-$
$q$
$Z/\gamma^*$
$l_2^+$
$q$
$l_1^-$
$Z/\gamma^*$
$l_1^+$
$g$



# 2 Computing Environment Setup

## 2.1 Remote Login

Besides the program execution that requires large computational power, almost all the scripts and executable files are run on the University of Michigan (UM) and CERN's server.

The login Bash script from local (terminal on Mac OS or Bash Shell on Windows OS) is straightforward as the following:

```
ssh –X user_name@umt3int01.physics.lsa.umich.edu
```

There are 4 equivalent servers available at the time:

- umt3int01.physics.lsa.umich.edu
- umt3int02.physics.lsa.umich.edu
- umt3int03.physics.lsa.umich.edu
- umt3int04.physics.lsa.umich.edu

For login convenience, one can also create a SSH key to prevent the recurring prompt for password when working interactively with the remote server. The procedure to set up the environment can be referred on the website.

## 2.2 VBFNLO

VBFNLO is a fully flexible parton level Monte Carlo program for the simulation of vector boson fusion, double and triple vector boson production in hadronic collisions at next to leading order in the strong coupling constant. VBFNLO includes Higgs and vector boson decays with full spin correlations and all off-shell effects. In addition, VBFNLO implements CP-even and CP-odd Higgs boson via gluon fusion, associated with two jets, at the leading order one loop level with the full top-quark and bottom-quark mass dependence in a generic two Higgs doublet model.

A variety of effects arising from beyond the Standard Model physics are implemented for selected processes. This includes anomalous couplings of Higgs and vector bosons and a Warped Higgsless extra dimension model. The program offers the possibility to generate Les Houches Accord event files for all processes available at leading order.[2]

VBFNLO version 2.7.1 is used for the event generation remotely on the UM's server.

## 2.3 LoopTools

In order to include the electroweak corrections, the program LoopTools is required for VBFNLO.

LoopTools is a package for evaluation of scalar and tensor one-loop integrals based on the FF package by G.J. van Oldenborgh. It features an easy Fortran, C++, and Mathematica interface to the scalar one-loop functions of FF and in addition provides the 2-, 3-, and 4-point tensor coefficient functions. [4]

The installation process is well detailed on the website.

## 2.4 ROOT

ROOT is a modular scientific software framework. It provides all the functionalities needed to deal with big data processing, statistical analysis, visualisation and storage. It is mainly written in C++ but integrated with other languages such as Python and R.

Pretty much all the research analysis is focused around the ROOT framework. The installation process is well detailed on the website.

## 2.5 Events Generation

Once the installation of LoopTools and ROOT is complete. The following command could be run to install VBFNLO:

```
   ./configure --prefix=[installation destination path] --enable-processes=all
--with-LOOPTOOLS=[LoopTools path] --with-root=[ROOT path] --enable-shared=no
```

Note that the default ROOT path on UM's server is `/cvmfs/atlas.cern.ch/repo/ATLASLocalRootBase /x86_64/root/6.04.14-x86_64-slc6-gcc49-opt/`. In addition, one can always enable more installation options referred in the manual. [3]

1 million events are then generated for signal (VBS) and background (QCD) respectively from VBFNLO with $e^+ \; e^- \; e^+ \; e^- \; jj$ final state. The generation level configuration can be referred in the following tables:

Table 1: VBFNLO Generation Level Setting (1)

| | VBS jet cuts | VBS jet cuts |
|---|---|---|
| min jet-jet R separation | 0 | 0 |
| max parton pseudorapidity | 5 | 5 |
| exponent of generalised $k_T$ algorithm | -1 | -1 |
| min jet transerverse momentum (pT) | 15 | 15 |
| max jet rapidity | 9 | 9 |

Table 2: VBFNLO Generation Level Setting (2)

| | VBS lepton cuts | VBS lepton cuts |
|---|---|---|
| max lepton rapidity | 2.8 | 2.8 |
| min lepton transerverse mome (pT) | 4.0 | 4.0 |
| min. $m_{l+l-}$ for any comb. of opposite charged leptons | 4 | 4 |
| max. $m_{l+l-}$ for any comb. of opposite charged leptons | 13000 | 13000 |
| min lepton-lepton R separation | 0.01 | 0.01 |
| max lepton-lepton R separation | 50.1 | 50.1 |

Note that one can always change the above configuration for ”tighter” or ”looser” event generation.

The configuration shown in the two tables above is recorded in the cuts and vbfnlo files in `.dat` format (`210_cuts.dat`, `210_vbfnlo.dat`, `3210_cuts.dat`, `3210_vbfnlo.dat`) in the same directory of the executable `vbfnlo` program. The cross-sections of the signals and backgrounds can be saved to a log file when running the Monte Carlo simulation by `./vbfnlo &> log` command. Note that It is always recommended to used `&` at the end of a command. One can always use `disown` command to continue the running task in background even after logging out the sever. Two commands here are recommended for Linux beginners to check the status of the running jobs. One is `ps`, which checks the ”process status” of the current job. The other one is `top`, which checks all the running jobs on the server.

## 2.6 Events Pre-selection

Once the events of the signals and backgrounds are generated and output as `.LHE` files. The `.LHE` files are then converted into ROOT trees by the command `./convert.py .LHE_file_name vbfnlo` with the Python script `convert.py`, where `vbfnlo` is the default tree name across all the scripts used in this research. The output file are named as `output_tree_10000_210_eeee.root` `output_tree_10000_3210_eeee.root` for signal events and background events respectively. One may then run ROOT by `root` command and load the C++ script by `.L VBFNLO_Fiducial_Preselect.cxx` command in ROOT. The following commands `fiducial("output_tree_10000_210_eeee.root", "210_eeee_fid_preselected.root")` and `fiducial("output_tree_10000_3210_eeee.root", "3210_eeee_fid_preselected.root")` specify the output ROOT file names after selecting events that satisfy the pre-selection criteria.

The following criteria are for the pre-selection that filter out the events are we interested in.

- Transverse momentum of lepton 1 (leading lepton) > 25 GeV

- Transverse momentum of leptons 2, 3, 4 > 7 GeV
- Pseudorapidity of leptons 1, 2, 3, 4 < 2.5
- Mass of Z boson (1) and Z boson (2) in the range of (66, 116) GeV

The event selection efficiency is then calculated in the following table:

Table 3: Event Pre-selection

| Selection Criteria | VBS Number of Events | VBS Selection Efficiency | QCD Number of Events | QCD Selection Efficiency |
|---|---|---|---|---|
| None | 890100 | 1 | 1014200 | 1 |
| Pre-selection | 209752 | 0.236 | 11562 | 0.114 |
| Pre-selection and m(jj) > 500 GeV and $\|\Delta\eta(jj)\| > 3$ | 125171 | 0.141 | 9509 | 0.009 |

# 3 Training a Classifier

## 3.1 Discriminant Variable Selection

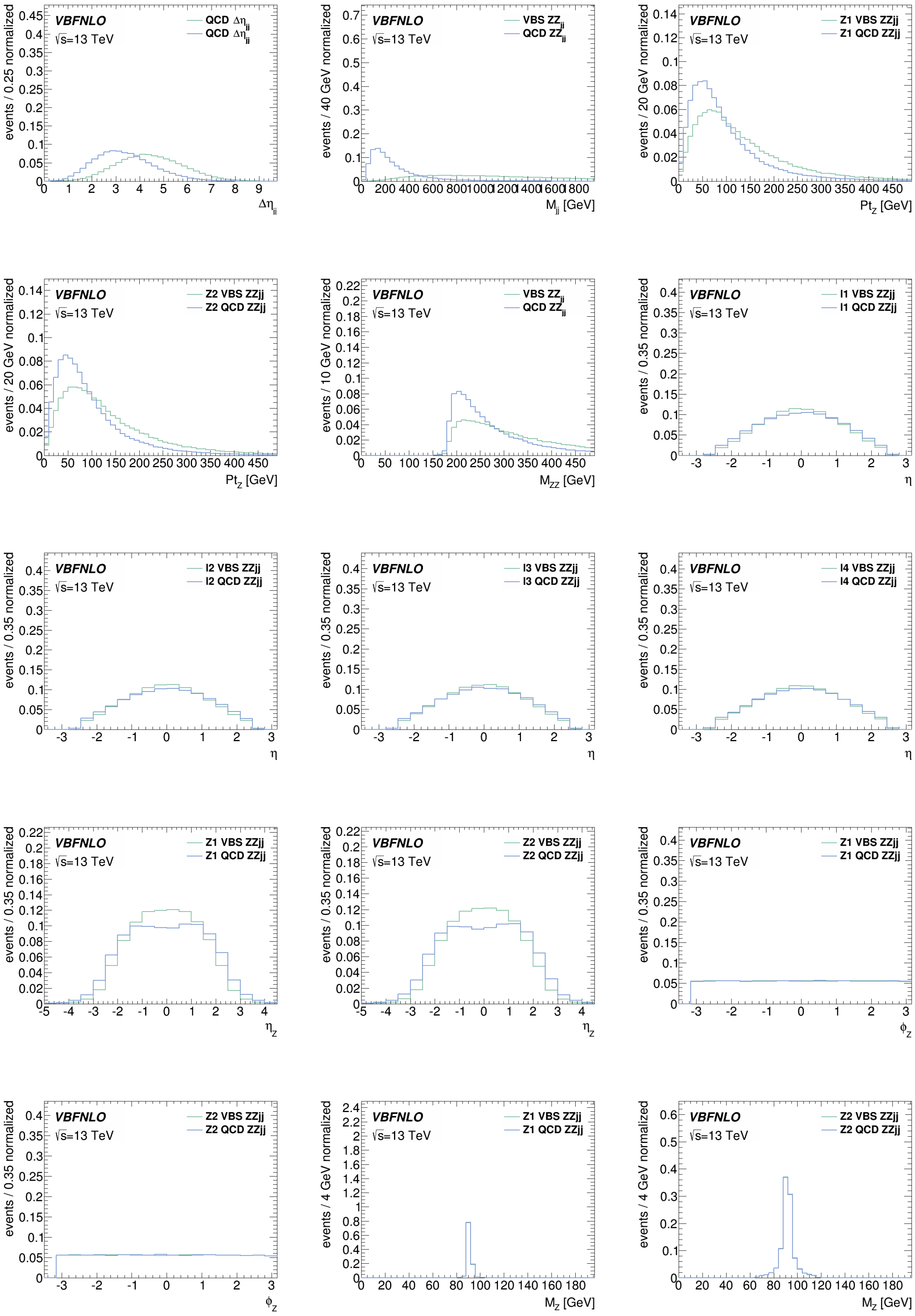
Following the pre-selection of events, we need to select the discriminant variables to be used to train a classifier. C++ script `VBFNLO_Cut.cxx` is used to perform the cumulative cuts according the pre-selection criteria and store the histograms of all the variables after pre-selection. One can use the commands `cut("output_tree_10000_210_eeee.root", "210_eeee_cut.root")` and `cut("output_tree_10000_3210_eeee.root", "3210_eeee_cut.root")` to specify the output ROOT file names and perform the cumulative cuts after loading the C++ script in ROOT, as described before.

The Python script `VBFNLO_VBS_QCD_eeee_comparison.py` is then used to iteratively plot the histograms of all the relevant variables across all cut stages: from no cuts, cuts on all non-leading leptons, cuts on pseudorapidity of all leptons, cuts on transverse momentum of the leading lepton, and cuts on the mass of Z boson (1) and Z boson (2) and the transverse momentum of the subleading jet.

In the Python script `VBFNLO_VBS_QCD_eeee_comparison.py`, an external plotting package `oneplot.py` (a Python-interface handler for plotting functions native in ROOT) is imported and used frequently, along with the two folders for the default ATLAS plotting style: `modules` and `atlas-style` in the same directory. Note that before running the Python script `VBFNLO_VBS_QCD_eeee_comparison.py`, one should run the Bash script `source setup.sh` to load all necessary files and package paths.

Also note that what we are interested is the cumulative cuts up to the satisfaction of the pre-selection criteria. But for cross-validation and helpfulness to debug, plots of histograms in different cuts stages are generated.

The following plots are the histograms of all the variables:


VBFNLO
√s=13 TeV
QCD Δη_jj
QCD Δη_jj
events / 0.25 normalized
Δη_jj
VBFNLO
√s=13 TeV
VBS ZZ_jj
QCD ZZ_jj
events / 40 GeV normalized
M_jj [GeV]
VBFNLO
√s=13 TeV
Z1 VBS ZZjj
Z1 QCD ZZjj
events / 20 GeV normalized
Pt_Z [GeV]
VBFNLO
√s=13 TeV
Z2 VBS ZZjj
Z2 QCD ZZjj
events / 20 GeV normalized
Pt_Z [GeV]
VBFNLO
√s=13 TeV
VBS ZZ_jj
QCD ZZ_jj
events / 10 GeV normalized
M_ZZ [GeV]
VBFNLO
√s=13 TeV
l1 VBS ZZjj
l1 QCD ZZjj
events / 0.35 normalized
η
VBFNLO
√s=13 TeV
l2 VBS ZZjj
l2 QCD ZZjj
events / 0.35 normalized
η
VBFNLO
√s=13 TeV
l3 VBS ZZjj
l3 QCD ZZjj
events / 0.35 normalized
η
VBFNLO
√s=13 TeV
l4 VBS ZZjj
l4 QCD ZZjj
events / 0.35 normalized
η
VBFNLO
√s=13 TeV
Z1 VBS ZZjj
Z1 QCD ZZjj
events / 0.35 normalized
η_Z
VBFNLO
√s=13 TeV
Z2 VBS ZZjj
Z2 QCD ZZjj
events / 0.35 normalized
η_Z
VBFNLO
√s=13 TeV
Z1 VBS ZZjj
Z1 QCD ZZjj
events / 0.35 normalized
φ_Z
VBFNLO
√s=13 TeV
Z2 VBS ZZjj
Z2 QCD ZZjj
events / 0.35 normalized
φ_Z
VBFNLO
√s=13 TeV
Z1 VBS ZZjj
Z1 QCD ZZjj
events / 4 GeV normalized
M_Z [GeV]
VBFNLO
√s=13 TeV
Z2 VBS ZZjj
Z2 QCD ZZjj
events / 4 GeV normalized
M_Z [GeV]

From the distribution of the variables, the following variables are finally selected for the training of the classifiers in later steps.

- mass of 2 jets ($m_{jj}$)
- change of pseudorapidity ($\Delta\eta$)
- mass of 2 Z bosons ($m_{ZZ}$)
- transverse momentum of 2 bosons ($pt_{Z1}$, $pt_{Z2}$)
- transverse momentum of 4 leptons ($pt_{l1}$, $pt_{l2}$, $pt_{l3}$, $pt_{l4}$)

Note that transverse momentum of 4 leptons are not plotted below, meaning that this variable is much less significant in the sense of discriminating power in the signal and backgrounds events than the other variables.

### 3.2 Toolkit for Multivariate Analysis (TMVA)

The Toolkit for Multivariate Analysis (TMVA) provides a ROOT-integrated machine learning environment for the processing and parallel evaluation of multivariate classification and regression techniques. TMVA is specifically designed to the needs of high-energy physics (HEP) applications, but should not be restricted to these. [5]

The package includes:

- Rectangular cut optimisation
- Projective likelihood estimation (PDE approach)
- Multidimensional probability density estimation (PDE - range-search approach)
- Multidimensional k-nearest neighbour classifier
- Linear discriminant analysis (H-Matrix and Fisher discriminants)
- Function discriminant analysis (FDA)
- Artificial neural networks (three different implementations)
- Boosted/Bagged decision trees
- Predictive learning via rule ensembles (RuleFit)
- Support Vector Machine (SVM)

In this research, Boosted/Bagged decision trees are chosen as the classifiers to be trained.

### 3.3 Boosted Decision Tree (BDT)

A decision tree is a binary tree structured classifier similar to the one sketched in Figure 1 below. Repeated left/right (yes/no) decisions are taken on one single variable at a time until a stop criterion is fulfilled. The phase space is split this way into many regions that are eventually classified as signal or background, depending on the majority of training events that end up in the final leaf node. The boosting of a decision tree extends this concept from one tree to several trees which form a forest. The trees are derived from the same training ensemble by reweighting events, and are finally combined into a single classifier which is given by a (weighted) average of the individual decision trees. Boosting stabilizes the response of the decision trees with respect to fluctuations in the training sample and is able to considerably enhance the performance w.r.t. a single tree. [6]

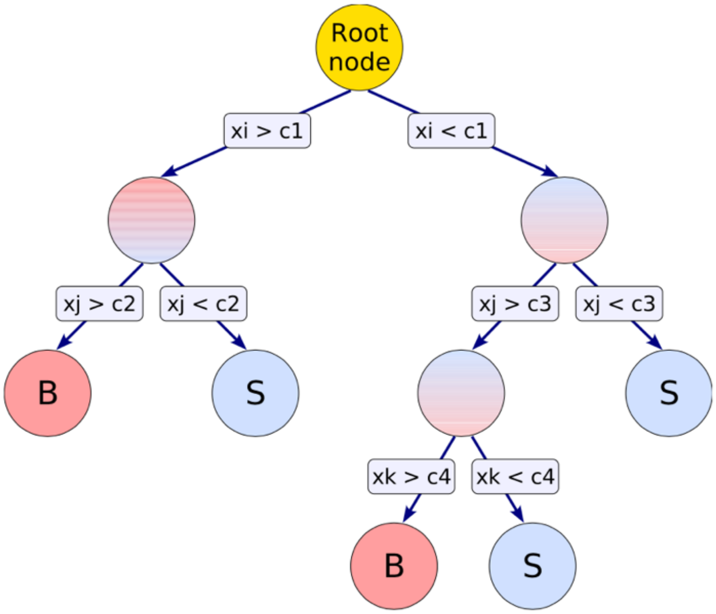


Figure 1: Schematic view of a decision tree. Starting from the root node, a sequence of binary splits using the discriminating variables $x_i$ is applied to the data. Each split uses the variable that at this node gives the best separation between signal and background when being cut on. The same variable may thus be used at several nodes, while others might not be used at all. The leaf nodes at the bottom end of the tree are labeled "S" for signal and "B" for background depending on the majority of events that end up in the respective nodes. [7]

## 3.4 Training a Boosted Decision Tree

### 3.4.1 Adaptive Boost (AdaBoost)

The most popular boosting algorithm is the so-called AdaBoost (adaptive boost). This is also the major algorithm used for this research. In a classification problem, events that were misclassified during the training of a decision tree are given a higher event weight in the training of the following tree. Starting with the original event weights when training the first decision tree, the subsequent tree is trained using a modified event sample where the weights of previously misclassified events are multiplied by a common boost weight. The boost weight is derived from the misclassification rate, err, of the previous tree,

$$\alpha = \frac{1 - err}{err} \tag{1}$$

The weights of the entire event sample are then renormalised such that the sum of weights remains constant. We define the result of an individual classifier as $h(x)$, with ($x$ being the tuple of input variables) encoded for signal and background as $h(x) = +1$ and 1, respectively. The boosted event classification $y_{Boost}(x)$ is then given by

$$y_{Boost}(x) = \frac{1}{N_{collection}} \sum_{i}^{N_{collection}} \ln(\alpha_i h_i(x)) \tag{2}$$

where the sum is over all classifiers in the collection. Small (large) values for $y_{Boost}(x)$ indicate a background-like (signal-like) event. Equation (2) represents the standard boosting algorithm.

AdaBoost performs best on weak classifiers, meaning small individual decision trees with a tree depth of often as small 2 or 3, that have very little discrimination power by themselves. Given such small trees, they are much less prone to overtraining compared to simple decision trees and as an ensemble outperform them typically by far. The performance is often further enhanced by forcing a "slow learing" and allowing a larger number of boost steps instead. The learning rate of the AdaBoost algorithm is controled by a parameter $\beta$ giving as an exponent to the boost weight $\alpha \rightarrow \alpha^{\beta}$ , which can be modified using the configuration option string of the MVA method to be boosted. [8]

### 3.4.2 Gradient Boost

The idea of function estimation through boosting can be understood by considering a simple additive expansion approach. The function $F(x)$ under consideration is assumed to be a weighted sum of parametrised base functions $f(x; a_m)$, so-called "weak learners". From a technical point of view any TMVA classifier could act as a weak learner in this approach, but decision trees benefit most from boosting and are currently the only classifier that implements GradientBoost (a generalisation may be included in future releases). Thus each base function in this expansion corresponds to a decision tree

$$F(x; P) = \sum_{m=0}^{M} \beta_m f(x; a_m); \quad P \in \{\beta_m a_m\}_0^M \tag{3}$$

The boosting procedure is now employed to adjust the parameters $P$ such that the deviation between the model response $F(x)$ and the true value $y$ obtained from the training sample is minimised. The deviation is measured by the so-called loss-function $L(F, y)$ a popular choice being squared error loss $L(F, y) = (F(x)y)^2$. It can be shown that the loss function fully determines the boosting procedure.

The GradientBoost algorithm attempts to cure this weakness by allowing for other, potentially more robust, loss functions without giving up on the good out-of-the-box performance of AdaBoost. The current TMVA implementation of GradientBoost uses the binomial log-likelihood loss

$$L(F, y) = \ln(1 + e^{2F(x)y}) \tag{4}$$

for classification. As the boosting algorithm corresponding to this loss function cannot be obtained in a straightforward manner, one has to resort to a steepest-descent approach to do the minimisation. This is done by calculating the current gradient of the loss function and then growing a regression tree whose leaf values are adjusted to match the mean value of the gradient in each region defined by the tree structure. Iterating this procedure yields the desired set of decision trees which minimises the loss function. Note that GradientBoost can be adapted to any loss function as long as the calculation of the gradient is feasible.

Just like AdaBoost, GradientBoost works best on weak classifiers, meaning small individual decision trees with a depth of often just 2 to 4. Given such small trees, they are much less prone to overtraining compared to simple decision trees. Its robustness can be enhanced by reducing the learning rate of the algorithm through the `Shrinkage` parameter, which controls the weight of the individual trees. A small shrinkage

(0.1–0.3) demands more trees to be grown but can significantly improve the accuracy of the prediction in difficult settings.

In certain settings GradientBoost may also benefit from the introduction of a bagging-like resampling procedure using random subsamples of the training events for growing the trees. This is called *stochastic gradient boosting* and can be enabled by selecting the `UseBaggedGrad` option. The sample fraction used in each iteration can be controlled through the parameter `BaggingSampleFraction`, where typically the best results are obtained for values between 0.5 and 0.8. [9]

### 3.4.3 Bagging

The term Bagging denotes a resampling technique where a classifier is repeatedly trained using resampled training events such that the combined classifier represents an average of the individual classifiers. A priori, bagging does not aim at enhancing a weak classifier in the way adaptive or gradient boosting does, and is thus not a “boosting” algorithm in a strict sense. Instead it effectively smears over statistical representations of the training data and is hence suited to stabilise the response of a classifier. In this context it is often accompanied also by a significant performance increase compared to the individual classifier.

Resampling includes the possibility of replacement, which means that the same event is allowed to be (randomly) picked several times from the parent sample. This is equivalent to regarding the training sample as being a representation of the probability density distribution of the parent sample: indeed, if one draws an event out of the parent sample, it is more likely to draw an event from a region of phase-space that has a high probability density, as the original data sample will have more events in that region. If a selected event is kept in the original sample (that is when the same event can be selected several times), the parent sample remains unchanged so that the randomly extracted samples will have the same parent distribution, albeit statistically fluctuated. Training several classifiers with different resampled training data, and combining them into a collection, results in an averaged classifier that, just as for boosting, is more stable with respect to statistical fluctuations in the training sample.

Technically, resampling is implemented by applying random Poisson weights to each event of the parent sample. [10]

## 3.5 Boosted Decision Trees Configuration

The following is the configuration of the BDT classifier:

- Transformations: Decorrelation; Principle Component Analysis (PCA); Gaussian, Decorrelation transformation:
- Training set: 60% of signal data and the same amount of background data
- Testing set: the rest of signal and background data
- Number of trees: 500/1000
- transverse momentum of 4 leptons
- Boost types:
  - Adaptive
  - Adaptive + Decorrelation
  - Adaptive + Fisher discriminant
  - Gradient
  - Bagging

- For Adaptive Boost: AdaBoostBeta = 0.5, use Bagged Boost, Bagged Sample Fraction = 0.5
- Separation type: Gini Index: p · (1 - p) where p is purity defined as $\frac{S}{S+B}$
- Number of cuts = 20/Optimal cuts
- Maximum depth = 5
- MinNodeSize = 2.5

Note that for Gini index G, "all signal node" → G = 0, "all background node" → G = 0, and the "perfect mixing" → G is maximum. We are trying to identify the variable (and the best cut) that maximizes the difference in Gini index between parent node and the left/right daughter leaves: $\Delta = G_{\text{Parent}} - G_{\text{Left}} - G_{\text{Right}}$

The C++ script `dtop.cpp` contains all the details of the BDT training in TMVA's framework. This script is associated with its header file `dtop.h` and its driver file `main.cpp`, in which 3 different training options are provided. A Makefile named `Makefile` is prepared for compiling. One can run the command `make clean` and then `make` and compile and run the executable `run.exe` by the command `./run.exe` to train and evaluate the classifier. Again, it is recommended to record the screen output as a log file for future reference.

With 3 different training options, there will be 3 output ROOT files:
`jj_m_zz_m_z1_pt_z2_pt_delta_eta_jj_dtop.root`, `jj_m_zz_m_z1_pt_z2_pt_delta_eta_jj_dtop.root`, and `jj_m_zz_m_z_pt_delta_eta_jj_l_pt_optimal_dtop.root`, each of which contains the relevant information of the classifier, such as the signal and background events distributions after classification, the correlation matrices of the variables, the receiving operator characteristics (ROC) curves, the variable distributions for the signal and background events etc.

In addition, I also adapt the scripts to a single Jupyter notebook `BDT+Classification.ipynb`, which has a much cleaner interface and could be run on CERN's SWAN (Service for Web based ANalysis) Service online. And for more flexible and portable machine learning and analysis environment, one can also try the rootpy package to convert ROOT data structures to NumPy arrays and then use data framework packages like Pandas or SFrame in Scikit-learn or Tensorflow frameworks.

# 4 Analysis

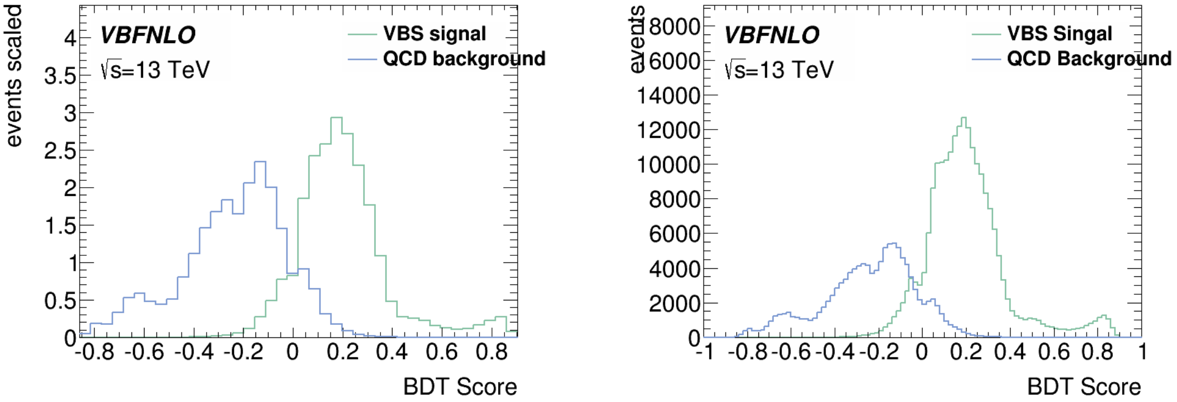
## 4.1 Events Distribution

Once we have the output of BDT classifiers, we first check the events distributions of VBS and QCD across the BDT scores. The Python script `plot_signal_background_distribution_from_TMVA.py` will read the signal and background events separately from the ROOT file output by TMVA and plot them on the same canvas. The plot obtained directly from TMVA will automatically scale the event numbers. To further verify the events distributions, one can also count directly from the data after pre-selection: `210_eeee_fid_preselected.root` and `3210_eeee_fid_preselected.root`. This verification step can be realized by running the C++ script `vbfnlo_driver.cxx`, which is the driver file of `vbfnlo.cxx` along with its header file `vbfnlo.h`. The exact command to compile and run the C++ script in ROOT is `root -l -b -q vbfnlo_driver.cxx`, where `-l` means loading without graphic interface, `-b` means loading in batch, and `-q` means running in quiet mode.

Note that to generate the event distribution histograms stored in ROOT files for both the signal and the background, one will need to configure the path in `vbfnlo.h` twice, one for the signal and one for the background. In addition, the driver file `vbfnlo_driver.cxx` contains different BDT options. It is important for the user to run one option at a time. Running 2 or more options at the time will mess up the memory of the stored variables in the previous run.

For the 3 different BDT options, one should be able to obtain the following ROOT files in the end: `BDT_Signal_histogram_1.root`, `BDT_Background_histogram_1.root`, `BDT_Signal_histogram_2.root`, `BDT_Background_histogram_2.root`, and `BDT_Signal_histogram_3.root`, `BDT_Background_histogram_3.root`.

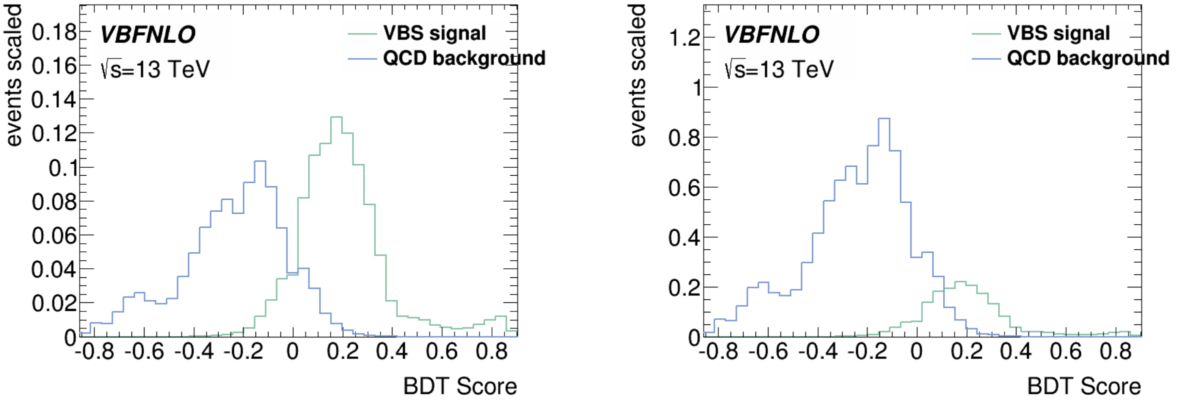
The Python scripts `plot_signal_background_distribution_from_BDT.py` can then be used plot the events distributions of the signal and background from direct counting.



(a) Events Distribution from TMVA

(b) Events Distribution from Direct Counting

Figure 2: Optimal BDT Training Results



(a) Events Distribution from TMVA
Normalized to Unity Area

(b) Events Distribution from Direct Counting
Normalize to 40fb-1

Figure 3: Optimal BDT Training Results with Normalization

## 4.2 BDT Score Scan

The normalized yield can be calculated in the following way:

- Yield(no selection) = Luminosity(40) * Cross-section(Signal/Background)
- Yield(after pre-selection) = Luminosity(40) * Cross-section(Signal/Background) * Eff(pre-selection)

where the cross-section for the signal events is around 0.18 and the cross-section for the background events is around 1.85. The numerical values are shown in the table below:

Table 4: Normalized Yield

| | N(S) | N(B) |
|---|---|---|
| NO SELECTION | 7.2 | 74 |
| AFTER PRE-SELECTION | 1.7 | 8.43 |

The following table records a scan of the signal region cut along the BDT scores from 0 to 0.52, each with increment at 0.04. The statistical uncertainty is calculated in the following way:

- Fractional Statistical Uncertainty = $\sqrt{\text{number of events}}$ / (number of events)
- Statistical Uncertainty = normalized events * Fractional Statistical Uncertainty

Table 5: BDT VS. Variable Cuts at m(jj) > 500 GeV and $|\Delta\eta(jj)| > 3$

| Classification | Signal region cut | N(S) | N(B) |
|---|---|---|---|
| by cuts on variables | m(jj) > 500 GeV and $|\Delta\eta(jj)| > 3$ cut | 1.352+/- 0.0033 | 0.901+/- 0.01217 |
| by BDT 500 trees | 0.00 | 1.539 +/- 0.0035 | 0.938 +/- 0.00827 |
| by BDT 500 trees | 0.04 | 1.460 +/- 0.0034 | 0.648 +/- 0.00688 |
| by BDT 500 trees | 0.08 | 1.309 +/- 0.0033 | 0.354 +/- 0.00508 |
| by BDT 500 trees | 0.12 | 1.145 +/- 0.0030 | 0.190 +/- 0.00372 |
| by BDT 500 trees | 0.16 | 0.964 +/- 0.0028 | 0.096 +/- 0.00264 |
| by BDT 500 trees | 0.20 | 0.763 +/- 0.0025 | 0.042 +/- 0.00176 |
| by BDT 500 trees | 0.24 | 0.576 +/- 0.0022 | 0.018 +/- 0.00114 |
| by BDT 500 trees | 0.28 | 0.419 +/- 0.0018 | 0.0073 +/- 0.00073 |
| by BDT 500 trees | 0.32 | 0.292 +/- 0.0015 | 0.0034 +/- 0.00050 |
| by BDT 500 trees | 0.36 | 0.204 +/- 0.0013 | 0.0015 +/- 0.00033 |
| by BDT 500 trees | 0.40 | 0.159 +/- 0.0011 | 0.00088 +/- 0.00025 |
| by BDT 500 trees | 0.44 | 0.103 +/- 0.0010 | 0.00073 +/- 0.00023 |
| by BDT 500 trees | 0.48 | 0.120 +/- 0.0009 | 0.00051 +/- 0.00019 |
| by BDT 500 trees | 0.52 | 0.103 +/- 0.0008 | 0.00029 +/- 0.00015 |

In this research, the BDT score cut at 0.08 is chosen as the classification threshold, events above which are classified as the signal and events below which are classifier as the background.

## 4.3 Variable Relation

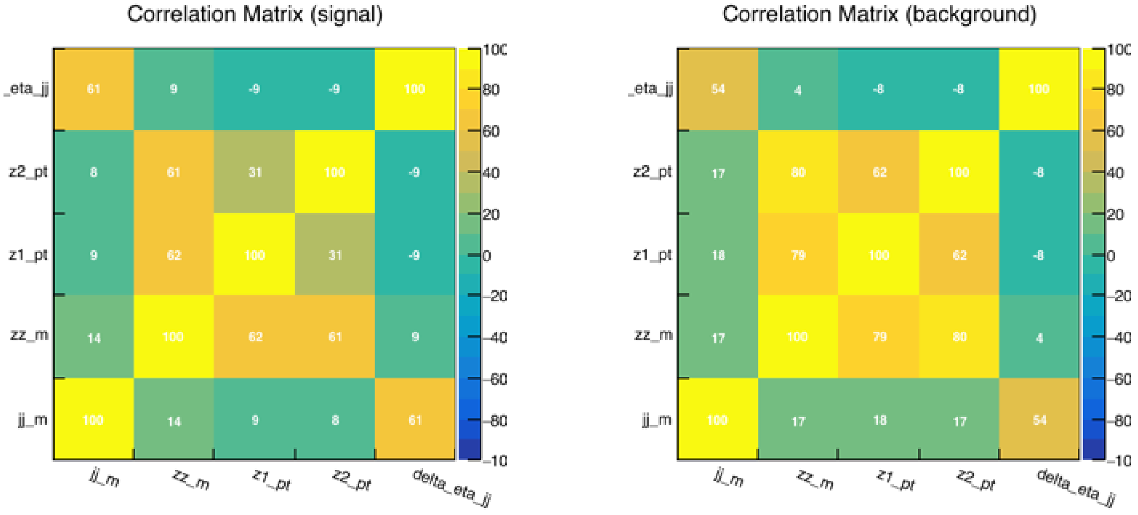
From the training log of TMVA, one can refer the ranking of variables listed in the table below:

Table 6: Ranking of Variables

| Rank | Variable | Variable Importance |
|---|---|---|
| 1 | change of pseudorapidity of 2 jets | 3.689e-01 |
| 2 | mass of 2 jets | 1.875e-01 |
| 3 | mass of 2 Z bosons | 7.758e-02 |
| 4 | transverse momentum of Z boson 1 | 6.298e-02 |
| 5 | transverse momentum of lepton 2 | 6.292e-02 |
| 6 | transverse momentum of Z boson 2 | 6.136e-02 |
| 7 | transverse momentum of lepton 1 | 6.100e-02 |
| 8 | transverse momentum of lepton 4 | 6.053e-02 |
| 9 | transverse momentum of lepton 3 | 5.731e-02 |

One can also view the correlation matrices of the variables using the commands `root` to start ROOT and then `TBrower b` open an interactive user interface. One can then quickly navigate to and click `CorrelationMatrixS` or `CorrelationMatrixB` in the TMVA output ROOT file, which is named `jj_m_zz_m_z_pt_delta_eta_jj_l_pt_optimal_dtop.root` by default when running the C++ script prepared before. By command `CorrelationMatrixS->Draw("COLZ")` in ROOT, the correlation matrix is drawn for the signal, and then by command `CorrelationMatrixS->Draw("TEXTsame")` in ROOT, the correlation in numerical values are overlaid on top of each color tiles. The same commands can be applied to the background by changing `CorrelationMatrixS` to `CorrelationMatrixB`.

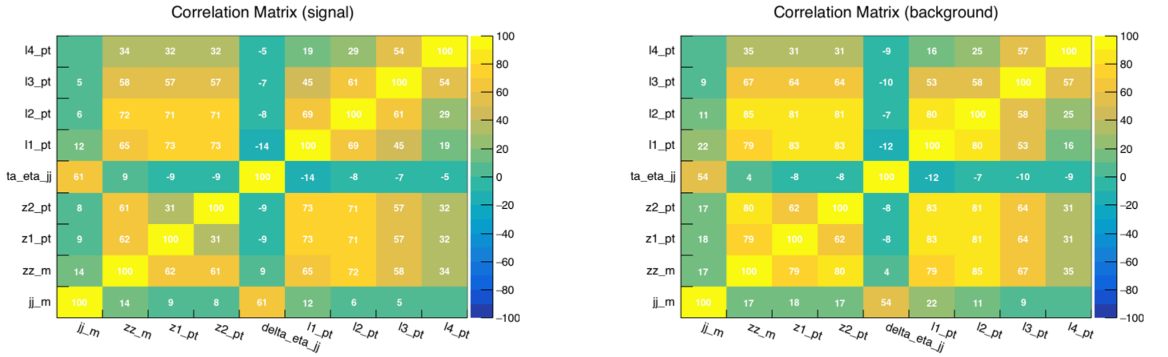
Some variable correlation matrices are plotted as the following:



(a) Variable Correlation Matrix for the Signal

(b) Variable Correlation Matrix for the Background

Figure 4: Variable Correlation Matrices for the BDT trained without leptons



(a) Variable Correlation Matrix for the Signal

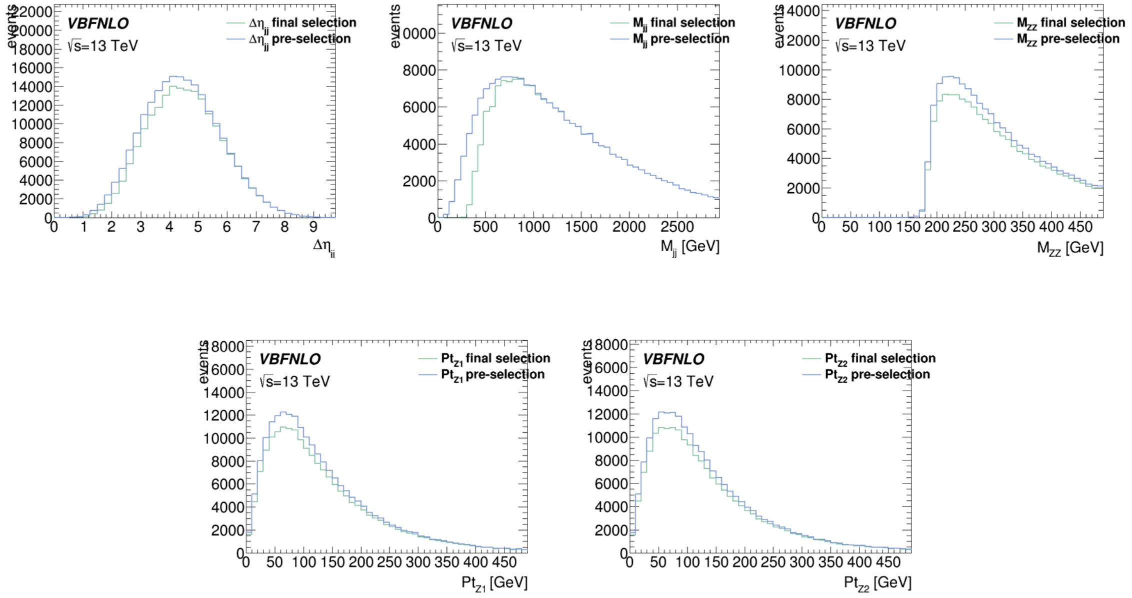
(b) Variable Correlation Matrix for the Background

Figure 5: Variable Correlation Matrices for the BDT trained including leptons

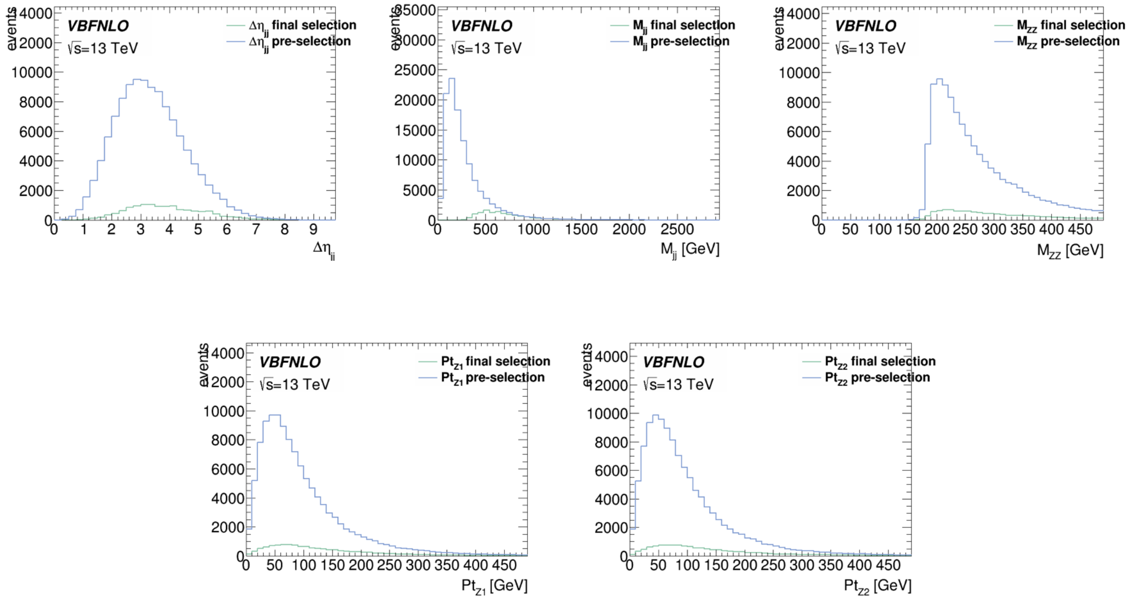
The yellowish colors in the matrices above represent higher correlation between the variables while the greenish color represent lower correlation.

One can also plot the variable distribution after BDT to visualize how are variables selected during the training process.

The following 5 plots show the variable selection for the signal for BDT score cut at 0.08:



The following 5 plots show the variable selection for the background for BDT score cut at 0.08:



# 5 Conclusion

The Python script `plot_signal_background_distribution_from_TMVA.py` could also be used to plot the ROC curve from ROOT file from the TMVA output. From the ROC curve of optimal BDT training result below, it is clear that BDT is slightly better than the direct variable cut of mjj $>$ 500 GeV and $|\Delta\eta_{jj}| > 3$:

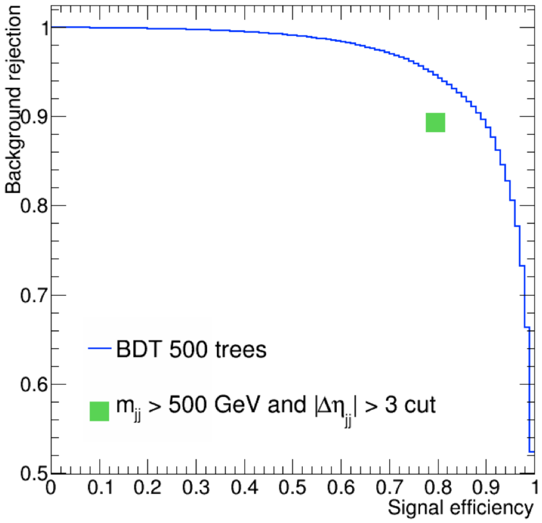
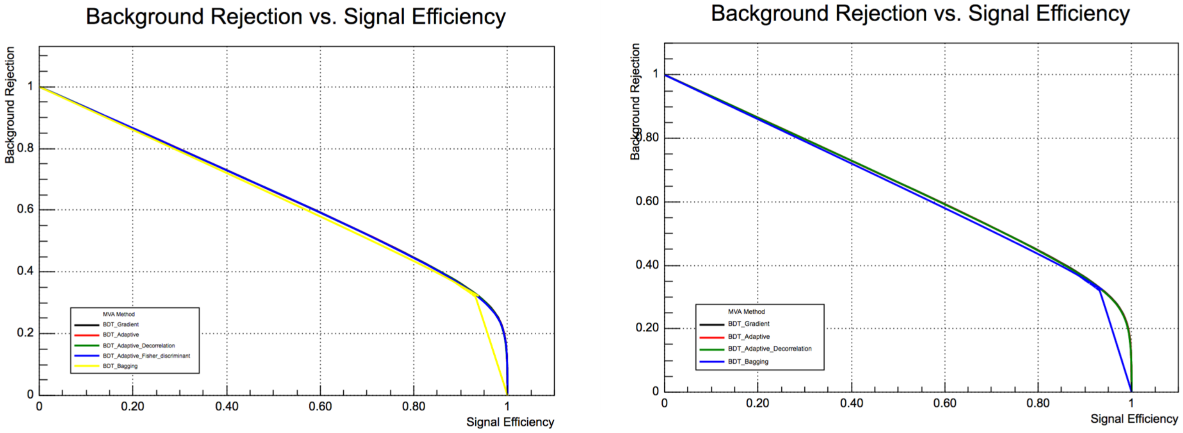


Figure 6: ROC for Optimal BDT compared with Direct Variable Cut



(a) BDT 500 Trees

(b) BDT 1000 Trees

Figure 7: Other ROC Curves with different training configurations obtained on SWAN

To quantify the advantages of using BDT over direct variable cut, we can tell that at signal efficiency of around 0.80, BDT is about 7% better than the direct cut and at background rejection of around 0.89, BDT is about 12.5% better than the direct cut.

Note that the final yields obtained in Table 5: BDT VS. Variable Cuts at m(jj) > 500 GeV and $|\Delta\eta(jj)|$ > 3 need to multiply 4 to account for all decay channels: eeee, eemumu, mumuee, and mumumumu. For 40 fb-1, we obtain N(S) ≈ 5.2, N(B) ≈ 1.1 at BDT score cut > 0.08. Considering 50% reconstruction efficiency, N(S) *approx* 2.6, N(B) ≈ 0.5. There is also potential to observe electroweak production of ZZ + 2 jets at 13 TeV (combined with other channels and using more luminosity in 2017).

Again, all the analysis in this research is based on the truth events generated from the theory-level simulation program. This research summary has focused on identifying the signal events and the backgrounds events we are interested in through a trained classifier. Once the classification result is proved to be reliable, we can then use this well-trained classifier to test on the detector-level simulation or extend and adapt the classifier for other data and processes.

# Acknowledgement

I would like to thank the Lounsbery Foundation and the University of Michigan for fully funded my research at the European Organization for Nuclear Research (CERN) in Geneva, Switzerland. I would also like to thank Prof. Bin Zhou and Dr. Yusheng Wu for supervising and guiding my search, as well as Prof. Jean Krisch and Dr. Steven Goldfarb for helping with the logistics here at CERN. In addition, I would also like to thank the graduate students who have greatly helped and inspired me along with way during my research, especially Weibin Zhang from USTC, and Thorben Quast from RWTH Aachen. Weibin has helped me tremendously on getting started with the VBFNLO Monte Carlo simulation software as well as the coding in Python and C++ with the ROOT framework. While Thorben has greatly inspired me on implementing codes with other frameworks like Tensorflow and batch jobs management tool Luigi luigi. This research could not have been conducted with all your help and support.

# References

[1] Classification using Boosted Decision Trees, *Data analysis*, December 13, 2016, https://indico.scc.kit.edu/indico/event/31/session/5/contribution/35/material/slides/0.pdf

[2] VBFNLO - A parton level Monte Carlo for processes with electroweak bosons, *Project Description*, December 13, 2016, https://www.itp.kit.edu/vbfnlo/wiki/doku.php?id=Overview

[3] J. Baglio, J. Bellm, G. Bozzi, et al., *2.2 Compilation and installation*, April 17, 2014, Vbfnlo: A parton level Monte Carlo for processes with electroweak bosons – Manual for Version 2.7.0 p. 6

[4] The LoopTools Visitor Center, December 13, 2016, http://www.feynarts.de/looptools/

[5] TMVA Toolkit for Multivariate Data Analysis with ROOT, *TMVA Executive Summary*, December 13, 2016, http://tmva.sourceforge.net/

[6] A. Hoecker, P. Speckmayer, J. Stelzer, J. Therhaag, E. von Toerne, H. Voss, *7.1 Adaptive Boost (AdaBoost)*, October 4, 2013, TMVA 4 Toolkit for Multivariate Data Analysis with ROOT Users Guide. p. 108

[7] A. Hoecker, P. Speckmayer, J. Stelzer, J. Therhaag, E. von Toerne, H. Voss, *7.1 Adaptive Boost (AdaBoost)*, October 4, 2013, TMVA 4 Toolkit for Multivariate Data Analysis with ROOT Users Guide. p. 109

[8] A. Hoecker, P. Speckmayer, J. Stelzer, J. Therhaag, E. von Toerne, H. Voss, *7.1 Adaptive Boost (AdaBoost)*, October 4, 2013, TMVA 4 Toolkit for Multivariate Data Analysis with ROOT Users Guide. p. 55

[9] A. Hoecker, P. Speckmayer, J. Stelzer, J. Therhaag, E. von Toerne, H. Voss, *7.3 Gradient Boost*, October 4, 2013, TMVA 4 Toolkit for Multivariate Data Analysis with ROOT Users Guide. p. 57

[10] A. Hoecker, P. Speckmayer, J. Stelzer, J. Therhaag, E. von Toerne, H. Voss, *7.3 Bagging*, October 4, 2013, TMVA 4 Toolkit for Multivariate Data Analysis with ROOT Users Guide. p. 58