Universiteit Utrecht

Faculteit Bètawetenschappen

Photon Conversion Classification by Boosting Decision Trees

Bachelor Thesis

Tijmen Schaapherder

Natuur- en Sterrenkunde

Supervisors:

Mike Sas
Institute for Subatomic Physics

Prof. Dr. Thomas Peitzmann
Institute for Subatomic Physics

June 13, 2018

Abstract

The ALICE detector located at CERN studies subatomic particles produced in heavy-ion collisions. These collisions generate enormous amounts of particles including photons. The data gathered from these collisions is contaminated with background. This research focuses on generating a viable and efficient machine-learning algorithm for selecting photon conversions and discriminating them from background. The method used is that of the boosted decision tree (BDT). A Monte Carlo simulation is used to train and test the BDT and afterwards to test the performance of the BDT. The Monte Carlo simulation consists of data taken from a simulated collision of 40%-60% centrality and a center of mass energy of $\sqrt{s_{\mathrm{NN}}} = 2.76$ TeV (Tera electron Volts).

Contents

1 Introduction

Located west of Geneva (Switzerland) on the border between Switzerland and France stands the CERN main headquarters. This is home to the world's largest and most powerful particle accelerator, namely the Large Hadron Collider (LHC). Working as of late 2008, it consists of a 27-kilometre ring of numerous superconducting magnets. Inside the accelerator the LHC can produce 2 high-energy particle beams, which travel in opposite directions close to the speed of light and are made to collide. These collisions produce many subatomic particles. The beams are directed around the accelerator by the superconducting magnets. The particle beams are made to collide at four locations around the accelerator. At these locations there each stands one giant particle detector, at which the results of a collision are measured[1].

1.1 A Large Ion Collider Experiment

A Large Ion Collider Experiment (ALICE) is one of the main detectors at the LHC. At ALICE the accelerators produce head-on collisions between heavy ions, such as lead nuclei. Its objective is to study the physics of strongly interacting matter at extreme high energy densities. The study of the properties of the quark-gluon plasma, a state of matter only existing at extreme high temperatures and energy densities, is one of its main occupations. ALICE consists of multiple subdetectors that use various techniques to detect and measure the particles produced in a heavy ion collision, which will be described below[2][3].

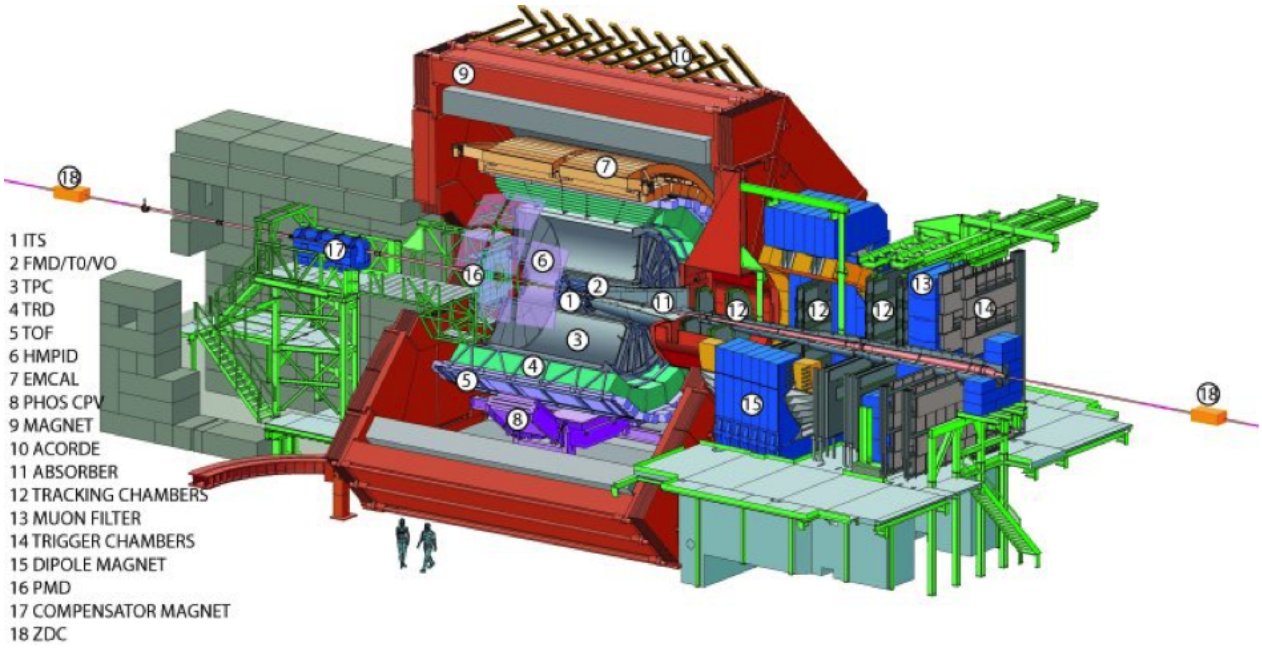


Figure 1: The ALICE detector with the locations of the subdetectors labeled.

1.1.1 The ALICE subdetectors

Charged particles in ALICE, like electrons and positrons, are primarily measured in the inner barrel. By measuring the electron and positron particles, photon conversions can be reconstructed. The charged particle detectors that are utilized in this research, are the Inner Tracking System, the Time Projection Chamber and the Time of Flight detector.
The **Inner Tracking System** (ITS) is made out of six layers of silicon detectors. The main purpose of the ITS is the tracking of particles in the vicinity of the initial collision and the reconstruction of the primary and secondary vertices. These vertices are the locations of the initial collision and particle decay, respectively. Additionally the ITS recovers particles missed by the external barrel detector[4].
The **Time Projection Chamber** (TPC) is the primary tracking detector of the central barrel. It consists of a gas filled cylinder divided in two drift regions. Charged particles pass through the TPC, which ionise the gas inside, consequently freeing electrons that flow under influence of an applied electrical field to the end plates of the cylinder. Here the electrons are registered and used to reconstruct the path of charged particles through the TPC[5].
The **Time of Flight** (TOF) detector measures the time of flight of charged particles and is composed of a large array of multigap resistive plate chambers (MRPCs) which forms a cylindrical surface in the inner barrel. Together with the momentum and track length measured by other detectors, the time of flight can be used to calculate the mass of a particle[6].

After a collision in ALICE, high-energy particles are produced. The corresponding energy of these particles is measured by the calorimeters in ALICE. The calorimeters used in this research will be discussed below.
The **Photon Spectrometer** (PHOS) is a high-resolution photon detector made out of lead tungstate crystals. Its main goal is to measure photons emitted by the quark-gluon plasma. If a high-energy photon collides with these crystals, the lead tungstate will start to glow, and this glow can be measured. This way the PHOS is able to detect photons originating at angles around 90° of the center-of-mass system[7].
The **Electromagnetic Calorimeter** (EMCal) is a calorimeter primarily consisting of scintillator layers and lead foils. Particles passing through the EMCal and interacting with the lead produce an electromagnetic shower. The photons originating from this electromagnetic shower are measured. This way, the EMCal is able to determine the energy of the interacting particles[8].

At ALICE the data is gathered in very brief time intervals, all data measured at such an time interval is ranked under the same event. The trigger detectors regulate the measurements at these intervals and are able to determine the event time very precisely.
The **V0 detector** serves as trigger detector and the collision centrality detector. At times of collisions particles do not always collide perfectly in the center they do not always collide perfectly in the center of one another. The centrality is a measurement of how much these particles 'overlap' at the time of collision. The V0 detector provides the number of collisions inside a specific centrality range[9].
The **T0 detector** serves as trigger detector and provides the start signal for some of the other detectors, such as the TOF detector. The T0 detector also gives a rough estimate of the centrality[10].

1.2 The standard model

The standard model is at present the best and most generally accepted theory of elementary particles. Its intent is to explain all phenomena regarding particle physics, excluding those resulting from gravity. It describes three of the four fundamental forces, namely the strong force, weak force and electromagnetic force. Elementary or fundamental particles are characterized as point-like particles, not having an internal structure or excited states. The primary attributes that describe elementary particles are the mass, spin and electric charge of the concerned particle. Particles with half integer spin and integer spin are called fermions and bosons, respectively. Fermions number a total of 24, consisting of 6 leptons, 6 quarks and their respective antiparticles. They are classified in 3 different generations, with the first generation being the only stable one[11].

Quarks are the 'building blocks' of matter. Quarks are divided in six different flavours and they carry colour charge. Multiple quarks can form hadrons, particles exclusively made up of quarks that are hold together by the strong force. Hadrons are subdivided into mesons and baryons. Mesons are particles made up of a quark and antiquark, while baryons are made up of three quarks. Quarks interact via the strong force and are kept together inside a hadron due to a process called confinement. A group of quarks forming a hadron combine in such a way that the resulting particle always will be colour neutral[12].

The leptons consist of the electron, muon and tau and their corresponding neutrinos. The electron, tau and muon interact via the electromagnetic force. The neutrinos interact solely via the weak force, due to the fact that they are electrically neutral.

According to the standard model bosons are divided into the gauge bosons and the Higgs boson. These gauge bosons or 'force carriers' mediate the fundamental forces. The photon and gluon are both massless particles that mediate the electromagnetic force and strong force, respectively. The W^+, W^- and Z bosons are the force carriers of the weak force and due to their charge the W+ and W- boson also couples to the electromagnetic force. The Higgs boson has 0-spin and is affiliated with the Higgs field, the latter explains why some fundamental particles have mass. The Higgs boson has been long since predicted by the standard model and has just recently been discovered. The discovery and confirmed existence of the Higgs boson is one of the major achievements of the LHC and CERN.

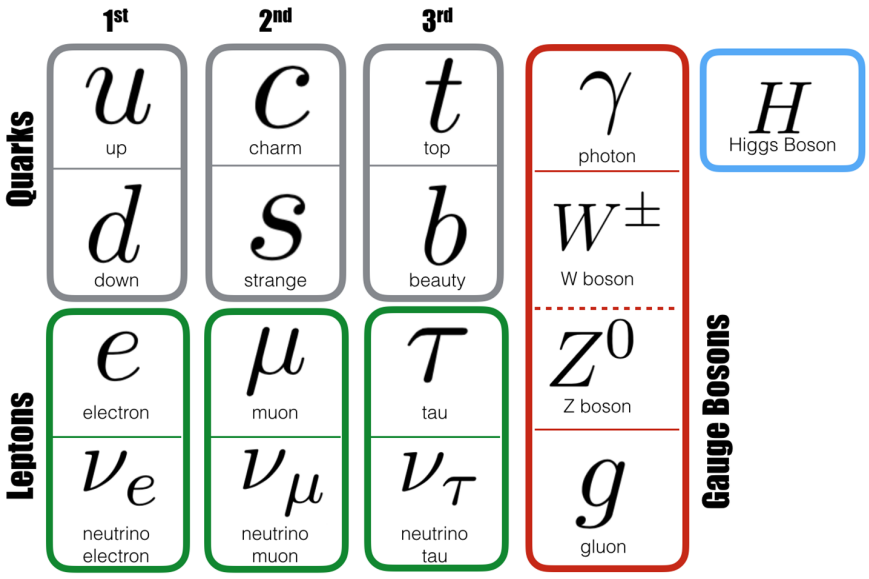


Figure 2: A schematic view of the standard model.

1.2.1 Photons

The photon is one of the gauge bosons and the force carrier for the electromagnetic force. It is a quantum of light, a bundle of light or electromagnetic wave consists of photons. Photons have a spin of 1. These massless particles carry the energy of the wave while moving with the speed of light. The energy of a photon can be calculated via:

$$E = \hbar\omega, \tag{1}$$

where $\hbar$ is the reduced Planck constant and ω the angular frequency. The total energy of an electromagnetic wave is equal to the total energy of the photons that the wave consists of. Due to having no electrical charge, the photon has no antiparticle, or depending on your point of view, is its own antiparticle. Despite of having no mass a photon does have a momentum, which can be calculated via:

$$p = \hbar\mathbf{k}, \tag{2}$$

where $\mathbf{k}$ is the wave vector and $\hbar$ the reduced Planck constant.

In the field of subatomic physics the role of a photon mainly lays as force carrier of the electromagnetic force, or as rest product from high-energy particle collisions. When fast moving particles at, for example, the LHC are made to collide, the result yields a lot of different particles flying of in all directions. Some of these particles will be photons, other particles may later decay to photons and even other, due to for example bremsstrahlung, may lose energy in the form of photons. The photons may convert, using detector material, to an electron-positron pair. Additionally an electron-positron pair is able to convert to two photons, this process is called electron-positron annihilation and is displayed as:

$$e^+ + e^- \rightarrow \gamma + \gamma. \tag{3}$$

In short, due to the massive energy accumulated at these collisions enormous amounts of photons will be produced. The produced photons at ALICE will be detected and identified by the PHOS detector, EMCal detector and via photons that convert in the central barrel.

1.3 Research objectives

The primary objectives of this research:

- Study the properties and procedure of the machine learning boosted decision tree method.
- Acquire a basic understanding of the ALICE detector and its subdetectors.
- Train and test the boosted decision tree using Monte Carlo data.

The main research goal:

- Use the boosted decision tree method to classify photons and evaluate the performance of the boosted decision tree.

2 Method

At the LHC or other particle accelerators, the measurements of the particle collisions will result in enormous amounts of data. Although in most cases there will be certain amounts of background contaminating the data. The aim of this research is to find a way to minimalize the amount of background contaminating the data. For example, let the data from Figure 3 be from a MC simulation. The signal and background are classified and when a cut is made at 0.0 for input variable *var4* a lot of background can be removed, while most of the signal will remain. Nevertheless it is clear that every arbitrary cut made at a variable from Figure 3 results in a substantial loss of signal while keeping a significant amount of background. To improve upon this method the concept of machine learning is used. The procedure is as follows; with the help of ROOT, a framework for high-energy data processing, a boosted decision tree (BDT) will be introduced. The boosting of a decision tree indicates the 'learning' of a tree from the missclassified candidates of the previously constructed decision trees. The ROOT-integrated Toolkit for Multivariate Analysis (TMVA) provides the necessary functions to construct a boosted decision tree. Through training and testing the BDT with data produced at a Monte Carlo (MC) simulation, the BDT will build a classifier. The classifier is constructed by training and testing over all the variables, which when all taken together will ultimately produce a so-called optimal cut value on the BDT output. This value is calculated on the basis of an as high as possible signal versus background ratio and an as high as possible signal efficiency. When applied on real data this cut value can be used to discard most of the background while keeping nearly all the signal. A Monte Carlo simulation is a simulated high-energy particle collision of which the background and signal particles are known. Consequently a MC simulation will train the BDT in separating the background and signal.

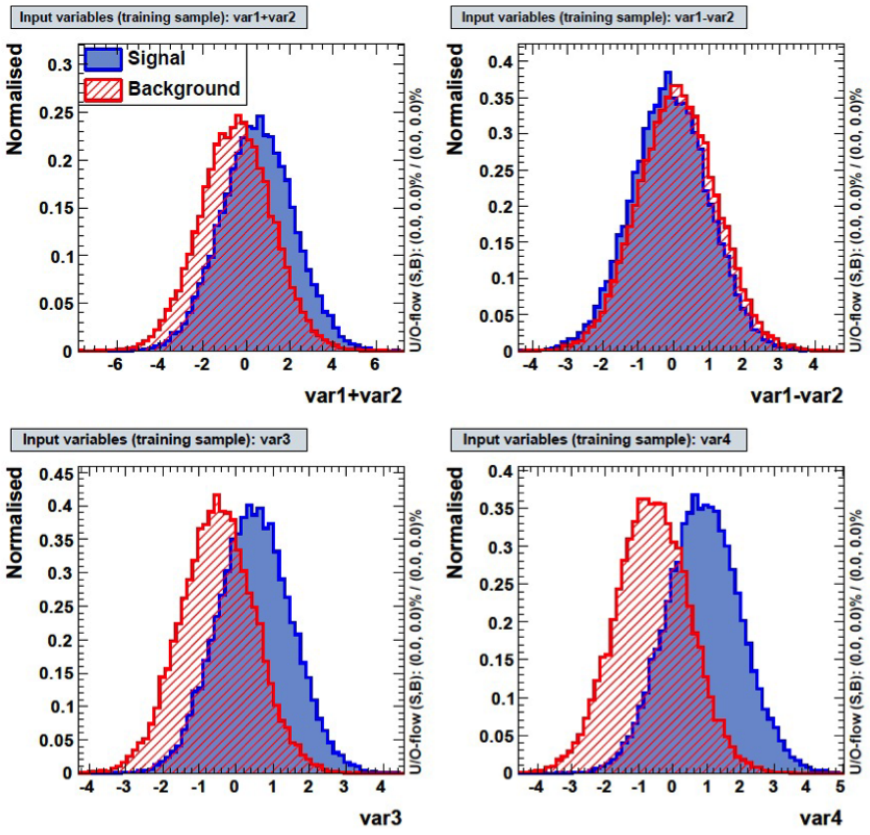


Figure 3: Four arbitrary distributions which indicate the difference between signal and background. For variable *var4* its is obvious to see a cut made at 0 results in the best ratio of signal versus background.

2.1 Decision tree

The fundamental concept behind a decision tree is the classification of data. In the case of this research the classification of background and signal particles. A decision tree consist of multiple nodes, starting off at the root node a cut will be made on a relevant variable that divide the node into two subsamples. This procedure continues hierarchical until the decision tree reaches the so-called end nodes of the tree. The aim is that these end nodes consist almost entirely of either background or signal. The end nodes will be labelled, depending on the majority of either signal or background in the nodes, accordingly as either signal or background. The nodes between the root node and end nodes are called the chance nodes. If a end node is classified as signal it will probably still contain an amount of background and vice versa. Consequently some data will be lost in the process. A way to compare the nodes with respect to the ratio between signal and background is to calculate the *signal purity* of a node via:

$$p = \frac{S}{S+B}, \tag{4}$$

where S and B stand for the amount of signal and background candidates, respectively. If the signal and background candidates are known, which is the case in a MC simulation, the goal is to achieve an as high as possible separation between background and signal. Figure 4 shows an schematic view of a decision tree with arbitrary cut variables.

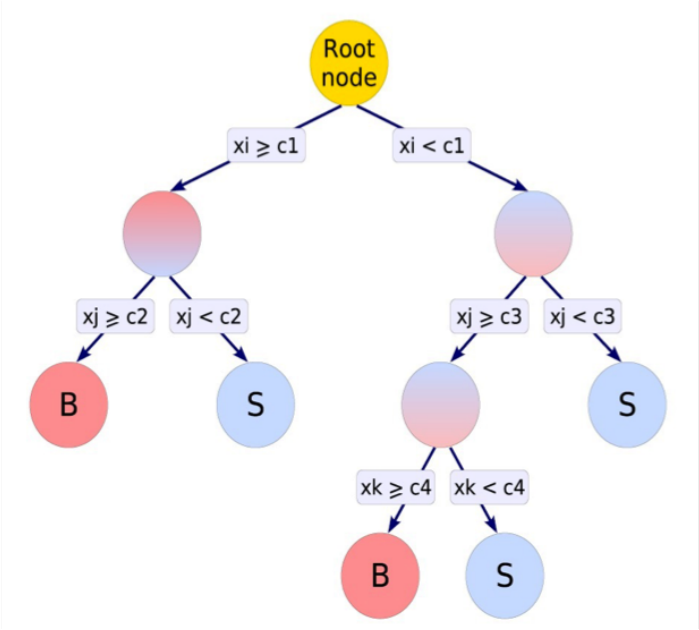


Figure 4: A schematic image of a decision tree. The root node will subdivided into two chance nodes depending on the cut of *c1*. Relative to the root node, one node will have a higher background to signal ratio, while the other has a higher signal to background ratio. Eventually the procedure of subdividing the node will accumulate in the end nodes. These are labelled S or B, corresponding to a signal node or background node respectively.

The purity is used to calculate the G_{ini} of a node, which is defined as:

$$G_{ini} = p(1-p). \tag{5}$$

The G_{ini} will help us to calculate the *separation gain*:

$$SeparationGain = N_p G_{ini}(Parent) - N_{d1} G_{ini}(Daughter1) - N_{d2} G_{ini}(Daughter2), \tag{6}$$

where N_p, N_{d1} and N_{d2} are the total number of candidates at the parent, first daughter and second daughter node, respectively. The highest possible separation for a node is achieved by maximizing the separation gain, in other words the focus is to minimalize the G_{ini} of the daughter nodes. This is done through the process of cutting the variables at different values. The value that returns the best separation will be used as cut value. This method is repeated for the other cut variables and the leftover nodes. This way, a decision tree is created that can be used to identify unclassified candidates [13].

2.2 Boosting

To produce a stable and efficient machine learning particle classifier the single decision tree method can be improved. To counteract this problem, the boosting of decision trees is used. When a decision tree is trained, some candidates will be misclassified. Boosting implies that the next tree being trained learns from the previous tree in a way that depends on the misclassification rate of that tree. The misclassified candidates are given a higher weight by multiplying the initial weight of the new tree with a so-called *boost weight*:

$$\alpha = \left(\frac{1-err}{err} \right)^{\beta}, \tag{7}$$

where the misclassification rate err is defined as:

$$err = \frac{\text{Total weight of misclassifications}}{\text{Total weight of entire tree}}, \tag{8}$$

and β a parameter for controlling the learning rate of the algorithm. To make sure the sum of the weights remains consistent, the weights of the entire event sample are renormalized. The process of classifying weights depending on the misclassification of the previous tree is repeated until a preferential number of trees is attained, which now form a so-called forest. This way the forest dictates the output value, eliminating the susceptibility of statistical fluctuations.

An individual classifier will yield an result concerning the classification of signal or background. This result called $h(x)$ produces the value of +1 for a signal event and -1 for a background event. Where x is a so-called tuple containing input variables. Consequently the $y_{boost}(x)$ value for the BDT output is generated:

$$y_{boost}(x) = \frac{1}{N_{trees}} \sum_{i=1}^{N_{trees}} \ln(\alpha_i) * h_i(x). \tag{9}$$

The summation over all classifiers in the collection produces the value of $y_{boost}(x)$, which, for small values, corresponds to background-like events, while values that are relatively high indicates signal-like events. The $y_{boost}(x)$ value for each event is used to build the BDT output classifier. A distribution is build with signal and background significantly separated. A appropriate made cut on the classifier can now discriminate background and signal to a much higher degree than with only one decision tree.

Due to the procedure described above a BDT is generally susceptible to overtraining. Overtraining arises when a BDT contains too few degrees of freedom, as a result of an algorithm containing too many modified parameters corresponding with too few data points. It leads to a seemingly increase of performance of a training sample, but a decrease of performance when measured over a test sample and results in a decreasing performance of the classification. To detect overtraining the results of the training and testing phase can be compared, when they don't correspond well overtraining is probably the cause. A different method is to conduct a Kolmogorov-Smirnov (KS) test, which compares the test and training distributions. An extremely low KS value indicates overtraining, while a value close to 1 corresponds to negligible overtraining. Overtraining can, for example, be countered by reducing the number of trees in the forest or adjusting the β parameter. An example of a overtrained BDT is displayed in Figure 5[14].

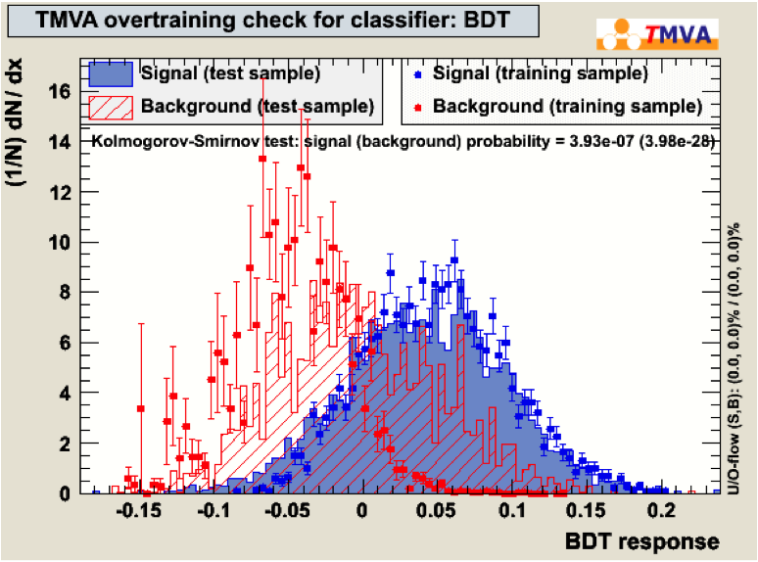


Figure 5: The result of a clearly overtrained BDT. The test sample does not correspond well with the points of the training sample. The KS test values are extremely low which also indicates a overtrained BDT.

2.3 Toolkit for Multivariate Analysis

The Toolkit for Multivariate Analysis (TMVA) is a ROOT-integrated package for the analysis of multivariate classification and regression problems. ROOT is a at CERN born framework written in the C++ programming language. It provides various kinds of functions to process and classify large amounts of data. The TMVA package provides the necessary components for machine learning classification techniques, like the BDT. It also provides the necessary functions to examine the effectiveness of the BDT, like the KS test for overtraining. The TMVA has the BDT separated in to two stages, a training and testing phase and a classification phase.

The first stage consists of setting up a script in C++ to train and test the Monte Carlo simulated data. To generate a working script, first the variables over which the BDT will train need to be determined and classified as either signal or background. The corresponding variables from the MC simulation should be added to the script. Hereafter the training can commence. The data will be split in a training sample and test sample later to be used to train and test the data. The training starts with instantiating a Factory object. So it generates the BDT forest depending on the training settings. These settings determine, for example, the number of trees in the forest and will be discussed later. Ultimately, after the trained variables are tested by the test sample, this results in several different generated control plots, allowing the user to check out the results of the training and testing stage.

In the last stage, the weights calculated at the training and testing phase will be used to classify data. At the end of the training and testing stage a weight file is generated. This file consists of weights for each decision tree, which needs to be loaded in a new C++ script. A data set of which the signal and background particles are unknown need to be loaded in the new C++ script. The variables corresponding to those that were used in the training stage need to be coded. To start the classification a reader class object is instantiated and will start classifying the unknown data on the basis of the loaded-in weight file. The result yields a number of histograms, depending on the settings at the start of the classification stage. Using the cut value generated at the training stage a selection of candidates is made, which ultimately yields in data of which most background is eliminated and most signal remains. A schematic view of both the BDT phases is shown in appendix A.

3 Training and Testing

Regarding the training and testing phase of the photon, an MC simulation is used with a collision centrality between 40% and 60%. The center of mass energy at the collisions per nucleon pair is 2.76 TeV. The data consist of 1507519 photon candidates that will be used to train and test the BDT.

3.1 Variables

At times of collision numerous photons will be produced of which some will convert to a electron-positron pair. Therefore the variables used for training should not only be the measurements of photons, but should include the electron and positron measurements as well. The selected variables and their conversions are shown in Table 1. This doesn't include every variable available from the MC simulation, as not every variable is particularly useful for signal and background selection. For example the Cartesian coordinates (x, y and z) are generally independent regarding a particle being background or signal. Consequently these are left out, while variables like the momentum of a particle are much more useful and thus included in the training variable selection.

photonQt	Q_t of a photon
photonPsiPair	ψ_{pair} of photon
photonCosPoint	$\cos(\theta_p)$ of photon
photonInvMass	Invariant mass of photon (M_γ)
dEdxPositronITS	$\frac{dE}{dx}$ of positron in ITS
dEdxPositronTPC	$\frac{dE}{dx}$ of positron in TPC
dEdxElectronITS	$\frac{dE}{dx}$ of electron in ITS
dEdxElectronTPC	$\frac{dE}{dx}$ of electron in TPC
clsTPCPositron	Number of clusters of positron in TPC
clsTPCElectron	Number of clusters of electron in TPC
ptPositron	P_t of positron
ptElectron	P_t of electron
photonR	R of photon

Table 1: BDT training input variables and their conversions.

After selecting the training variables, the spectator variables could be selected. These are variables that are not especially useful regarding the training stage, but could have some value at the end of the training and testing stage. When declared spectator a variable is not used in the training, but will appear in the final output and can consequently be used to evaluate the BDT training output. The variables selected as spectators and their conversions are shown in Table 2. Figure 6 displays some of the converted variables used in the BDT.

photonPt	P_t of photon
photonPhi	ϕ of photon
photonEta	η of photon

Table 2: The variables selected as spectator and their conversions.

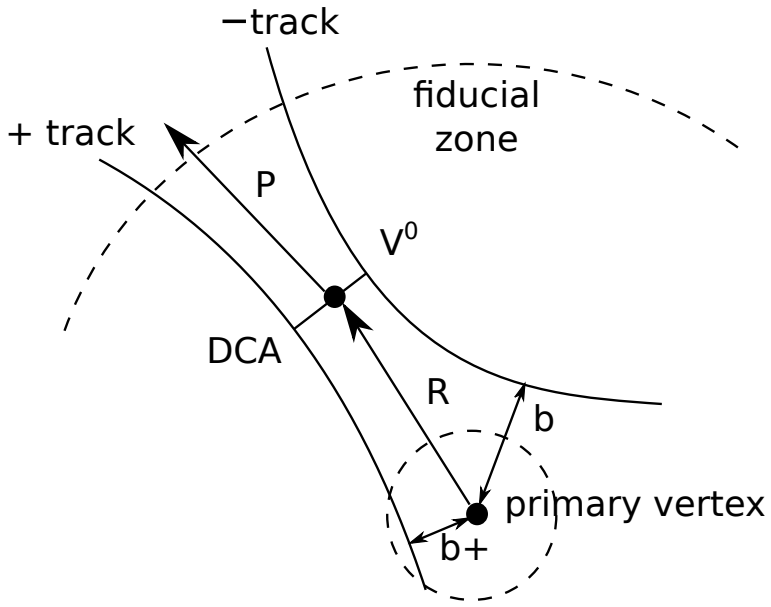


Figure 6: A schematic view of the conversions of some of the input variables.

The variables selected for training need to be coded to separate signal and background candidates. After this additional cuts can be made to each candidate. Some variables consist of parts with a high particle concentration, while other parts are extremely low concentrated. If these low concentrated areas exist on the boundary values of a variable, these parts could be cut away. This could lead to a better performance of the BDT and decreased computation time. The cuts can be selected independently for signal and background candidates. Table 3 displays the variables that are cut and their cut values.

Signal Candidate Cuts	**Background Candidate Cuts**
photonQt < 0.05	photonQt < 0.25
photonInvMass < 0.1	photonInvMass < 1
dEdxPositronITS < 1050	dEdxPositronITS < 1050
ptPositron < 2.5	ptPositron < 2.5
ptElectron < 2.5	ptElectron < 5
dEdxElectronITS < 1050	dEdxElectronITS < 1050
$40 <$ dEdxPositronTPC < 100	$40 <$ dEdxPositronTPC < 120
$40 <$ dEdxElectronTPC < 100	$40 <$ dEdxElectronTPC < 120
photonCosPoint > 0.8	
$-0.5 <$ photonPsiPair < 0.5	

Table 3: The cuts made per variable.

At the start of the training, the BDT has to determine the ideal cut values for each input variables. This goes as follows; each variable distribution will be cut in multiple parts of equal size. The number of cuts made can be specified in the BDT settings. For each cut the separation gain (defined in section 2.1) is calculated. The cut with the highest separation gain is used in the BDT. The histograms of the input variables are displayed in appendix B.

3.2 BDT Configuration

The last steps to be completed before the training and testing stage can be initiated are to code the C++ script with proper settings. To train the BDT will take a sample of 200000 signal and 200000 background photon candidates. For the testing stage the BDT will use 140000 signal and 140000 background candidates. These candidates are all randomly selected from the Monte Carlo data. The BDT itself needs to be configured as well. To obtain the optimal configuration the BDT is tested with various settings. To evaluate the performance the significance is calculated. The significance is explained in section 3.3.4. Table 4 and 5 shows various settings tested with their corresponding significance.

Ntrees:	400	850	1200
MaxDepth:			
2	31.04	31.09	31.10
3	31.13	31.15	31.16
4	31.16	31.17	31.18
5	31.17	31.18	31.19

Table 4: Test settings and significance for BDT for $nCuts = 20$.

Ntrees:	400	850	1200
MaxDepth:			
2	31.10	31.13	31.14
3	31.15	31.18	31.18
4	31.17	31.20	31.20
5	31.19	31.20	31.21

Table 5: Test settings and significance for BDT for $nCuts = 40$.

Ntrees sets the number of trees generated in the BDT forest. *MaxDepth* sets the allowed depth of a tree. For illustration, the example in figure 4 has a *MaxDepth* of 3. The number of cuts made per input variable to determine its ideal cut value is specified by *nCuts*. Higher settings for *Ntrees*, *MaxDepth* and *nCuts* generally mean a better performing BDT at the cost of a greater risk of overtraining the BDT. Hence these settings need to be considered carefully as they also determine the computation time for a large part. The testing of the settings has displayed that higher settings do not show a significant improvement of the calculated significance. Hence the settings of the BDT of this research have accordingly been kept close to the default settings. The settings used in the BDT are shown in Table 6. The complete list of settings that can be used in the BDT are shown and explained in appendix C.

Option	**Value**
NTrees	850
MinNodeSize	2.5%
MaxDepth	3
BoostType	AdaBoost
AdaBoostBeta	0.5
BaggedSampleFraction	0.5
SeparationType	GiniIndex
nCuts	20

Table 6: Settings of the BDT.

3.3 Test and Training Results

In this section the results of the training and testing phase will be discussed. The various plots produced by the BDT are displayed and elaborated. The training and testing is done through data taken from a Monte Carlo simulation, which is stored in a *.root* file. At the end of the training and testing a weight file is created that will be used in the classification stage. The first decision tree generated by the BDT is displayed in Figure 7. In this example four out of the thirteen variables are used to construct this tree.

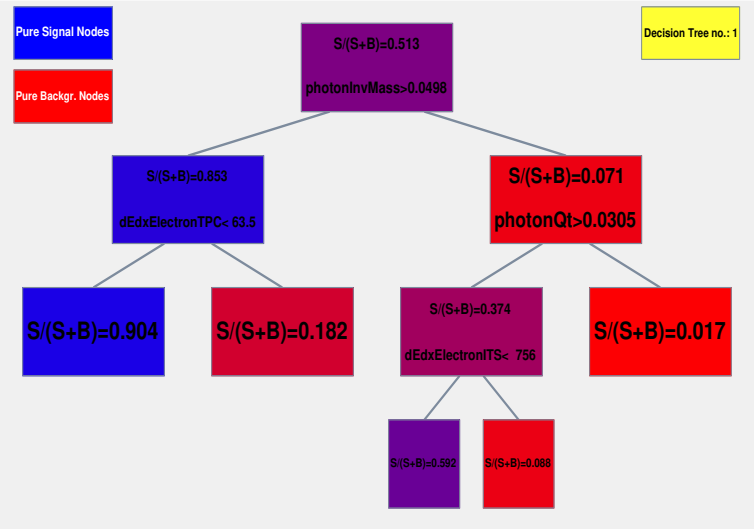


Figure 7: The first decision tree. The candidate cuts and purity of the nodes are shown.

3.3.1 Control Plots

The error fraction, also called the misclassification rate (defined in section 2.2), of each individual decision tree is displayed in Figure 8. The first number of trees has an error fraction relatively low with respect to the remaining trees. The error fraction of a tree never surpasses the 0.5, which is expected. The error fraction is used to calculate the boost weight for each tree. The boost weight is defined in section 2.2 and displayed per tree in Figure 9. Clearly dependent on the error fraction the first number of trees has a boost weight relatively much higher as the remaining trees. After approximately the first 100 trees the boost weight for each tree, just like the error fraction, remains somewhat stable.

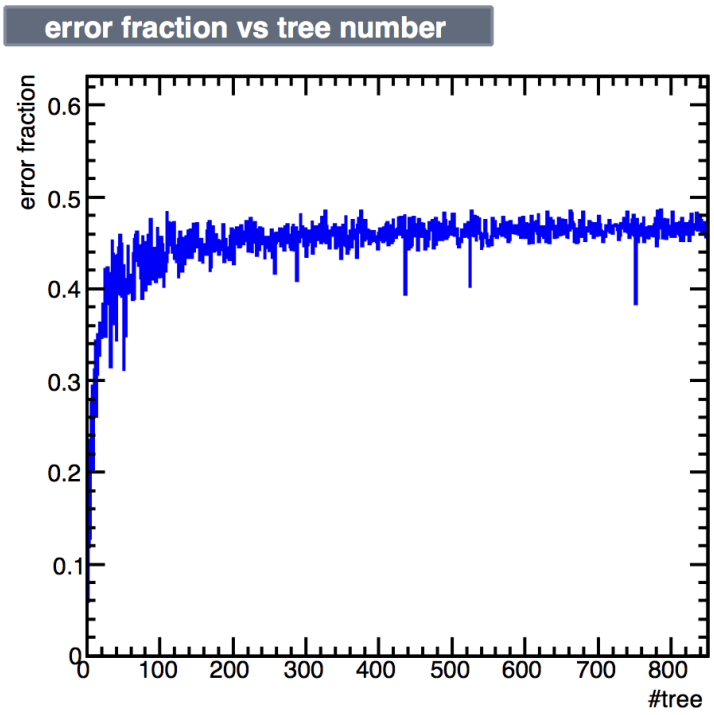


Figure 8: The error fraction per decision tree.

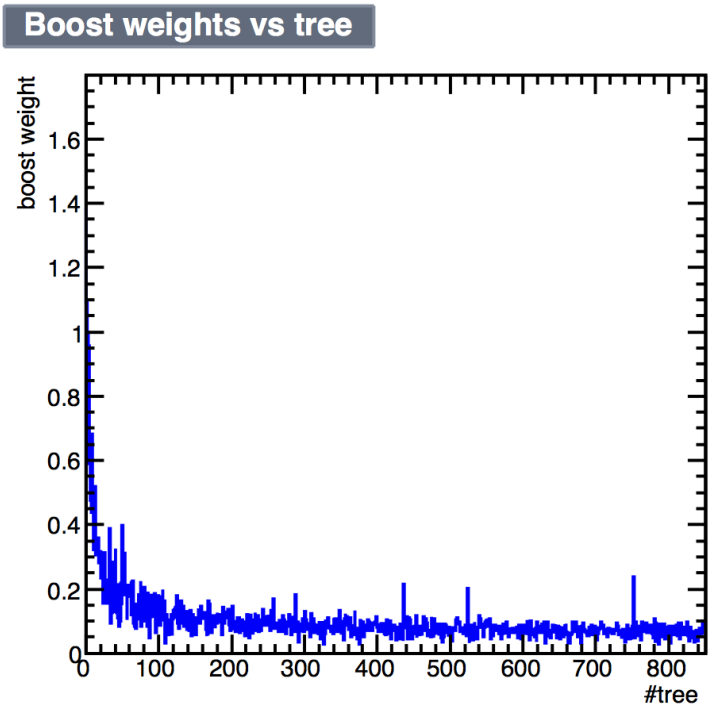
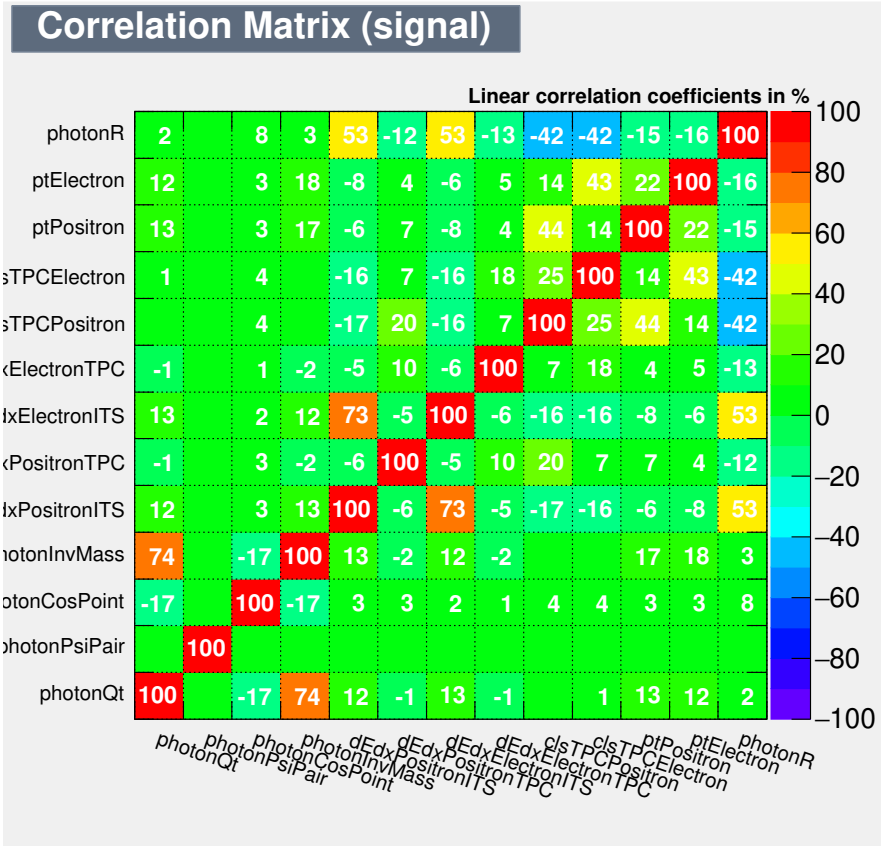


Figure 9: Boost weight per decision tree.

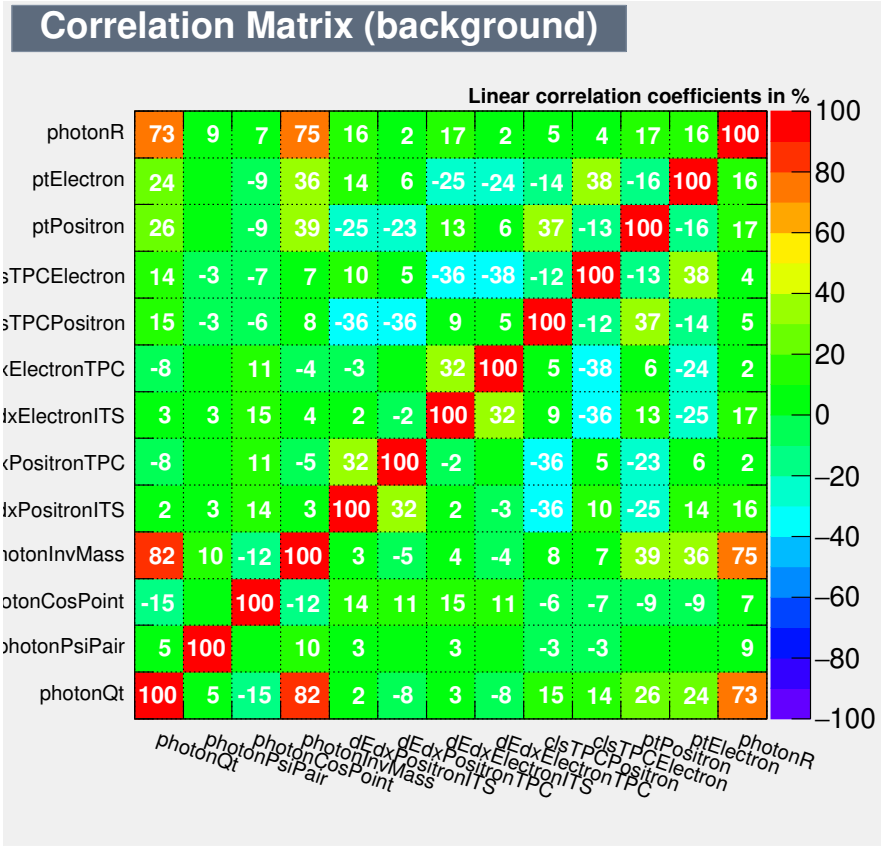
3.3.2 Correlation Matrix

The BDT calculates the linear correlation between each variable. The correlations of signal and background candidates are separated and shown in Figure 10a and 10b. The variables with values close to zero or displayed in green are not closely correlated with each other. The values close to a hundred or viewed in red are positively correlated, while values close to minus one hundred or viewed in blue are negatively correlated. If a number is not displayed in the figures, it indicates a correlation value very close to zero, consequently having practically no correlation with each other. Closely correlated variables are generally not very useful for constructing a BDT due to the candidate cuts being correlated as well.

As is clear from the figures most variables do not truly correlate with each other. The ones that do show signs of correlation are most notably the correlations of *photonInvMass-photonQt* in signal and background candidates, and *photonR* with both *photonInvMass* and *photonQt* in background candidates. Using the correlation between two random highly correlated variables, the two variables could be combined into one. This procedure is not performed in this research.



(a)



(b)

Figure 10: (a) Correlation matrix for signal candidates. (b) Correlation matrix for background candidates.

3.3.3 BDT Response

The BDT response, also called BDT output, is displayed in Figure 11. It shows the BDT response distribution of signal and background, where signal tends to be closer to 1 compared to background. The BDT response value is determined by formula 9 in section 2.2. This distribution is however not an indication of the ratio of signal and background candidates due to the exactly same number of signal and background candidates used in the training en testing phase. It is evident from the figure that the cut value should be close to zero. Additionally the BDT is checked for overtraining. The KS test value of 0.153 for signal and 0.065 for background are easily within the acceptable boundaries and thus indicate no significant overtraining of the BDT. A KS test value of 1 indicates that the two distributions correspond exactly, while a lower value points out less correspondence between the samples. The test sample and training sample correspond quite well as is clear from the figure.

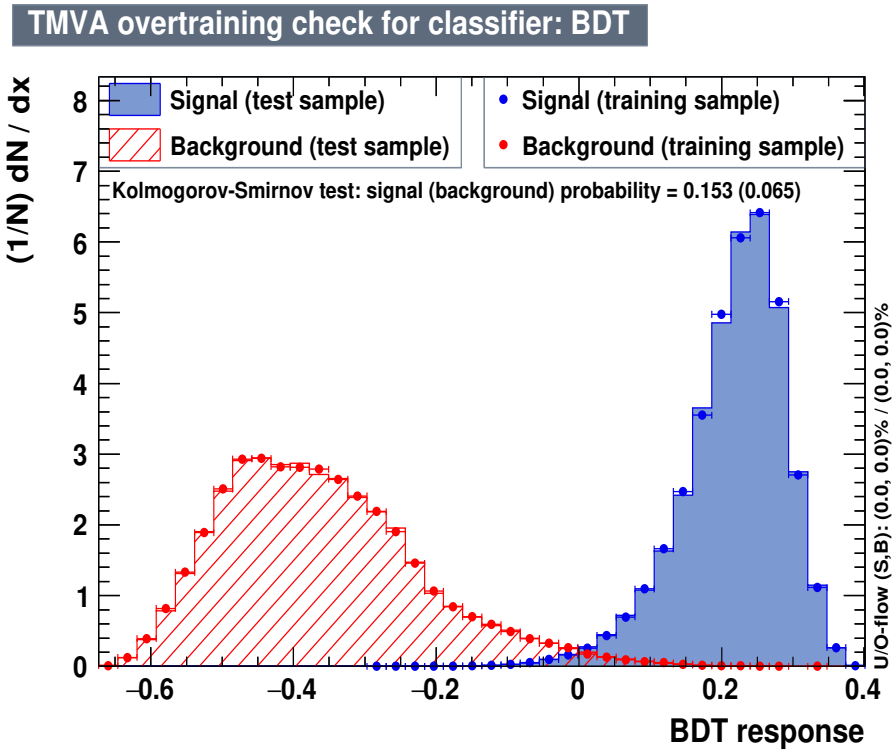


Figure 11: The MVA BDT response.

3.3.4 Optimal Cut Value

The data taken from the MC simulation is prone to statistical variations. Consequently the BDT is sensitive to these variations, which in turn could decrease the performance of the BDT. For acquiring the best possible performance for the BDT the highest possible significance is calculated. The significance is defined as:

$$\text{Significance} = \frac{S}{\sqrt{S+B}}. \tag{10}$$

As displayed in Figure 12, the highest significance is calculated to be 31.15 at an optimal cut value of 0.0. The significance is displayed by the green curve, which should be rather smooth and stable at the optimal cut value to counteract the statistical variations. The optimal cut value is used in the application phase to select the unidentified signal or background candidates.

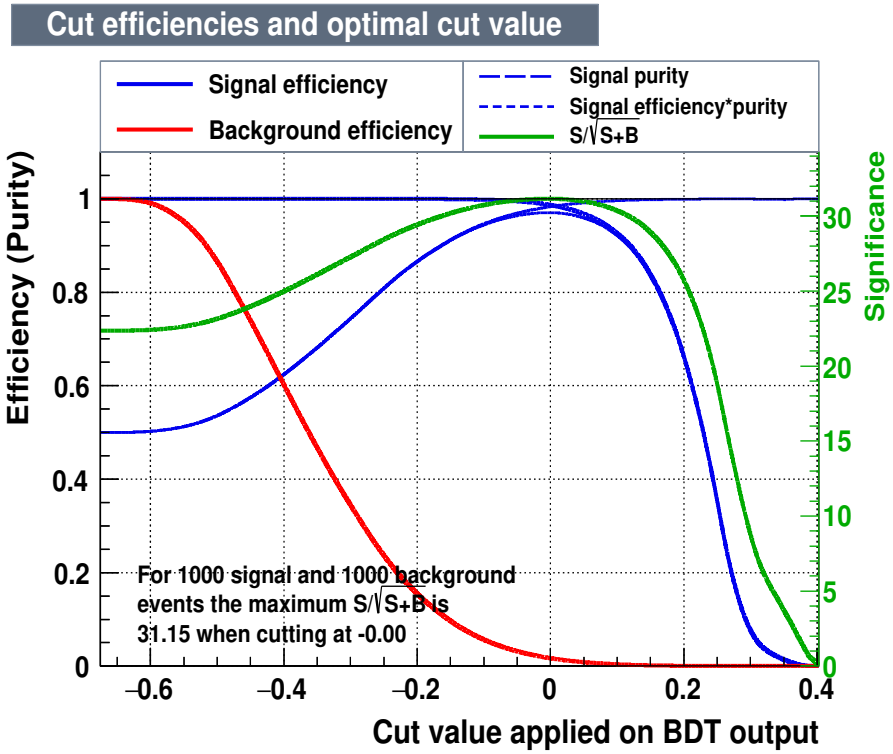


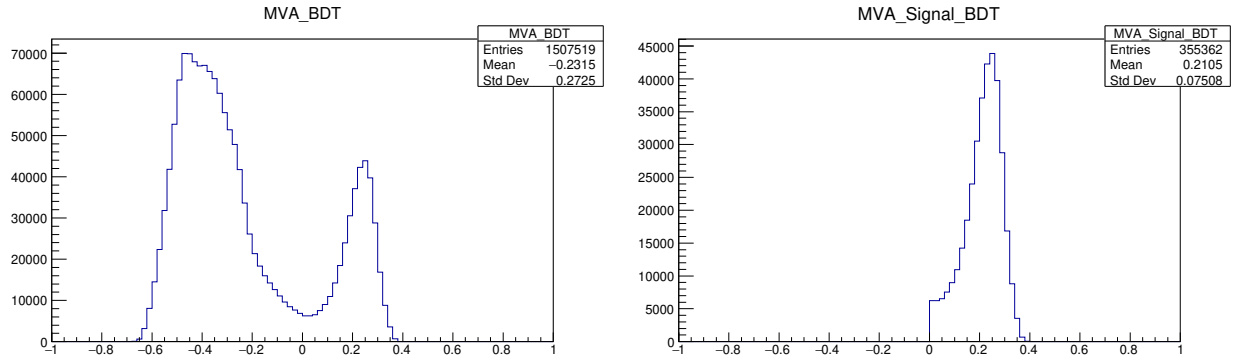
Figure 12: Efficiencies per applied cut value. The optimal cut value is given.

4 Classification

After the training and testing phase is completed the BDT should be ready to start classifying unidentified data. A new C++ script should be developed where the variables used in the training and testing stage are coupled to the variables from the unidentified data and the spectators defined in the training in testing stage should be included in the new script. The same candidate cuts used to train the BDT ought to be defined in the script. Additionally the *.xml* weight file created at the end of the training and testing phase should be loaded in the new C++ script. The weight file contains all the necessary information needed to successfully classify the unknown data. A reader class object, a TMVA function, is implemented to classify the data on the basis of the weight file. For classifying the data the same data from the Monte Carlo simulation is used. Hence the collision centrality is 40% - 60% for a center of mass energy of 2.76 TeV. A MC simulation, from which the background and signal particles are known, is used to easily calculate the performance of the BDT.

4.1 Results

The output of the BDT, where the MC simulation is treated as data, is displayed in Figure 13. Figure a shows the complete distribution of all classified photon candidates, so this includes the background particles. Figure b shows the BDT distribution after applying the optimal cut value derived from the BDT training. In a good performing BDT this should include most signal candidates of the original complete data, while most background is eliminated. The selected candidates should contain a to some degree negligible amount of data contaminating background. Hence the new data encloses a high amount of useful data.



(a) The application output without selection.

(b) The application output after selection.

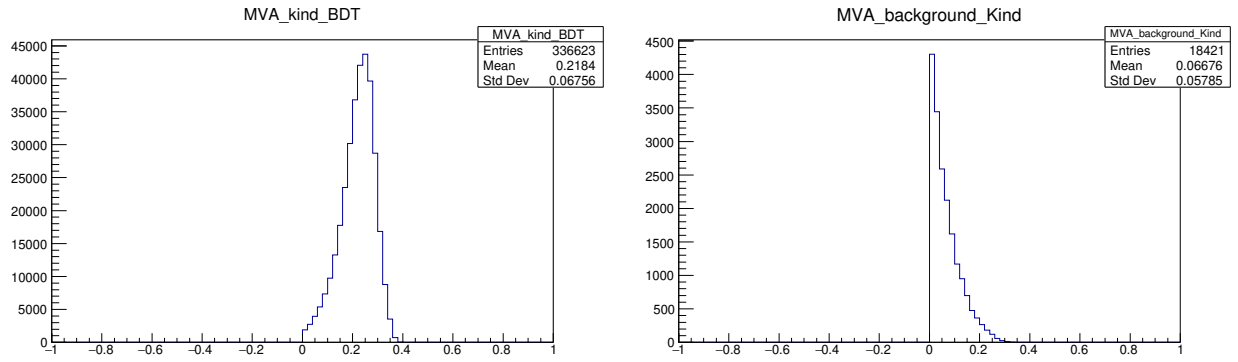
Figure 13: The BDT application output.

4.1.1 Performance

A BDT performance can be based on the amount of data 'lost' after a selection of data and the purity of the selection. The purity is explained in section 2.1 for decision tree nodes, but it can be likewise applicable to the whole of the BDT output. To evaluate the lost data, the signal efficiency is calculated through:

$$\text{Efficiency} = S_{\text{after}}/S_{\text{before}}, \tag{11}$$

where S_{before} and S_{after} are the number of signal candidates before and after the selection, respectively. Figure 14 shows the distribution of signal and background after the selection. It is evident most signal is selected, while most background is cleared away. The precise figures are displayed in Table 7. Almost 99% of the signal is selected, while about 98% of the background is removed. The total signal purity is almost 95%.



(a) Signal after selection.

(b) Background after selection.

Figure 14: Signal and background in selected output.

	Signal	**Background**	**Total**	**Purity**
Before selection	341449	1166070	1507519	0.2928
After selection	336623	18421	355362	0.9481
Efficiency	0.9859	0.0158		

Table 7: BDT performance.

5 Conclusion

The boosted decision tree method has been trained and tested and used to classify data taken from a Monte Carlo simulation of lead-lead collisions. The simulated data has a centrality of 40% - 60% with a center of mass energy of 2.76 TeV. The photons have been selected regarding signal and background to produce a more manageable and useful amount of data. The Monte Carlo simulation has been used to classify the data to generate an evaluation of the boosted decision tree method. Thirteen input variables are used to train the boosted decision tree while three so-called spectator variables are applied to the data.

Training and testing the boosted decision tree produced an optimal cut value of 0 to classify the data. The results of the training and testing stage indicated no significant overtraining and produced a stable operating boosted decision tree. Most variables are relatively uncorrelated which indicates a good performance of the boosted decision tree. In the training phase 200 000 background and signal randomly selected candidates were used and for testing 140 000 background and signal candidates.

The classification of photon particles taken from the Monte Carlo data is primarily implemented to evaluate the performance of the boosted decision tree. The data was selected using the optimal cut value acquired in from the training and testing stage. This eventually resulted in a signal efficiency of 0.9859 and a signal purity of 0.9481.

In my opinion the boosted decision tree method is a viable option for selecting data. When familiar with ROOT, TMVA and C++ the boosted decision tree method is relatively accessible and time saving. In this research training and testing of the data taken from the Monte Carlo simulation, which consisted of about 1 500 000 entries, generally took less than ten minutes. The performance results were also quite satisfying, as I believe that an efficiency of almost 99% and purity of almost 95% are quite impressive. Although I don't have produced any comparison with other machine learning methods, I certainly think that the boosted decision tree method is applicable in the data processing of particle physics data.

Appendix A

Flowchart for the two phases

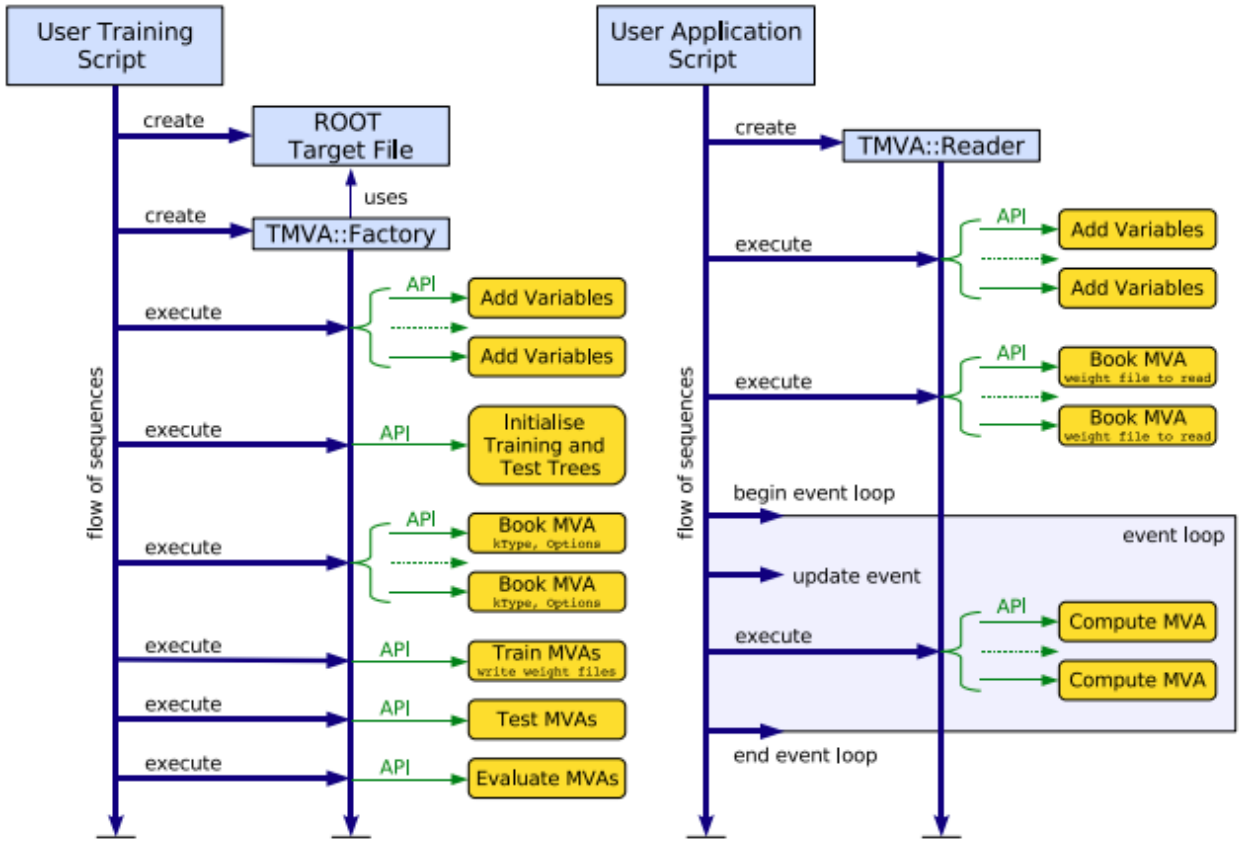


Figure 15: TMVA flowchart.

Appendix B

Input Variables

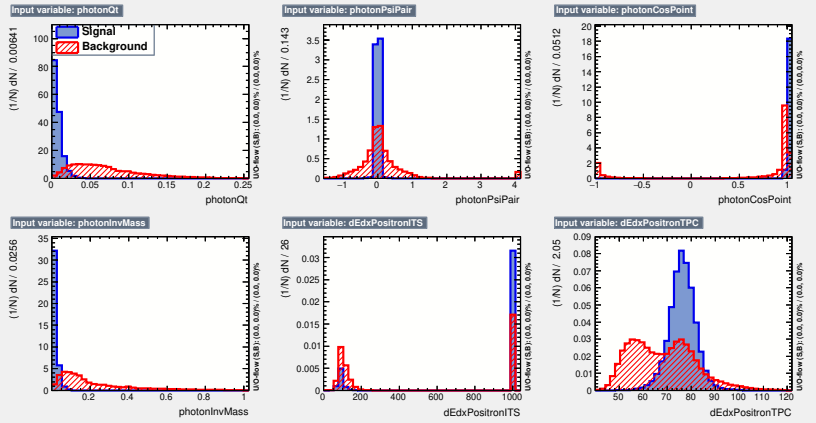


Figure 16: Input Variables(1).

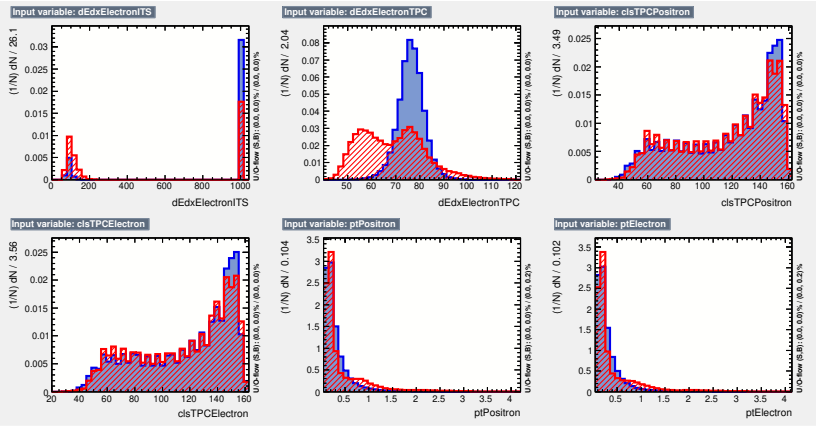


Figure 17: Input Variables(2).

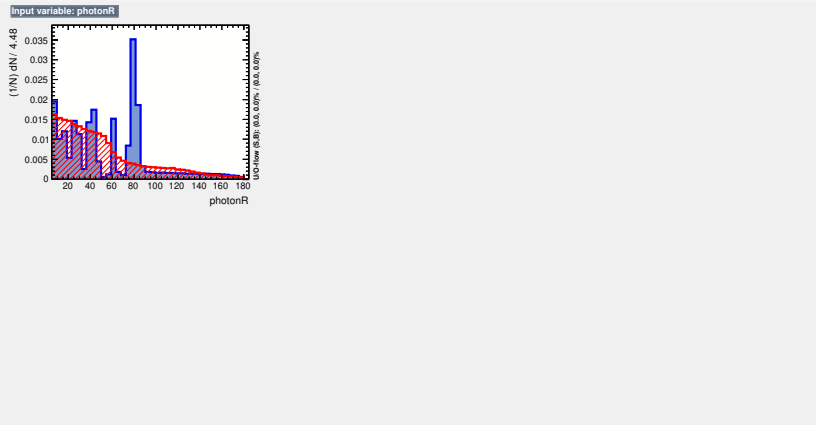


Figure 18: Input Variables(3).

Appendix C

Options table taken from the TMVA website[15]

Configuration options reference for MVA method: *BDT*				
Option	**Array**	**Default value**	**Predefined values**	**Description**
V	No	False	–	Verbose output (short form of VerbosityLevel below - overrides the latter one)
VerbosityLevel	No	Default	Default, Debug, Verbose, Info, Warning, Error, Fatal	Verbosity level
VarTransform	No	None	–	List of variable transformations performed before training, e.g., D_Background,P_Signal,G,N_AllClasses for: Decorrelation, PCA-transformation, Gaussianisation, Normalisation, each for the given class of events ('AllClasses' denotes all events of all classes, if no class indication is given, 'All' is assumed)
H	No	False	–	Print method-specific help message
CreateMVAPdfs	No	False	–	Create PDFs for classifier outputs (signal and background)
IgnoreNegWeightsInTraining	No	False	–	Events with negative weights are ignored in the training (but are included for testing and performance evaluation)
NTrees	No	800	–	Number of trees in the forest
MaxDepth	No	3	–	Max depth of the decision tree allowed
MinNodeSize	No	5%	–	Minimum percentage of training events required in a leaf node (default: Classification: 5%, Regression: 0.2%)
nCuts	No	20	–	Number of grid points in variable range used in finding optimal cut in node splitting
BoostType	No	AdaBoost	AdaBoost, RealAdaBoost, Bagging, AdaBoostR2, Grad	Boosting type for the trees in the forest
AdaBoostR2Loss	No	Quadratic	Linear, Quadratic, Exponential	Type of Loss function in AdaBoostR2

Figure 19: BDT options(1).

AdaBoostR2Loss	No	Quadratic	Linear, Quadratic, Exponential	Type of Loss function in AdaBoostR2
UseBaggedBoost	No	False	–	Use only a random subsample of all events for growing the trees in each iteration.
Shrinkage	No	1	–	Learning rate for GradBoost algorithm
AdaBoostBeta	No	0.5	–	Learning rate for AdaBoost algorithm
UseRandomisedTrees	No	False	–	Determine at each node splitting the cut variable only as the best out of a random subset of variables (like in RandomForests)
UseNvars	No	2	–	Size of the subset of variables used with RandomisedTree option
UsePoissonNvars	No	True	–	Interpret UseNvars not as fixed number but as mean of a Possion distribution in each split with RandomisedTree option
BaggedSampleFraction	No	0.6	–	Relative size of bagged event sample to original size of the data sample (used whenever bagging is used (i.e. UseBaggedGrad, Bagging,)
UseYesNoLeaf	No	True	–	Use Sig or Bkg categories, or the purity=S/(S+B) as classification of the leaf node -> Real-AdaBoost
NegWeightTreatment	No	InverseBoostNegWeights	InverseBoostNegWeights, IgnoreNegWeightsInTraining, PairNegWeightsGlobal, Pray	How to treat events with negative weights in the BDT training (particular the boosting) : IgnoreInTraining; Boost With inverse boostweight; Pair events with negative and positive weights in traning sample and *annihilate* them (experimental!)
NodePurityLimit	No	0.5	–	In boosting/pruning, nodes with purity > NodePurityLimit are signal; background otherwise.
SeparationType	No	GiniIndex	CrossEntropy, GiniIndex, GiniIndexWithLaplace, MisClassificationError, SDivSqrtSPlusB, RegressionVariance	Separation criterion for node splitting

Figure 20: BDT options(2).

DoBoostMonitor	No	False	–	Create control plot with ROC integral vs tree number
UseFisherCuts	No	False	–	Use multivariate splits using the Fisher criterion
MinLinCorrForFisher	No	0.8	–	The minimum linear correlation between two variables demanded for use in Fisher criterion in node splitting
UseExclusiveVars	No	False	–	Variables already used in fisher criterion are not anymore analysed individually for node splitting
DoPreselection	No	False	–	and and apply automatic pre-selection for 100% efficient signal (bkg) cuts prior to training
RenormByClass	No	False	–	Individually re-normalize each event class to the original size after boosting
SigToBkgFraction	No	1	–	Sig to Bkg ratio used in Training (similar to NodePurityLimit, which cannot be used in real adaboost
PruneMethod	No	NoPruning	NoPruning, ExpectedError, CostComplexity	Note: for BDTs use small trees (e.g.MaxDepth=3) and NoPruning: Pruning: Method used for pruning (removal) of statistically insignificant branches
PruneStrength	No	0	–	Pruning strength
PruningValFraction	No	0.5	–	Fraction of events to use for optimizing automatic pruning.
GradBaggingFraction	No	0.6	–	deprecated: Use *BaggedSampleFraction* instead: Defines the fraction of events to be used in each iteration, e.g. when UseBaggedGrad=kTRUE.
UseNTrainEvents	No	0	–	deprecated: Use *BaggedSampleFraction* instead: Number of randomly picked training events used in randomised (and bagged) trees

Figure 21: BDT options(3).

References

[1] Large Hadron Collider, URL https://home.cern/topics/large-hadron-collider.

[2] ALICE, URL https://home.cern/about/experiments/alice.

[3] ALICE, URL http://alice.web.cern.ch/.

[4] ALICE ITS, URL http://alice.web.cern.ch/detectors/more-details-alice-its.

[5] ALICE TPC, URL http://alice.web.cern.ch/detectors/more-details-alice-tpc.

[6] ALICE TOF, URL http://alice.web.cern.ch/detectors/more-details-time-flight.

[7] ALICE PHOS, URL http://alice.web.cern.ch/detectors/more-details-alice-photon-spectrometer.

[8] ALICE EMCal, URL http://alice.web.cern.ch/detectors/more-details-emcal.

[9] ALICE V0, URL http://alice.web.cern.ch/detectors/more-details-alice-v0-detector.

[10] ALICE T0, URL http://alice.web.cern.ch/detectors/more-details-about-t0.

[11] B. R. Martin, *Nuclear and Particle Physics* (2009), 2nd edition, (ISBN-13: 978-0470742754).

[12] E. M. Henley and A. Garcia, *Subatomic Physics* (2007), 3rd edition, (ISBN-13: 978-9812700575).

[13] M. Sas, *Particle identification with boosted decision trees* (2014), bachelor Thesis.

[14] A. Hoecker, P. Speckmayer, J. Stelzer, J. Therhaag, E. von Toerne, and H. Voss, *Tmva user guide*, october 4, 2013, URL http://tmva.sourceforge.net/docu/TMVAUsersGuide.pdf.

[15] URL http://tmva.sourceforge.net/optionRef.html.