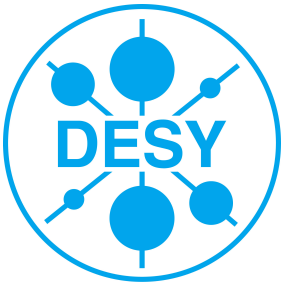


A study of several multivariate analysis techniques in the context of a search for $t\bar{t}H$ production and new physics at the CMS experiment

Maximilian Welsch

Karlsruhe Institute of Technology

Germany

September 6, 2017

Abstract

In this study different MVA techniques are applied to the separation of $t\bar{t}H(b\bar{b})$ signal and $t\bar{t}$ background. First a short introduction to the $t\bar{t}H(b\bar{b})$ analysis and different machine learning algorithms is given. Then some new features of the TMVA package are tested. After that neural networks are used to separate between the signal and background processes. There several sophisticated optimization libraries are tested. In the end new input variables for the boosted decision trees are being explored.

Contents

1 Theory

The following sections gives an introduction to topics dealt with in this study. First there is an short overview of the $t\bar{t}H(b\bar{b})$ analysis. After that, the necessary machine learning techniques are explained in order to understand the studies.

1.1 $t\bar{t}H(b\bar{b})$ Analysis

The $t\bar{t}H$ production offers an a direct access to the coupling of the Higgs boson to a Top quark. This process is a test for the theory of the fermion masses in the standard model. An observation would further confirm the Yukawa coupling of the Higgs boson to fermions. In this subsection, the relevant signal and background processes and the general analysis strategy in a search for $t\bar{t}H$ production, with a focus on for this study relevant topic, are explicated.

In this study, events will be classified, where a Higgs boson is produced in association with a Top quark pair. Top quarks almost exclusively decay into Bottom quarks and W bosons.Therefor, the decay of the $t\bar{t}$ pair can be characterized by the decay of the W bosons. If both W bosons decay into quarks, the decay is called hadronic. In the case one W boson decays into quarks while the other W boson decays into a lepton neutrino pair, the decay is called semi leptonic. The remaining case, where both W boson decay into leptons and neutrinos is called dileptonic. The focus in this study is on the dilepton channel of the $t\bar{t}H$ production. Furthermore only the decay of the Higgs boson into a Bottom quark pair $H \rightarrow b\bar{b}$ is being considered. This decay channel distinguishes itself with the largest branching ratio of all Higgs decays. In leading order, the signal final state on parton level consist of four Bottom quarks und to oppositely charged leptons and two neutrinos. A Feynman diagram is shown in Fig. 1a. The leptons allow to distinguish the signal from QCD background processes containing only jets. The neutrino is only weakly interacting. For this reason it cannot be observed in the detector. Its presence is characterized by missing transversal energy. The quarks in the final state hadronize and can be observed as jets in the detector. Jets originating from Bottom quarks can be identified by using b tagging algorithms.

There are several background processes resulting in a similar final state to the $t\bar{t}H(b\bar{b})$ process. A background reduction is possible by placing cuts on the number of jets and b tags. In this study, the background considered is the production of Top quark pairs with additional jets in the dilepton channel. Additional jets can result from gluon radiation. Because of the identical final state as the signal, the production of a Top quark pair in association with a Bottom quark pair results in a irreducible component of the background (see Fig. 1b). A separation between $t\bar{t}H(b\bar{b})$ and $t\bar{t}$ +$b\bar{b}$ events is only possible by comparing e.g. kinematic quantities. Also, the $t\bar{t}$ +$b\bar{b}$ cross section is about eight times larger than the signal cross section (NLO, $\sqrt{s} = 14$TeV). Further, the production of a $t\bar{t}$ pair with additional jets is another background which cannot be neglected, because jets could be falsely identified as b jets. The $t\bar{t}$ cross section is about three

magnitudes larger than the $t\bar{t}H(b\bar{b})$ cross section.

The search for $t\bar{t}H(b\bar{b})$ events at the CMS experiment is challenging because of the small signal cross section and the kinematically very similar background. Moreover it is difficult to find event variables with enough separation power allowing to distinguish between signal and background. Therefore multivariate analysis (MVA) techniques like boosted decision trees (BDT) or neural networks (NN) are used in order to combine weakly separating variables into a more powerful discriminant.

The first step in the $t\bar{t}H(b\bar{b})$ analysis is the reconstruction and selection of events. Details to this step can be found in the actual $t\bar{t}H(b\bar{b})$ analysis [1]. Then events are grouped into mutually exclusive category characterized by the number of jets and b tags. In each category MVA methods are used to create a powerful discriminant to separate signal from background. Besides the machine learning techniques, also physics motivated likelihood method like the matrix element method is deployed. In order to calculate a limit on the signal strength, a maximum Likelihood fit is performed simultaneously in all category. For more details on the matrix element method and the statistical tools used in this analysis have a look in the analysis itself [1].

1.2 Machine Learning Techniques

Separating signal and background events is a binary classification problem. Machine learning algorithms can be used to classify new events. Events are characterized by some event variables $\mathbf{x}$ and a true class y. In the following the two machine learning algorithms used in this study will be explained. Also receiver operator curves (ROC) are introduced in order to measure the performance a classifier.

1.2.1 Boosted Decision Trees

A single decision tree (DT) is shown in Fig. 2. The functionality of a DT is straightforward. An event is either classified as signal or background by either passing or not passing a cut on a specific node until a decision is made. In order to determine these cuts, the decision tree is grown starting from the root node. Here, the training dataset is recursively split into further subsets, which become more signal or background like. At each node, the cut on a single variable is determined by the cut value which optimizes the information gain $\Delta I = I(t_p) - p_{t_l} I(t_l) - p_{t_r} I(t_r)$ between the parent node and the two daughter nodes. p_t is the probability of an event to either end up in the left or right daughter node. The information $I(t)$ of a node can be quantified e.g. by the Gini index $G = 1 - p_s^2 - p_b^2$, where $p_s = s/(s+b)$ is the signal and $p_b = 1 - p_s$ the background fraction in each node. The splitting continues until a stopping criteria e.g. the minimum number of events or maximum depth of the tree is reached. In the final nodes, the classification is based by the majority of the class in these nodes.

Unfortunately a single DT is a very weak classifier and it is also prone to statistical fluctuations in the dataset. It is better to us a "forest" of decision trees. DT are trained iteratively with reweighted events. Using the so called AdaBoost algorithm, events are reweighted based on the missclassification rate err in the previous treed. Missclassified events are then reweighted according to $w \to w \cdot \alpha$ with $\alpha = (1 - err)/err$. The final classifier then is taken as a weighted average over all DT according to

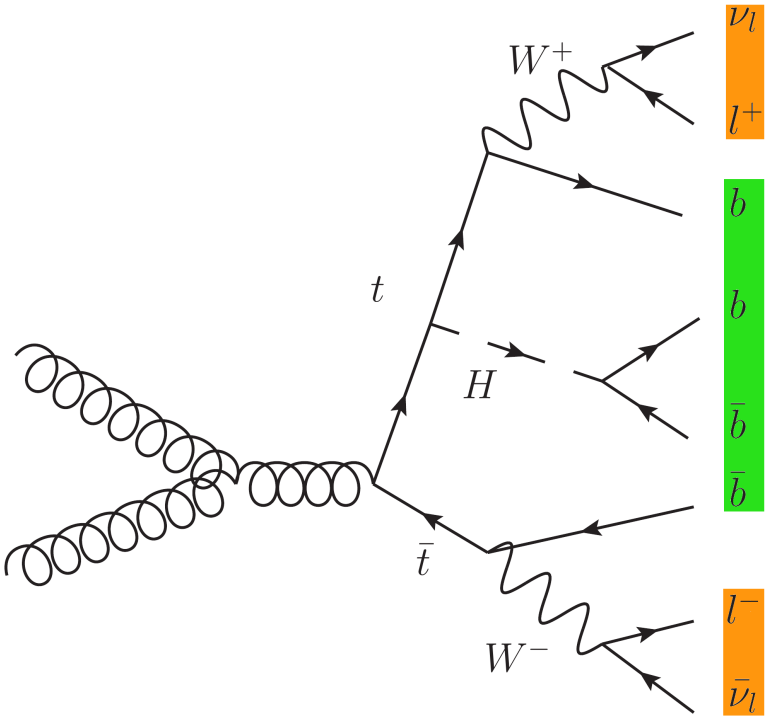
$$\hat{y}(\mathbf{x}) = (1/N_{\text{Trees}}) \cdot \sum_{k=1}^{N_{\text{Trees}}} \ln(\alpha_k) \cdot t_k(\mathbf{x}) \tag{1}$$

1.2.2 Neural Networks

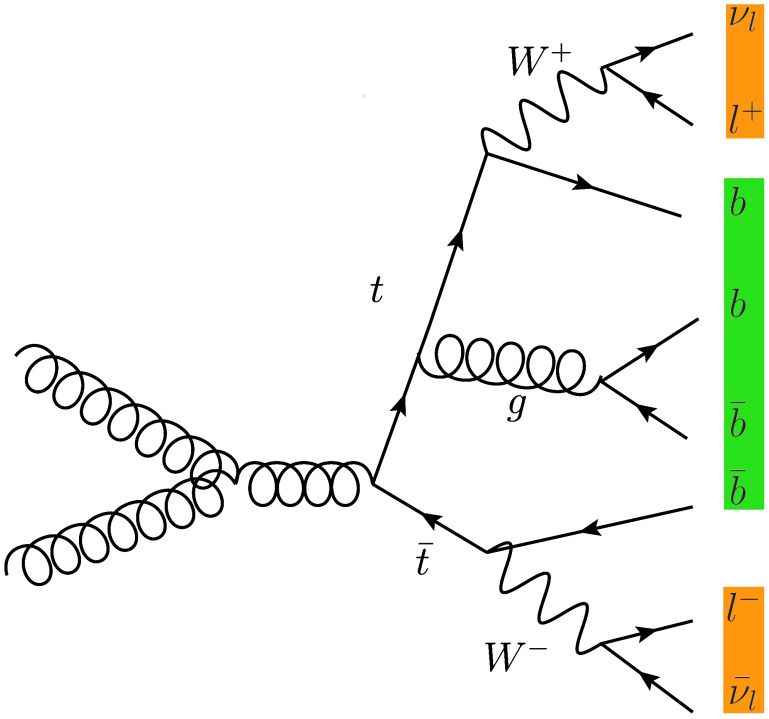
A neural network (NN) consists of so called neurons arranged in layers, where each neuron in a certain layer is connected with all neurons in the previous layer. The structure of a NN is shown in Fig. 3. The functionality of a neuron is explained in Fig. 4a. A neuron takes a input $\mathbf{x}$ and takes the weighted sum $z = \sum_i x_i w_i + b$, where the bias b is an extra parameter. Then a non linear activation function $g(z)$ is applied. Typical activations functions are shown in Fig. 4b. In the end the output is propagated further through the network. The training of a NN consists of minimizing a loss function $L(\hat{y}, y)$ with respect to the network weights $\mathbf{w}$ and $\mathbf{b}$.

1.2.3 Receiver Operator Curves

To measure the performance of a binary classifier the integral of the receiver operator curve (ROC) is used. The ROC is calculated by varying cuts on the MVA output where the signal efficiency and the background rejections is calculated. A ROC is shown in Fig. 5. A perfect classifier would have an ROC integeral of 1 whereas random guessing would result in a ROC integral of 0.5.



(a)



(b)

Figure 1: **Leading order Feynman diagrams of the $t\bar{t}H$ process (a), and the $t\bar{t}$ +$b\bar{b}$ process (b).**

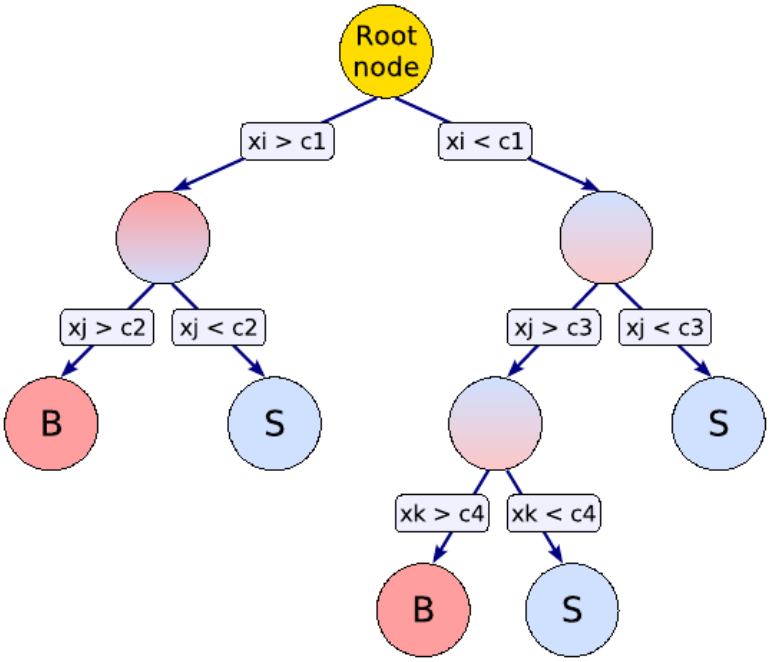


Figure 2: **A single binary decision tree.**

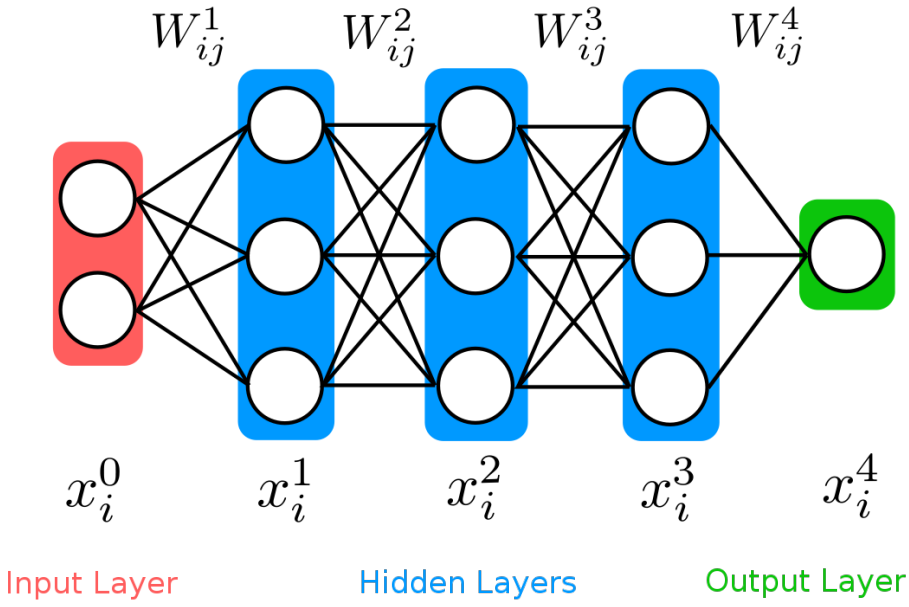
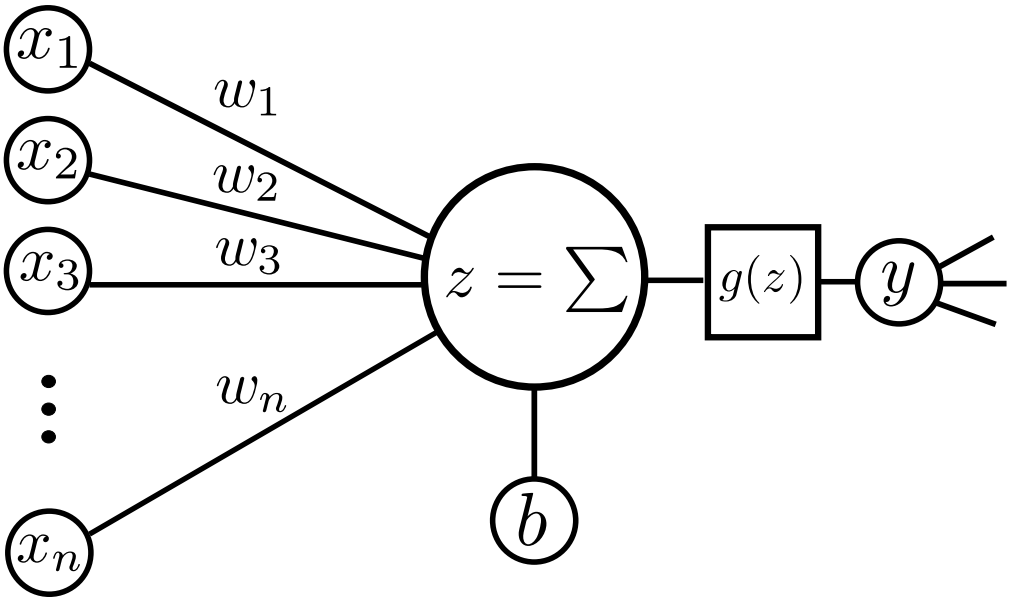
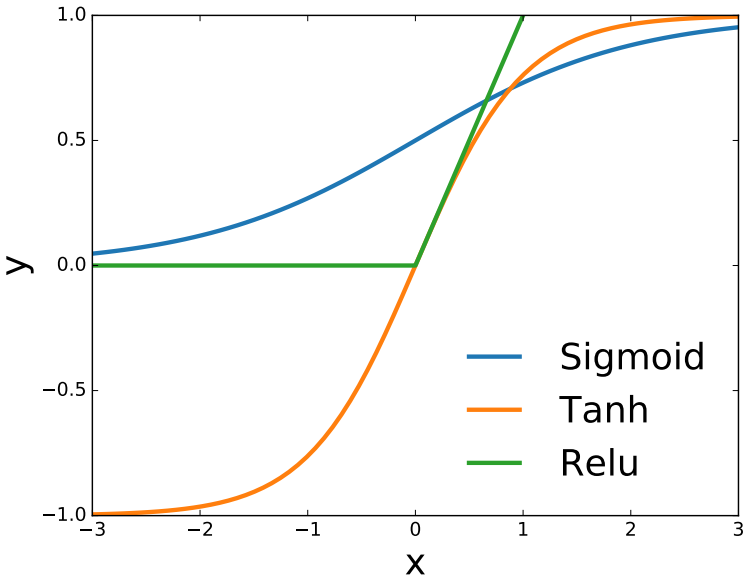


Figure 3: **Structure of a neural network.**

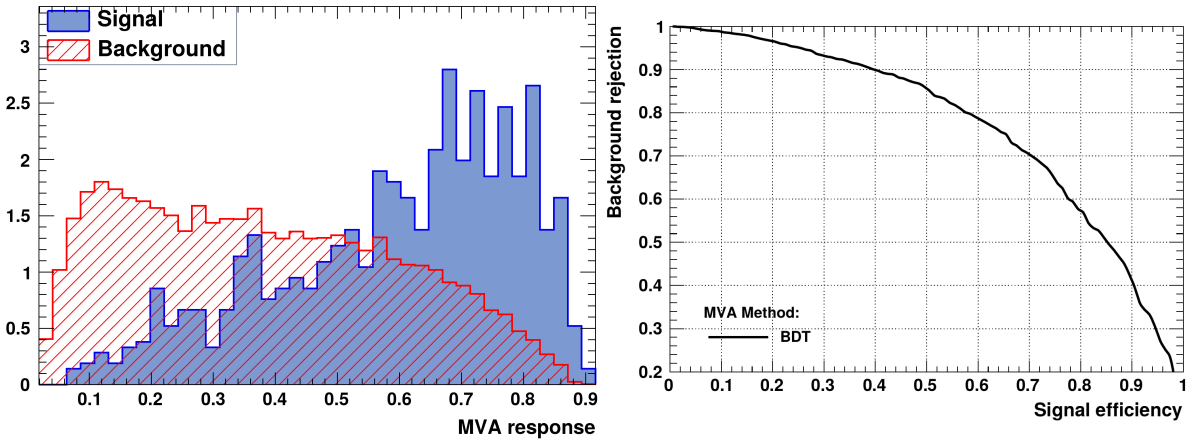


(a)



(b)

Figure 4: **The functionality of a single neuron (a) and typical activation functions (b)**



(a) (b)

Figure 5: **A typical MVA output (a) and the resulting ROC (b).**

2 New TMVA features

This section describes two new features of the TMVA package [2]. First a method called cross-validation [3] for evaluating model robustness will be discussed. After that the functionality for optimizing model hyperparameters [4] will be explained.

2.1 Cross-Validation

For training and testing a MVA method the dataset is usually split into two independent datasets. One is used for training the MVA method while the other one is used for evaluating the performance of the trained model. Using this method, only half of the available data is used for training. This might be sufficient if the size of the whole dataset is large enough, but with smaller datasets it might be better to whole dataset for training. The problem is, that it is also necessary have method to measure the model's performance and that there is a check for overtraining.

A possible way out of this problem is using a k-fold cross-validation (CV) to evaluate the model's performance. Here the dataset is split into k randomly sampled subsets (see fig. 6), also called folds. Then the MVA method is trained with $k-1$ folds and tested with the remaining fold. This procedure is repeated k times, until all folds have been used for testing. In the end the overall performance of the classifier can be estimated by the mean of all achieved ROC integrals on different folds. The advantage of CV is, that the whole dataset can be used for testing and training of a MVA method. Furthermore the robustness of a model can be estimated. If the ROC integrals, evaluated on the different folds, show a large variance, the model with the chosen hyperparameters does not perform very stable and maybe a simpler model should be chosen.

There is a new class for performing CV on a given MVA method implemented in TMVA. As an example, the CV has been used in the ≥ 4 jets, ≥ 4 tags category. Table 1 and Fig. 7 summarize the result of the CV. The CV results in a mean ROC integral of 0.742 ± 0.022. This is the same value, compared to the ROC integral of 0.742 achieved on the test dataset, while using the other half of the data for training. But additionally there is also information about the variance of the ROC integral.

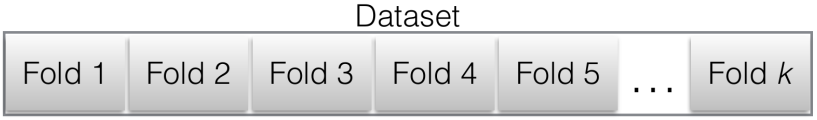


Figure 6: **Diagram of dataset split into k subsets (folds).** Taken from [3]

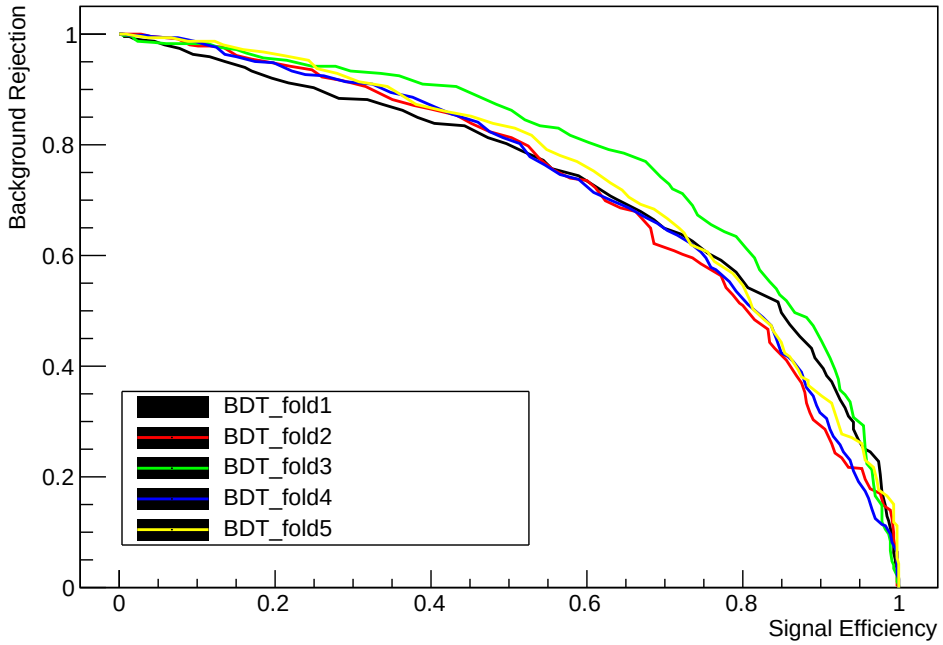


Figure 7: **Result of the CV of a BDT used to separate $t\bar{t}H$ signal from $t\bar{t}$ Background in the ≥ 4 jets, ≥ 4 tag category.**

2.2 Hyperparameter Optimization

Before training a MVA method, the right set of hyperparameters, e.g. the number of trees or the maximum depth of a BDT, have to chosen. It is not trivial, to find the "right" set of parameters that give the best classifier. One simple way to optimize the hyperparameters is a manual grid-search, where the parameter space is scanned manually. One disadvantage of this method is that the number of different classifiers, which have to be trained and evaluated, grows really fast with the number of parameters to optimize.

There is a new class for hyperparameter optimization implemented in the TMVA package. Unfortunately there is no documentation available for this process. In addition, it seems like that only the AdaBoost option for the BDTs in TMVA can be optimized. Therefore this class is only compared to the BDT used in the $= 3$ jets, $= 3$ tags category, where a AdaBoost BDT has been optimized with the particle swarm optimization [5]. Some information, about how the class is used, can be found here [4]. The resulting ROCs calculated from the BDTs outputs on the test dataset and the corresponding ROC integrals for three hyperparameter suggestions are shown in Fig. 8 and Tab. 2. In addition the results for a BDT with default parameters and the BDT optimized with the PSO are shown also. All BDTs have been trained with the same input variables selected by the PSO. The optimization provided by TMVA can give as good results as

Table 1: **Result of the CV of a BDT used to separate $t\bar{t}H$ signal from $t\bar{t}$ Background in the ≥ 4 jets, ≥ 4 tag category.**

Evaluated on	ROC Integral
Fold 1	0.734
Fold 2	0.724
Fold 3	0.779
Fold 4	0.728
Fold 5	0.746
Average	0.742
Std-Dev	0.022

the PSO.

If the development of the TMVA hyperparameter optimization will be continued in the future so that different MVA method options could be chosen in advance, like also gradient boosting instead of only AdaBoost, this class would provide a good starting point when optimizing MVA hyperparameters. Because of the lack of documentation, it is not really clear how to optimization procedure is carried out.

2.3 PyMVA Interface

PyMVA provides a interface between TMVA and the Python machine learning library scikit-learn [6]. This section compares the GradientBoostingClassifier and AdaBoostClassifier from scikit-learn with their equivalent MVA methods in TMVA.

For the comparison, a BDT with gradient boosting from TMVA and the GradientboostingClassifier from scikit-learn are trained in the ≥ 4 jets, ≥ 4 tags category. Additionally a BDT with AdaBoost from TMVA and the AdaBoostclassfier from scikit-learn are

Table 2: **Resulting ROC integral for different parameter sets suggested from the TMVA hyperparameter optimization compared to the particle swarm optimization and the default values in the $= 3$ jets, $= 3$ tags category.**

Parameter Sets	ROC Integral
PSO	0.760
THPO_2	0.759
THPO_3	0.759
THPO_1	0.750
DEFAULT	0.736

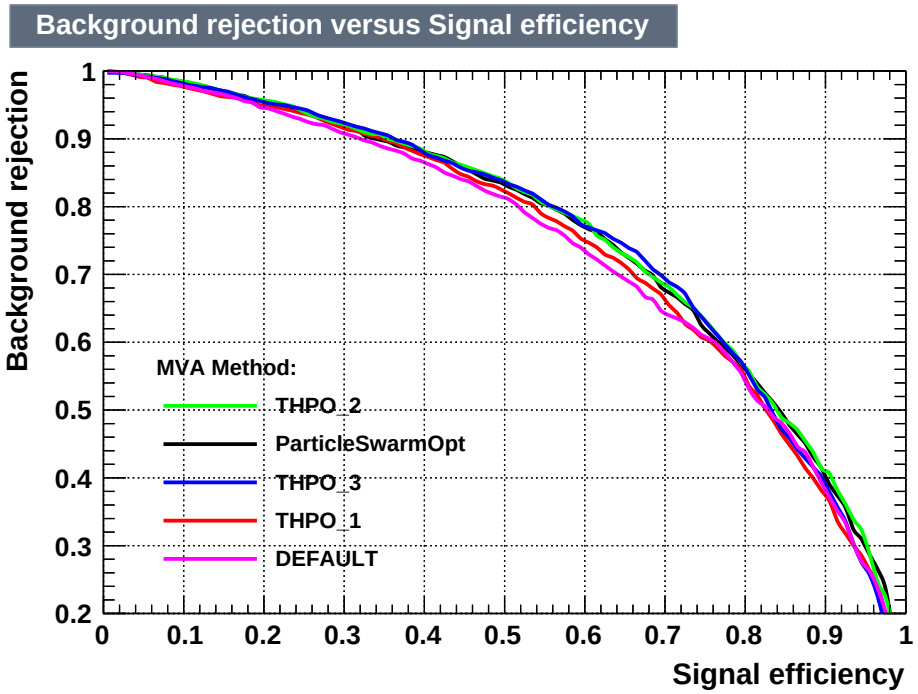


Figure 8: **Different BDT configurations resulting from the TMVA hyperparameter optimization compared to the default parameters and the result from the standalone particle swarm optimization.** Results are shown for the = 3 jets, = 3 tags category. The particle swarm optimization also selects the input variables.

trained in the = 3 jets, = 3 tags category. The same input variables and hyperparameters, selected by the PSO in the corresponding category, have been used.
The ROC and MVA output distributions in the $\geq$ 4 jets, $\geq$ 4 tags category are shown in Fig. 9 and 10. Both curves have an integral of 0.742. Moreover, the shape of the two MVA output distributions are really similar. There are also no significant differences in training time and evaluation time for the TMVA gradient BDT and the scikit-learn GradientBoostingClassifier.
The ROC and MVA ouput distributions in the = 3 jets, = 3 tags category are shown in Fig. 11 and 12. There are significant differences between the TMVA AdaBoost BDT and the scikit-learn AdaBoostClassifier both in the ROC and the mva output distributions. The TMVA AdaBoost BDT scores a ROC integral of 0.760 on the test dataset, whereas the scikit-learn AdaBoostClassifier scores a ROC integral of 0.720. Also the distribution of the scikit-learn AdaBoostClassifier peaks around 0.5 and a small width and an additional accumulation around 0.3. The output distribution of the TMVA AdaBoost BDT shows an expected behaviour with no conspicuous features. There are no differences in the training time of both MVA methods, but when evaluating, the scikit-learn AdaBoostClassifier needs about 100 times longer compared to the TMVA AdaBoost BDT.

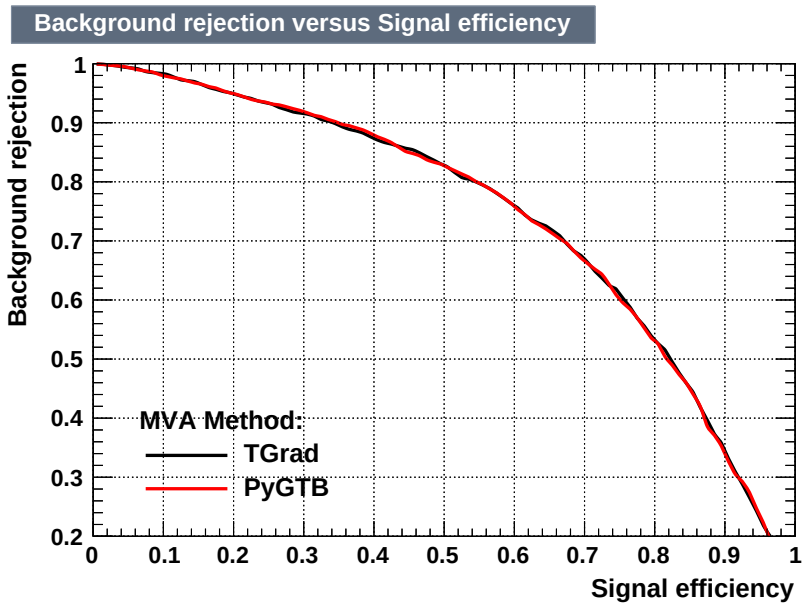
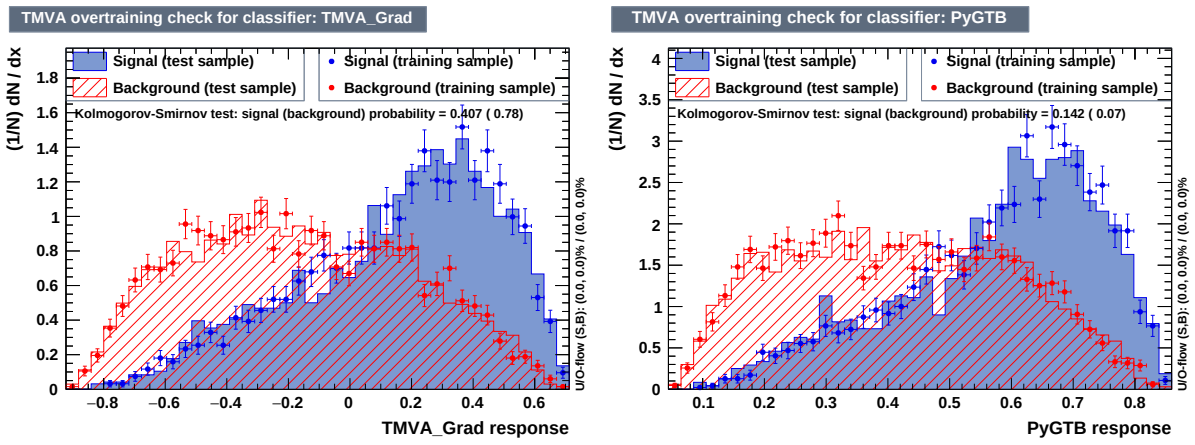


Figure 9: **ROC of the TMVA gradient BDT and the scikit-learn GradientBoostingClassifier in the ≥ 4 jets, ≥ 4 tags category.** Evaluated on the test dataset.

In conclusion the TMVA gradient BDT and the scikit-learn GradientBoostingClassifier seem to perform equivalently. Therefore both methods could be are interchangeable and one could use the extra functionality around scikit-learn to tune parameters for the GradientBoostingClassifier and then use it later in TMVA.



(a) (b)

Figure 10: **MVA output of the TMVA gradient BDT and the scikit-learn GradientBoostingClassifier in the ≥ 4 jets, ≥ 4 tags category.**

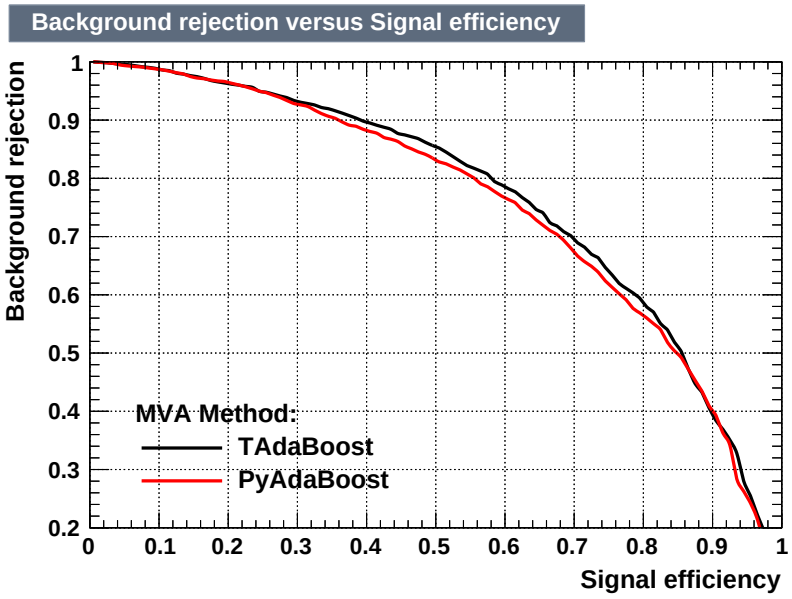
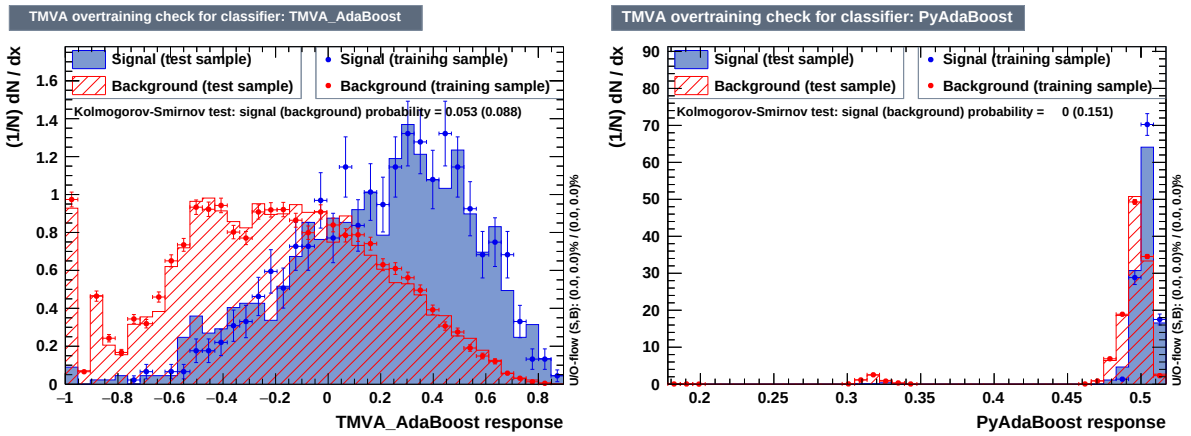


Figure 11: **ROC of the TMVA AdaBoost BDT and the scikit-learn AdaBoostClassifier in the = 3 jets, = 3 tags category.** Evaluated on the test dataset.



(a) (b)

Figure 12: **MVA output of the TMVA AdaBoost BDT and the scikit-learn AdaBoostClassifier in the $= 3$ jets, $= 3$ tags category.**

3 Neural Network Hyperparameter Optimization

The first part of this section gives a short overview over the new interface between TMVA and Keras. The rest of this section deals with the optimization of the hyperparameters of the neural networks (NN) for the separation of $t\bar{t}H$ signal and $t\bar{t}$ background in the dilepton channel.

3.1 PyKeras

With PyKeras it is possible to use the Keras [7] interface for the deep learning library TensorFlow [8]. In order to train neural networks with TensorFlow in TMVA, a neural network model has to be defined with Keras in advance. The training and testing can be perfomed in TMVA like with any other MVA method. Using PyKeras has the advantage, that one could easily try out new techniques for training neural networks and make use of several python libraries around Keras for tuning model parameters. Additionally the training of a neural network can be performed with both CPU and GPU (if available), which can significantly accelerate the training process

3.2 Hyperparameter Optimization

Like other MVA methods, a NN also has several hyperparameters, e.g.

- network architecture:
  - number of hidden layers
  - number of neurons per layer
  - type of the activation function
  - type of the weight initialization
- regularization parameters:
  - weight decay strength
  - dropout rate
- training parameters:
  - learning rate
  - batch size
  - number of training epochs

Concerning the number of hyperparameters (HP), it can be difficult find the optimal set of hyperparameters for a given classification problem. In order to determine an initial set of HP, one could make an educated guess or perform a grid search. The parameters optimized in this process are:

- number of hidden layers

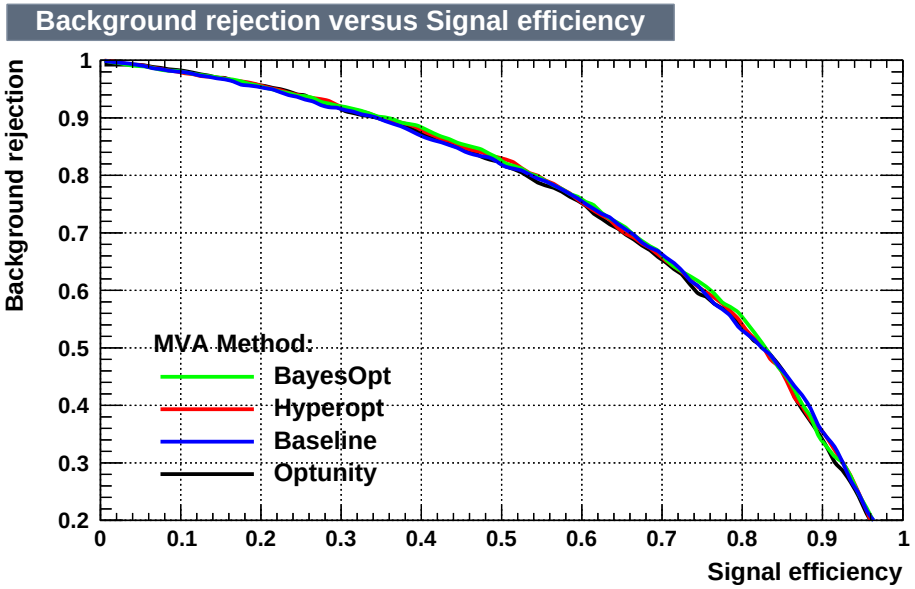
Table 3: **Resulting HP set in the ≥ 4 jets, ≥ 4 tags category**

	layers	neurons	learning rate	dropout rate	L2 norm	ROC integral
Baseline	1	300	1.0e-3	0.0	1.0e-3	0.727
BayesOpt	1	320	3.8e-3	0.23	2.3e-3	0.722
Hyperopt	2	70	7.4e-3	0.10	2.7e-3	0.735
Optunity	6	210	2.4e-3	0.07	4.0e-3	0.724

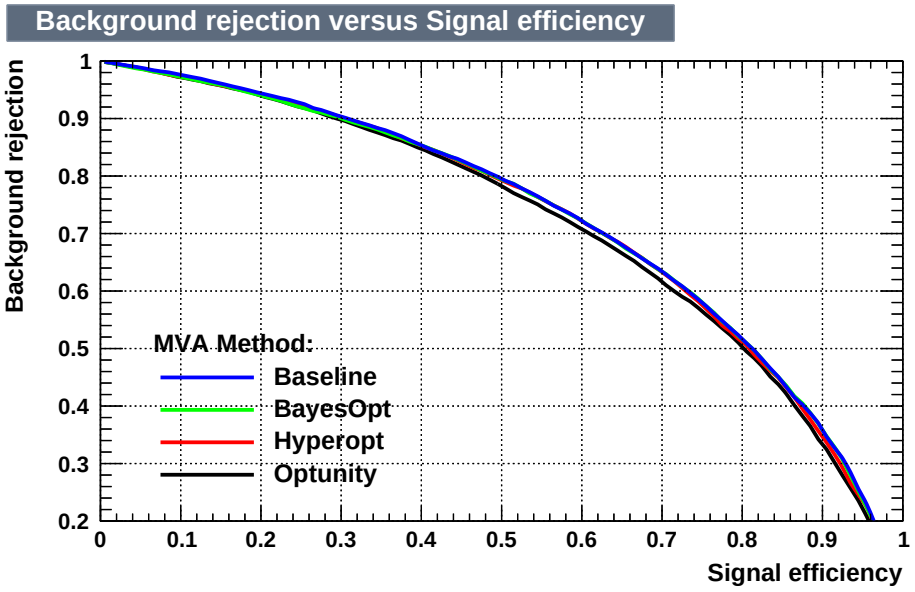
- number of neurons per layer
- dropout rate
- weight decay strength
- learning rate

The Relu function ($g(z) = \max\{0, z\}$) is chosen as activation function for all neurons in the hidden layers. AdamOptimizer is used to minimize the loss function. The training was performed for 50 epochs with a batch size of 128. A grid search is basically a scan of a given parameter space. Hence the number of NN with different HP to train grows exponentially with the number of parameters, a grid search can be computationally expensive and may no be reasonable. For this reason more sophisticated optimization techniques like particle swarm optimization (Optunity [9]) or bayesian optimization techniques (Hyperopt [10], Bayesopt [11]) have been used also. The figure of merit for this optimization is a 3 fold cross validated ROC integral. The HP optimization has been performed for ≥ 4 jets, ≥ 4 tags and ≥ 4 jets, $= 3$ tags categories, the two most signal like categories, where the same input variables determined in the optimization process for the BDT have been used.

The resulting set of HP are summarized in Tab. 3 and 4. The corresponding ROC of these NN are show in Fig. 13. The NN with the resulting HP set barely perform better than the Baseline HP set determined in the grid search. Deep NN, with more than one hidden layer, show no significant improvement to shallow NN at least for the same input variables used for the BDTs. When evaluated on the test dataset, there is no improvement in the integral of the ROC. All NN perform basically the same as the equivalent BDT used until now.
The shape of the output for NN with several hidden layers also show considerable differences to the output shape of shallow NN. In Fig. 14d the output of the NN with the HP set determined by the particle swarm optimization is shown. This distribution has a single peak around 0.7 in contrast to the smoother output distributions from the other NN.



(a)



(b)

Figure 13: **ROC of NN with resulting HP from the optimization process.** Evaluated on the on the test dataset. (a) shows the ROC for the ≥ 4 jets, ≥ 4 tags, and (b) shows the ROC for the ≥ 4 jets, $= 3$ tags category.

Table 4: **Resulting HP set in the ≥ 4 jets, $= 3$ tags category**

	layers	neurons	learning rate	dropout rate	L2 norm	ROC integral
Baseline	1	300	1.0e-3	0.0	1.0e-3	0.727
BayesOpt	2	430	1.0e-2	0.10	0.1e-3	0.724
Hyperopt	3	994	8.4e-4	0.36	6.5e-3	0.717
Optunity	1	153	3.2e-3	0.31	0.5e-3	0.724

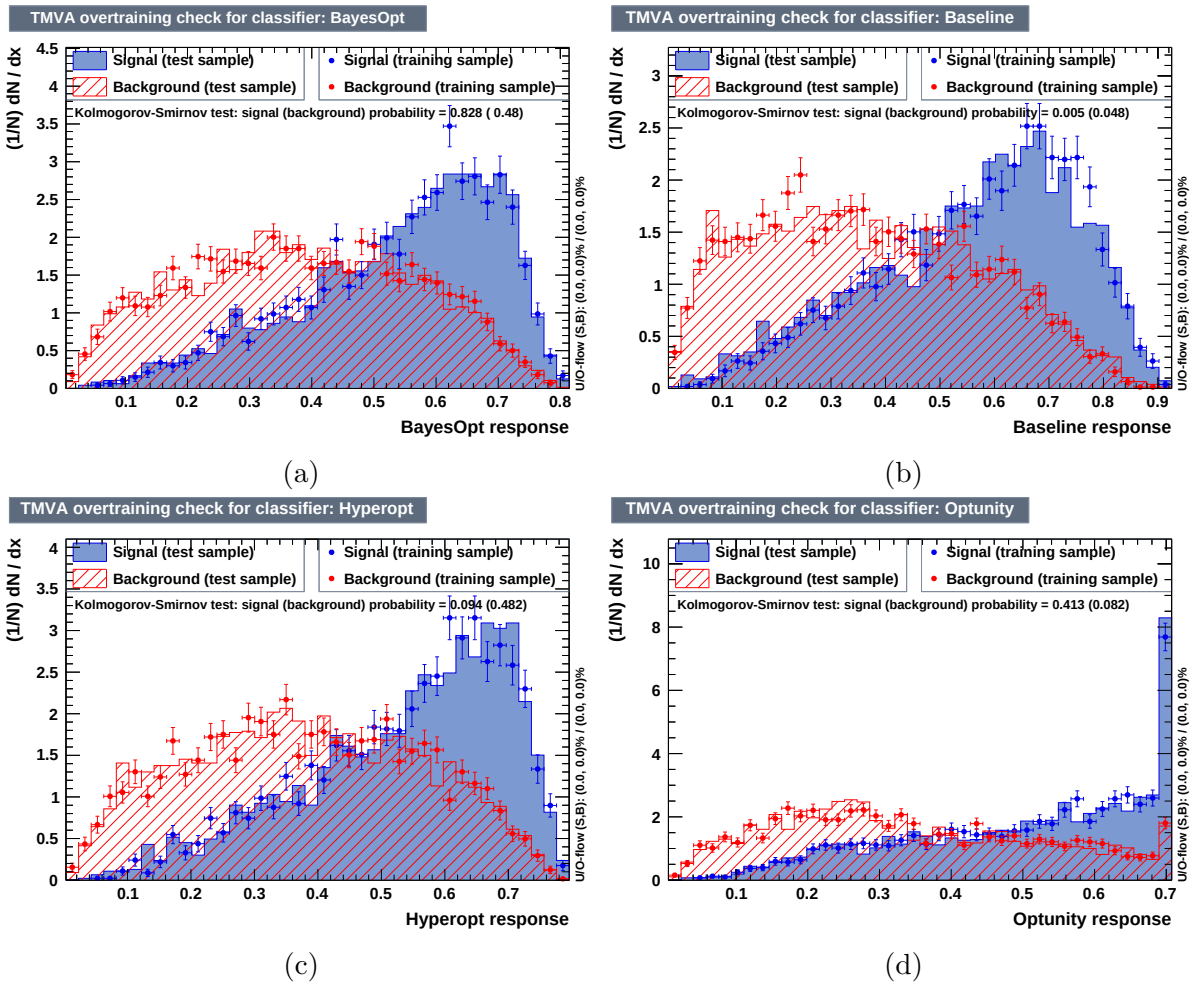


Figure 14: **NN output with resulting hyperparameters from the optimization process in the ≥ 4 jets, ≥ 4 tags category.**

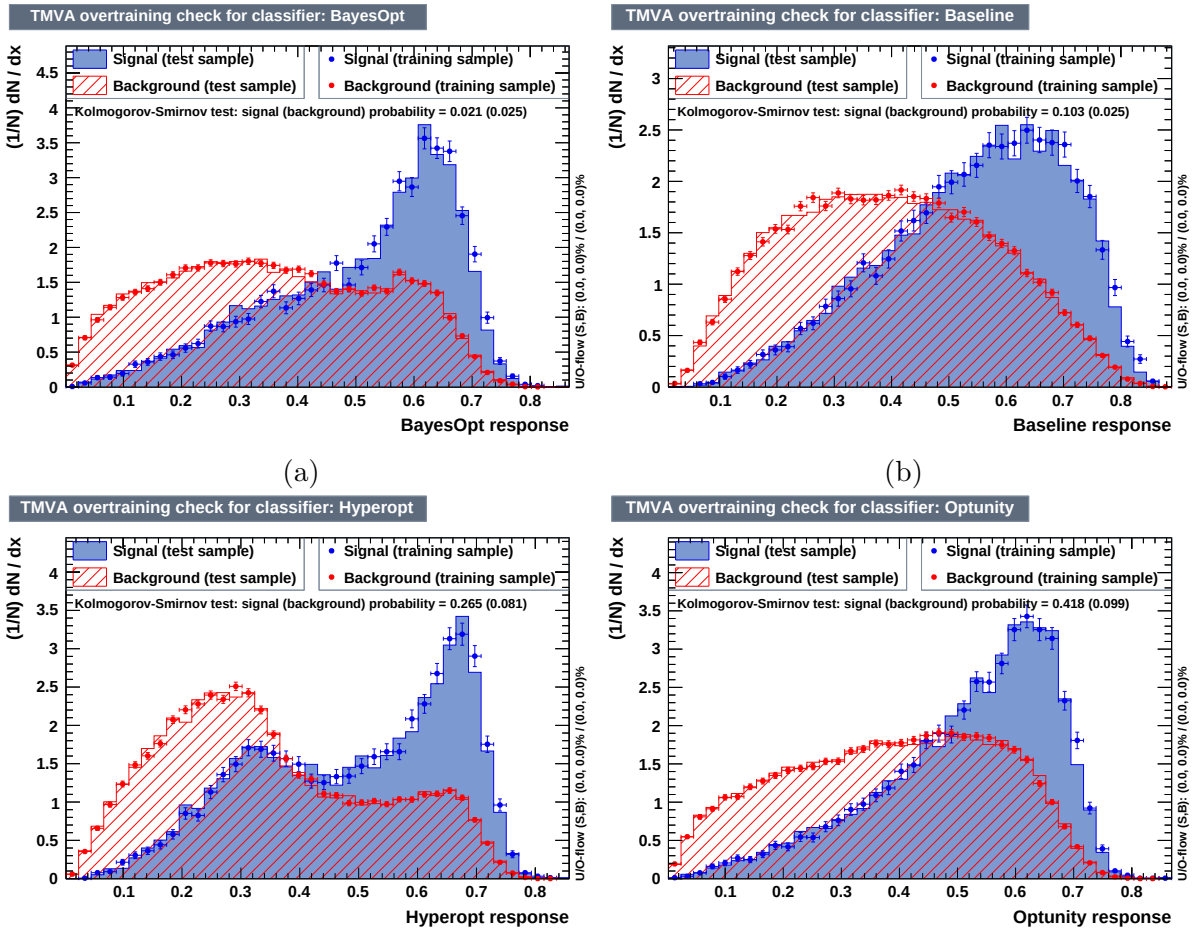


Figure 15: **NN output with resulting hyperparameters from the optimization process in the ≥ 4 jets, $= 3$ tags category.**

4 New boosted decision tree input variables

The classification performance with NN does not show any improvement compared to the BDT when the same input variables are used. This might hint that there is still physics information missing. Therefore new input variables for BDT are calculated. These new variables combine information of jets, which are close together in the η plane, because jets originating from the b quarks of the Higgs boson decay H $\rightarrow$ bb tend to be more collimated compared to the background. The new input variables are shown in Fig. 17. The correlation coefficients with the other BDT input variables in the ≥ 4 jets, ≥ 4 tags category are shown in Fig. 18-20.

The resulting ROC of the BDT trained with the new input variables are summarized in Fig. **??** and Tab. **??**. Only the mass of two tagged jets with minimum $\Delta\eta$ can improve the integral of the ROC. A training with the other variables result in no improvement regarding the ROC integral.

Table 5: **ROC integral achieved with the new BDT input variables.** Evaluated on the test dataset.

New Variable	ROC Integral
None	0.742
mass of 2 tagged jets with min DeltaEta	0.745
None	0.742
mass of 2 tagged jets with max DeltaEta	0.742
multiplicity of Dijets with DeltaEta < 1.5	0.741
All	0.745

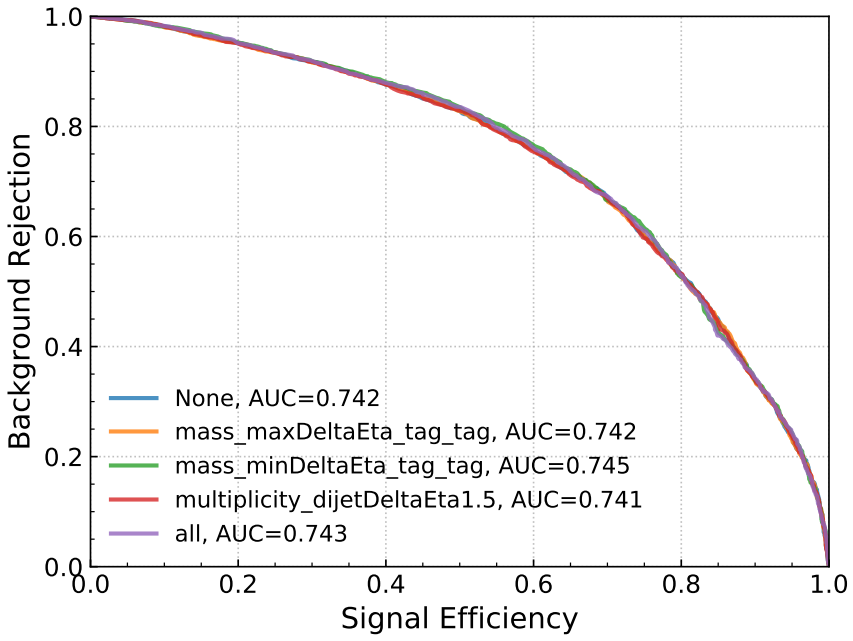
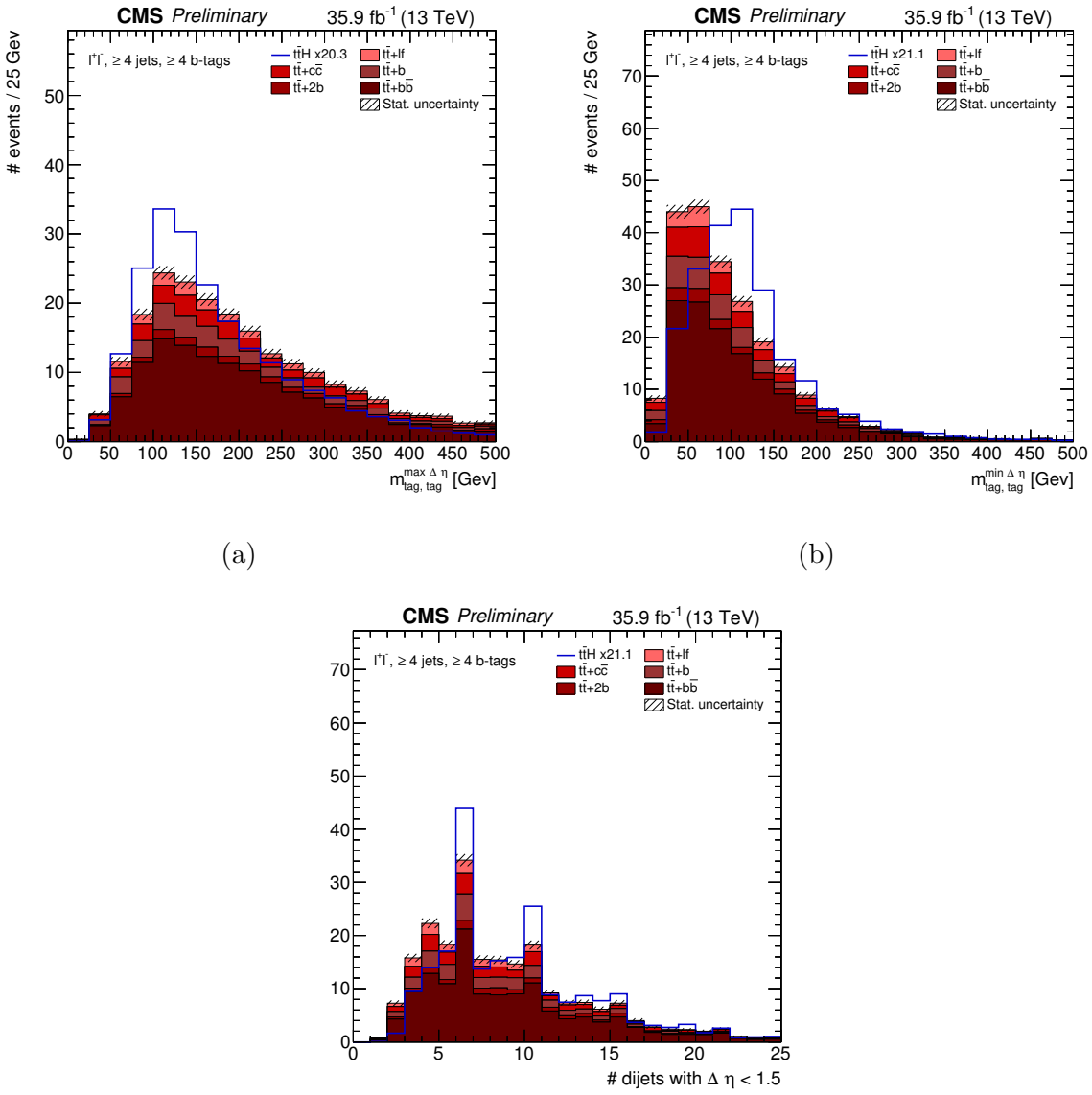


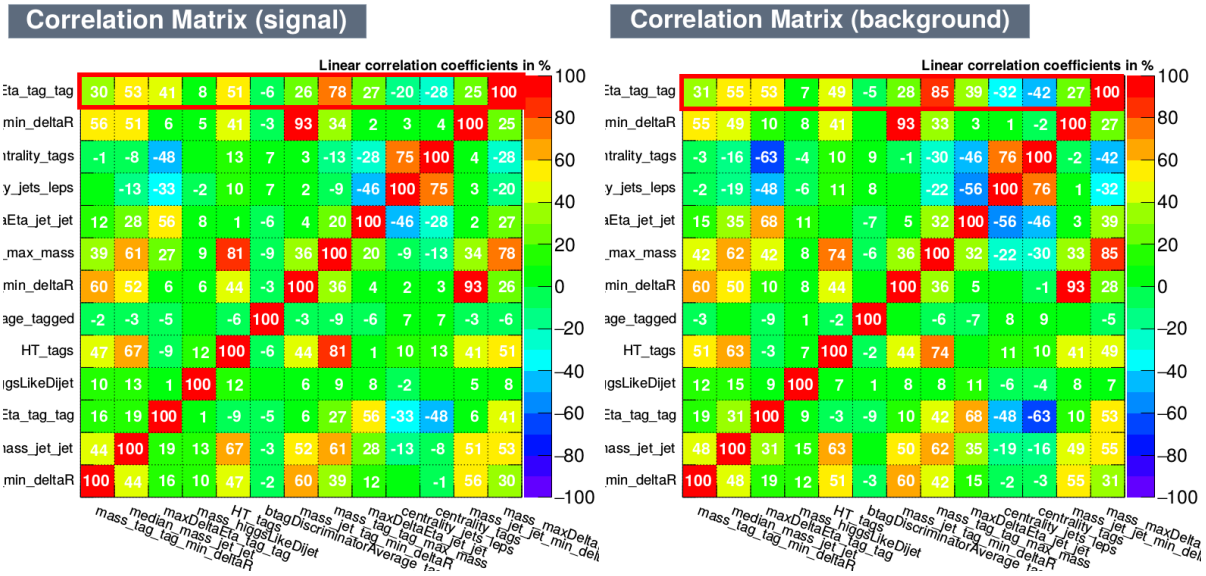
Figure 16: **ROC of the BDT trained with the new input variables.** Evaluated on the test dataset.



(a) (b)

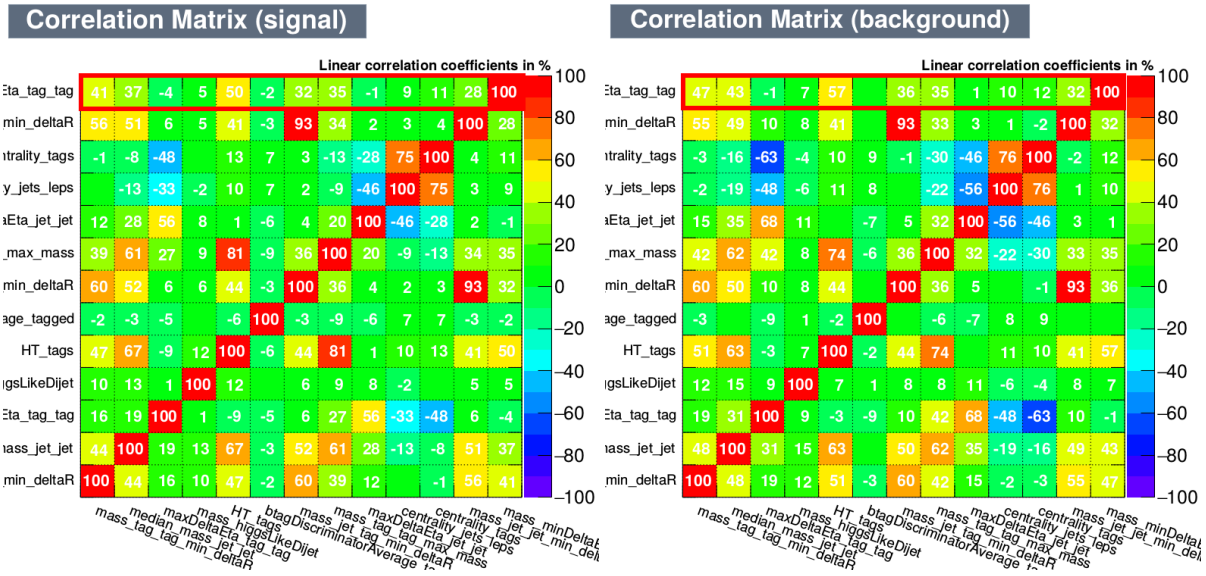
(c)

Figure 17: **New BDT input variables in the ≥ 4 jets, ≥ 4 tags category.** Shown are the mass of two tagged jets with maximum $\Delta\eta$ (a), the mass of two tagged jets with minimum $\Delta\eta$ (b) and the multiplicity of dijets with $\Delta\eta \leq 1.5$ (c)



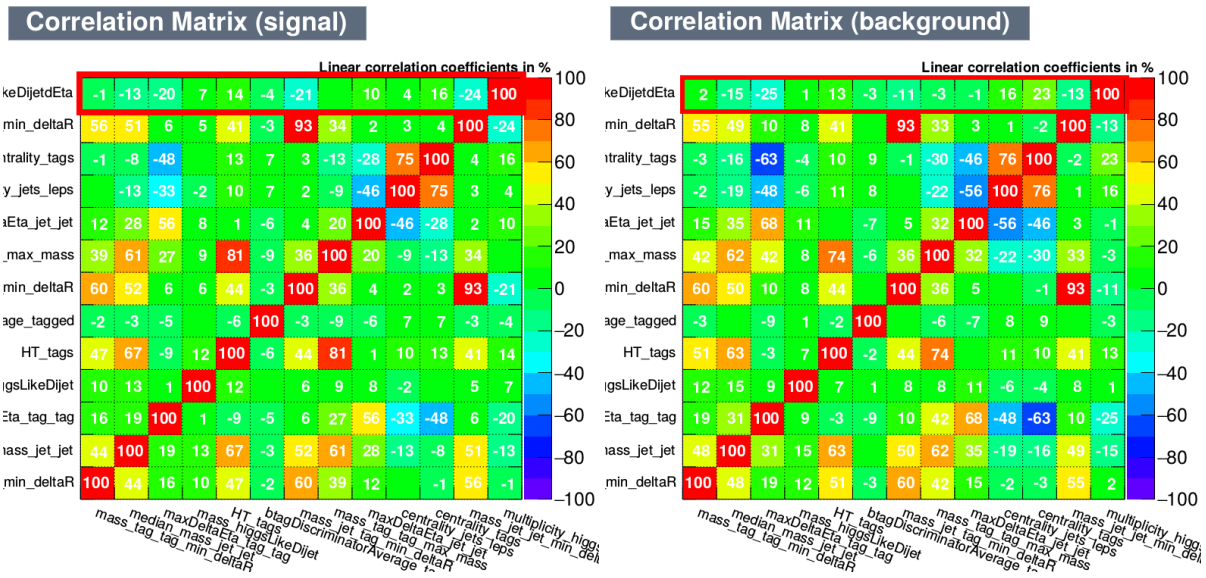
(a) (b)

Figure 18: **Correlations coefficients of the mass of two tagged jets with maximum $\Delta\eta$ with other BDT input variables in the ≥ 4 jets, ≥ 4 tags category.** The correlations of the new variable are shown in the first row.



(a) (b)

Figure 19: **Correlations coefficients of the mass of two tagged jets with minimum $\Delta\eta$ with other BDT input variables in the ≥ 4 jets, ≥ 4 tags category.** The correlations of the new variable are shown in the first row.



(a) (b)

Figure 20: **Correlations coefficients of the multiplicity of dijets with $\Delta\eta \leq 1.5$ with other BDT input variables in the ≥ 4 jets, ≥ 4 tags category.** The correlations of the new variable are shown in the first row.

References

[1] "Search for $t\bar{t}H$ production in the $H \to b\bar{b}$ decay channel with 2016 pp collision data at $\sqrt{s} = 13$ TeV," CERN, Geneva, Tech. Rep. CMS-PAS-HIG-16-038, 2016. [Online]. Available: https://cds.cern.ch/record/2231510

[2] A. Hoecker et al., "TMVA: Toolkit for Multivariate Data Analysis," *PoS*, vol. ACAT, p. 040, 2007.

[3] T. Stevenson, "Cross-Validation," September 2016. [Online]. Available: https://indico.ph.qmul.ac.uk/indico/getFile.py/access?contribId=3&resId=0&materialId=slides&confId=129

[4] ——, "Cross-Validation in TMVA," October 2016. [Online]. Available: https://indico.cern.ch/event/571102/contributions/2342484/attachments/1359710/2057400/CV_IML_Oct2016.pdf

[5] K. El Morabit, "ParticleSwarmOptimization." [Online]. Available: https://github.com/kit-cn-cms/ParticleSwarmOptimization/tree/master/PSO

[6] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[7] F. Chollet *et al.*, "Keras," https://github.com/fchollet/keras, 2015.

[8] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

[9] M. Claesen *et al.*, "Optunity," 2014. [Online]. Available: http://optunity.readthedocs.io/en/latest/index.html

[10] J. B. et al, "Hyperopt," 2013. [Online]. Available: https://github.com/hyperopt/hyperopt

[11] R. Martinez-Cantin, "Bayesopt: A bayesian optimization library for nonlinear optimization, experimental design and bandits," *Journal of Machine Learning Research*, vol. 15, pp. 3735–3739, Nov 2014. [Online]. Available: https://github.com/rmcantin/bayesopt