Master Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Machine learning for $t\bar{t}H$ mechanism Higgs boson detection from CERN ATLAS data

Bc. Jan Presperín

Supervisor: prof. Dr. Ing. Jan Kybic
Supervisor–specialist: doc. Dr. André Sopczak
Field of study: Open Informatics
May 2022

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení:	**Presperín**	Jméno: **Jan**	Osobní číslo: **465913**
Fakulta/ústav:	**Fakulta elektrotechnická**		
Zadávající katedra/ústav:	**Katedra počítačů**		
Studijní program:	**Otevřená informatika**		
Specializace:	**Datové vědy**		

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Metody strojového učení pro detekci ttH mechanismu produkce Higgsova bosonu

Název diplomové práce anglicky:

Machine learning for ttH mechanism Higgs boson detection from CERN ATLAS data

Pokyny pro vypracování:

The CERN proton accelerators LHC produces collisions every 25ns. These collisions are recorded by the ATLAS detector and lead to so-called events describing the collision products and its properties. Events of interest (signal events) are currently recognized by a rule-based system based on hand-engineered and complex features. The task is to recognize the signal events automatically using the techniques of machine learning, and
possibly deep learning, and thus increase the ratio of correctly identified events. We shall use lower level features, namely the energies and momentum vectors of the collision products. The project is a part of the effort to analyse the properties of the Higgs
boson using the complete data-set recorded in the LHC Run-2 operation.
Instructions:
1. Get familiar with the data and basic principles of searching for particles in high-energy physics.
2. Get familiar with the existing implementation of a classifier to separate events of interest from the background,using high-level features and classical machine-learning techniques.
3. Design and implement a classifier based on high-level features
using deep learning. Evaluate its performance on simulated data and compare with existing approaches.
4. Design and implement a classifier based on low-level features using either classical machine learning or deep-learning or both.
Evaluate their performance on simulated data and compare withexisting approaches.
5. Apply the classifier on recorded real data and compare the number of detected signal events with the expectation.

Seznam doporučené literatury:

[1] https://atlas.cern (“learn more”) An introduction to the ATLAS experiment for the public
[2] Dan Guest et al. Deep Learning and its application to LHC Physics. Annu. Rev. Nucl. Part. Sci. 2018, 68:1-22
[3] Pierre Baldil et al: Searching for Exotic Particles in High-Energy Physics
[4] ATLAS Collaboration: Observation of Higgs boson production in association with a top quark pair at the LHC with the ATLAS detector. Physics Letters B, vol. 784, pp. 173-191, 2018
[5] R.Duda,P. Hart, D.Stork: Pattern classification. Willey-Intersciencee, 2000.
[6] I. Goodfellow, Y. Bengio, A. Courville: Deep learning. MIT Press. 2016
[7] J. Malý: "Automatic event recognition for Higgs boson detection",Master thesis, CTU, 2020

Jméno a pracoviště vedoucí(ho) diplomové práce:

prof. Dr. Ing. Jan Kybic, algoritmy pro biomedicínské zobrazování FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

doc. Dr. André Sopczak, ÚTEF ČVUT

Datum zadání diplomové práce: **12.02.2021** Termín odevzdání diplomové práce: ______________

Platnost zadání diplomové práce: **30.09.2022**

prof. Dr. Ing. Jan Kybic podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

.	
Datum převzetí zadání	Podpis studenta

Acknowledgements

I would like to thank both my supervisors, Prof. Dr. Ing. Jan Kybic and doc. Dr. André Sopczak, for providing me with valuable support, constructive feedback and guidance during the work as well as giving me access to the necessary resources which were used during the work.

Declaration

Abstract

One aspect of studying subatomic particles by observing proton-proton collision is being able to identify those collisions where the particles of interest occur, since thousands of collisions are happening in an accelerator such as the Large Hadron Collider (LHC) at any given time. Machine learning methods have shown the potential to improve the performance of the detection while using either hand-engineered features or low-level measurements from the detector as input features. One such particle, which has been studied by multiple research groups, is the Higgs boson. The aim of this thesis is to test and compare several machine learning algorithms and compare the usage of hand-engineered features with the usage of direct measurements of the detector on the task of detecting Higgs boson events, namely the $t\bar{t}H$ process. Gradient boosting, multi-layered perceptron (MLP) and TabNet algorithms were tested and the results show superior performance of gradient boosting algorithms. Hand-engineered features show superior performance as opposed to direct measurements from the detector. Combination of all types of features show the best performance. We also show that classifiers training with only the most important features can achieve results with only a small performance decrease, while on the other hand providing benefits in terms of training time and model simplicity. In addition, it is shown that for an increased amount of training data, the performance of the classifiers is expected to improve.

Keywords: CERN, ATLAS, Higgs boson, classification, neural networks, machine learning

Supervisor: prof. Dr. Ing. Jan Kybic

Supervisor-specialist: doc. Dr. André Sopczak

Abstrakt

Jedním z aspektů zkoumání subatomárních částic pomocí studia protonových srážek je schopnost identifikovat ty srážky, kde vznikají částice, které nás zajímají, jelikož v akcelerátoru, jako je Large Hadron Collider (LHC), v každém okamžiku dochází k tisícům srážek. Metody strojového učení ukázaly potenciál zlepšit úspěšnost detekce při použití jak příznaků vytvořených na základě doménové znalosti (doménové příznaky), tak těch odpovídajícím přímým měřením z detektoru (nízkoúrovňové příznaky). Jednou z částic, jejímž studiem se zabývá několik výzkumných skupin, je Higgsův boson. Cílem této práce je otestovat a porovnat několik algoritmů strojového učení a porovnat doménové příznaky oproti nízkoúrovňovým příznakům na úloze detekce kolizí, kde vzniká Higgsův boson, konkrétně jeho produkce zvaná $t\bar{t}H$ proces. Byly testovány algoritmy založené na gradient boostingu, více-úrovňové neuronové sítě (MLP) a TabNet, přičemž metody gradient boostingu dosahují nejlepších výsledků. Ukazuje se rovněž dominance doménových příznaků nad nízkoúrovňovými, nejlepších výsledků je dosaženo při jejich kombinaci. Je také ukázáno, že klasifikátory používající pouze několik nejdůležitějších příznaků mohou dosáhnout téměř tak dobrých výsledků jako ty používající všechny příznaky, s dodatečnou výhodou nižších časů trénování a větší jednoduchosti výsledného modelu. Rovněž je ukázáno, že při získání více tréninkových dat lze od klasifikátorů na příslušné úloze očekávat zlepšení výkonu.

Klíčová slova: CERN, ATLAS, Higgsův boson, klasifikace, neuronové sítě, strojové učení

Překlad názvu: Metody strojového učení pro detekci $t\bar{t}H$ mechanismu produkce Higgsova bosonu

Contents

Figures

Tables

Chapter 1

Introduction

Researchers at CERN and other institutions have been uncovering new insights regarding how our universe works, which is often done by examining the particles that make up the matter itself. One such experiment, which deals with searching for new particles, is the CERN ATLAS experiment [A$^+$08], which registers particles created in proton-proton collisions from the Large Hadron Collider (LHC).

The collisions where massive particles of interest occur constitute only a small fraction of all events that happen in the detector, so it is important to be able to select what subset of events is suitable for further analysis and which events can be ignored.

This task, called event selection, is performed on multiple levels. Given that the ATLAS detector detects over a billion particles per second, which would not be possible to store into a permanent storage, it features a two-stage trigger system reducing the total number of events. The first stage of this system is hardware-based and immediately after an event is created, makes a decision whether to keep or discard it. Roughly 100 000 events per second are passed from the first stage to the second, which is composed of a number of computers that perform a finer selection of roughly 100 events per second for permanent storage. More custom analysis-specific selections are then performed on the stored data.

Event selection can be framed as a signal versus background classification problem, where the goal is to sucessfully detect particle collisions where some physics process of interest happened (signal events) from those collisions where it did not happen or where another process occured (background events). Each collision event is described by a set of parameters and either is the physics process of interest (positive class) or not (negative class). By framing the problem as such, machine learning methods can be used to solve it.

This thesis in particular is dealing with the problem of event selection applied on one particle that has been studied a lot in recent years in high energy physics, the Higgs boson. This work aims to apply neural networks to separate events where the Higgs boson was produced from the events which have similar properties but the Higgs boson was not produced. In addition the results are compared to previous results. Particularly, deep learning methods and decision tree ensembles are the main focus.

The aim of this study is to improve and further advance the study of the $t\bar{t}H$ process [Col18] by creating a functional tool for the event selection task, enabling more precise measurements of the properties of the Higgs boson. Given that the event selection task is performed in the same way as other processes that are studied in high energy physics, the knowledge gained in this study related to the performance of different methods and algorithms is transferrable to other studies that involve the task of separating signal from background events.

1.1 Task description

The task of this thesis is to find a classifier that takes as input signal and background events and classifies them such that significance is maximized as computed in chapter 3. The signal class in this case are the events belonging to $t\bar{t}H$ process, while the background constitutes of events of other processes, namely $t\bar{t}W$, $t\bar{t}Z$, $t\bar{t}$, VV and *other*. The *other* class represents multiple merged less frequent background processes. Each event is described by a set of parameters, which are either some direct detector measurements such as energies or momenta of the collision products, or some higher-level constructed numerical characteristics based on the measurements. The task itself is then a multiclass classification. Although the goal is the maximization of significance (as defined in chapter 3), other evaluation metrics are also used to keep track of the classifier performance in the context of more frequently used (standard) machine learning metrics, such as the ROC curve. This helps to see whether the classifier is performing well from the machine learning point of view.

1.2 Related Work

The usage of machine learning and deep learning methods has been gaining popularity in many fields and particle physics is no exception. Currently there are many cases where these methods are used, such as hit reconstruction - mapping particles to individual hits of the detector sensors, track finding - identification of measurements belonging to the same track, object identification - determining which particles were present or event selection, which is the main focus of this thesis. An overview of machine learning cases in high energy physics is given in [GCW18]. In [A$^+$92] for example, a neural network was used for separating a sample of selected hadronic decays. In event classification in general, physics-inspired features are often created and then fed into the models. [B$^+$04] showed that deep networks, which receive as input only the low-level measurements made by the particle detector, can achieve better results, suggesting that manually creating features only results in an information loss. Another successful usage of neural networks in high energy physics was demonstrated in [A$^+$16], where a convolutional neural network was used to select neutrino events in the NOvA neutrino detector [Bia13].

This thesis is focused mostly on the machine learning part of the physics problem and therefore the network optimization is a significant part of it. The optimization of neural networks is a complex problem involving the choice of many different hyperparameters and although there is currently no way to say beforehand what the best choice of these hyperparameters is, works such as [Ben12] and [YA20] give practical recommendations based both on theory and empirical research.

Deep neural networks have been achieving astonishing results in recent years in involving language tasks, text, audio or video data [DCLT19, GPAM$^+$14, B$^+$20]. One of the most common types of data today is tabular data, where the ensembles of decision tree models such as those based on gradient boosting [Fri01], are considered to be the state-of-the-art methods and can frequently be seen on the top ranks of charts in many machine learning competitions [SZA22]. Despite that, there have been attempts of researchers to create specialized deep neural network architectures for tabular data such as TabNet [AP21] or TabTransformer [HKCK20] to improve the performance of deep networks on "classical" tabular data, although [SZA22] shows that even these sophisticated models are often incapable of beating the gradient boosting-based ensembles.

Chapter 2

Data description

2.1 Physical meaning of the data

2.1.1 Proton-proton collisions

When two particles collide, new, heavier particles than those which were involved in the interaction are produced and almost instantly decay into lighter particles. In the detector, it is therefore only possible to measure physical properties of these lighter particles called decay products. These can be electrically-charged leptons (electrons or muons) and particle jets (streams of particles originating from quarks or gluons). The fact that we can only measure the properties of the decay products brings with itself a problem where we have to deal with a situation where different physical processes can result in decay products containing similar particles and therefore they appear in our working channel (defined in chapter 2.1.3) and are not easily distinguished by the measured properties, which is what makes the correct event classification difficult [Kel19].

2.1.2 $t\bar{t}H$ production

There are many different ways how a Higgs boson can be produced, such as through the gluon fusion process or vector-boson fusion process [Col16], but this thesis focuses on the so-called $t\bar{t}H$ production, which is a rare process where a pair of top quarks emits a Higgs boson or a fusion of a top quark-antiquark pair creates a Higgs boson, which is shown in Figure 2.1. This production mode counts for roughly 1% of all Higgs productions. Examining this process is crucial to understanding the properties of the coupling of the Higgs boson to the top quark, the heaviest known fermion.

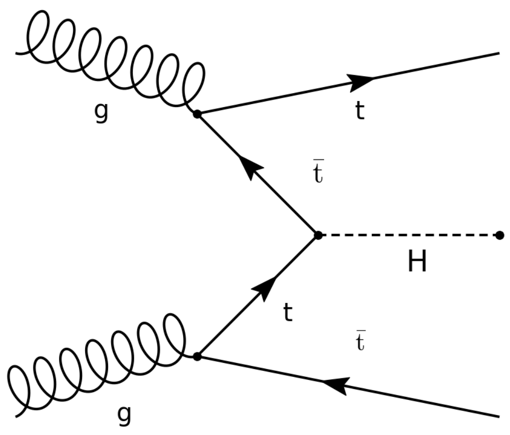


Figure 2.1: Higgs boson production with a pair of top quarks [Ant21], the so-called $t\bar{t}H$ process, shown as a Feynman diagram.

2.1.3 Channel & preselections

It is important to mention that the analysis done in this thesis is performed on a so-called 2LSS1Tau channel. This means, that only events that satisfy the condition of having two same-sign light leptons and one hadronically decaying τ are considered, others are filtered out of the input files and not considered for training at all. This filter on the data is called a preselection and different experiments can vary in the preselection. The exact preselection of an experiment is always specified with it. The reason why we do not consider all events is that concurrently, there are multiple research groups within CERN that study the Higgs boson and the work is divided among them such that each group works on a different channel and research findings are then merged together to produce an aggregate analysis result.

2.1.4 Background for $t\bar{t}H$ production

As mentioned in the previous chapter, when searching for the occurence of a particular physical process, one has to be aware of processes with similar decay products. In the case of $t\bar{t}H$ Higgs production, there can be many such processes, but the main ones are $t\bar{t}W$, $t\bar{t}Z$ and $t\bar{t}$, which constitute the majority of events with similar decay products happening in reality, therefore they are used in our dataset to train and evaluate the classifier along with the VV background and a class of backgrounds called *other*, which reflects more processes which are less frequent and therefore merged together to a single category.

From the machine learning point of view, signal, in this case $t\bar{t}H$, is the positive class while background processes are the negative classes. In the previous study on the topic [Mal20], only $t\bar{t}W$ and $t\bar{t}Z$ background processes were used for training and other background was held as a constant during evaluation. The $t\bar{t}W$ process results in two W bosons and two b-quarks as decay products, while the $t\bar{t}Z$ process can result in many different decay modes. The VV process stands for di-vector boson production, either a pair of W or Z bosons is produced. In the $t\bar{t}H$ process, two t-quarks are produced which subsequently decay into a W and b, thus two W bosons and two b-quarks are created. Figure 2.2 shows the diagrams for $t\bar{t}W$, $t\bar{t}Z$, $t\bar{t}$ backgrounds.

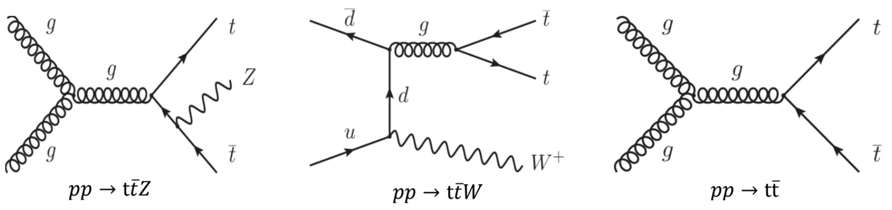


Figure 2.2: Diagrams of $t\bar{t}W$, $t\bar{t}Z$ and $t\bar{t}$ processes [Ant21]

2.2 Training & Testing

2.2.1 Simulation data vs. real data

In order to train a classifier to discriminate between samples of different classes of events, a training dataset with known labels is necessary. To address this need, ATLAS uses programs for generating events such as Pythia [S+14] and uses simulation frameworks that mimic the behavior of the detector itself [A+10]. These simulations generate simulated data of each class of events that are used in our training set. It is up to us to select which classes (physics processes) should be included in our training set based on the particular use case.

The fact that simulated data is used during training brings with itself some specific aspects we need to take into account. First, the distribution of classes in our dataset is completely different than the distribution of the occurence of individual decay processes in reality because more signal events are explicitly generated to provide more examples to the classifier. Second, the expected numbers of events occuring in reality are usually much smaller than the number of training samples we have from the simulations. Due to this fact, weights are assigned to each event during training and evaluation such that the data is scaled to the real class distribution and expected number of events in real data. More details about weighing are given in chapter 2.2.2.

2.2.2 Event weights

As it was mentioned in the chapter describing the nature and differences of real and simulated data 2.2.1, the number of simulated data that the classifiers are trained on exceed the number of events ocurring in real data, so a number is computed for each event in our simulated dataset that is a probability of the event occuring in reality during our measurement period, while the weighted count of the events in the end produce a number of events corresponding to the number of events recorded in the real data. This number calculated for each event is called an event weight and is denoted as (w_e), which means the weight of event (e), where event is a tuple of features describing it.

Event weights are used in two ways. First, they are used when computing the loss function during training, so that the relative probability of each event occuring is calculated. Both with and without-the-weight training was experimented with. Second, the weights are always used when computing the weighted confusion matrix (C_w) during the test phase, which is the main element used to evaluate the classifier. The element $C_w^{i,j}$ corresponding to a value in the weighted confusion matrix on the row i and column j is calculated as:

$$C_w^{i,j} = \sum_{x \in C^{i,j}} w_x \text{ , where} \begin{cases} w_x \text{ is the weight of test example } x \\ C \text{ is a classical confusion matrix} \end{cases} , \tag{2.1}$$

that is, the element on position $(C_w^{i,j})$ is a weighted count (sum of weights in other words) of samples in the test set, whose true class was (i) and the predicted class was (j). The fact that the weights can be floating point numbers also results in the weighted confusion matrix possibly having non-integer values.

The calculation of weight for each event is based on a formula, which is reflecting both the origin of the simulation file (different years correspond to different detector configurations) and the size of the simulated sample. The exact formula for calculating event weights is specified in appendix A.1 and was taken from the $t\bar{t}H$ working group internal note [Col19]. One important note is that some events end up having negative weights, with details about this property specified in [VYM$^+$21]. Negative weight events are, based on group consensus, always removed from training. Based on the characteristics of the simulated data, negative weight events are left in the test set, in order to obtain the correct number of expected events. The important note is that overall after applying weights, the weighted count of events should match the expected number of events recorded in the real data resulting from proton-proton collisions.

2.2.3 Input data

The simulated data and also real data files come in the form of so-called n-tuples, which are essentially ROOT format (*.root*) files, which is a file format used extensively in the scientific community of high energy physics. Details about the ROOT ecosystem are explained in more detail in the previous work on this task [Mal20].

One ROOT file contains events of a single or multiple classes. There can also be multiple files present for one class. Each simulated event is characterized by a set of parameters describing the event. These are either some precalculated characteristics of the event such as the number of b-jets or they are a low-level information related to some physical measurable property such as kinetic energy of some of the particles making up the decay product of the event. Given the fact that this data comes from a simulation, the events also contain some truth information that was fed into the simulation as input and these variables must not be used as features for training a model, since they are not available in the real data. Features that are used throughout this thesis can be separated into two groups:

- Low-level features - (IDs and explanations in Table B.1): Features which correspond to the direct measurements of the detector.
- High-level features - (IDs and explanations in Tables B.2 and B.3): Features which correspond to metrics calculated using domain knowledge.

New simulations are produced regularly, with each simulation being more precise than the last one and with possible addition of new features. It is therefore advisable to always use the newest production. Each simulation is also based on a particular configuration of the detector, which has been changing throughout years. This is marked by the tags *mc16a* (2015 and 2016 configuration), *mc16d* (2017 configuration) and *mc16e* (2019 configuration).

The Table 2.1 shows the exact properties of the dataset used throughout out experiments. It amounts to 73 370 training examples which correspond to 105.87 expected events in real data.

Table 2.1: Training data overview - *v0605_v3* dataset. Weighted count of training examples corresponds to the expected count of events in real data.

Class	Number of training examples	Weighted count of training examples
$t\bar{t}H$	29538	22.75
$t\bar{t}W$	10289	24.84
$t\bar{t}Z$	27410	18.22
$t\bar{t}$	186	22.16
VV	2805	6.45
other	3142	11.42
Total	73370	105.87

2.2.4 Preprocessing

Before using the data in any machine learning algorithm in this thesis, the ROOT files are preprocessed such that multiple files are combined to form a single tabular dataset that can be directly used as input for machine learning algorithms. At first the data is scattered into multiple ROOT files, each containing a subset of the whole dataset and it is not convenient to use it in this way. The goal of the preprocessing is thus to take a set of n-tuples and produce a single tabular dataset usable for training. The so-called data pipeline is used for this purpose. In general, the pipeline consists of 3 steps, separated into 3 scripts, *data_convert.py* , *data_weights.py* and *data_prepare.py*. More details related to the functionality of the scripts is given in the appendix C.

Chapter 3
Significance

Unlike traditional machine learning metrics that are usually optimized in classification tasks, such as accuracy, sensitivity or specificity, in the $t\bar{t}H$ event selection, a special metric native to particle physics is used, called significance.

To illustrate the meaning of significance, let there be a classifier, which classifies events detected in a particle detector (described by a number of features related to the event) into two classes, *selected* events and *rejected* events.

In the context of this thesis, the *selected* events would be all events marked by the classifier as $t\bar{t}H$ events while *rejected* events would be all other events. Let us then have a null hypothesis H_0, which states that a particular decay process ($t\bar{t}H$) does not exist and, based on a physics model we have, some number of events b are expected to be observed in our dataset. We then perform an experiment and observe a higher number of events than b, call this number n. There is a possibility that this was just a statistical fluctuation, so we are interested in determining, whether this was the case or that the reason for observing a higher number of events than b was that there exists some unknown process that our theoretical physics model predicts. If the number of events n is much higher than the expected number of events b, we may conclude that it is not just a statistical fluctuation. In other words, if the

$$n - b$$

is significant based on a result of a statistical test, we can reject H_0. The result of the hypothesis test is the test statistic, from which the p-value can be calculated. The meaning of p-value is the probability of observing such or more extreme observation given H_0 is true. If the p-value is low enough, we reject H_0. This p-value is then reported, with the only difference being that it is expressed as S, such that a zero-mean normally distributed random variable with standard deviation 1 exceeds S with probability p-value. Table 3.1 lists p-values and their corresponding values of significance.

The detailed formula for calculating significance is given in [Kor08], but due to the computational complexity and given the assumption that the selected events come from a normal distribution, we can use simplified formulas for estimating significance. These are used in many CERN publications as well.

Table 3.1: Relationship of significance and p-value [Kor08]. 1 sigma standard deviation corresponds to 68% confidence level. 2 sigma standard deviation corresponds to 95% confidence level.

significance	1	2	3	4	5
p-value	16%	2.3%	0.14%	$3 \cdot 10^{-5}$	$3 \cdot 10^{-7}$

$$S = \frac{s}{\sqrt{s+b}} \tag{3.1}$$

or in its simplified form, which can be used when the amount of background largely exceeds signal:

$$S = \frac{s}{\sqrt{b}} \ , \tag{3.2}$$

where

$$s = n - b \ . \tag{3.3}$$

In the classification task performed in this thesis, which is the task of discriminating between signal events ($t\bar{t}H$ class) and background events ($t\bar{t}W$, $t\bar{t}Z$, $t\bar{t}$, VV, *other* classes), the significance is calculated using a trained classifier and a test set. In particular, the **weighted confusion matrix** (defined in chapter 2.2.2) values are used to calculate the significance.

Signal s in our case is calculated as:

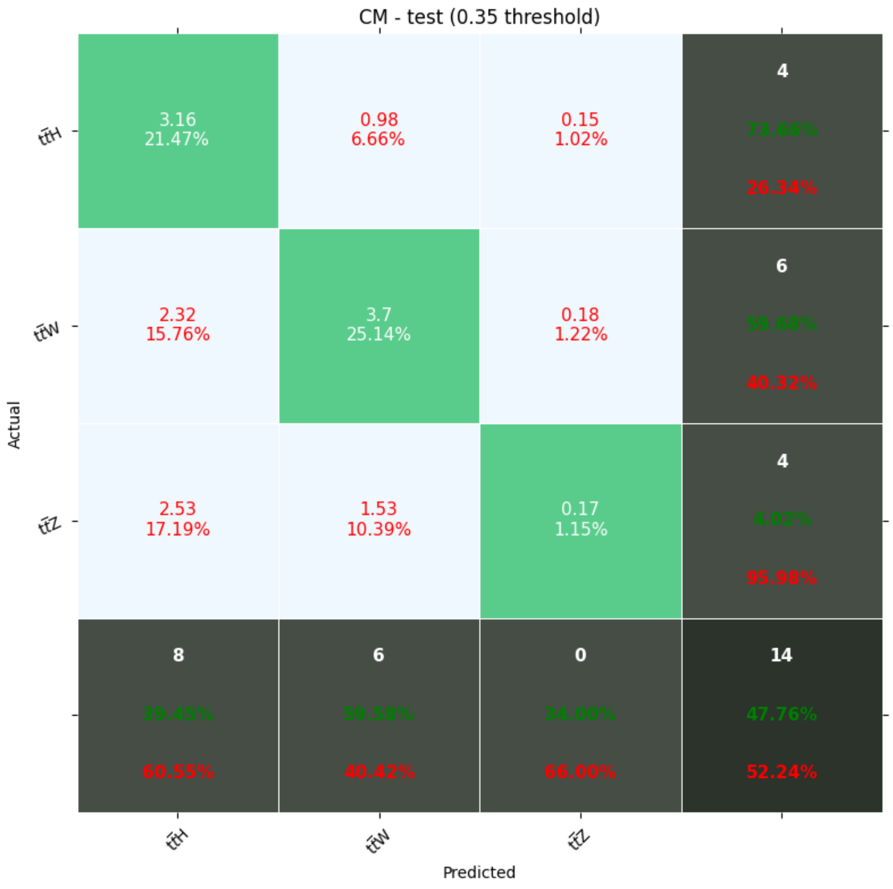
$$s = F \cdot \sum_{x \in P_t} w_x \ , \text{where} \begin{cases} w_x \text{ is the weight of test example } x \\ P_t \text{ is a set of True Positive examples} \\ F \text{ is the scaling factor, defined below} \end{cases} , \tag{3.4}$$

while Background b is calculated as:

$$b = F \cdot \sum_{x \in P_f} w_x \ , \text{where} \begin{cases} w_x \text{ is the weight of test example } x \\ P_f \text{ is a set of False Positive examples} \\ F \text{ is the scaling factor, defined below} \end{cases} \tag{3.5}$$

When significance is calculated, it has be calculated such that it corresponds to a significance on a dataset of real data recorded in some specific period. When using a test set to calculate it, this condition does not hold because we only have a subset of the accordingly weighted simulated dataset. The scale factor F is used to account for this. When the significance is calculated from the weighted confusion matrix, it is necessary to scale the numbers in the weighted confusion matrix such that they correspond to the expected count of events in real data recorded in that specific period. To achieve that, we calculate the scale factor as:

$$F = \frac{size_f}{size_t}\text{, where}\begin{cases} size_t \text{ is the size of the testset,} \\ size_f \text{ is the size of the simulated dataset} \end{cases} \tag{3.6}$$



.

Figure 3.1: Example of a weighted confusion matrix for significance calculation. The percentages in the light blue and green cells show the ratio of the cell value to the sum of all cell values in green and blue cells. The 2-tuple of percentages in the dark cells show the ratios of correctly (green percentage) and incorrectly (red percentage) classified examples in the specific row/column

Figure 3.1 shows an example of a weighted confusion matrix (C_w) from which the significance is calculated. (the term $C_w^{i,j}$ corresponds to a value in the weighted confusion matrix on the row i and column j (indexing goes from zero)). The size of the test set was 20% of the simulated data, thus, the scale factor $F = 5$. Equation 3.7 shows the significance computation for the matrix specified in Figure 3.1.

$$
\begin{aligned}
S &= \frac{s}{\sqrt{b}} \\
&= \frac{F \cdot C_w^{0,0}}{\sqrt{F \cdot (C_w^{1,0} + C_w^{2,0})}} \\
&= \sqrt{F} \cdot \frac{C_w^{0,0}}{\sqrt{(C_w^{1,0} + C_w^{2,0})}} \\
&= \sqrt{5} \cdot \frac{3.16}{\sqrt{(2.32 + 2.53)}} \\
&= 3.208
\end{aligned}
\tag{3.7}
$$

The example described above is considered as a good result in terms of significance on our particular task.

- **NOTE** Significance was also used to evaluate classifiers in the previous thesis on this topic [Mal20], but the way how it is computed has changed due to addition of sample weights in this thesis.

Chapter 4
Methods

4.1 Optimal threshold search

The decision policy q of the classifiers for example x is as follows during the training and testing:

$$q(x) = \arg\max_{y \in \boldsymbol{C}} p(y|x), \text{ where } \begin{cases} C = \{t\bar{t}H, t\bar{t}W, t\bar{t}Z, t\bar{t}, VV, other\} \\ p(y_i|x) = \text{ probability that} \\ \quad \text{example } x \\ \quad \text{belongs to class } y_i \end{cases} \tag{4.1}$$

In other words, the classifier predicts that an example is in a class which has the highest probability. But this decision strategy is not always optimal to achieve the best performance if we optimize for significance, so the same strategy is used as in [Mal20] to find the best model. The strategy is also displayed below.

$$q(x,t) = \begin{cases} t\bar{t}H & \text{if } p(t\bar{t}H|x) \geq t \\ \arg\max_{y \in \boldsymbol{R}} p(y|x), \text{ where } R = C \setminus \{t\bar{t}H\} & \text{otherwise} \end{cases} \tag{4.2}$$

After a model is trained, the test set is taken and 100 evenly distributed decision thresholds (t) from 0 to 1 are generated. Then for each threshold (t), the class predictions are calculated on the test set based on this classification rule, and the weighted confusion matrix is then calculated, which is used to calculate the metric of interest. The threshold (t) with the best value of the significance is taken as the optimal threshold.

4.2 Hyperparameter optimization

During training of the classifiers, a hyperparameter optimization was performed to find the optimal model. Each model used in our experiments had their parameter grids (specified with the relevant experiments) and a variant of coordinate descent algorithm was used to find the optimal value for each hyperparameter. The method works in such a way that during one iteration, the best value of a single hyperparameter is being selected while keeping the others fixed. After the best value is found, it is fixed and the procedure continues with another hyperparameter. Altogether four rounds which go over all hyperparameters are performed with an early stopping behaviour in place to stop the optimization if it does not improve further. Due to each experiment being run with a random training/testing split, a 4-fold cross-validation is performed so that each experiment run with a certain set of hyperparameters is described by 4 individual training/testing phases to smooth out the random selection process.

4.3 Algorithms description

The goal of this chapter is to describe classification algorithms used in this thesis for the signal-background classification problem. Namely these are algorithms based on gradient boosting machines such as XGBoost [CG16], feed-forward neural networks, and some other specific neural network architectures suitable for tabular data such as TabNet [AP21]. The method used for calculating the importance of features by using the multi-layer perceptron network is also discussed.

4.3.1 Neural networks - overview

Neural networks, also called artificial neural networks, are a type of machine learning algorithm which can be used for a variety of prediction tasks such as classification, regression, anomaly detection, computer vision or natural language processing. There are many different types of networks, but in general, they are composed of so-called layers and if the network is composed of multiple layers, we speak about deep learning. Each layer is a set of computational units, also called neurons. Each neuron can be represented by some mathematical function, which receives an input and produces some output. Each neuron also has a set of parameters which parametrize the function it implements. Neurons in each layer are connected to neurons in another layer, meaning that their outputs serve as inputs to the other neurons. There are different types of architectures of the neural networks and they differ substantially in how the individual layers are connected to each other, what kind of mathematical functions the neurons implement, how many layers and neurons are in the network and many other design aspects.

To put it more technically, the whole neural network is a mathematical function:

$$f : X^{\mathbb{R}^n} \to y^{\mathbb{R}^m} \tag{4.3}$$

mapping some input (X) (received by the first layer of neurons) to some output (y) (produced by the last layer of neurons), where the nature of (X) and (y) differs based on the task. For example, in the task of recognizing the licence plates from images the input (X) is an image and (y) a letter sequence, which corresponds to the predicted licence plate. On the other hand, in a task of predicting age of a person from the measurements of their health indicators, the input (X) is a vector of numerical measurements and (y) is a number.

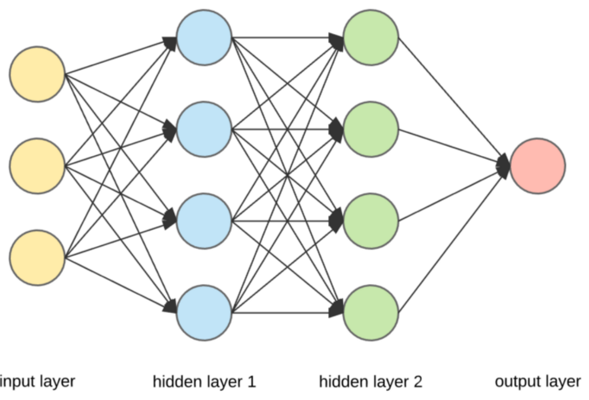


Figure 4.1: Schema of a multilayered perceptron neural network with 4 layers [Der17]

4.3.2 Multi-layer perceptron

Multi-layer perceptron (MLP) is a type of feed-forward neural network that is one of the algorithms used for the classification task in this thesis. Every neuron in one layer is connected to every other neuron in the next layer, as it is seen in chapter 4.1. The first layer is called the input layer and the last layer is called the output layer. The layers in between are called hidden layers. It is possible to make the network bigger and capable of representing more complex relationships by adding new hidden layers and thus making the overall mathematical function that the networks represents more complex. The MLP network could also be viewed as a directed acyclic computational graph.

Each node of the graph, as displayed in Figure 4.1, is a single neuron implementing a linear transformation of the inputs along with a nonlinear function called activation function:

$$A(\sum_{i=1}^{N} w_i \cdot x_i) \ , \tag{4.4}$$

where A is the nonlinear activation function.

4.3.3 Activation functions

The goal of activation functions in neural networks is to add a nonlinear element, which enables the network to represent a more complex mapping between inputs and outputs. There can be many different activation functions and it also plays a role in which layer we want to use it. For the input and hidden layers, the most common activation functions are Rectified Linear Unit(ReLU), Sigmoid or Hyperbolic tangent(Tanh). For multinomial classification, Softmax is often used on the output.

- Rectified linear unit function (ReLU)
$$f(x) = \max\{x, 0\}. \tag{4.5}$$
- Sigmoid function
$$f(x) = \frac{1}{1 + e^{-x}}. \tag{4.6}$$
- Hyperbolic tangent function
$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{4.7}$$
- Softmax function
$$f(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \tag{4.8}$$

4.3.4 Feature Importance

When working with classification algorithms, one of their desirable properties, besides being able to classify as many samples correctly as possible, is being able to quantify the influence of a particular input dimension. This enables us to say which features are important and play a key role in determining the class of a sample. Not only does this allow us to reduce the number of features without compromising the resulting performance of the classifier, but it also indicates to us which features should we be interested in with respect to the inputs-outputs relationship we are examining.

Deep neural networks are often thought of as black-box models which on one hand can achieve outstanding results on some tasks, but on the other hand it is often not clear why exactly they make some particular decision. In tree-based models the feature importance for a parameter can easily be calculated based on the reduction in the metric used to select split points in the trees and given that gradient boosting algorithms (described more in detail in chapter 4.3.5) sometimes use decision trees as weak learners, they share this property [HTF09].

Although interpretability is not particularly a strength of neural networks, there exist approaches to extract which attributes are the most important. One such method is called Integrated Gradients [STY17], which is also used to extract feature importance in some of our experiments.

For each feature in a particular sample, this method assigns a score describing whether the value of this feature contributed to the sample being classified as the positive class or negative class and also by what extent. The total score of the feature is then averaged over all samples.

The attribution for a sample works in the following way:

1. A baseline input is taken. This is an input for which the prediction is neutral.
2. A series of inputs are generated such that these are a linear interpolation between our input of interest and the baseline input.

3. Model outputs are calculated for these inputs based on our trained model.
4. For each model output corresponding to one interpolated input, the gradient with respect to the inputs is calculated.
5. Average of those gradients for a single feature is the feature attribution for that parameter.

4.3.5 Gradient boosting algorithms

Ensembling

Some algorithms used by our experiments, namely the XGBoost, are based on the technique of gradient boosting, which is a form of model ensembling. Ensembling is a general technique used in machine learning to combine prediction outputs of multiple (simpler) models, so-called weak learners, together to form the final prediction output. It can be achieved by two approaches:

- Bagging: In bagging we sample multiple training sets t_i from our training set T with replacement (ensuring independance among the samples) and train R **high variance, low bias predictors** - one model r_i on one dataset t_i - which together creates the model ensemble. When classifying a new sample x_k, the output of the whole model is the average of the output of all models r_i:

  $$q(x_k) = \frac{1}{R} \cdot \sum_{r_i}^{R} r_i(x_k) \tag{4.9}$$

  Notable algorithm using the principle of bagging is the Random Forest.
- Boosting: In boosting we sequentially train **low variance, high bias predictors**, which are learned sequentially so that subsequent learners aim to fix the mistakes made by previous learners. The most common implementation of boosting is Adaboost, which in each step assigns higher weights to incorrectly classified examples in order for the next weak learner to improve performance on them.

Gradient Boosting

The idea of gradient boosting is best described on a gradient boosting algorithm performing regression. In simple general terms, gradient boosting model is created in the following way:

1. A baseline model is taken. This can be a model predicting a constant or a model minimizing the mean squared error (as our example consists of a regression problem).

2. Residuals are calculated as the partial derivative of the baseline model predicted outputs with respect to its inputs.

3. A new weak learner is fitted to the negative of these residuals using squared loss.

4. Output of the new model is taken as the baseline model output + output of the weak learner from the previous step.

5. Calculation of residuals of the new model and subsequent fitting of another weak learner to the residuals is repeated for a desired number of steps.

This procedure enables us to generalize for a classification task using slight modifications and replacement of the loss function [Drc22] in the following way:

1. True label of any given example x_i is represented as a distribution

$$p(y|x_i), \text{ where } \begin{cases} p(y_i|x_i) = 1 \text{ if } x_i \text{ belongs to class } y_i \\ p(y_i|x_i) = 0 \text{ otherwise} \end{cases} \tag{4.10}$$

2. Given the number of classes is M, we create M models, one for each class y_j in our dataset. During classification, each respective model m_j is used to predict a score for its respective class y_j. The scores are then normalized to represent a distribution. Result q of the classification of example x_i is determined by:

$$q(x_i) = \arg\max_y p(y|x) \tag{4.11}$$

3. A difference between the predicted distribution and the true one is calculated, for example using KL-divergence [Joy11]. This is the metric we wish to minimize over the training set.

4. Now the same approach as with training the regressor shown above is used. The only difference is that we have M models, which aim to predict scores corresponding to the values of the distribution $p(y|x)$ for their respective class over the whole training set. After each step, weak learners are added to compensate for the residuals of the M models .

4.3.6 TabNet transformer network

For classification tasks on tabular data, gradient boosting algorithms such as XGBoost or LightGBM are often the go-to strategy [SZA22], but some specific architectures of the neural networks other than the multi-layered perceptron have been designed specifically to tackle tabular data. One such architecture is the TabNet network [AP21].

The TabNet is a transformer neural network architecture - a network based on encoder and a decoder part using attention units [VSP$^+$17]. The encoder part of TabNet is composed of a feature transformer, an attentive transformer and feature masking steps. The decoder is composed of a feature transformer block and a fully connected layer. The feature transformer blocks have a part which has common weights throughout all feature transformer blocks in the network and a part which has independent weights.

The TabNet model is composed of so-called steps, as it is shown in Figure 4.2. The number of steps is a configurable hyperparameter and the more steps there are, the more components the model has and thus a larger capacity to learn, but also more likely it is to overfit. The steps mimic the behavior of weak learners in ensembles. One part of a step is a feature transformer, which itself is a neural network composed of many layers of three types - fully connected layer, Batch normalisation layer and a Gated Linear Unit activation [DFAG16]. Another component in a single step is the attentive transformer, which is again a neural network having three types of layers - fully connected layer, Batch normalisation layer and a Sparsemax layer. This component performs feature selection while also getting as input weights which the previous block gave to each example, thus propagating the information through steps.

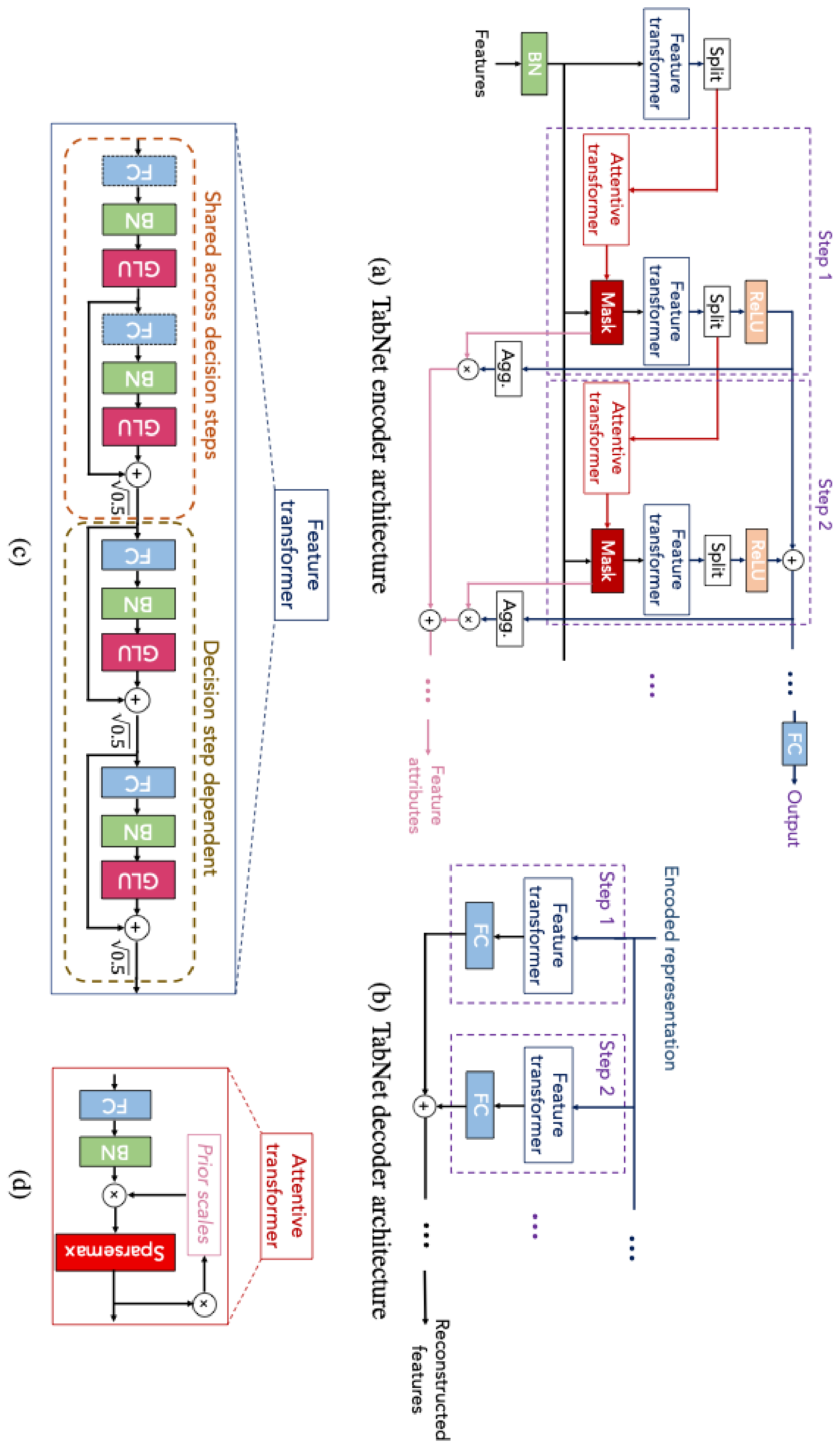


Figure 4.2: TabNet architecture overview, taken from [AP21]

Chapter 5

Experiments

5.1 Overview

This chapter shows the results of the experiments, where different models are tested on our dataset, the feature importance is determined and the performance dependence on the amount of statistics is explored. The experiments use n-tuples version $v0605\_v3$ (see Table 2.1) and apply the tight lepton preselection as specified in the Appendix A.2.

5.2 Experiment 1 - model and feature comparison

5.2.1 Overview

This experiment's goal is to compare different types of models for the $t\bar{t}H$ event selection task and also compare how low-level detector measurement features perform against high-level features constructed by domain experts. Namely, the types of models whose performances were compared are:

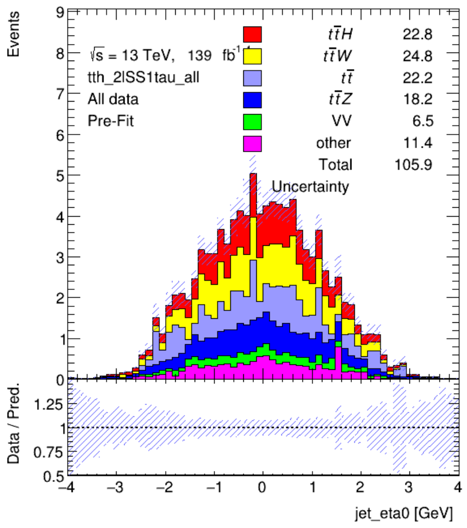
- Multi-layer perceptron neural network
- XGBoost
- TabNet neural network

Three particular sets of features were tested - a detailed list of features used is given in the respective table for each feature set:

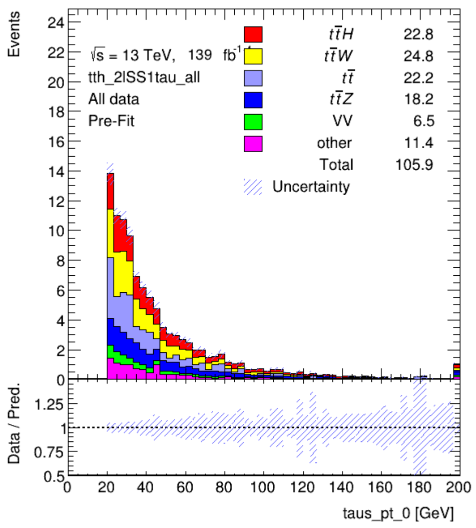
- Low-level features - (IDs and explanations in Table B.1): Features which correspond to the direct measurements of the detector. Altogether there is 17 low-level features available. Later referred to as **L-L**.
- High-level features - (IDs and explanations in Tables B.2 and B.3): Features which correspond to metrics calculated using domain knowledge. Altogether there is 66 high-level features available. Later in the plots referred to as **H-L**.

- All features - union of previous two. Later in the plots referred to as **ALL**.

Each algorithm was tested using each feature set, so altogether it accounts for nine experiments.



(a) : Distribution of jet_eta0 in all (training+testing) data



(b) : Distribution of taus_pt_0 variable in all (training+testing) data

Figure 5.1: Distribution of events based on selected input features - *v0605_v3* dataset. Numbers in the legend correspond to the expected numbers of events in real data, in this case corresponding to the sum of weights of simulated events of each particular process. Numbers correspond to the input data Table 2.1

5.2.2 TabNet - Parameter grid

Table 5.1 shows the parameter grid used during hyperparameter optimization of the TabNet algorithm.

Table 5.1: Parameter Grid - TabNet

Parameter name	Parameter description	Values
n_a	Width of the decision prediction layer	numpy.arange(8, 64, 12)
n_d	Width of the attention embedding for each mask	value always equal to n_a
n_steps	Number of steps in the architecture	numpy.arange(3, 10, 3)
gamma	Number of steps in the architecture	numpy.linspace(1, 2, 4)
n_independent	Number of independent Gated Linear Units layers at each step	numpy.arange(1, 6, 2)
n_shared	Number of shared Gated Linear Units at each step	numpy.arange(1, 6, 2)
lambda_sparse	Extra sparsity loss coefficient	0.001, 0.05
momentum	Momentum for batch normalization	0.01, 0.02, 0.1, 0.2, 0.3

5.2.3 MLP - Parameter grid

Table 5.2 shows the parameter grid used during hyperparameter optimization of the MLP algorithm.

Table 5.2: Parameter Grid - MLP

Parameter name	Parameter description	Values
learning_rate	Learning rate	0.05, 0.001, 0.1, 0.01
layers	Network architecture	List of network architectures as specified in B.4
max_epoch	Maximum amount of epochs during training	150 - with early stopping enabled
batch_size	Network batch size	8, 16, 32, 64, 128, 512
optimizer	Optimization algorithm used for training	Adam, Stochastic Gradient Descent
use_weights_in_loss	Whether weights examples should be weighted in the loss function	True, False

5.2.4 XGBoost - Parameter grid

Table 5.3 shows the parameter grid used during hyperparameter optimization of the XGBoost algorithm.

Table 5.3: Parameter Grid - XGBoost

Parameter name	Parameter description	Values
gamma	Minimum loss reduction required to make a further partition on a leaf node of the tree	0, 6, 12, 25, 100, 200
learning_rate	Learning rate	0.01, 0.03, 0.1, 0.25, 0.5
max_depth	Maximum depth of a tree	5, 10, 15, 25, 35
n_estimators	Number of gradient boosted trees	50, 100, 130, 150
reg_alpha	L1 regularization term on weights	0, 0.4, 3.2, 12, 25, 51, 102, 200
reg_lambda	L2 regularization term on weights	0, 0.4, 3.2, 12, 25, 51, 102, 200

5.2.5 Results

Running these 9 experiments involving 3 classifiers and 3 sets of features showed superior performance of higher-level parameters as opposed to the low-level ones. The best results were achieved by combining both sets of features. With respect to algorithms, the XGBoost showed the best results compared to the other algorithms. Close to XGBoost was the MLP network using all features.

Results are provided in the form of boxplots showing the performance of each algorithm-feature combination in terms of significance (Figures 5.2 and 5.3 show two different significance approximations based on definitions in chapter 3), AUC (Figure 5.4) and accuracy (Figure 5.5).

Each boxplot shows results of 20 cross-validation runs of the best model for a particular algorithm and feature set in terms of the given metric. The boxplots are sorted based on the mean value (green triangle in the plots) of the metric of these 20 cross-validation runs.

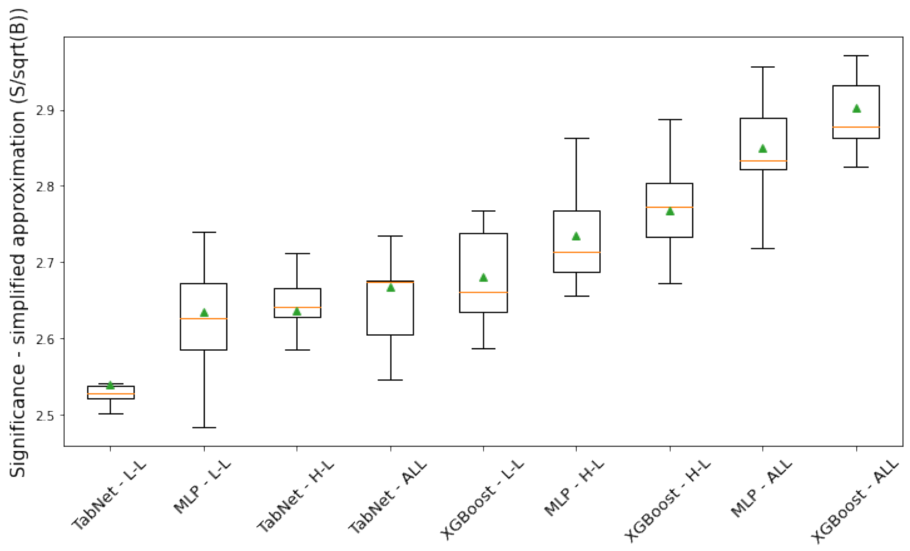


Figure 5.2: Significance dependance on algorithm and feature selection - training with weights, simplified S approximation according to Equation 3.2 - Exp.1

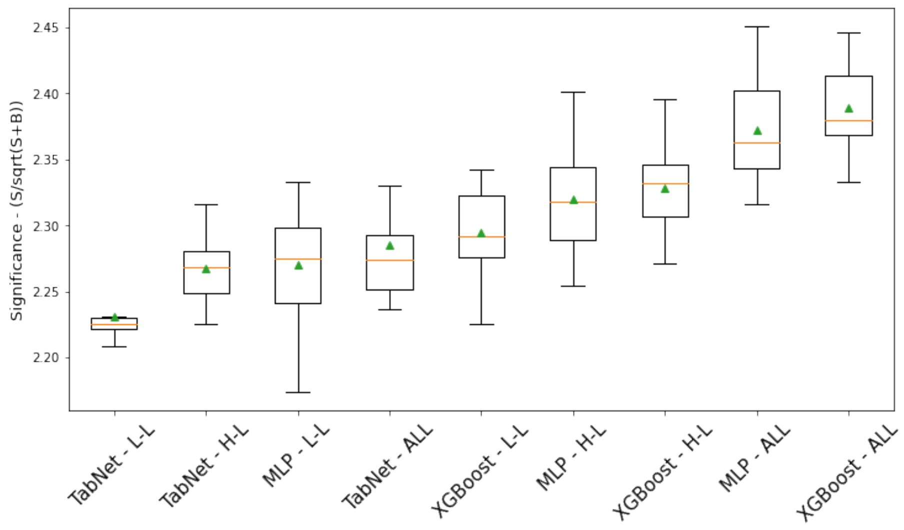


Figure 5.3: Significance dependance on algorithm and feature selection - training with weights, S approximation according to Equation 3.1 - Exp.1

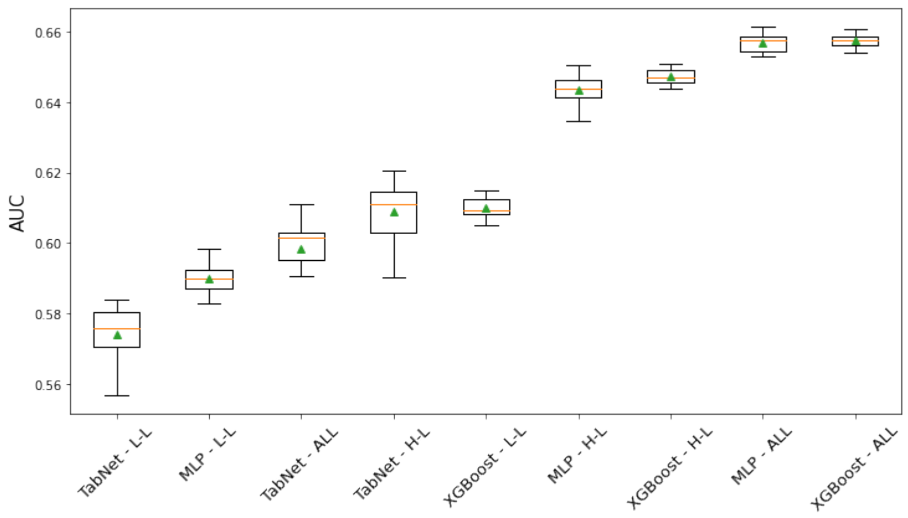


Figure 5.4: AUC dependance on algorithm and feature selection - training with weights - Exp.1

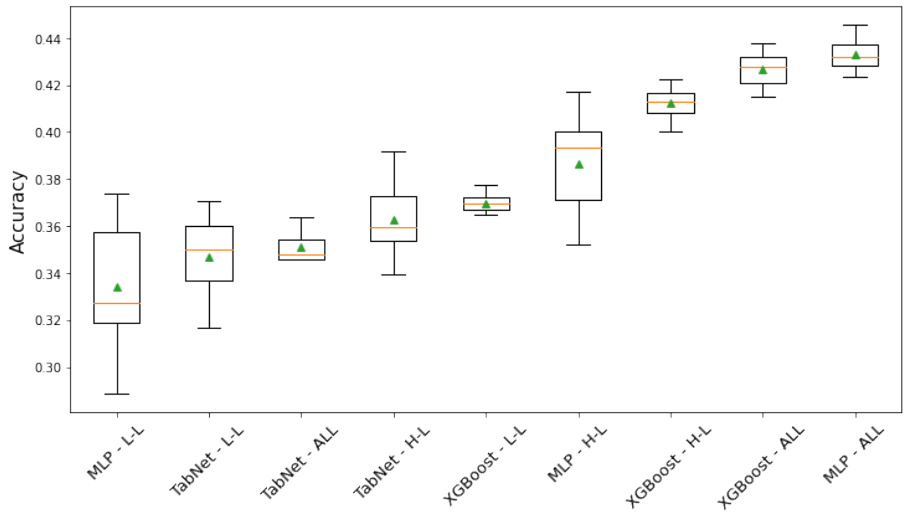


Figure 5.5: Accuracy dependance on algorithm and feature selection - training with weights - Exp.1

XGBoost algorithm achieved the highest significance of 2.90 when measured based on the simplified formula for significance. This result proved the strength of this gradient boosting ensemble on tabular data, where it often outperforms the neural network algorithms. The details of one run of the best XGBoost model are given in the confusion matrix (Figure 5.6), AUC plot (Figure 5.7), and the plot showing significance dependance on the threshold (Figure 5.8).

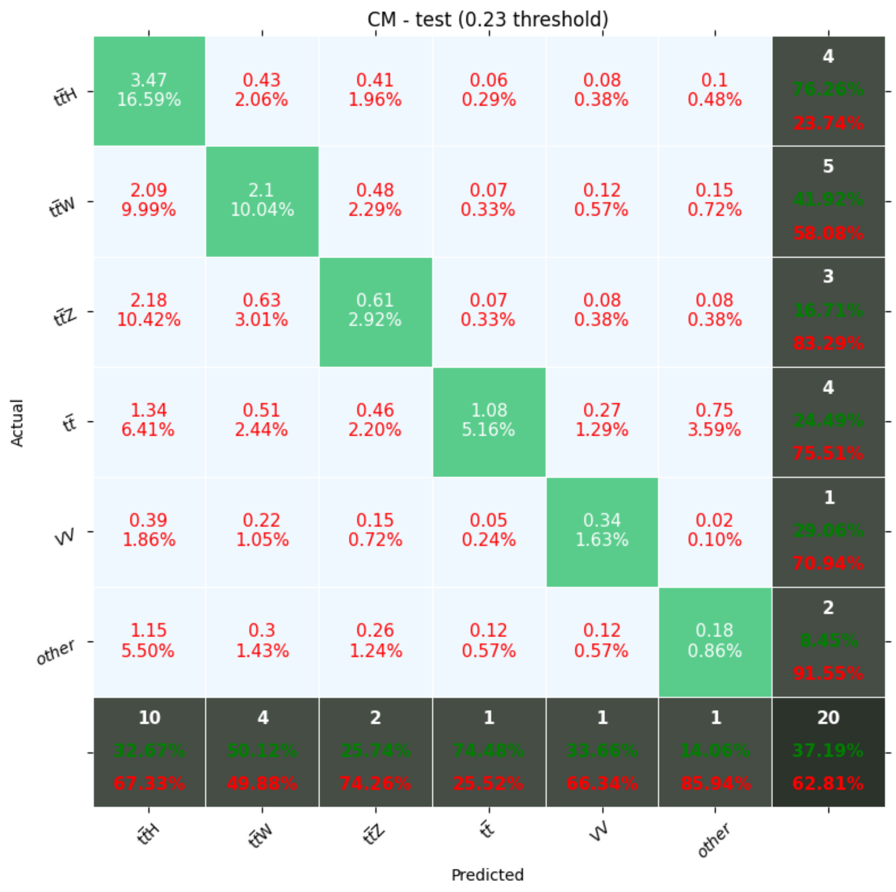
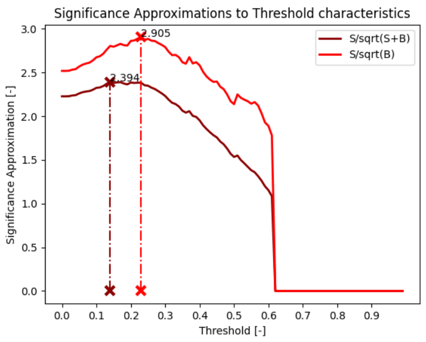
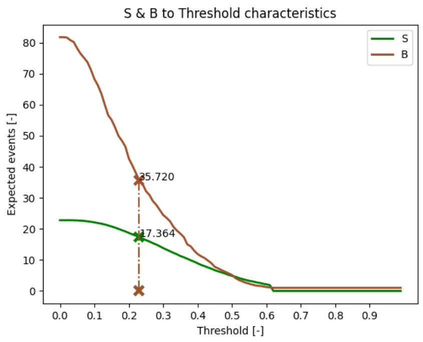


Figure 5.6: Weighted confussion matrix (XGBoost with all features) - Exp.1



(a) : Significance on thresholds



(b) : Signal/Background on different decision thresholds

Figure 5.8: Dependence of signal/background/significance on threshold (XGBoost with all features) - Exp.1

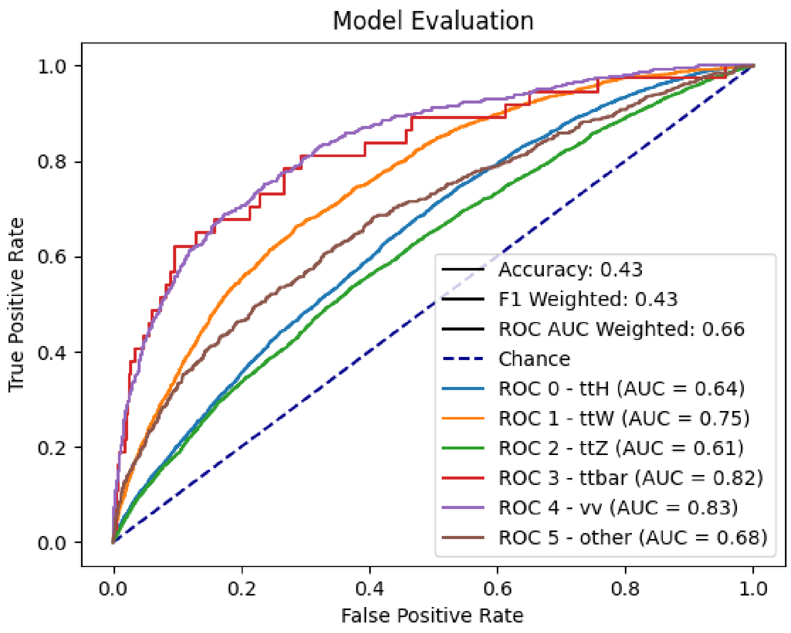
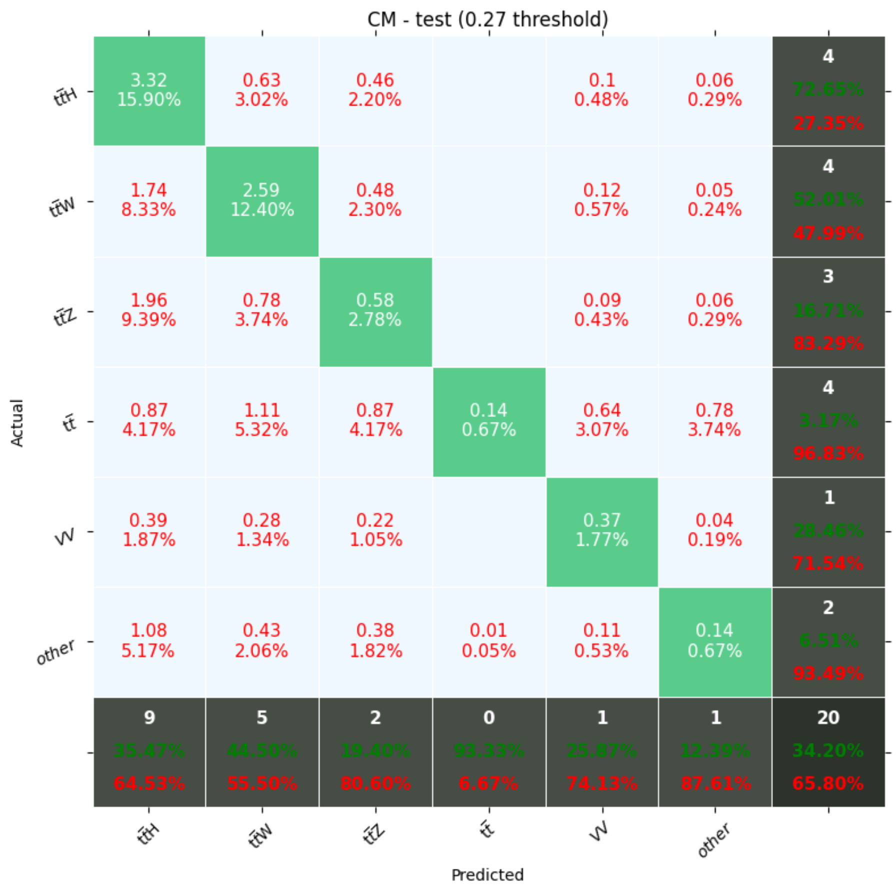


Figure 5.7: Network test scores (XGBoost with all features) - Exp.1. The AUC score for each class is computed using the one vs. rest strategy

The second place in terms of significance is taken by the MLP network. The overview of five best MLP models using the set of all features is given in Table B.5. The performance of the MLP model using all features - which was the second best model overall - is given in more detail in Figures 5.9, 5.10,5.11, 5.12. The stacked histograms (Figures 5.13, 5.14, 5.15, 5.15) provide an overview of the distribution of the events in the test set and events selected by the MLP network as signal.


CM - test (0.27 threshold)
Actual
Predicted

	$t\bar{t}H$	$t\bar{t}W$	$t\bar{t}Z$	$t\bar{t}$	VV	other	
$t\bar{t}H$	3.32 15.90%	0.63 3.02%	0.46 2.20%		0.1 0.48%	0.06 0.29%	4 72.65% 27.35%
$t\bar{t}W$	1.74 8.33%	2.59 12.40%	0.48 2.30%		0.12 0.57%	0.05 0.24%	4 52.01% 47.99%
$t\bar{t}Z$	1.96 9.39%	0.78 3.74%	0.58 2.78%		0.09 0.43%	0.06 0.29%	3 16.71% 83.29%
$t\bar{t}$	0.87 4.17%	1.11 5.32%	0.87 4.17%	0.14 0.67%	0.64 3.07%	0.78 3.74%	4 3.17% 96.83%
VV	0.39 1.87%	0.28 1.34%	0.22 1.05%		0.37 1.77%	0.04 0.19%	1 28.46% 71.54%
other	1.08 5.17%	0.43 2.06%	0.38 1.82%	0.01 0.05%	0.11 0.53%	0.14 0.67%	2 6.51% 93.49%
	9 35.47% 64.53%	5 44.50% 55.50%	2 19.40% 80.60%	0 93.33% 6.67%	1 25.87% 74.13%	1 12.39% 87.61%	20 34.20% 65.80%

Figure 5.9: Weighted confussion matrix (MLP with all features) - Exp.1

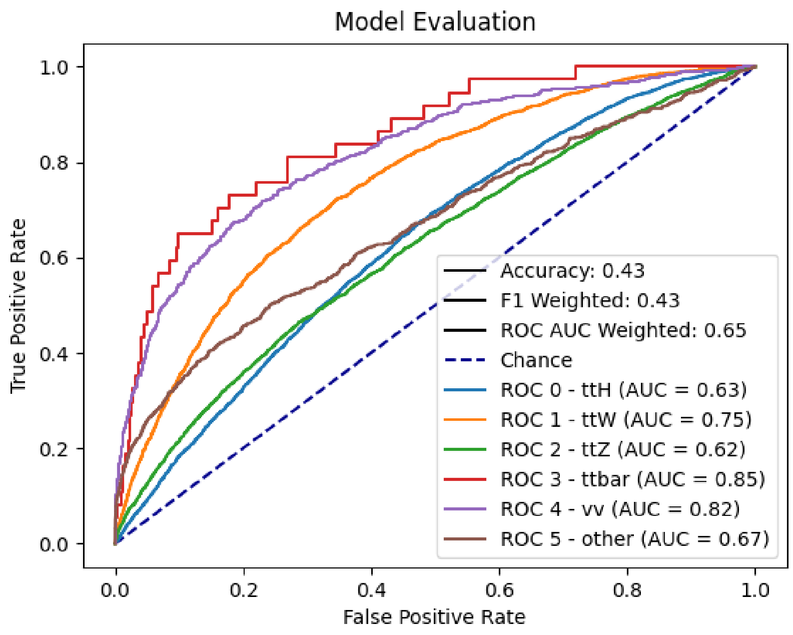
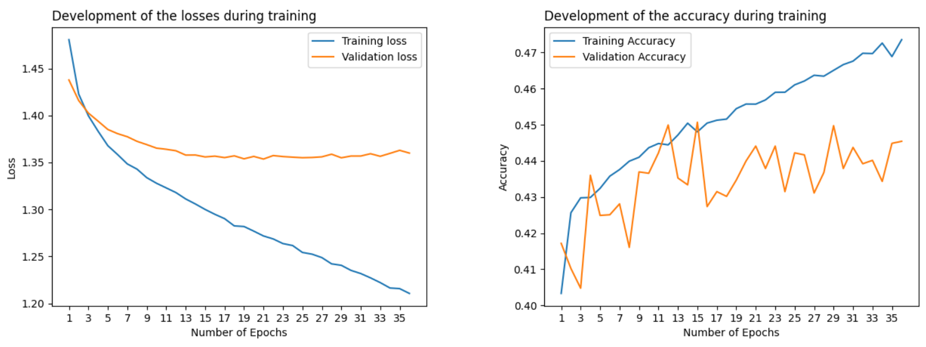
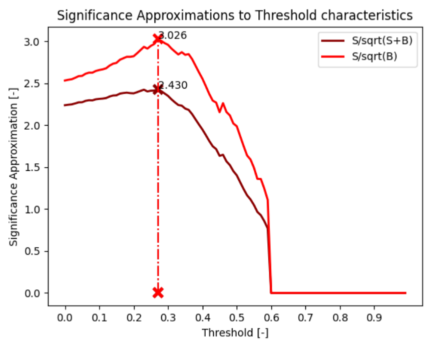


Figure 5.10: Network test scores (MLP with all features) - Exp.1. The AUC score for each class is computed using the one vs. rest strategy

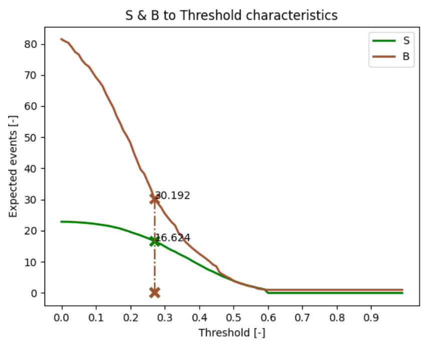


(a) : Training - loss function **(b) :** Training - validation accuracy

Figure 5.11: Network training progress metrics (MLP with all features) - Exp.1

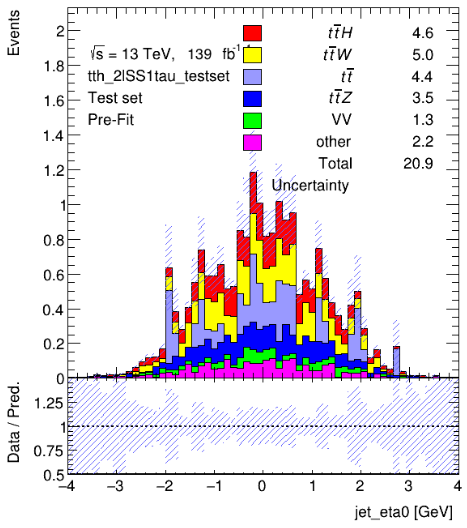


(a) : Significance on thresholds

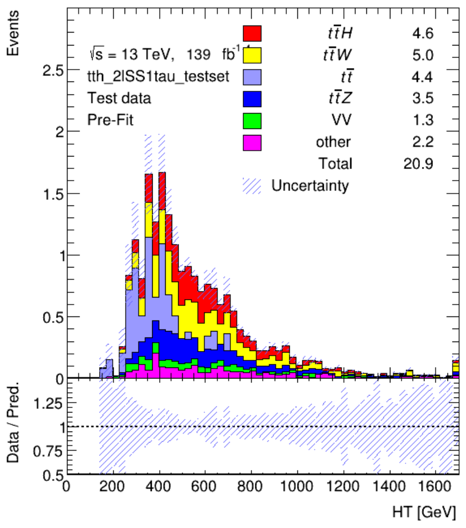


(b) : Signal/Background on different decision thresholds

Figure 5.12: Dependence of signal/background/significance on threshold (MLP with all features) - Exp.1

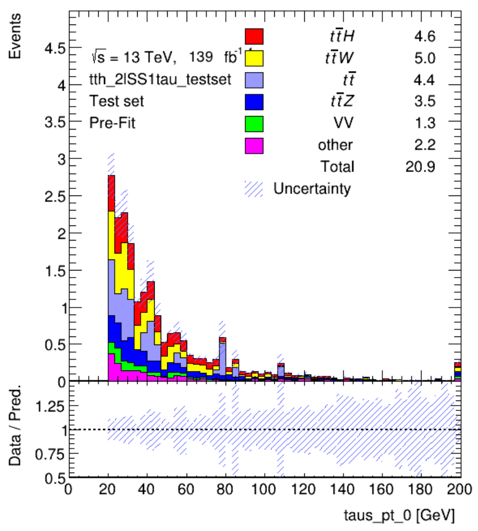


(a) : Distribution of jet_eta0 variable

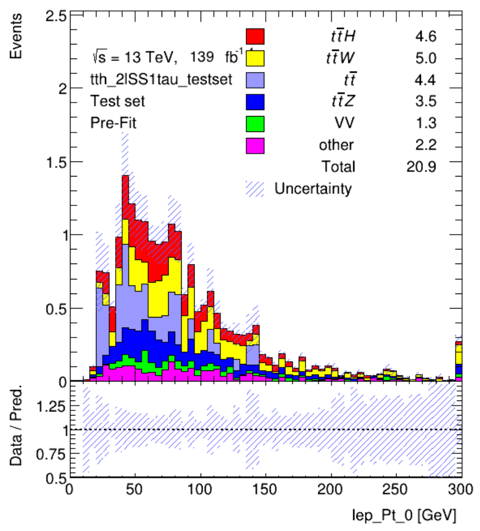


(b) : Distribution of HT variable

Figure 5.13: Distribution of test events - MLP network with all features (1/2) - Exp.1

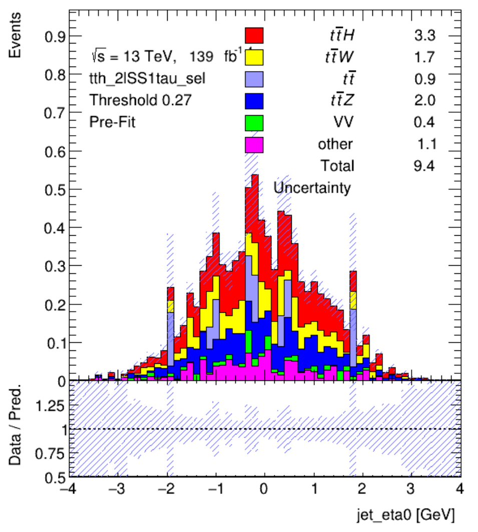


(a) : Distribution of taus_pt_0 variable

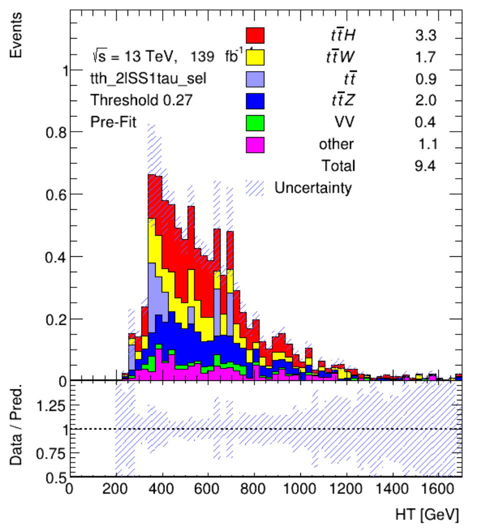


(b) : Distribution of lep_Pt_0 variable

Figure 5.14: Distribution of test events - MLP network with all features (2/2) - Exp.1

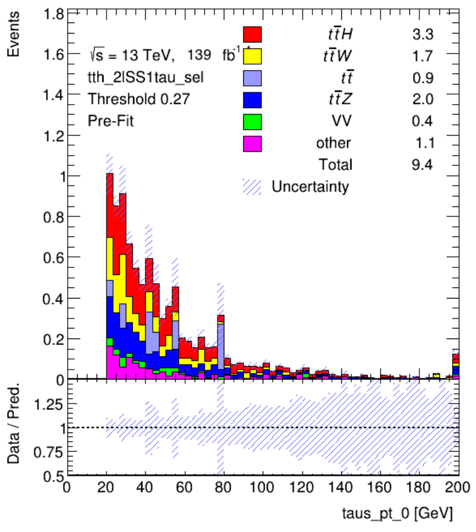


(a) : Distribution of jet_eta0 variable

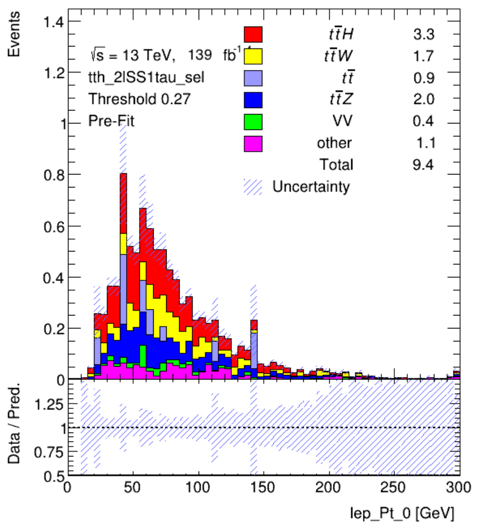


(b) : Distribution of HT variable

Figure 5.15: Distribution of events classified as signal - MLP network with all features (1/2) - Exp.1



(a) : Distribution of taus_pt_0 variable



(b) : Distribution of lep_Pt_0 variable

Figure 5.16: Distribution of events classified as signal - MLP network with all features (2/2) - Exp.1

Overall based on the result of the experiment the classifiers using high-level features outperformed classifiers using low-level features and both performed worse than classifiers using all features. It showed that better significance can be achieved by adding more information to the network. The best result was a mean significance of 2.90 using the simplified approximation in formula 3.2. The resulting significance was confirmed by an independent calculation using the TRExFitter program [Col21]. The mean significance of 2.90 was obtained at a 95% confidence level.

A comparison with previous results is also presented. The significance of 2.90 is an improvement over a BDT study performed within the $t\bar{t}H$ working group which reported a significance of 1.94 [BCG$^+$21]. The previous result corresponds to a measurement precision of $\mu = 1 + 0.72/ - 0.54$, determined with TRExFitter. The expected measurement precision in this thesis has improved and it is $\mu = 1 + 0.42/ - 0.37$ [1]. The value μ is defined as the ratio of measured compared to expected $t\bar{t}H$ events. As we are only using simulated data, the value of μ is unity, thus, only the uncertainty of the μ determination is relevant for this thesis.

To give more context to the presented results of the classifiers, which all use weights during training as described in chapter 2.2.2, hyperparameter optimization and subsequent selection of the 9 best models was also performed without the weights to see how our observed metrics (significance, AUC,

[1]Similarly to the case of the significance computation, the expected limits calculated by TRExFitter (or actually be CommonStatTools) are not "signal-blind". In order to build the Asimov data-set used to extract the expected exclusion limit, the values of the nuisance parameters fitted in the real data are used. This gives a more realistic value to compare with the observed one, especially in cases where large nuisance-parameter pulls change the background prediction significantly with respect to the pre-fit case [Col21].

accuracy) change. As expected, the significance is reduced when compared on the basis of the same algorithm-feature set tuple, as it is shown in the Figures 5.17 and 5.18. This is because algorithms using weights achieve better performance on highly-weighted samples. On the other hand, AUC and accuracy are better when not using weights when compared among the same algorithms-feature set combinations, as shown in the Figures 5.19 and 5.20 show.

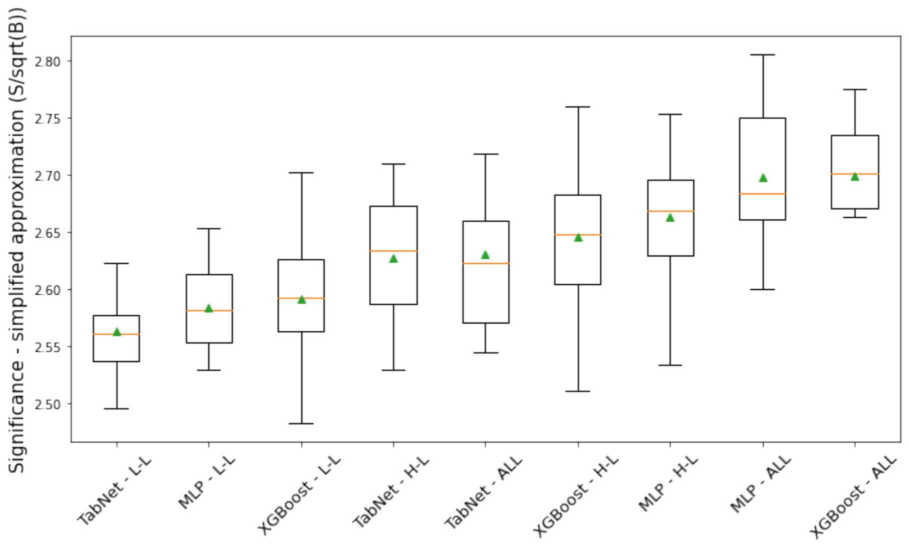


Figure 5.17: Significance dependance on algorithm and feature selection - training without weights, simplified S approximation - Exp.1

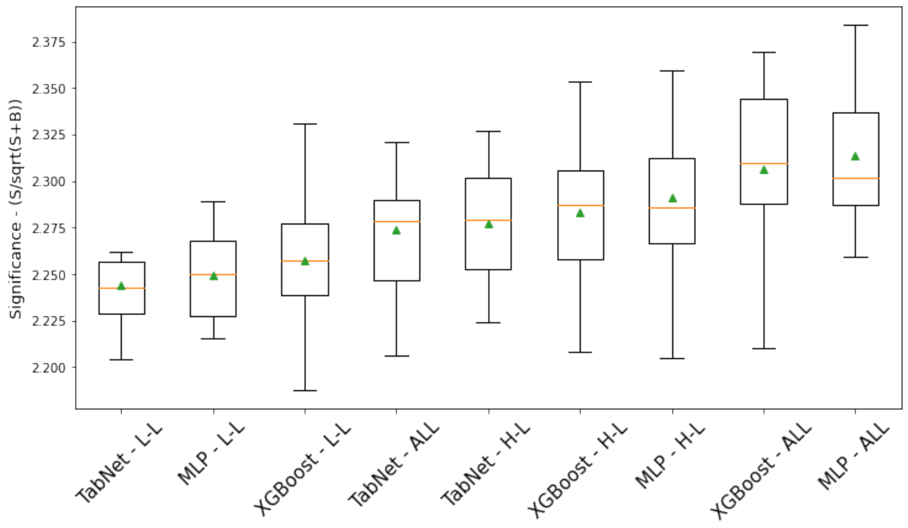


Figure 5.18: Significance dependance on algorithm and feature selection - training without weights - Exp.1

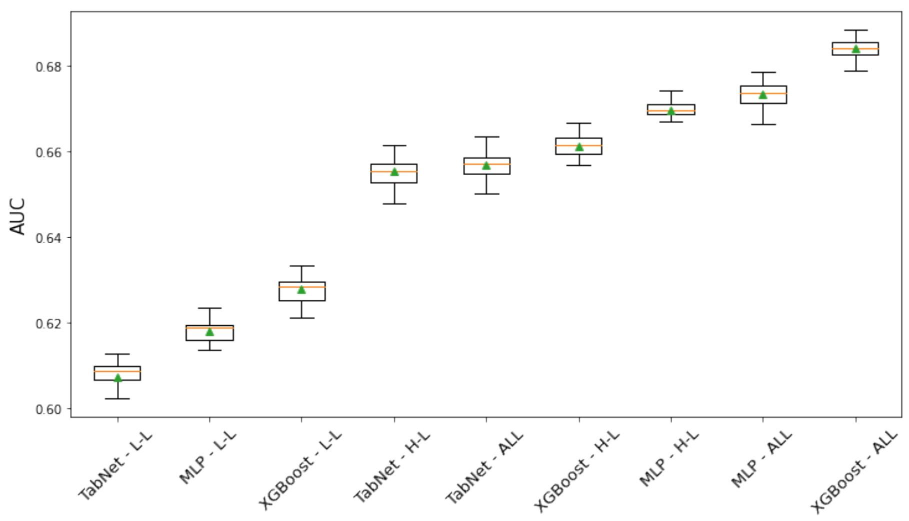


Figure 5.19: AUC dependance on algorithm and feature selection - training without weights - Exp.1

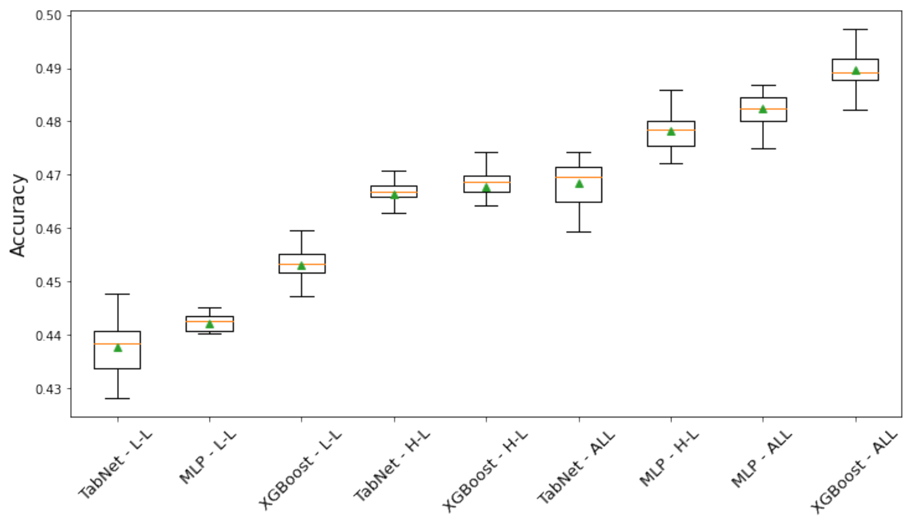


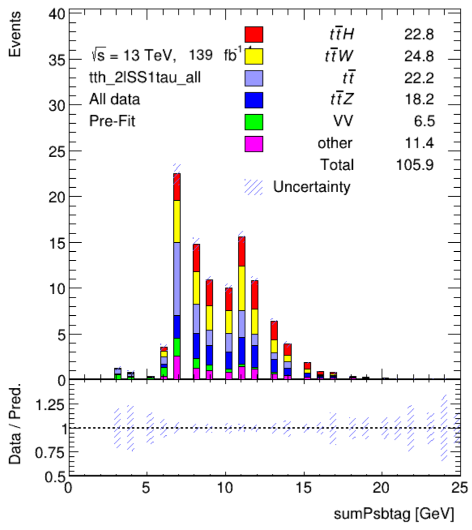
Figure 5.20: Accuracy dependance on algorithm and feature selection - training without weights - Exp.1

5.3 Experiment 2 - reduced feature set

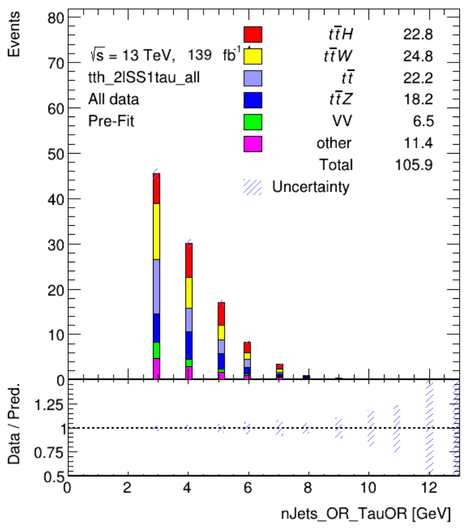
5.3.1 Overview

Based on the results of the previous experiment, which suggested that adding variables will improve our result, it was suggested to improve the usability of the models, mainly their training times, by finding the most important features and building a classifier with this subset of them, which would enable us to train faster with only a small possible reduction in performance.

In the first step, the MLP classifier using all features from previous experiment was taken and it was trained again, this time with the inclusion of a step which calculated Integrated Gradients (IG) score for each feature. The computed scores are given in Tables B.6 and B.7. The most important features in Integrated Gradients are those with the highest absolute value. Distributions of the four most important features are shown in Figures 5.21 and 5.22.

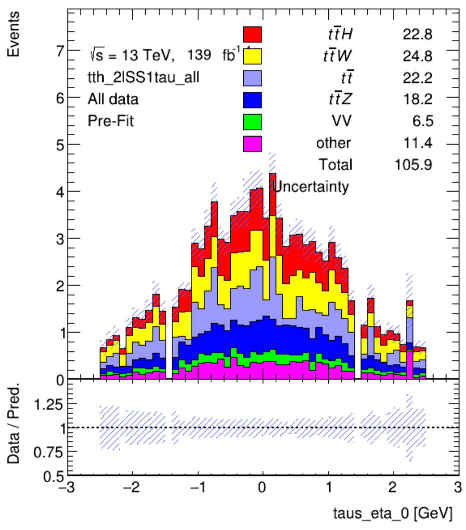


(a) : Distribution of sumPsbtag variable

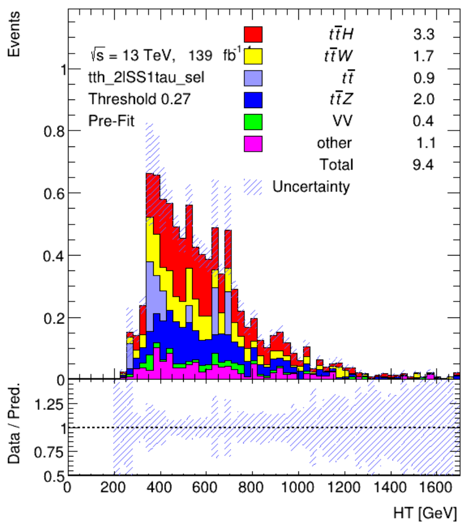


(b) : Distribution of nJets_OR_TauOR variable

Figure 5.21: Distribution of all simulated events in the simulated dataset - most important variables 1/2



(a) : Distribution of taus_eta_0 variable



(b) : Distribution of HT variable

Figure 5.22: Distribution of all simulated events in the simulated dataset - most important variables 2/2

Five experiments were performed to test how much significance can be achieved when using smaller numbers of the most important features. The five experiments which using the same model (MLP) included the 5, 10, 15, 20 and 25 most important features based on the importance measured by the absolute value of the IG score. Their performance was compared.

5.3.2 Results

Figures 5.23, 5.24 and 5.25 show the AUC, significance and accuracy dependance on the number of used features. All three metrics show a gradual increase, as more information is given to the classifier. With 25 features, the significance is already larger than the 2.8 threshold.

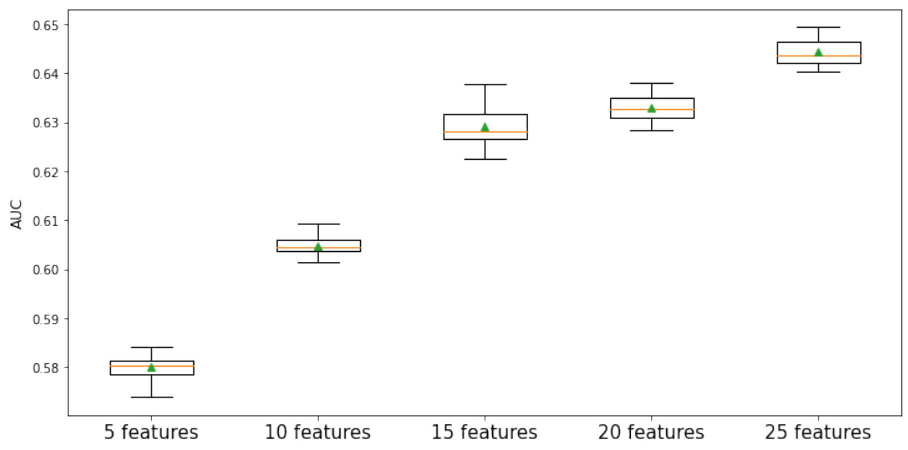


Figure 5.23: Dependance of AUC on number of used features - Exp.2

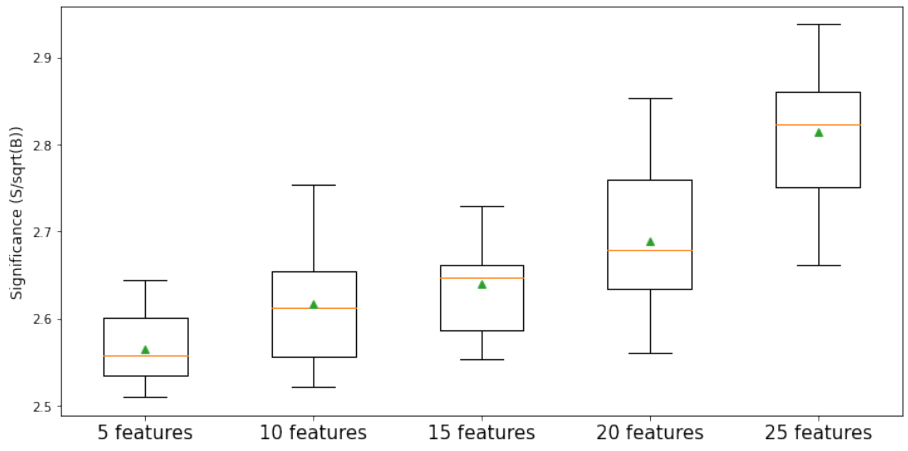


Figure 5.24: Dependance of significance on number of used features - Exp.2

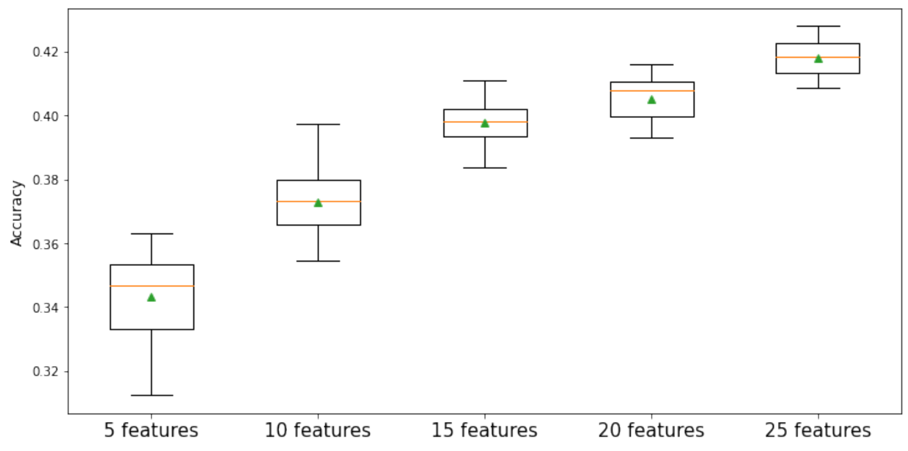


Figure 5.25: Dependance of accuracy on number of used features - Exp.2

5.4 Experiment 3 - dependance of significance on simulated data size

5.4.1 Overview

Given that our data used for training and evaluation comes from a simulation, we set out to understand how the amount of simulated data influences the results we get, both in terms of classical metrics such as AUC and the one we optimize for - significance. Therefore the following experiment was performed.

We performed training on variously sized subsets of our simulated dataset, in particular we took datasets composed of random 20 %, 40 %, 60 % and 80 % of the simulated data and trained a classifier on these subsets and then compared the results. The procedure was such that we took the best MLP classifier using all features from Experiment 1 and on each dataset size performed 40 experiments composed of training and testing phase to account for the effect of the random example selection. On each of the 40 experiments, different random subset of the data of the specific size was taken and trained/tested on. Results are therefore presented in the form of boxplots. The calculation of significance was changed due to a new aspect being brought by reducing the simulated data size.

Our full simulated dataset from the $v0605\_v3$ (Table 2.1) n-tuples with applied preselection contains 73 370 simulated examples, which correspond to 105.87 expected events. When evaluating experiments for significance, the goal is always to calculate with numbers that correspond to expected numbers of events in real data (105.87). Therefore, the scale factor F (defined in chapter 3) was employed in previous experiments which accounted for the fact that significance evaluation is happening only on the test set, which is a subset of our full simulated dataset corresponding to the expected number of events. In this experiment, an additional scale factor G has to be employed in order to account for the fact that our simulated dataset is smaller. It is defined as:

$$G = \frac{1}{size_{sub}} \tag{5.1}$$

where $size_{sub}$ is the relative size of the simulated data subset compared to the full simulated dataset. If we are training on a simulated dataset which is 50 % of our original simulated dataset corresponding to the 105.87 expected events, the scale factor G will be:

$$\begin{aligned} G &= \frac{1}{size_{sub}} \\ &= \frac{1}{0.5} = 2 \end{aligned} \tag{5.2}$$

Significance S in this experiment is then calculated as:

$$
\begin{aligned}
S &= \sqrt{F} \cdot \frac{G \cdot s}{\sqrt{G \cdot b}} \\
&= \sqrt{F} \cdot \sqrt{G} \cdot \frac{s}{\sqrt{b}}
\end{aligned}
\tag{5.3}
$$

5.4.2 Results

We present 3 boxplots showing accuracy, AUC and simplified significance scores for each dataset size. All plots show a gradual increase in the given metric. The increasing trend can be most clearly seen in the AUC boxplot 5.26 while the boxplot with significances 5.27 shows gradual improvement with some slight drops. This instability can be explained by the fact that altogether there is only 186 $t\bar{t}$ training examples in the whole dataset, but they account for 22.16 expected events, which is about one fifth of all expected events. With such a small number of training examples and the amount of features, the classifiers are not able to learn this class properly. Therefore even classifiers using smaller subset of data, which happen to perform well on this class, get a huge boost in terms of the performance on the significance metric, therefore the trend of increasing significance is not that stable as for example AUC or accuracy which do not involve the weights.

It is also visible that as the dataset size grows, the variance of the achieved metrics among experiments of the same data size is getting lower. Overall these results suggest that by generating more simulated data from the distribution will enable us to get more stable and better results.

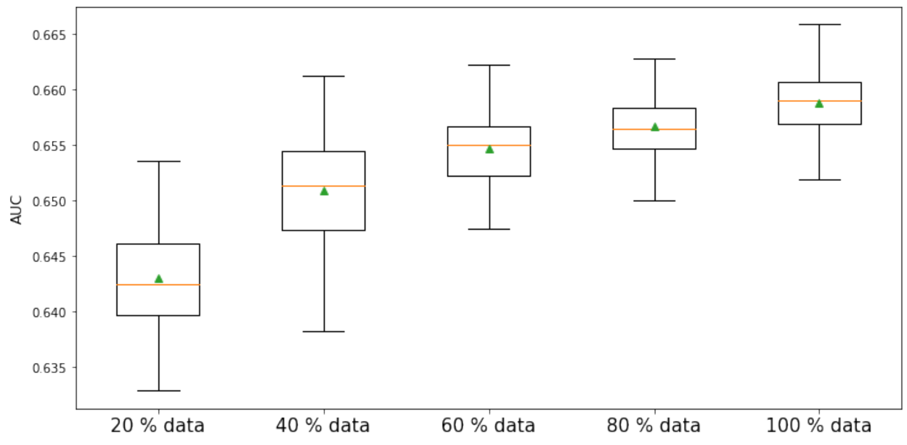


Figure 5.26: Dependance of AUC on simulated dataset size - Exp.3

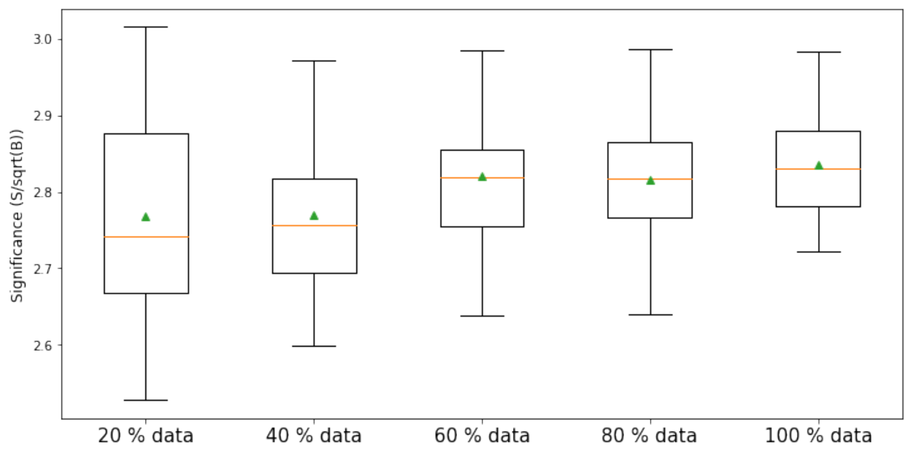


Figure 5.27: Dependance of significance on simulated dataset size - Exp.3

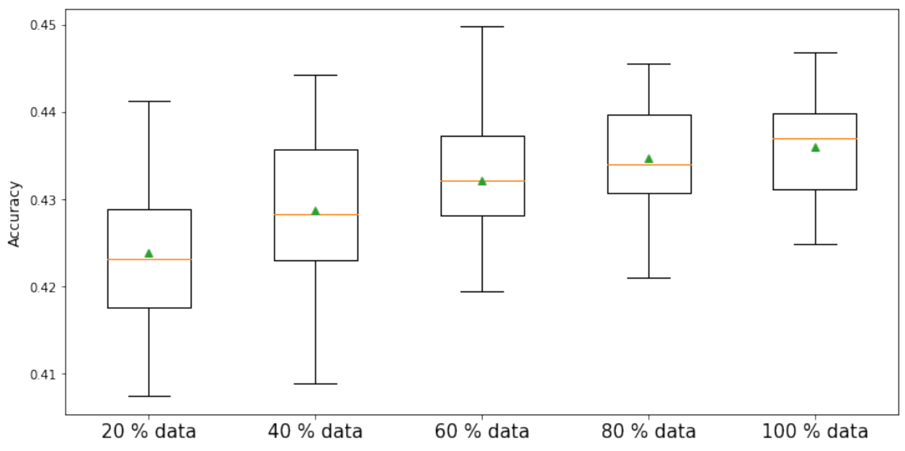


Figure 5.28: Dependance of accuracy on simulated dataset size - Exp.3

Chapter 6
Conclusion

This thesis deals with the task of separating signal ($t\bar{t}H$) events from background events ($t\bar{t}W$, $t\bar{t}Z$, $t\bar{t}$, VV, $other$). It uses events satisfying the 2LSS1Tau preselection, that is two same-sign light leptons and one hadronically decaying τ. The main goal was to test the performace of low-level features as opposed to high-level hand-engineered features created by domain experts. Experiments were conducted such that three models (multi-layered perceptron, TabNet, XGBoost) along with the two feature sets (low-level, high-level) were tested on this task and their performance was compared. In terms of the algorithms, XGBoost classifier showed the best performance, while TabNet had the worst result. As far as the types of features are concerned, classifiers using the high-level features showed better results than those using low-level features. The best results were obtained when the feature sets were merged and the XGBoost classifier was trained on this union, which resulted in a mean significance of 2.90. The resulting significance was confirmed by an independent calculation using the TRExFitter program. This is an improvement over a previous BDT study performed within the $t\bar{t}H$ working group which reported a significance of 1.94. The feature importance was also determined for the best MLP model using the Integrated Gradients method, which provided a feature ranking. Based on this ranking, several experiments were performed with differently sized subsets of the most important features and the experiment with the 25 most important features topped the mean significance level of 2.8. Given that the training and testing of the classifiers was performed on simulated data whose size after the preselection was 73 370 events, experiments were also made to determine whether it is worth generating more simulated events from their respective distributions. All evaluation metrics including AUC, accuracy and most importantly in our case, significance, have been improving as the simulated dataset size was increased, suggesting that further improvements can be obtained by producing more simulated events.

Appendix A
Formulas

A.1 Event weight formula

The formula for calculating weight of an event is based on a number of parameters of the events, given in Table A.1.

Table A.1: List of event features used for event weight calculation

ID	Parameter name	Parameter explanation
f_0	RunYear	Year of detector configuration
f_1	custTrigSF_LooseID_FCLooseIso_DLT	-
f_2	weight_pileup	-
f_3	jvtSF_customOR	-
f_4	bTagSF_weight_DL1r_70	-
f_5	weight_mc	-
f_6	xs	Cross-section
f_7	totalEventsWeighted	Total number of events in the file
f_8	lep_SF_CombinedTight_0	-
f_9	lep_SF_CombinedTight_1	-

The formula for calculating weight w for event e (w_e) is as follows:

$$w_e = \frac{L \cdot f_1^e \cdot f_2^e \cdot f_3^e \cdot f_4^e \cdot f_5^e \cdot f_6^e \cdot f_8^e \cdot f_9^e}{f_7^e} \tag{A.1}$$

where,

$$L = \begin{cases} 36207.66 \text{ if } f_0^e = 2015 \vee f_0^e = 2016, \\ 44307.4 \text{ if } f_0^e = 2017, \\ 58450.1 \text{ if } f_0^e = 2018 \end{cases} \tag{A.2}$$

is the luminosity of the detector in inverse picobarn $[pb^{-1}]$, which changes based on the year that the simulation configuration was based upon and

$$f_i^e = \text{ value of parameter with ID } f_i \text{ for event } e. \tag{A.3}$$

The numerator is a product of multiple variables corresponding to parameters of the event and luminosity. The number is then normalized by the size of the simulated data set.

A.2 Tight lepton preselection

Formula A.4 is a boolean condition on multiple variables determining whether an event passes the preselection or not (filter of events). Due to the long nature of the formula, the names of features that it is composed of were shortened by substituting their IDs as specified in Table A.2.

Table A.2: List of features used for tight lepton 2LSS1Tau preselection

ID	Parameter name
f_0	lep_ID_0
f_1	lep_isMedium_0
f_2	lep_isolationFCLoose_0
f_3	passPLIVVeryTight_0
f_4	lep_isTightLH_0
f_5	lep_chargeIDBDTResult_recalc_rel207_tight_0
f_6	lep_ID_1
f_7	lep_isMedium_1
f_8	lep_isolationFCLoose_1
f_9	passPLIVVeryTight_1
f_{10}	lep_isTightLH_1
f_{11}	lep_Mtrktrk_atPV_CO_0
f_{12}	lep_RadiusCO_0
f_{13}	lep_Mtrktrk_atConvV_CO_0
f_{14}	lep_Mtrktrk_atPV_CO_1
f_{15}	lep_Mtrktrk_atConvV_CO_1
f_{16}	lep_ambiguityType_1
f_{17}	nTaus_OR_Pt25
f_{18}	lep_RadiusCO_1
f_{19}	nJets_OR_TauOR
f_{20}	nJets_OR_DL1r_70
f_{21}	dilep_type
f_{21}	lep_ambiguityType_0

$$selection\_formula = (A \vee B) \wedge (C \vee D) \wedge J \wedge (f_{17} >= 1) \wedge G \wedge H \tag{A.4}$$

, where the individual components are themselves boolean formulas.

$$A = ((|f_0| = 13) \wedge (f_1 > 0) \wedge (f_2 > 0) \wedge (f_3 > 0)) \tag{A.5}$$

$$B = ((|f_0| = 11) \wedge (f_2 > 0) \wedge (f_4 > 0) \wedge (f_5 > 0.7) \wedge (f_3 > 0)) \tag{A.6}$$

$$C = ((|f_6| = 13) \wedge (f_7 > 0) \wedge (f_8 > 0) \wedge (f_9 > 0)) \tag{A.7}$$

$$D = ((|f_6| = 11) \wedge (f_8 > 0) \wedge (f_{10} > 0) \wedge (f_5 > 0.7) \wedge (f_9 > 0)) \tag{A.8}$$

$$E = ((|f_0| = 11) \wedge (f_{22} = 0) \wedge (\neg E_1 \wedge \neg E_2)) \tag{A.9}$$

$$E_1 = (((f_{11} < 0.1) \wedge (f_{11} > 0)) \wedge \neg((f_{12} > 20) \wedge E_3)) \tag{A.10}$$

$$E_3 = ((f_{13} < 0.1) \wedge (f_{13} > 0)) \tag{A.11}$$

$$E_2 = ((f_{12} > 20) \wedge ((f_{13} < 0.1) \wedge (f_{13} > 0))) \tag{A.12}$$

$$F = ((|f_6| = 11) \wedge f(_{16} = 0) \wedge \neg F_1 \wedge \neg F_2) \tag{A.13}$$

$$F_1 = ((f_{18} > 20) \wedge ((f_{15} < 0.1) \wedge (f_{15} > 0))) \tag{A.14}$$

$$F_2 = (((f_{14} < 0.1) \wedge (f_{14} > 0)) \wedge \neg F_3) \tag{A.15}$$

$$F_3 = ((f_{18} > 20) \wedge ((f_{15} < 0.1) \wedge (f_{15} > 0))) \tag{A.16}$$

$$G = ((f_{19} > 2) \wedge (f_{20} > 0)) \tag{A.17}$$

$$H = ((f_{21} > 0) \wedge ((f_0 \cdot f_6) > 0)) \tag{A.18}$$

$$J = (((|f_0| = 13) \vee E) \wedge (F \vee (|f_6| = 13))) \tag{A.19}$$

Appendix B

Tables

B.1 Used features

Table B.1: List of low-level features which correspond to the direct measurements of the detector

ID	Feature name	Explanation
0	jet_eta0	pseudorapidity of the leading jet
1	jet_eta1	pseudorapidity of the subleading jet
2	jet_eta2	pseudorapidity of the sub-subleading jet
3	jet_pt0	transverse momentum of the leading jet
4	jet_pt1	transverse momentum of the subleading jet
5	lep_E_0	energy of the leading lepton
6	lep_E_1	energy of the subleading lepton
7	lep_Eta_0	pseudorapidity of the leading lepton
8	lep_Eta_1	pseudorapidity of the subleading lepton
9	lep_Phi_0	azimuthal angle of the leading lepton
10	lep_Phi_1	azimuthal angle of the subleading lepton
11	lep_Pt_0	transverse momentum of the leading lepton
12	lep_Pt_1	transverse momentum of the subleading lepton
13	taus_charge_0	charge of the leading tau
14	taus_eta_0	pseudorapidity of the leading tau
15	taus_phi_0	azimuthal angle of the leading phi
16	taus_pt_0	transverse momentum of the leading tau

Table B.2: List of high-level features corresponding to constructed metrics based on domain knowledge - 1/2

ID	Feature name	Explanation
0	nTaus_OR	-
1	taus_width_0	-
2	taus_decayMode_0	Decay mode of the leading Tau
3	taus_BDTJetScore_0	-
4	taus_BDTJetScoreSigTrans_0	-
5	taus_JetBDTSigLoose_0	-
6	taus_JetBDTSigMedium_0	-
7	taus_JetBDTSigTight_0	-
8	taus_RNNJetScore_0	-
9	taus_RNNJetScoreSigTrans_0	-
10	taus_JetRNNSigLoose_0	-
11	taus_JetRNNSigMedium_0	-
12	taus_JetRNNSigTight_0	-
13	taus_numTrack_0	number of Tau tracks
14	taus_DL1r_0	-
15	taus_fromPV_0	-
16	taus_passJVT_0	-
17	taus_passEleOLR_0	-
18	taus_passEleBDT_0	-
19	best_Z_Mll	-
20	best_Z_other_Mll	-
21	best_Z_other_MtLepMet	-
22	DeltaR_min_lep_jet	-
23	DeltaR_min_lep_jet_fwd	-
24	dEta_maxMjj_frwdjet	-
25	dilep_type	-
26	DRll01	special distance of jet and lepton
27	eta_frwdjet	-
28	HT	sum of transverse momentum of all objects
29	HT_fwdJets	sum of transverse momenta of jets
30	HT_inclFwdJets	HT including FwdJets
31	HT_lep	sum of transverse momenta of leptons
32	lep_DFCommonAddAmbiguity_0	-

Table B.3: List of high-level features corresponding to constructed metrics based on domain knowledge - 2/2

ID	Feature name	Explanation
33	lep_DFCommonAddAmbiguity_1	-
34	lep_EtaBE2_0	-
35	lep_EtaBE2_1	-
36	lep_ID_0	-
37	lep_ID_1	-
38	lep_Mtrktrk_atConvV_CO_0	-
39	lep_Mtrktrk_atConvV_CO_1	-
40	lep_Mtrktrk_atPV_CO_0	-
41	lep_Mtrktrk_atPV_CO_1	-
42	lep_nInnerPix_0	-
43	lep_nInnerPix_1	-
44	lep_nTrackParticles_0	-
45	lep_nTrackParticles_1	-
46	lep_sigd0PV_0	-
47	lep_sigd0PV_1	-
48	lep_Z0SinTheta_0	-
49	lep_Z0SinTheta_1	-
50	max_eta	-
51	met_met	-
52	met_phi	-
53	minDeltaR_LJ_0	-
54	minDeltaR_LJ_1	-
55	minDeltaR_LJ_2	-
56	mjjMax_frwdJet	-
57	MLepMet	-
58	Mll01	invariant mass of lepton pair (leading and subleading)
59	MtLepMet	-
60	nFwdJets_OR_TauOR	-
61	nJets_OR_TauOR	number of jets
62	nTaus_OR_Pt25	-
63	Ptll01	-
64	sumPsbtag	-
65	total_charge	total charge

B.2 Experiment 1

Table B.4: Parameter Grid - MLP layers architecture - Exp.1

ID	Layer architecture
1	[[num_input_features, 16], [16, num_output_classes]]
2	[[num_input_features, 64], [64, num_output_classes]]
3	[[num_input_features, 128], [128, num_output_classes]]
4	[[num_input_features, 256], [256, num_output_classes]]
5	[[num_input_features, 400], [400, num_output_classes]]
6	[[num_input_features, 800], [800, num_output_classes]]
7	[[num_input_features, 100], [100, 100], [100, 100], [100, num_output_classes]]
8	[[num_input_features, 50], [50, 50], [50, 50], [50, num_output_classes]]
9	[[num_input_features, 200], [200, 200], [200, 200], [200, num_output_classes]]
10	[[num_input_features, 400], [400, 400], [400, 400], [400, num_output_classes]]
11	[[num_input_features, 800], [800, 800], [800, 800], [800, num_output_classes]]
12	[[num_input_features, 100], [100, 100],[100, 100], [100, 100], [100, num_output_classes]]
13	[[num_input_features, 50], [50, 50], [50, 50],[50, 50], [50, num_output_classes]]
14	[[num_input_features, 200], [200, 200], [200, 200],[200, 200], [200, num_output_classes]]
15	[[num_input_features, 400], [400, 400], [400, 400], [400, 400], [400, num_output_classes]]
16	[[num_input_features, 800], [800, 800], [800, 800], [800, 800], [800, num_output_classes]]
17	[[num_input_features, 100], [100, 100], [100, 100], [100, 100],[100, 100], [100, num_output_classes]]
18	[[num_input_features, 50], [50, 50], [50, 50],[50, 50], [50, 50], [50, num_output_classes]]
19	[[num_input_features, 200], [200, 200], [200, 200], [200, 200],[200, 200], [200, num_output_classes]]
20	[[num_input_features, 400], [400, 400], [400, 400], [400, 400],[400, 400], [400, num_output_classes]]

Table B.5: Top 5 MLP models using all features * (BS|OPT|EP|W|LR = batch size|optimizer|epochs|weights in loss|learning rate

Rank	Layers	*BS|OPT|EP|W|LR	Significance	Signif.-simple
1	[[83, 400], [400, 6]]	8|sgd|150|True|0.001	2.372809	2.859039
2	[[83, 128], [128, 6]]	16|sgd|150|True|0.001	2.377316	2.886544
3	[[83, 256], [256, 6]]	16|sgd|150|True|0.001	2.377527	2.866111
4	[[83, 16], [16, 6]]	8|sgd|150|True|0.001	2.387446	2.864654
5	[[83, 200], [200, 200], [200, 200], [200, 200], [200, 6]]	16|sgd|150|True|0.001	2.371628	2.863886

B.3 Experiment 2

Table B.6: List of features based on their Integrated Gradient score 1/2 - Exp.2

Feature	IG Score	Feature	IG Score
taus_charge_0	1.0	lep_sigd0PV_1	0.6538
taus_passJVT_0	0.6119	lep_ID_1	0.6089
DeltaR_min_lep_jet	0.6067	taus_JetBDTSigMedium_0	0.604
dEta_maxMjj_frwdjet	0.601	lep_sigd0PV_0	0.567
lep_ID_0	0.549	lep_DFCommonAddAmbiguity_1	0.5418
lep_Pt_0	0.5338	lep_Eta_0	0.5111
lep_Mtrktrk_atPV_CO_0	0.5102	lep_nTrackParticles_0	0.4853
lep_DFCommonAddAmbiguity_0	0.461	taus_width_0	0.4604
mjjMax_frwdJet	0.4598	taus_BDTJetScoreSigTrans_0	0.4488
DeltaR_min_lep_jet_fwd	0.4434	lep_Mtrktrk_atPV_CO_1	0.4421
minDeltaR_LJ_2	0.442	taus_RNNJetScore_0	0.4319
total_charge	0.4282	taus_JetRNNSigTight_0	0.4101
lep_Phi_0	0.4094	jet_pt0	0.402
taus_BDTJetScore_0	0.4014	nTaus_OR_Pt25	0.4012
taus_RNNJetScoreSigTrans_0	0.4002	taus_numTrack_0	0.3983
taus_JetBDTSigTight_0	0.3891	eta_frwdjet	0.3861
taus_phi_0	0.3845	taus_fromPV_0	0.3825
dilep_type	0.3751	taus_decayMode_0	0.3628
lep_Eta_1	0.3437	lep_Z0SinTheta_0	0.3366
taus_JetBDTSigLoose_0	0.3274	minDeltaR_LJ_1	0.3236
lep_Mtrktrk_atConvV_CO_0	0.3228	jet_eta0	0.3228
lep_Mtrktrk_atConvV_CO_1	0.3195	taus_DL1r_0	0.312
taus_JetRNNSigMedium_0	0.3065	taus_passEleOLR_0	0.3025
taus_JetRNNSigLoose_0	0.3023	best_Z_Mll	0.3023
nTaus_OR	0.3023	taus_passEleBDT_0	0.3023
best_Z_other_Mll	0.3023	best_Z_other_MtLepMet	0.3023
lep_nInnerPix_0	0.3009	lep_Z0SinTheta_1	0.2951
lep_nInnerPix_1	0.2933	lep_E_0	0.286
lep_EtaBE2_0	0.2776	lep_Phi_1	0.2757
taus_pt_0	0.2713	lep_E_1	0.2699
HT_lep	0.268	MtLepMet	0.2551
Mll01	0.2285	jet_eta2	0.2199
DRll01	0.2123	jet_eta1	0.2075

Table B.7: List of features based on their Integrated Gradient score 2/2 - Exp.2

Feature	IG Score	Feature	IG Score
Ptll01	0.2065	max_eta	0.2051
lep_EtaBE2_1	0.1846	met_phi	0.1779
nFwdJets_OR_TauOR	0.1296	lep_nTrackParticles_1	0.1284
minDeltaR_LJ_0	0.1247	MLepMet	0.0753
HT_fwdJets	0.0485	jet_pt1	-0.0103
lep_Pt_1	-0.067	HT_inclFwdJets	-0.0939
met_met	-0.0996	HT	-0.1447
taus_eta_0	-0.4432	nJets_OR_TauOR	-0.6413
sumPsbtag	-1.0		

Appendix C

Code description

C.1 Overview

As far as the code is concerned, it is available at `https://gitlab.fel.cvut.cz/prespjan/cern-machine-learning`. Within the project, a flexible data conversion pipeline was created, whose main goal is to take a number of ROOT files and produce a dataset. The main requirement on the pipeline was the ability to create new datasets for training quickly based on specifying the required n-tuples (*.root* files), features that should be used, number of classes and other optional filters (preselections) applied on the data. The pipeline consists of three scripts:

- **data_convert.py** serves as the first step, in which a set of root files in a specific directory structure is converted to *.csv* files which contain only the features we need in either the training itself or for the later stages of the data processing pipeline (for filtering the data on some values of particular columns, in other words, preselection). The requirement on the directory structure of the input files is as follows. There should be a parent directory containing a set of directories, each corresponding to a directory with files of particular class of events. The name of these subdirectories will later also serve as the class name. Another input of this script is a directory path, which specifies where the resulting *.csv* files will be saved. The output directory then contains a list of *.csv* files, which have an added column (y) corresponding to the class of events contained in the respective file.

- **data_weights.py** serves two purposes. The first task this script accomplishes is that it calculates the weight for each individual event based on the formula in 2.2.2. The second purpose is that it peforms filtering of rows based on specified condition, so-called preselection 2.1.3. The most widely used condition in this thesis is on the 2LSS1Tau channel, which is specified by multiple conditions on many variables.

- **data_prepare.py** takes as input the *.csv* files from the previous section where columns correspond to individual features and combines them into three *pickle* files that serve as a single dataset, which is then used by the neural network training program. The first one is *X.pkl*, which contains a $(n\_event \times n\_features)$ matrix ($\mathbf{X}$), the second file is *y.pkl*, which contains a $(n\_event,)$ vector $\mathbf{y}$ of true labels. The third file is *f.pkl*, which contains a $(n\_features)$ long list ($\mathbf{f}$) of features used in this dataset described by their name. Along with these *pickle* files, *.csv* file called *big_dataframe.csv* is created, which serves as a user-friendly interface to the dataset, because it contains the feature names in the columns and is mainly used for exploratory analysis of the data using data analytics and visualization libraries in Python or other languages. This step also removes the columns which were only used to perform the row-wise filtering and are not used as features for the predictions.

Another essential component of the code is the classifier training component. Again, the goal was to create a flexible program, that could be used to train a classification model on some of the datasets. It was designed with extendability in mind, meaning that the program is composed of various components based on interfaces, so that if one wants to create a new type of model (agent), one just implements an interface that specifies the methods. This interface can be found in `agents/base.py`. An example of that could be switching a fully connected network for a LSTM neural network. Later the architecture was extended to accomodate training other types of classifiers using the same interface.

The program is capable of getting a set of hyperparametes and their values as input and then doing a coordinate ascent-based hyperparameter optimization to select the best combination. It produces an output in the form of a set of directories with the experiment results and an file summarizing the results with the key metrics for each experiment.

C.1.1 Classifier training program overview

This part serves as a brief description of how the project is structured, while further details can be found in the code itself in the form of method and module descriptions (comments).

The root directory of the project contains the entry point of the program, the `main.py` script, along with a number of packages (`utils`, `models`, `agents`, `datautils`, `common`) and directories (`data`, `experiments`).

The configuration necessary before each experiment lies in the `main.py` script itself, particularly all that is necessary to create a configuration is to fill out the desired configuration parameters to the object of type `models.ExperimentConfiguration` returned by the function `user_generated_configuration()`. The documentation to the whole `models.ExperimentConfiguration` object regarding the particular values of the attributes is specified in the class docstrings.

When the `main.py` is run, it performs it performs multiple iterations over hyperparameter values using the coordinate descent approach and for each tested combination of hyperparameters, performs multiple iterations (based on cross-validation settings) of : **1. Training**, **2. Testing** and **3. Optimal working point search**.

1. **Training:** The training is a procedure which performs repetitive iterations over all training examples and aims to minimize the specified loss function.

2. **Testing:** The testing phase is a procedure which computes class predictions of the trained model on the test set, which is then used as input to the `finalize()` method of the subclasses of the `agents.base.BaseAgent` class. This method produces some of the result plots and saves the resulting predictions and probabilities for further use.

3. **The working point search:** The working point search is a procedure, which takes as input a trained model, and iterates over all decision thresholds and specifies the optimal one, which is the one that maximizes the metric optimized for on the test data as described in chapter 4.1.

After the hyperparameter optimization phase finishes the search for optimal hyperparameter values, an overview in the form of an Excel file is generated to have a clear view of the experiment results.

C.1.2 Project modules and directories description

As mentioned in the previous section C.1.1, the project is composed of a number of modules, which aim to serve as changeable components of the whole program. The goal of this section is to give an overview on these modules, while further documentation can be found in the code.

- `utils` (module): This module contains the `utils.root_processing_scripts` submodule, which contains the scripts related to the data processing pipeline (chapter 2.2.4), `utils.significance` submodule containing all methods related to the computation of significance and other model evaluation functions, `confmatrix_prettyprint.py` script, which deals with the generation of the confussion matrices, and other helper scripts containing various functions used throughout the project.

- `models` (module): This module contains a set of files, each containing a class subclassing the `torch.nn.Module` class. These classes serve to build a PyTorch model based on some input architecture and also implement a forward pass of data for that particular model. Forward pass of data through the model can be defined as a series of steps that should be applied to the input tensor corresponding to a single batch of input examples to produce an output of the network. These classes contain all models that are defined and when it is required to use a particular model in an experiment, the model class that was created is usually imported to the constructor of `agents.some_agent.SomeAgentClass` class usually in the following form as a class attribute: `self.model=NameOfTheNewlyDefinedClass`.

- `agents` (module): This module contains a set of files, each containing a class subclassing the `agents.base.BaseAgent` class. This class defined methods which create the structure of the whole training and testing process. Each method in the `agents.base.BaseAgent` class contains its proper documentation.

- `datautils` (module): This module contains two significant files, `dataloaders.py` and `datasets.py`. The `dataloaders.py` file contains classes that serve as custom data loaders for training - practically these create custom `torch.utils.data.dataloader.DataLoader` objects containing the necessary data for training and evaluation. Example of one such class is the `datasets.dataloaders.tth_ttw_ttz_DataLoader`. The second file, `datasets.py`, contains classes that subclass the `torch.utils.data.Dataset` class, for details see [P+19].

- `common` (module):This module contains a set of directories and a set of Python files. The first directory is named `cutvariable_variations`, which holds the *.csv* files, each corresponding to a set of *.root* file parameters (features) based on which a row filtering should be done in the data processing stage. More details can be found in the code documentation in `utils.root_processing_scripts.data_convert.py` file. The second directory is named `feature_variations`, which holds *.csv* files, each corresponding to a set of *.root* file parameters (features) which should be used as features in a dataset generated by the data processing pipeline. More details can be found in the code documentation in `utils.root_processing_scripts.data_convert.py` file. Among the files contained in this module are two files, `conf.py` and `constants.py`. The `conf.py` file contains a series of methods for setting up project logging and loading the configuration file. These functions are executed at the beginning of the whole program. The `constants.py` file contains global constants used throughout the project.

- `experiments` (directory): When the program is run, this directory serves as the storage location for the experiment files. When an experiment is run, it generates an experiment name and each individual training/testing phase which is using a certain set of hyperparameters produces a subdirectory, so that in the end one experiment is stored as `experiment/experiment-id/EXP_F_0_OPT_experiment-run-id`, along with a set of subdirectories for each run.

- `data` (directory): This directory serves as a storage for training/testing datasets. It contains three subdirectories `raw`, `converted` and `tabular`. The `raw` directory is used to store raw *.root* files. If one wants to create a new dataset, one first has to create a directory in the `data/raw` with a subdirectory structure, where each subdirectory corresponds to a storage place for files containing events of a single class. For example, if we have a number of *.root* files containing $t\bar{t}H$, $t\bar{t}W$, $t\bar{t}$ and $t\bar{t}Z$ and want to create a dataset using these files, we create `data/raw/some_dataset_name` directory and in it, we create four directories each corresponding to a single class. See the directory for $t\bar{t}H$ as an example: `data/raw/some_dataset_name/tth`. Then each *.root* file has to be placed to the specific directory according to its class. The other two directories `converted` and `tabular` are used for storing converted, filtered data that results from the data processing pipeline outputs. Further documentation can be found in the Python pipeline scripts themselves.

Appendix D
Bibliography

[A$^+$92] P. Abreu et al., *Classification of the hadronic decays of the Z0 into b and c quark pairs using a neural network*, Phys. Lett. B **295** (1992), 383–395.

[A$^+$08] G. Aad et al., *The ATLAS Experiment at the CERN Large Hadron Collider*, JINST **3** (2008), S08003.

[A$^+$10] G. Aadamowicz et al., *The atlas simulation infrastructure*, The European Physical Journal C **70** (2010), no. 3, 823–874.

[A$^+$16] A. Aurisano et al., *A convolutional neural network neutrino event classifier*, Journal of Instrumentation **11** (2016), no. 09, P09001–P09001.

[Ant21] G. Antonius, *Higgs boson production with a pair of top quarks*, `http://gkantonius.github.io/feynman/auto_examples/index.html`, September 2021.

[AP21] S. Ö. Arik and T. Pfister, *Tabnet: Attentive interpretable tabular learning*, Proceedings of the AAAI Conference on Artificial Intelligence **35** (2021), no. 8, 6679–6687.

[B$^+$04] P. Baldi et al., *Searching for exotic particles in high-energy physics with deep learning*, Annu. Rev. Nucl. Part. Sci. 2018 **68** (2004), 1–22.

[B$^+$20] T. Brown et al., *Language models are few-shot learners*, Advances in Neural Information Processing Systems (H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, eds.), vol. 33, Curran Associates, Inc., 2020, pp. 1877–1901.

[BCG$^+$21] N. Bruscino, A. Chomont, S. Gentile, N. Huseynov, and A. Sopczak, *Study on BDT use in 2lSS1tau channel for ttH analysis*, December 2021, presented by A.Chomont at the ATLAS ttH working group meeting 7 December 2021, CERN.

[Ben12] Y. Bengio, *Practical recommendations for gradient-based training of deep architectures*, pp. 437–478, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[Bia13] J. Bian, *The nova experiment: Overview and status*, arXiv: Instrumentation and Detectors (2013).

[CG16] T. Chen and C. Guestrin, *XGBoost: A scalable tree boosting system*, Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (New York, NY, USA), KDD '16, ACM, 2016, pp. 785–794.

[Col16] The ATLAS Collaboration, *Measurements of the higgs boson production and decay rates and coupling strengths using pp collision data at $\sqrt{s} = 7$ and 8 TeV in the ATLAS experiment*, The European Physical Journal C **76** (2016), no. 1.

[Col18] ______, *Observation of higgs boson production in association with a top quark pair at the LHC with the ATLAS detector*, Physics Letters B **784** (2018), 173–191.

[Col19] ______, *Search for the associated production of a higgs boson and a top quark pair in multilepton final states in pp collisions at $\sqrt{s} = 13$ TeV with the atlas detector*, ATL-COM-PHYS-2018-410 (2019).

[Col21] ______, *Trexfitter*, `https://gitlab.cern.ch/TRExStats/TRExFitter`, September 2021.

[DCLT19] J. Devlin, M. Chang, K. Lee, and K. Toutanova, *Bert: Pre-training of deep bidirectional transformers for language understanding*, ArXiv **abs/1810.04805** (2019).

[Der17] A. Dertat, *Applied deep learning - part 1: Artificial neural networks*, Medium - Towards Data Science (2017).

[DFAG16] Y. Dauphin, A. Fan, M. Auli, and D. Grangier, *Language modeling with gated convolutional networks.*

[Drc22] J. Drchal, *Lecture notes in statistical machine learning*, `https://cw.fel.cvut.cz/b211/_media/courses/be4m33ssu/ensembling_ws2021.pdf`, February 2022.

[Fri01] J. H Friedman, *Greedy function approximation: a gradient boosting machine*, Annals of statistics (2001), 1189–1232.

[GCW18] D. Guest, K. Cranmer, and D. Whiteson, *Deep Learning and its Application to LHC Physics*, Ann. Rev. Nucl. Part. Sci. **68** (2018), 161–181.

[GPAM$^+$14] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, *Generative adversarial nets*, Advances in Neural Information Processing Systems (Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, eds.), vol. 27, Curran Associates, Inc., 2014.

[HKCK20] X. Huang, A. Khetan, M. W. Cvitkovic, and Z. S. Karnin, *Tab-transformer: Tabular data modeling using contextual embeddings*, ArXiv **abs/2012.06678** (2020).

[HTF09] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference and prediction*, 2 ed., Springer, 2009.

[Joy11] J. M. Joyce, *Kullback-leibler divergence*, pp. 720–722, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[Kel19] J. S. Keller, *Measurements of ttH and tH production at ATLAS + CMS*, `https://cds.cern.ch/record/2680951`, Jul 2019.

[Kor08] A. Korytov, *Introduction to elementary particle physics*, `http://www.phys.ufl.edu/~korytov/phz6355/note_A13_statistics.pdf`, September 2008.

[Mal20] J. Maly, *Automatic event recognition for Higgs boson detection*, `https://cds.cern.ch/record/2722145`, May 2020, Presented 25 Jun 2020.

[P$^+$19] A. Paszke et al., *Pytorch: An imperative style, high-performance deep learning library*, Advances in Neural Information Processing Systems 32 (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), Curran Associates, Inc., 2019, pp. 8024–8035.

[S$^+$14] T. Sjöstrand et al., *An Introduction to PYTHIA 8.2. An Introduction to PYTHIA 8.2*, Comput. Phys. Commun. **191** (2014), 159–177. 19 p, 45 pages.

[STY17] M. Sundararajan, A. Taly, and Q. Yan, *Axiomatic attribution for deep networks*, Proceedings of the 34th International Conference on Machine Learning (Doina Precup and Yee Whye Teh, eds.), Proceedings of Machine Learning Research, vol. 70, PMLR, 06–11 Aug 2017, pp. 3319–3328.

[SZA22] R. Shwartz-Ziv and A. Armon, *Tabular data: Deep learning is not all you need*, Information Fusion **81** (2022), 84–90.

[VSP$^+$17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, *Attention is all you need*, Proceedings of the 31st International Conference on Neural Information Processing Systems (Red Hook, NY, USA), NIPS'17, Curran Associates Inc., 2017, p. 6000–6010.

[VYM$^+$21] A. Valassi, El. Yazgan, J. McFayden, S. Amoroso, J. Bendavid, A. Buckley, M. Cacciari, T. Childers, V. Ciulli, R. Frederix, S. Frixione, F. Giuli, A. Grohsjean, Ch. Gütschow, S. Hoeche,

W. Hopkins, P. Ilten, D. Konstantinov, F. Krauss, and G. Stewart, *Challenges in Monte Carlo Event Generator Software for High-Luminosity LHC*, Computing and Software for Big Science **5** (2021).

[YA20] L. Yang and S. Abdallah, *On hyperparameter optimization of machine learning algorithms: Theory and practice*, Neurocomputing **415** (2020), 295–316.