

Study of BDT Training Configurations with an Application to the $Z/H \rightarrow \tau\tau \rightarrow ee$ Analysis

David Ciupke, University of Göttingen, Germany

6 September 2012

Abstract

We investigate the performance of boosted decision trees (BDT) with respect to Monte Carlo samples of the $Z/H \rightarrow \tau\tau \rightarrow ee$ and $Z/H \rightarrow ee$ analysis implemented via the Toolkit for Multivariate Analysis (TMVA). Following a general introduction to the methods of BDTs and the possible optimizations supported by TMVA we present the results of the study of the different configuration options of TMVA with respect to the two different data sets. We conclude that randomization of the training process can greatly enhance the results, but one relies on chance.

Contents

1	Introduction	3
2	Theory	3
2.1	Physics Background	3
2.2	Decision Trees	4
2.3	Boosting	5
2.4	Overtraining and Pruning	6
2.5	Toolkit For Multivariate Analysis (TMVA)	7
3	Analysis Results	9
3.1	Analysis for $Z \rightarrow \tau\tau \rightarrow ee$	9
3.2	Analysis with cut on DiLeptonMass	16
4	Conclusions	17
	References	18
A	Appendix	19

1 Introduction

So far the CMS $H \rightarrow \tau\tau$ analysis implies no excess [1], but more sensitivity is needed. This study is part of the analysis of the $H \rightarrow \tau\tau \rightarrow ee$ contribution. The “standard candle” for the Higgs-analysis is the $Z \rightarrow \tau\tau \rightarrow ee$ channel, where one can probe the possibilities for analysis of the Higgs-channel. The goal of this study is to discriminate the $Z \rightarrow \tau\tau \rightarrow ee$ channel from the $Z \rightarrow ee$ background with highest possible performance. The majority of signal events is in a mass window of 25 to 70 GeV of the invariant mass of the two leptons and the other background contributions in this region are negligible. One question in particular is then, if a precut to the mass window can give any performance boost in the discrimination.

One of the most important challenges to the preselection is to find suitable variables with high discrimination power. An example of how important the choice of variables is can be seen in sec. 3.1, where a variable with very high discrimination power is added to the classification problem.

Many classification problems are too complex to use linear cuts on event-variables as classification and typically involve a very large background compared to the signal. This makes a detailed study of the classification problem necessary. The current study refers only to classifications via boosted decision trees (BDT), see sec. 2.2. Decision trees are binary structured classifiers, that involve a sequence of decisions, so called nodes, on a input event, where each decision is made on a single event-variable. The sequence will terminate at a leaf, which classifies the event as background or signal. This way the phase space of events is nonlinearly cut into different regions, which are either signal or background. BDTs are known to have high performance in nonlinear classification problems, but they often suffer from overtraining (see sec. 2.2) and take a relatively long time to be trained. Typically BDTs require only little tuning of the configuration to achieve good results [2], a statement, that we confirm in our analysis, see sec. 4.

For the training of BDTs we use the Toolkit for Multivariate Analysis (TMVA) version 4.1.0, which is implemented in ROOT. Throughout the analysis the version 5.28 of ROOT was used.¹

2 Theory

2.1 Physics Background

The Higgs boson emerges from electroweak symmetry breaking in the Standard Model (as well as other models). Since the discovery of a neutral boson at a mass of 125 GeV last July in other final states the study of the $H \rightarrow \tau\tau$ channel has become very interesting, as it offers the possibility of detecting crucial properties of the boson and may indicate new physics.

¹The more recent version of ROOT was not able to properly implement the TMVA-Graphical User Interface (GUI).

This study is part of the $H \rightarrow \tau\tau$ -channel analysis in the di-electron final state. This channel is particularly challenging because of the small branching ratio of the $\tau \rightarrow e$ and the dominant Drell-Yan background $Z \rightarrow ee$. Before the actual $H \rightarrow \tau\tau \rightarrow ee$ -analysis the $Z \rightarrow \tau\tau \rightarrow ee$ -channel will be considered since the topology of these events is very similar to the higgs signal. Discrimination between signal and DY-background will be done via the MVA method of Boosted Decision Trees. The discriminating variables used for the BDT training describe the kinematics of the final state electrons:

- **DiLepEta:** The pseudorapidity of the di-electron system
- **PosMETDPhi:** The azimuthal angle between the direction of the positively charged electron three-momentum and the missing transverse energy
- **DiElePtRatio:** Ratio of missing transverse moment of both electrons
- **EleDCASig:** Inter electron distance of closest approach significance
- **ValidDiTau:** Set to one if the collinear approximation for the reconstruction of tau lepton pair kinematics yields a physical solution for the neutrino energies, otherwise the variable takes the value 0
- **CosPosDiLepton:** The decay angle of the positively charged electron in the rest frame of both electrons
- **CosPosDiLeptonPlane:** The angle between the three-momentum of the e^+ and the plane spanned by the three-momentum vector of the di-electron system and the protonbeam axis.

2.2 Decision Trees

Decision trees are used for classification problems and have a binary structure. For the problem at hand, they sequentially cut on discrimination variables. In this way an event is processed through the tree yielding a tree response of signal or background, see fig. ???. Via this method the space of discrimination variables is split in many rectangular regions, which are labeled by signal or background. The trees are grown respectively trained in a successive manner, i.e. starting with the root node. The cut value and discrimination variable at each node is chosen, such that it provides the best separation between signal and background. The training stops at a node as soon as the a critical lower bound of events (**nEventsMin**, see sec. 2.5) is reached at that node. The leafs, i.e. the final nodes, determine whether the event is signal or background and they are labeled according to their purity $p = S/(S + B)$. When $p > 0.5$ the event is considered signal if $p < 0.5$ it is considered background. The quality of separation is defined by the so called impurity function, some examples are the following:

- *Gini Index:* $p \cdot (1 - p)$
- *Cross entropy:* $-p \cdot \ln(p) - (1 - p) \cdot \ln(1 - p)$

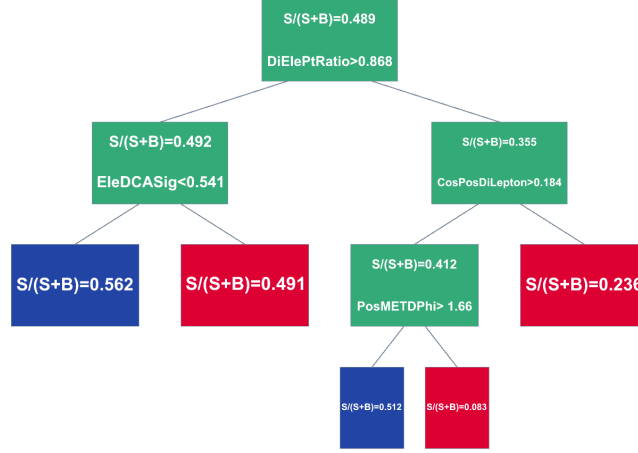


Figure 1: Example of a decision tree showing intermediate nodes (green) as well as signal (blue) and background (red) leafes. The variables used stem from the Z/H data analysis. The picture was produced by TMVA.

- *Misclassification error:* $1 - \max(p, 1 - p)$
- *Statistical significance:* $S/\sqrt{S+B}$.

The training procedure then chooses the variable and cut value, that maximizes the increase in the impurity function between an initial node and the sum of the daughter nodes weighted with their corresponding fraction of events. TMVA does this with a granularity set by the parameter `NCuts`. Setting this parameter to the value -1 TMVA will determine the optimal value of it, i.e. the one yielding highest performance.

2.3 Boosting

Decision trees are sensitive to statistical fluctuations of the training data sample. A way to make decision trees more robust and also increase the performance is to use boosting, a method that is generally applicable to any classifier [2]. The idea always is to combine a collection of weak classifiers into one that is strong. TMVA provides three different boosting types: Adaptive boosting, gradient boosting and bagging. Adaptive Boosting (**AdaBoost**) creates a strong learner by training weak learners sequentially in a way that they “learn“ from the errors of previously grown trees. First an initial tree is grown. Afterwards the fraction of misclassified events f_1 is determined and all misclassified events will be reweighed with a weight $\alpha_1 = (1 - f_1)/f_1 \geq 1$. A second tree will be trained with the reweighed data, the sample will again be reweighed and another tree be grown. This process will go on until a certain `NTrees` are grown, this is then referred to as a forest.

Depending on the particular TMVA configuration an event x , which is put into a single tree, will either give a response of signal or background, corresponding to values $h(x) = \pm 1$, or will respond a purity value. The corresponding option is called

UseYesNoLeaf. Adaptive boosting then yields a response defined by

$$y_{Ada}(x) = \frac{1}{\text{NTrees}} \sum_{i=1}^{\text{NTrees}} \ln(\alpha_i) h_i(x) . \quad (2.1)$$

When using bagging, a forest of trees is grown in such a way, that each tree is grown with a resampled training sample. Thereby the strong learner is a statistical combination of the weak learners.

Gradient boosting on the other hand is not demonstrative, but implemented in a rather technical way [2]. It is generally more robust than adaptive boosting and in TMVA also implemented with the option of extra bagging via the option **UseBaggedGrad**. More options related to it are listed in tab. 1.

Another option that is available for each boosting type is **UseRandomisedTrees**. If the option is set to **True**, each tree in the forest will be grown in a way that at each node splitting only a random subset of all training variables will be used, the total amount of variables at each node can be set via the option **UseNvars**. Further randomised trees allows for a bagging-type resampling of the training data set for each tree grown. The number training events is determined via the option **UseNTrainEvent**. A randomization only occurs, if less events are used than there are in the whole training sample.

2.4 Overtraining and Pruning

When classifiers are too much adapted to a specific training sample, similar samples can have a completely different response from the trained classifier. In this case one talks of overtraining. Overtraining is problematic as the real data will be classified different from the training sample and the BDT output distribution can not be used to determine the optimal cut value. BDTs are in general sensitive to overtraining. Different techniques called "pruning" have been developed to deal with this problem in the case of BDTs. Pruning usually refers to the process of cutting down the tree from the leaf-nodes further into the tree removing statistically insignificant nodes. In TMVA pruning is only available for **AdaBoost** and provides two methods **CostComplexity** and **ExpectedError**.

The Cost Complexity pruning cuts away subtrees, whose gain in classification is not "justified" by the number of extra nodes required for it. It is defined via the so called cost complexity for a node

$$\rho = \frac{R(\text{node}) - R(\text{subtree below node})}{\#(\text{subtree}) - 1} , \quad (2.2)$$

where $R = 1 - \max(p, 1-p)$ is the misclassification error. Nodes with $\rho < \text{PruneStrength}$ are then iteratively cut out of the tree.

Expected Error pruning on the other hand recursively cuts off leaf nodes from the tree, whose combined statistical error

$$\text{err} = \sqrt{\frac{p(1-p)}{N}} , \text{ where } N \text{ is the number of training events in the node.}$$

of the leaf nodes is higher than the error of the parent node. The statistical error might be manipulated by multiplying it with `PruneStrength`.

In the case of `CostComplexity` TMVA provides the option `PruneStrength = -1`, which automatically determines the value for `PruneStrength`, that yields the highest performance, by using a subsample of the training sample. This option is not available for `ExpectedError`. It should also be remarked, that pruning in TMVA is only available for `AdaBoost`.

2.5 Toolkit For Multivariate Analysis (TMVA)

As mentioned in the introduction TMVA provides a framework for multivariate analysis implemented in `root`. In particular it has incorporated boosted decision trees with some of the options already explained in the previous sections. Fig. ?? shows the different elements of the BDTs in TMVA and their corresponding configuration variables such as pruning or boosting. TMVA can be initialized via a simple `root` script. For the preparation of the testing and training samples the MC data samples for every event class considered ($Z \rightarrow \tau\tau \rightarrow ee$ for signal and $Z \rightarrow ee$ for background) are reweighed according to their cross section and number of generated events. For the $Z \rightarrow \tau\tau \rightarrow ee$ analysis two different data sets were included, one having a transverse momentum p_T of 50 GeV, the other a transverse momentum in the range from 10 to 50 GeV. TMVA also has the option of applying precuts to the sample. The preparation is done via the class `factory`, i.e.

```
factory → PrepareTrainingAndTestTree(cuts, "options");
```

This includes the precuts and splits the entire sample into test and train sample via the option `SplitMode`. The splitting can be chosen as randomised via the value `Random`. The randomised division is implemented via the random generator `TRandom3`, that is the Mersenne Twister. Using this option the random seed can be set to a certain value via `SplitSeed`. Using the value `SplitSeed = 0` determines a seed by random, such that the randomization yields a different result every time it is used. One may also split the sample just into two blocks via the value `Block` or use alternating via `Alternate`. Further the ratio of test and train samples can be fixed via the options `nTrain_Signal`, `nTest_Signal`, `nTrain_Background` and `nTest_Background`, the standard configuration here is a split in half testing and training samples. Further one can normalize the event weights via the option `NormMode`. If the value `NumEvents` is chosen, the sample is normalized to the number of events.

After preparing the sample one can book the MVA method via the function

```
factory → BookMethod(TMVA :: Types :: kBDT, "BDT", "options"); ,
```

where the first argument specifies the MVA method, in this case BDT, the second assigns a name to the method and the last is used for further options. Some of the options have already been mentioned in the previous sections, a full table of booking

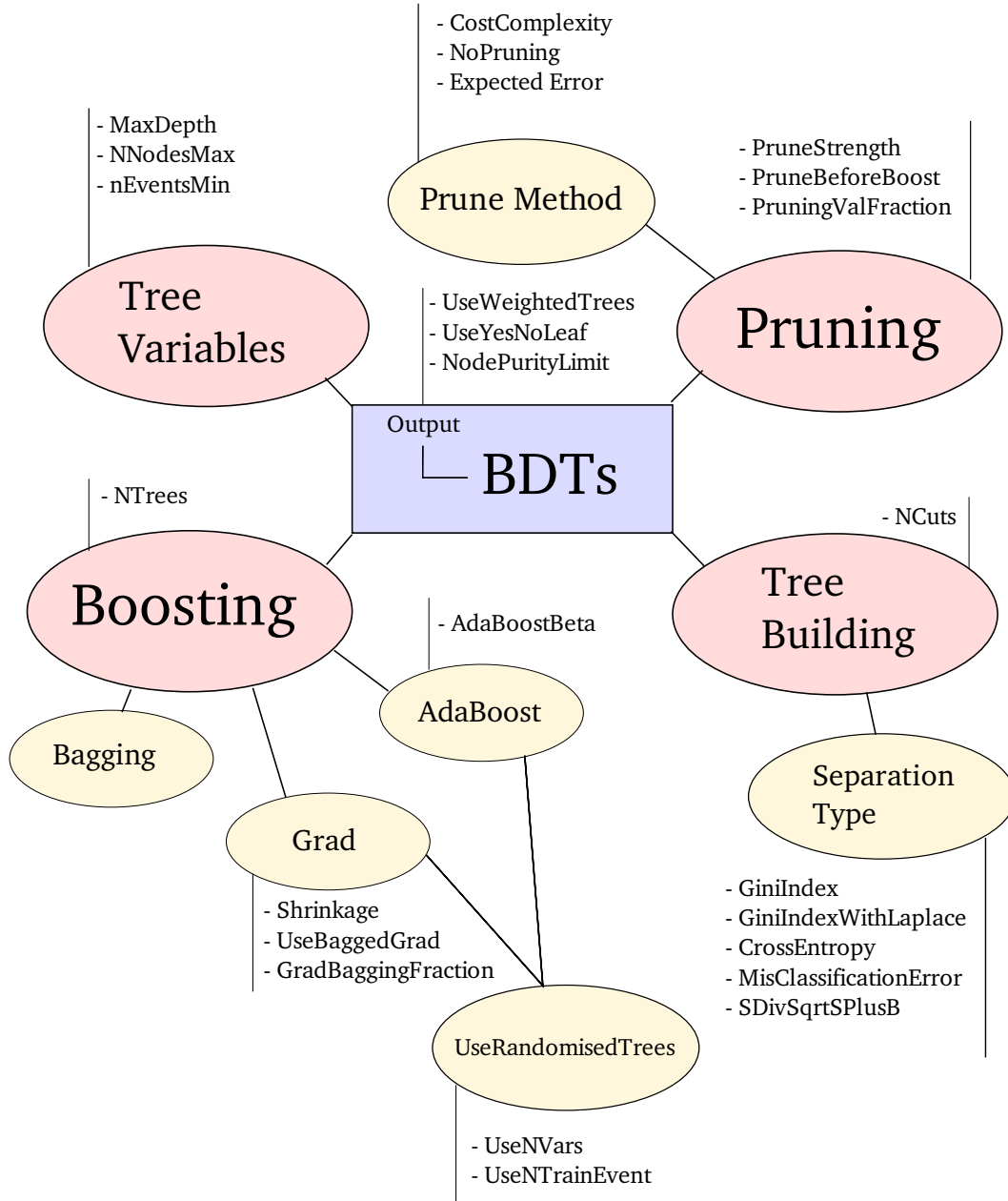
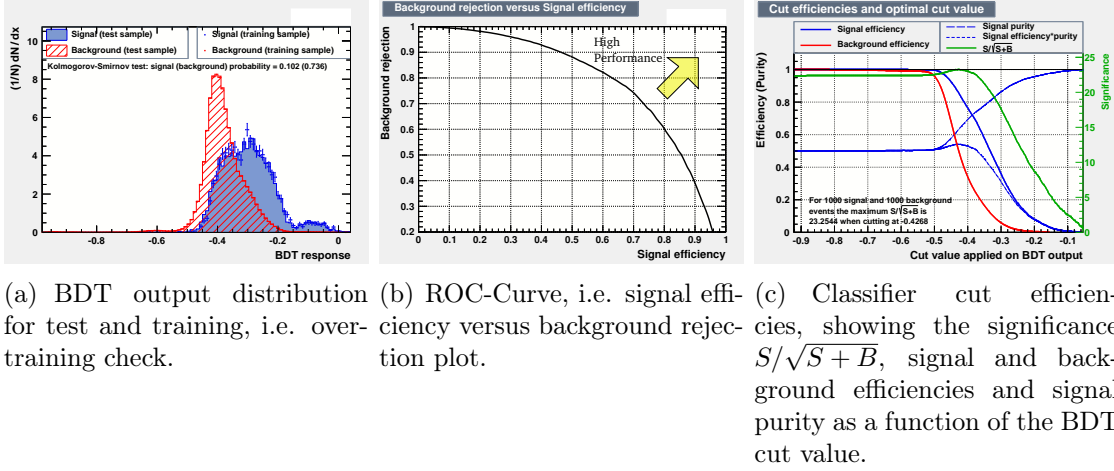


Figure 2: Overview over the different building blocks of the TMVA-BDT-configurations and the corresponding configuration variables of TMVA.

options for BDT can be found in tab. 1.

TMVA comes with a graphical user interface (TMVAGui.C) with a menu of different plots implemented. On one hand it shows the distributions of the input variables, their scatter plots and correlations, e.g. see fig. 11 and fig. 9 in the appendix. On the other hand it shows the classifier output distributions of signal and background, i.e. the probability density functions of the classifier, see fig. 3. In order to check overtraining the BDT output distribution plot is available showing



(a) BDT output distribution for test and training, i.e. over-training check. (b) ROC-Curve, i.e. signal efficiency versus background rejection plot. (c) Classifier cut efficiencies, showing the significance $S/\sqrt{S+B}$, signal and background efficiencies and signal purity as a function of the BDT cut value.

Figure 3: Different plots provided by the TMVA graphical user interface.

the training and testing samples superimposed. Further plots include the so called ROC-curve showing the signal efficiency versus background rejection and the classifier cut efficiencies, see fig. 3. The latter consists of a plot displaying significance $S/\sqrt{S+B}$ as a function of the BDT cut value for a specific number of signal and background events. This value is the parameter indicating the performance of the BDT-training most accurately. However one also has to take statistics into account, which means that one should keep in mind the signal efficiency that corresponds to the optimal cut value. If this value is very low, a lot of signal is cut away. If one has low statistics in the real data, one can not be sure anymore of the quality of the cut, since the output distribution is different from the one of the training sample. TMVA also prints out the integral over the ROC-curve for every booked MVA method. This value can be used as a first indicator of the quality of a training. The TMVA-Gui also includes plots showing each decision tree of the forest as well as tree control plots, see fig. ?? and fig. 10. The control plots show boost weights and the fraction of misclassified events per tree as well as the number of nodes before and after pruning.

3 Analysis Results

3.1 Analysis for $Z \rightarrow \tau\tau \rightarrow ee$

Our studies of BDT configurations for TMVA involve mostly probing the impact and behaviour of distinct configuration variables. A systematic study covering all possible configurations on a grid of most their values would need much more computational power and time. Especially computing power was limited in our studies as a typical training would take between 30 and 100 minutes. However we can deduce approximate statements about the importance of the configuration variables and their interplay.

The input variables used for the training of the sample are added in the appendix, fig. 11. Their correlation plots are shown in fig. 9. We begin our analysis with the

Option	Default	Values	Description
NTrees	200	-	Number of trees in the forest
BoostType	AdaBoost	AdaBoost , Bagging, Grad	Boosting type for the trees in the forest
UseBaggedGrad	False	-	Use only a random subsample of all events for growing the trees in each iteration.
GradBaggingFraction	0.6	-	Defines the fraction of events to be used in each iteration when UseBaggedGrad=True.
Shrinkage	1	-	Learning rate for GradBoost algorithm
AdaBoostBeta	1	-	Parameter for AdaBoost algorithm
UseRandomisedTrees	False	-	Choose at each node splitting a random set of variables
UseNvars	4	-	Number of variables used if randomised tree option is chosen
UseNTrainEvent	N	-	Number of Training events used in each tree building if randomised tree option is chosen
UseWeightedTrees	True	-	Use weighted trees or simple average in classification from the forest
UseYesNoLeaf	True	-	Use Sig or Bkg categories, or the purity as classification of the leaf node
NodePurityLimit	0.5	-	In boosting/pruning, nodes with purity > NodePurityLimit are signal; background otherwise.
SeparationType	GiniIndex	CrossEntropy, GiniIndex, GiniIndexWithLaplace, MisClassificationError, SDivSqrtSPlusB,	Separation criterion for node splitting
nEventsMin	20	-	Minimum number of events required in a leaf node (default uses given formula)
nCuts	20	-	Number of steps during node cut optimisation
PruneStrength	-1	-	Pruning strength
PruneMethod	CostComplexity	NoPruning, ExpectedError, CostComplexity	Method used for pruning (removal) of statistically insignificant branches
PruneBeforeBoost	False	-	Flag to prune the tree before applying boosting algorithm
PruningValFraction	0.5	-	Fraction of events to use for optimizing automatic pruning.
NNodesMax	100000	-	Max number of nodes in tree
MaxDepth	100000	-	Maximal depth of the decision tree allowed

Table 1: BDT configuration variables together with their standard values, predefined options and further explanations.

options for the preparation of the test and training sample. The option `NormMode=NumEvents` was used throughout the analysis as well as `SplitMode=Random`. The change of the split seed however yielded quite different results depending on the booking options. We will consider changes in performance, that are indistinguishable from the regular deviations, i.e. with respect to the “standard configuration”

```
factory → BookMethod(TMVA :: Types :: kBDT, "BDT", "NTrees = 2000 : nEventsMin
= 150 : MaxDepth = 3 : BoostType = AdaBoost : AdaBoostBeta = 0.5 : nCuts = 20
: PruneMethod = NoPruning"); ,
```

due to a change of split seed, as insignificant. Tab 2 shows the ROC-curve integral for `SplitSeed=0,50,100` and `NTrees=100,500,2000`, one can see deviations of up to 0.01. For the significance the deviation can be up to 3 %. The discrepancy between

SplitSeed	Int ROC(100)	Int ROC(500)	Int ROC(2000)
0	0.774	0.771	0.777
50	0.772	0.773	0.775
100	0.779	0.780	0.783

Table 2: Difference in the ROC-curve integral for different values of the split seed and number of trees, indicated through the brackets. A stabilization with increase in trees is not seen.

the distributions of the test and training events is given in fig. 12 for the example of the variables `CosPosDiLepton` and `PosMETDPhi` for each signal and background events. The discrepancy is much smaller for larger statistics, i.e. background events. This results in the fact, that the BDT performance of the signal is correlated to the random seed, that has been used for the generator. Using more trees smoothens out this effect, but it can be enhanced for certain configurations. In fact using the option `UseRandomisedTrees=True` strongly enhances the discrepancy between individually different trees due to statistical fluctuations, such that the probability density function of the BDT response has either no overtraining and high performance or significant overtraining and considerably less high performance, we will get back to this point later on.

The splitting of the test and training sample was done with the standard configuration, i.e. a splitting into 50 % test and 50 % training sample. A test run over a granularity of $i \cdot 10\%$ of test data for $i = 1, \dots, 9$ did not yield any significant change in performance.

Let us now proceed to the study of the booking options. The boosting type `BoostType=Bagging` could not be compiled in the used root and TMVA versions and could therefore not further be investigated. `AdaBoost` and `Grad` were usually studied separately.

The first variable investigated is `NTrees` w.r.t `AdaBoost`. An increase of this variable is expected to yield a stabilization of the BDT output distribution with respect to

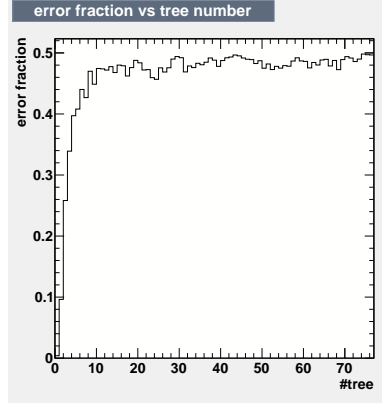


Figure 4: Error fraction as a function of the tree index: zoom of plot in fig. 10.

statistical fluctuations and make the distribution more smooth. This behaviour was confirmed. Also the center of the distributions would be shifted to greater values of the BDT response for a larger number of trees. This is due to the fact, that the boost weights usually have an attractor of 1 for increasing tree number as can be seen for instance in fig. 10. If the boost weights are all approximately the same, the distributions are shifted towards the value of 0 with increasing number of trees, since the trees classify signal/background only about 50 % of the times correctly. This process is slow due to the fact that in the forest response the weights are incorporated via a logarithm and therefore nearly vanish. Further it was observed that the overtraining is increasing with larger forests. This does strongly depend on the other configuration parameters like pruning or `MaxDepth`. Similar results were produced for `Grad` with the difference that the BDT response PDF already stabilizes for small forests as `Grad` is more robust than adaptive boosting.

It is somewhat surprising that adaptive boosting performs so bad, i.e. it does not enhance the trees but rather makes them worse as can be seen from the control plot in fig. 4, where the first trees have a low misclassification error, but the error becomes larger with increasing tree index. One should keep in mind that a tree, that misclassifies 50 % of the events, is as good as a random guess.

The options `UseWeightedTrees` and `UseYesNoLeaf` were tested for `AdaBoost` and the results of the BDT response distributions can be seen in fig. 5. We again see that adaptive boosting misclassifies nearly 50 % of all events. Using unweighted classifiers yielded a worse performance and more overtraining. The option `UseYesNoLeaf=False` made the PDF more peaky and did not increase performance.

For all used data sets, that is with and without extra cut on `DiLeptonMass`, the different options for `SeparationType` showed approximately the same hierarchy in performance. The options `GiniIndex` and `CrossEntropy` yielded the same performance, while `GiniIndexWithLaplace` performed slightly and `SDivSqrtSPlusB` as well as `MisClassificationError` significantly worse. The granularity for node splitting, namely `nCuts`, is insignificant, the performance saturates already at values around 20. Also the TMVA provided option `nCuts=-1` does not yield any significant benefit, but drastically increases computation time.

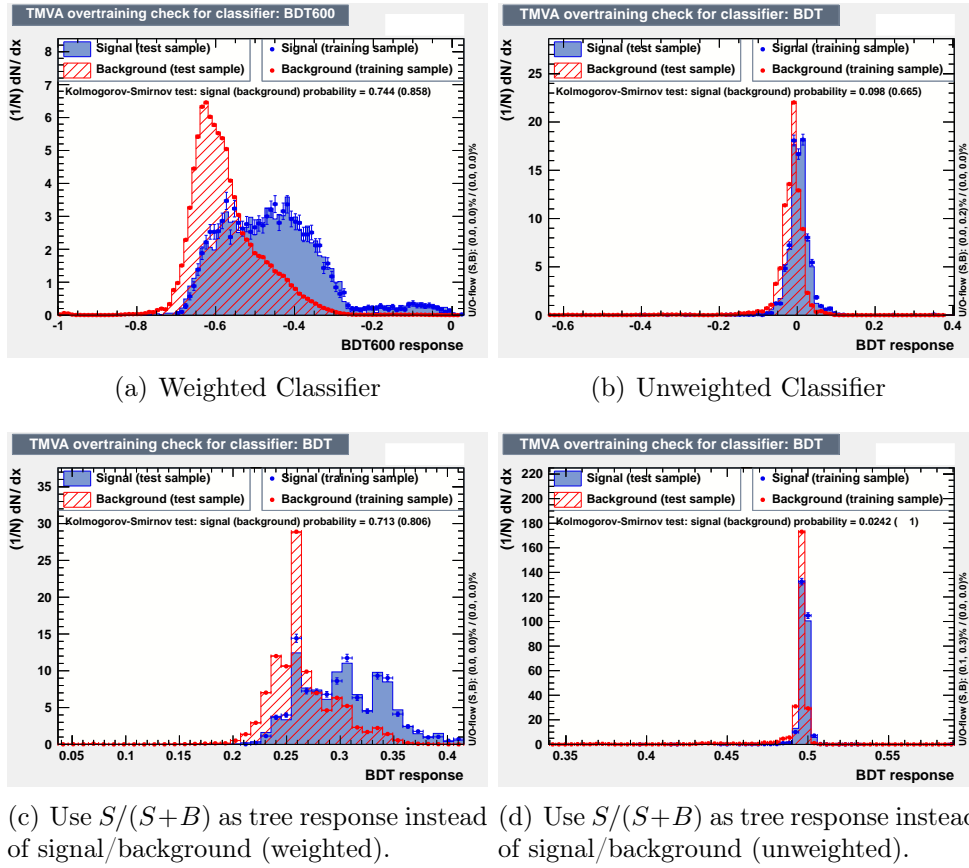


Figure 5: Difference in options `UseWeightedTrees` and `UseYesNoLeaf` with respect to a training configuration using 600 Trees, AdaBoost, MaxDepth= 3 and no pruning. Fig. 5(b) implies, that signal and background are misclassified in almost 50 % of all trees.

A study of the boosting parameters `AdaBoostBeta` and `Shrinkage` revealed that values of `AdaBoostBeta` around 0.5 perform best, but changing the value is insignificant. The deviation of performance for `Shrinkage` is larger, the best performance can be achieved with values below 0.1. Usually the increase in performance saturates at a level of 0.05. Note also that decreasing this values will further increase computation time.

Let us now proceed to the building block of controlling overtraining. In general one has two different possibilities of dealing with the latter by pruning or changing parameters such as `MaxDepth` or `nEventsMin`. Several trainings showed, that pruning does not work together with values of `MaxDepth` lower or equal than 5, an example of this can be seen in fig. 10. For higher values it does work, but determining the optimal value of `PruneStrength` is required. For lower values overtraining is still present, while for too high values the BDT response PDFs become discrete. Having determined the optimal pruning strength the BDT output distributions are still wiggly and overtraining is considerably high, usually below the “critical” value of the Kolmogorov-Smirnov test of 0.01. Because trees have to have greater depth for the pruning to work, the computation time is higher.

Controlling the overtraining via fine-tuning of the parameters `MaxDepth`, `nEventsMin` and `NNodesMax` proved to be more reasonable. A general choice of these values for any data sample can not be given, but in fact requires a detailed study. It is worth mentioning that these parameters have a different granularity in dealing with the overtraining. `MaxDepth` is the coarsest parameter, followed by `NNodesMax` and `nEventsMin`, such that one would start by finding a reasonable range for this value first. For the given data set both the choices `MaxDepth`=3 and 4 greatly reduce overtraining without compromising performance. The simulations were in favor of leaving `NNodesMax` untouched while tuning `nEventsMin`. This is due to the fact that `NNodesMax` is not much different from `MaxDepth` and the data set had enough statistics to be fairly resilient against overtraining. Values of `nEventsMin` between 1000 and 2000 performed best in dealing with remaining overtraining.

Further there was an inconsistency with the usage of `ExpectedError`-pruning observed. Fig. 14 shows two configurations that use a different value of `PruneStrength`. As already mentioned pruning did not work together with too small values of `MaxDepth` like in this case, so one would expect both plots to coincide. In fact the result for `PruneStrength`=50 could not be reproduced and every training yielded exactly the output of fig. 14(b). This strongly suggests that fig. 14(a) is the result of an inconsistency in TMVA or in the graphical user interface.

The option `UseRandomisedTrees=True` is of particular interest since it exhibits the ability to enhance the performance of the training significantly. Fig. 13 and tab. 3 show the output distributions, ROC-curves and significances of trainings using randomised trees together with `Grad` for `SplitSeed`=0 and 100. One can see that the training performs very well for `SplitSeed`=0 and `NVars`=2, ..., 6, but performs only average for `NVars`=7 and `SplitSeed`=100. At the same time the great performance

goes along with almost no overtraining, while the average performance simulations are overtrained. This behaviour was already mentioned. Similar behaviour for AdaBoost was observed.

One may think, that there is no advantage in using `UseRandomisedTrees=True`, since it decreases the separation power of the training. In fact the training procedure only takes into account which variable yields the greatest separation power at the corresponding node, but ignores the further splitting of the subtree. Randomised trees can give in advantage in the sense, that it may cause scenarios, where variables are chosen, that do not necessarily give the best separation at a particular node, but yield better subtrees.

The option `UseBaggedGrad=True` was also able to enhance the performance similar to randomised trees, but was not studied to a great extent, since it does not offer as many possibilities as `UseRandomisedTrees=True` and is not implemented for AdaBoost.

Final Training Configuration

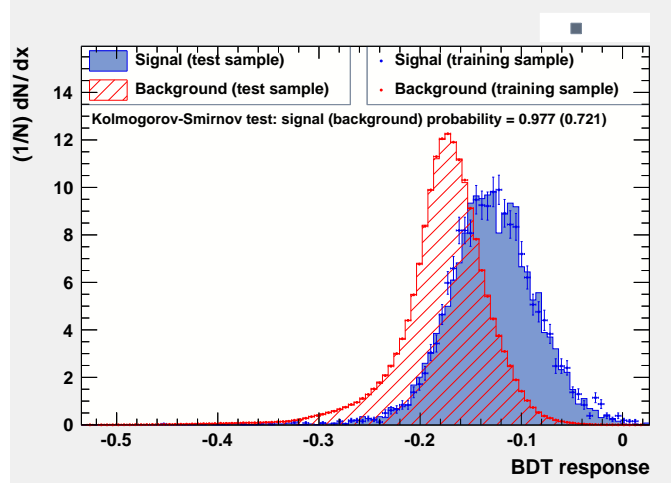
The result of the study is a configuration had the following booking options (with `SplitSeed=0`)

```
NTrees = 2000 : nEventsMin = 2000 : MaxDepth = 4 : BoostType = AdaBoost :
AdaBoostBeta = 0.6 : UseRandomisedTrees = True : UseNVars = 6 : nCuts
= 2000 : PruneMethod = CostComplexity : PruneStrength = -1 ,
```

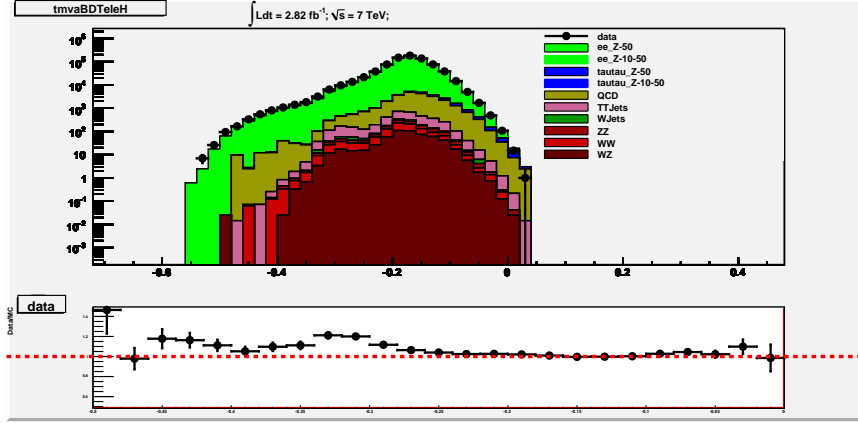
the corresponding BDT PDFs can be found in fig. 6. This configuration includes pruning even though it is just making the training set smaller, since the option `PruneStrength=-1` was chosen. The quality of the training was produced by chance, similar to the example for the usage of randomised trees earlier. The optimal cut value is at -0.13 and yields a boost in significance of 10 % with respect to the standard configuration, reaching a value of $S/\sqrt{S+B} = 16.5$ with 1360 signal events at an integrated luminosity of $\int L dt = 2.82 \text{ fb}^{-1}$. In the proximity of the optimal cut value the data and MC seem to agree well with each other. The corresponding DiLeptonMass distributions including all channels before and after applying the corresponding cut are shown in fig. 15 showing the mass window, where the majority of signal events lie. Further also the BDT control plots are found in fig. 10.

Analysis including DiLeptonMass as a training variable

Training the sample including DiLeptonMass will not be implemented in the analysis, but shows how much the discrimination quality determines the performance of the BDTs. The fact that BDTs ignore weakly discriminating variables further enhances this effect with respect to this example. From fig. 7 one can clearly see the large boost in discrimination power. In terms of significance this is roughly a boost from a range of 7 to values greater than 22.



(a) BDT output distribution for the final training configuration.



(b) BDT output distribution for the final training configuration including all background channels.

Figure 6: BDT output distributions for the result of the $Z \rightarrow \tau\tau \rightarrow ee$ study.

3.2 Analysis with cut on DiLeptonMass

Selecting a precut of $25 < \text{DiLeptonMass} < 70$ greatly reduces the amount of background from 2334154 to 113760 events. In this sense the training is different as the statistics are worse. The training exposed to be in favor of the value $\text{MaxDepth}=3$, but the overtraining was in general larger than in the previous study and harder to get under control. This made the usage of the parameter NNodesMax feasible. For this the training configuration

```
NTrees = 2000 : nEventsMin = 1000 : NNodesMax = 3 : BoostType = Grad :
Shrinkage = 0.04 : UseBaggedGrad = True : UseNVars = 6 : nCuts = 2000 :
MaxDepth = 3 ,
```

proved to give high performance. Using 2300 signal and 40000 background events yielded a significance of 12.95 at a cut of -0.818 corresponding to 1306 signal events

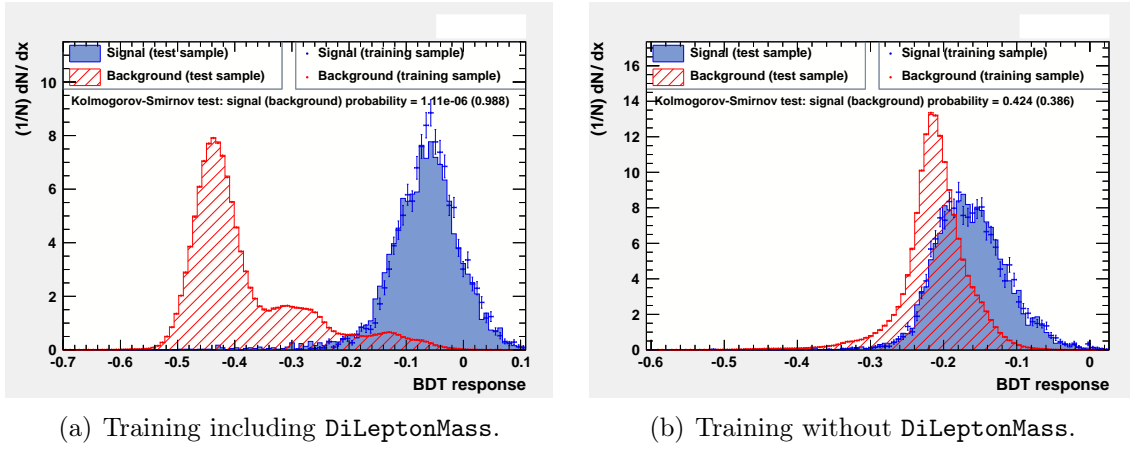


Figure 7: BDT output distribution for a training including DiLeptonMass and one without. The training was done with AdaBoost.

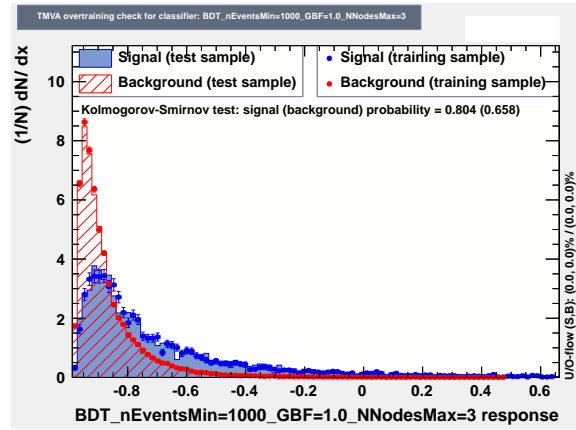


Figure 8: BDT output distribution for the training configuration mentioned in sec. 3.2 with respect to the precutted data sample.

after applying the cut. This implies a worse performance than the training without cut. On the other hand the choices taken do not involve `UseRandomisedTrees=True` and therefore a better training seems likely, but will take longer to be found due to the usual overtraining. Also the option `NNodesMax=3` is quite drastic, leaving the training with extremely short trees. Further studies will have to probe, if the training after applying the cut has in fact a worse performance than applying the cut after the training.

4 Conclusions

For the training of the discrimination between $Z \rightarrow \tau\tau \rightarrow ee$ and $Z \rightarrow ee$ it could be concluded that the training without precut on DiLeptonMass performs better than with precut. For the TMVA configurations in particular it was found that AdaBoost and Grad had no significant deviation in performance or overtraining and were in total very comparable.

The comparison between pruning and adjusting tree parameters like `MaxDepth` implies that one should use tree parameters to reduce overtraining or rely on chance using the option `UseRandomisedTrees=True`, since this greatly enhances performance. In every case the options `MaxDepth = 3,4` proved to be valuable as they did not decrease the performance, but reduced overtraining.

Most of the chosen BDT configurations performed equally well or were indistinguishable from the gain or loss in performance induced by choosing a different seed for the randomised selection of test and training events from the entire sample using the “standard” configuration. It was shown that the options `UseRandomisedTrees=True` and `UseBaggedGrad=True` can significantly enhance the performance of the BDT training, but require repetitive trainings as the outcome depends on chance.

First test runs with respect to the $H \rightarrow \tau\tau \rightarrow ee$ sample showed that the obtained results also apply to the sample and yielded high performance.

A possible optimization would be to use artificial neural networks as they have a higher theoretical best performance in problems that involve many well discriminating variables. However the problem at hand involves mainly poorly discriminating variables, that would have to be removed such that the neural networks perform well enough [2]. A detailed study would be necessary to test, if the performance boost of the neural networks overcompensated the loss of performance due to the cut-down of variables.

Acknowledgements

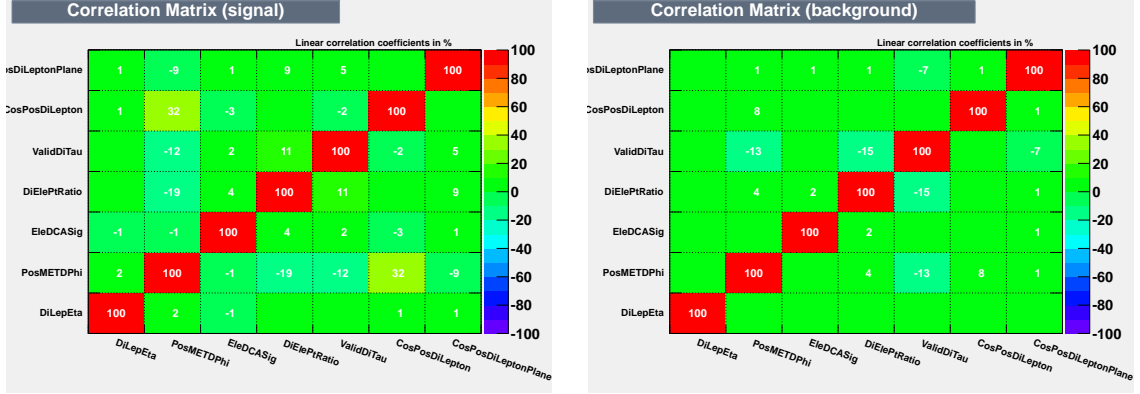
I would like to thank the DESY organisational team for this summerschool program, I enjoyed it very much. Further I want to thank my supervisors Alexei Raspereza, Jakob Salfeld-Nebgen and Agni Bethani for the great time I had working in their group and helping me whenever I encountered any problems. Their company was much appreciated and I felt very welcome. I wish we could have spent more time together as Jakob was not here during most of my stay. Also I would like to thank Gregor Hellwig, who was helping me with some problems.

Further I wish to thank my friends Jože Zobec and the other people from the wooden hostel for all the fun we had and in particular my officemate Khilesh Mistry, who was a great source of distraction and gossip.

References

- [1] S. Chatrchyan et al. [CMS Collaboration]: Phys. Lett. B [arXiv:1207.7235 [hep-ex]]
- [2] A. Hoecker et al.: *TMVA 4 Users Guide*, 2009 [arXiv:physics/0703039]

A Appendix



(a) Input variable correlation matrix for signal.

(b) Input variable correlation matrix for background.

Figure 9: Input variable correlations for the uncut sample.

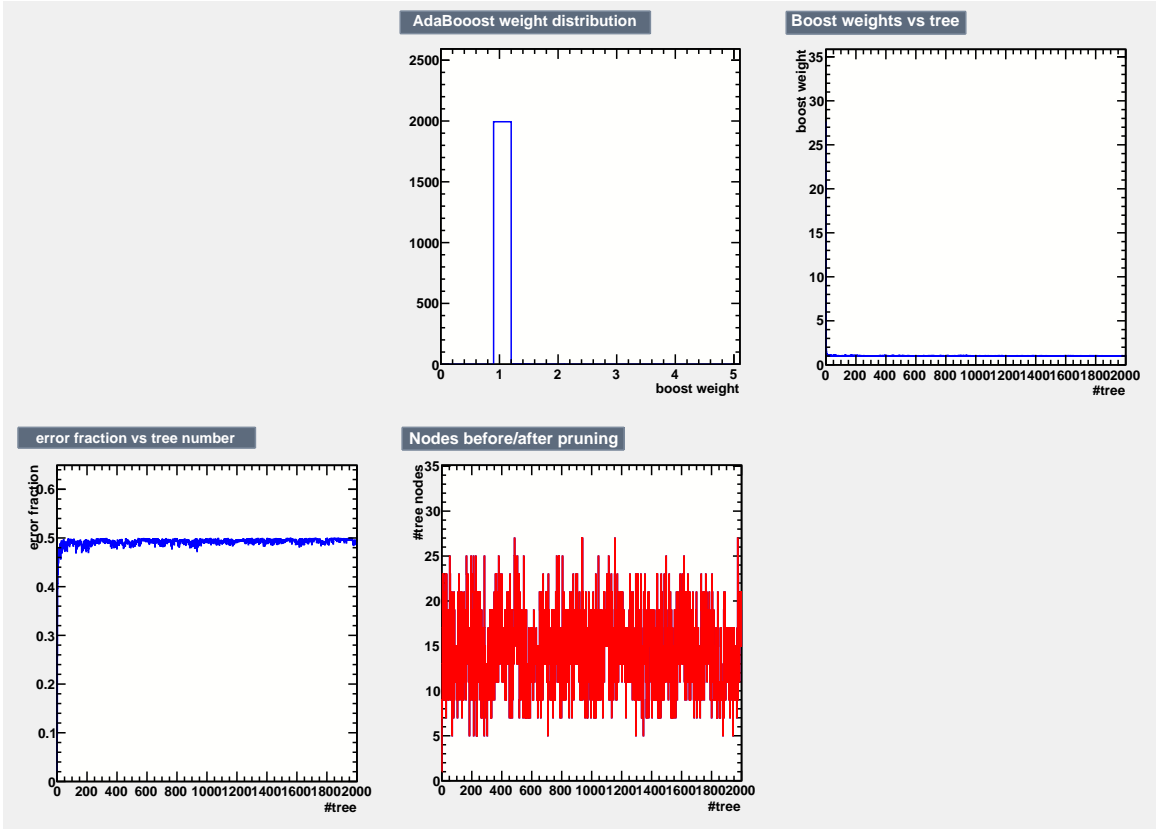


Figure 10: BDT control parameters for the final result of the uncut Z study. The plots imply that no tree was pruned and AdaBoost reaches an approximate limit of boost weight 1 after a certain number of trees are grown. This is also visible in the error fraction plot.

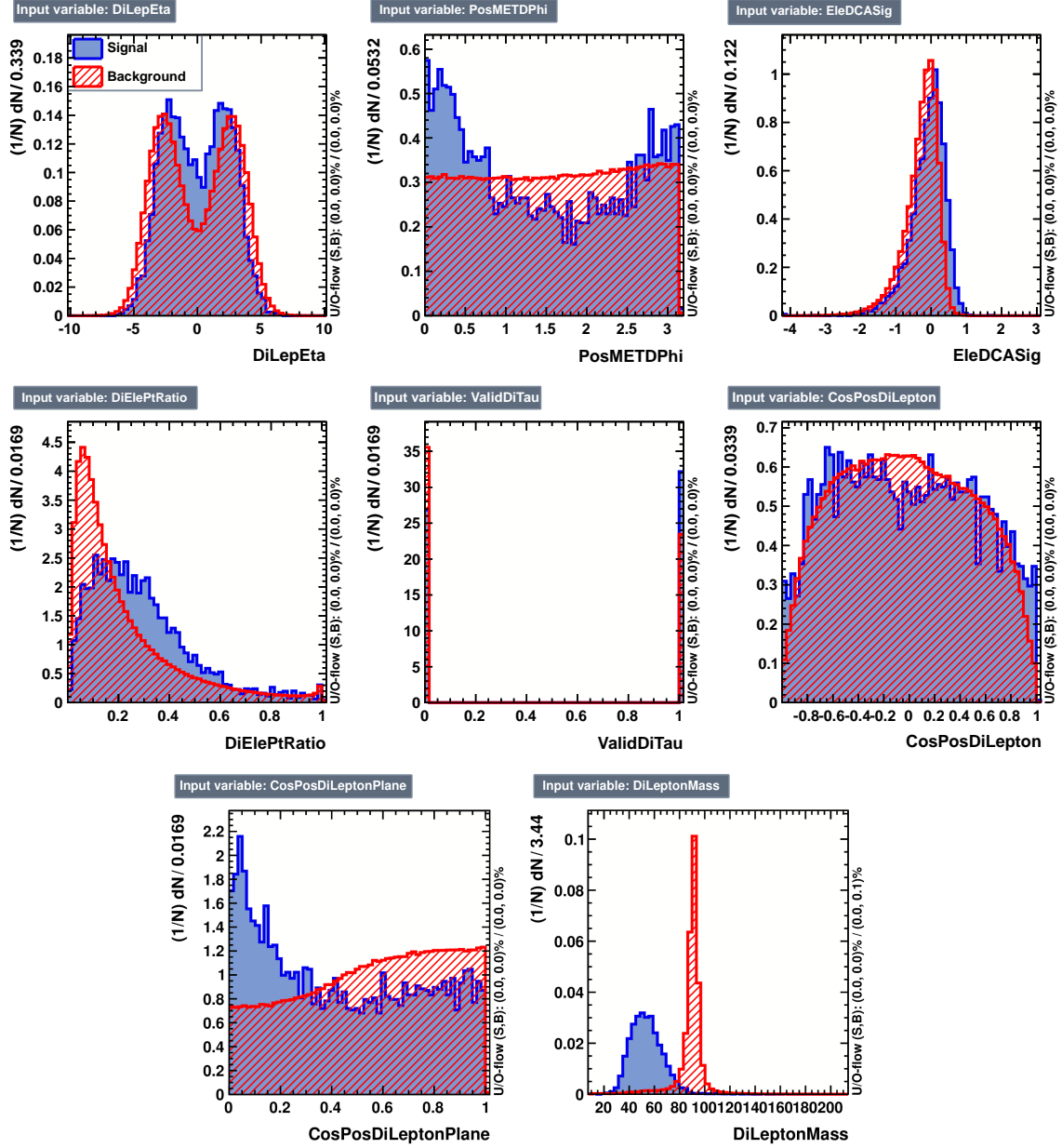
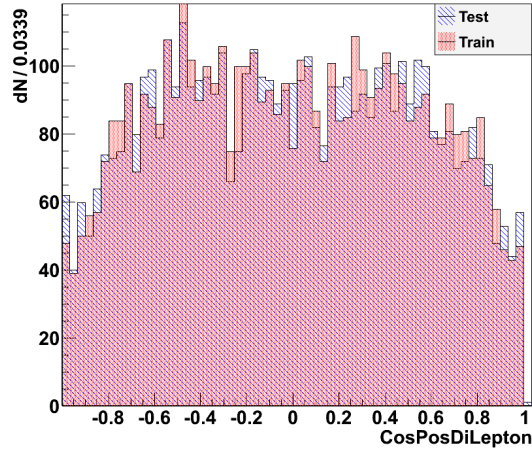
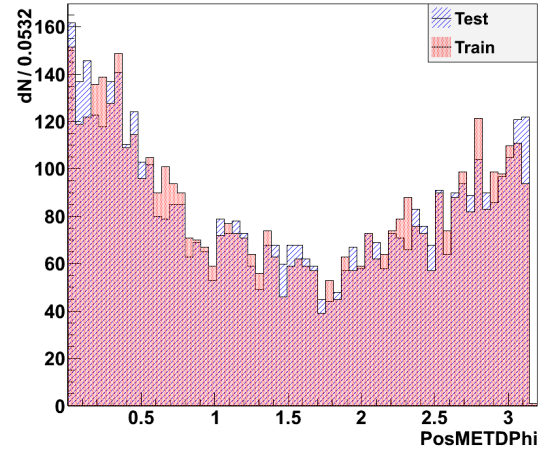


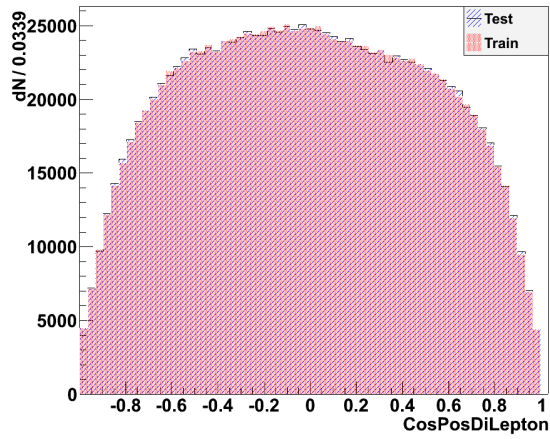
Figure 11: Distributions of input variables for signal (blue) and background (red) including DiLeptonMass , which is usually used only as a spectator variable. Normalizations are arbitrarily chosen.



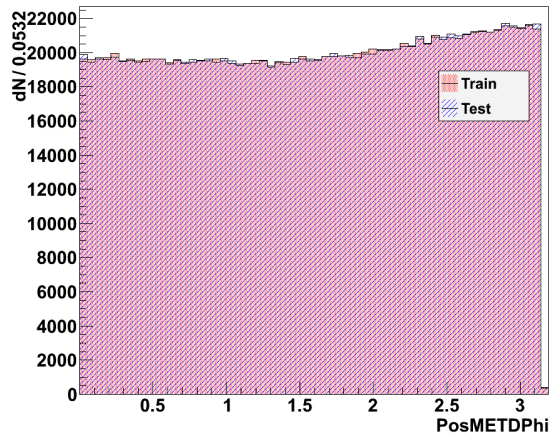
(a) Randomised Test and Train distributions of CosPosDiLepton for signal events.



(b) Randomised Test and Train distributions of PosMETDPhi for signal events.

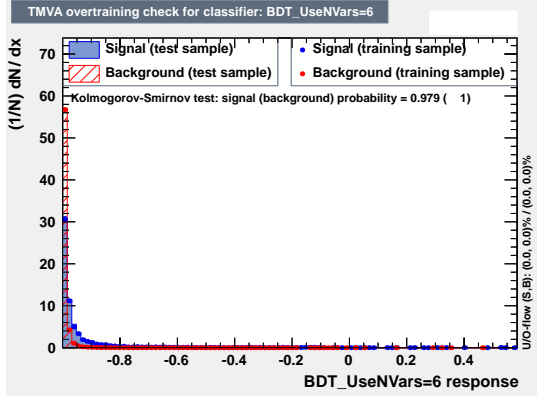


(c) Randomised Test and Train distributions of CosPosDiLepton for background events.

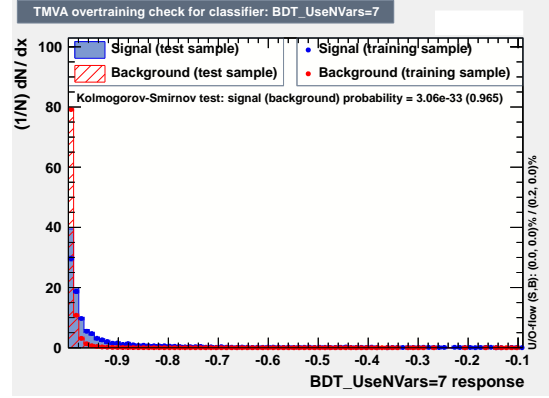


(d) Randomised Test and Train distributions of PosMETDPhi for background events.

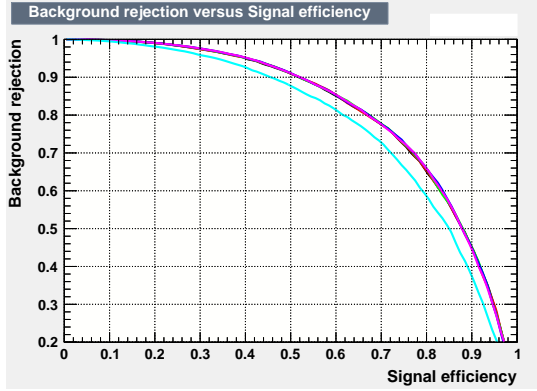
Figure 12: Difference of test and train variable distributions after randomization of the entire sample for the variables CosPosDiLepton and PosMETDPhi for signal and background.



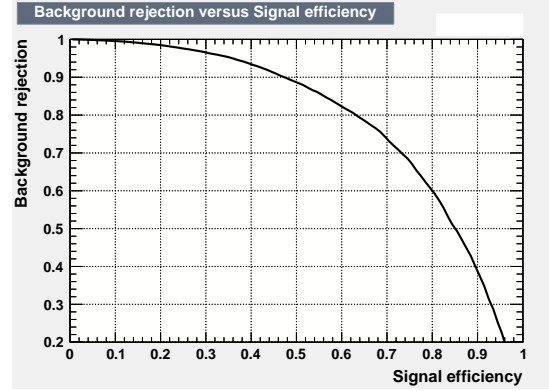
(a) BDT output distribution for booking options Grad, UseRandomisedTrees=True, UseNVars=6 and with SplitSeed=0. The behaviour is very similar for the same booking configuration, but with UseNVars=2, ..., 5



(b) BDT output distribution for booking options Grad, UseRandomisedTrees=True, UseNVars=7 and with SplitSeed=0. The behaviour is very similar for the same booking configuration, but with SplitSeed=100 and UseNVars=2, ..., 7



(c) ROC-Curve for booking options Grad, UseRandomisedTrees=True and with SplitSeed=0. The light blue curve corresponds to UseNVars=7 while the others correspond to values UseNVars=2, ..., 6.

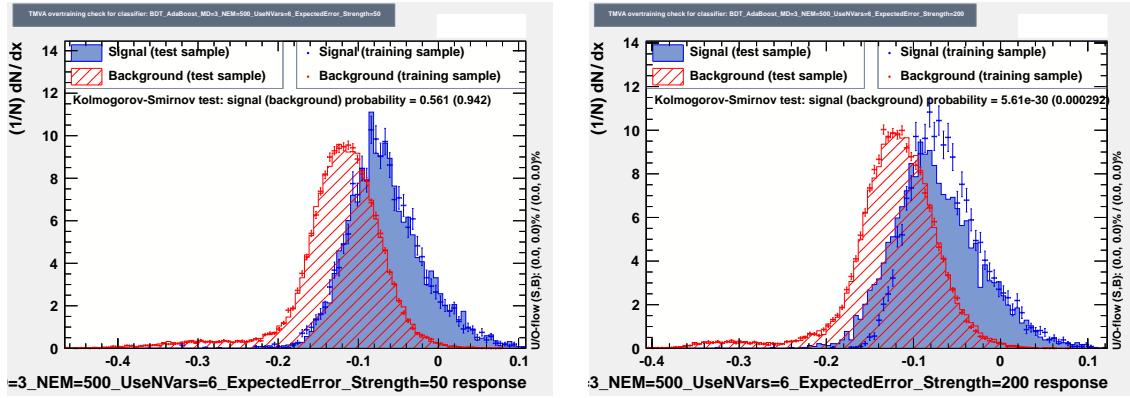


(d) ROC-Curve for booking options Grad, UseRandomisedTrees=True and with SplitSeed=100 for UseNVars=2, ..., 7

Figure 13: BDT output distribution and ROC-curves for UseRandomisedTrees=True indicating the strong sensitivity that comes along with this option.

	SplitSeed=0		SplitSeed=100	
UseNVars	$S/\sqrt{S+B}$	Int ROC	$S/\sqrt{S+B}$	IntROC
2	9.00422	0.812	7.21696	0.787
3	9.14339	0.813	7.21696	0.787
4	9.424	0.813	7.17362	0.786
5	9.57263	0.814	7.15674	0.786
6	9.62671	0.813	7.10917	0.785
7	6.72443	0.779	7.04257	0.785

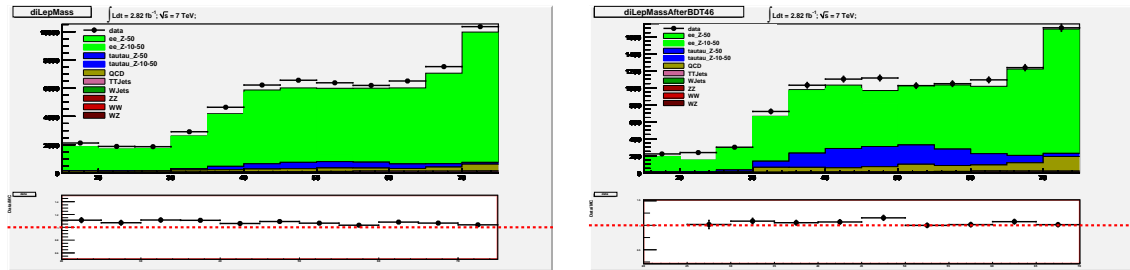
Table 3: Difference in ROC-curve integral and significance for the booking configurations of fig. 13.



(a) BDT output distribution for NTrees=2000, AdaBoost, UseRandomisedTrees=True, UseNVars=6, PruneMethod=ExpectedError, MaxDepth=3, nEventsMin=500, SplitSeed=100 and PruneStrength=50

(b) BDT output distribution for NTrees=2000, AdaBoost, UseRandomisedTrees=True, UseNVars=6, PruneMethod=ExpectedError, MaxDepth=3, nEventsMin=500, SplitSeed=100 and PruneStrength=200

Figure 14: Two BDT output distributions for ExpectedError-pruning.



(a) Distribution of DiLeptonMass before applying the optimal cut value corresponding to the final BDT training of the $Z \rightarrow \tau\tau \rightarrow ee$ training.

(b) Distribution of DiLeptonMass after applying the optimal cut value corresponding to the final BDT training of the $Z \rightarrow \tau\tau \rightarrow ee$ training.

Figure 15: Distribution of DiLeptonMass before and after applying the BDT cut of the final training configuration in fig. 6.