ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení:	**Vatrt**	Jméno: **Martin**	Osobní číslo: **465862**
Fakulta/ústav:	**Fakulta informačních technologií**		
Zadávající katedra/ústav:	**Katedra aplikované matematiky**		
Studijní program:	**Informatika**		
Specializace:	**Znalostní inženýrství**		

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Optimalizace metod pro separaci tH(bb) signálů s využitím strojového učení

Název diplomové práce anglicky:

Performance Optimisation of tH(bb) Signal and Background Separation Using Machine Learning

Pokyny pro vypracování:

The particle accelerator at CERN produces a high number of so-called events, describing the collision products and their properties. The task is to recognize the events of interest automatically using the techniques of machine learning, and possibly deep learning, and thus increase the ratio of correctly identified events. The project is a part of the effort to analyse the properties of the Higgs boson. Simulated data are available for tH(bb) production and background reactions.
Tasks:
(1) Familiarise yourself with the existing code of the tH(bb) analysis.
(2) Develop machine learning algorithms to separate signal and background events.
(3) Implement and apply the algorithms on the provided data.
(4) Study the performance of the algorithms and compare them to the original analysis performance.
(5) Determine the feature importance ranking and study the effect of feature reduction on the performance.
Bonus task: study the uncertainty on the signal and background separation.

Seznam doporučené literatury:

Jméno a pracoviště vedoucí(ho) diplomové práce:

doc. Dr. André Sopczak katedra softwarového inženýrství FIT

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

Datum zadání diplomové práce: **13.04.2022** Termín odevzdání diplomové práce: ______________

Platnost zadání diplomové práce: ______________

doc. Dr. André Sopczak	Ing. Karel Klouda, Ph.D.	doc. RNDr. Ing. Marcel Jiřina, Ph.D.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	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

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Performance Optimisation of tH(bb) Signal and Background Separation Using Machine Learning

Bc. Martin Vatrt

Department of Applied Mathematics
Supervisor: doc. Dr. André Sopczak

February 16, 2023

Acknowledgements

I want to thank my supervisor doc. André Sopczak for his kindness and help whenever it was needed, over the whole scope of this thesis.

Declaration

In Prague on February 16, 2023 . . . .

Citation of this thesis

Vatrt, Martin Vatrt. *Performance Optimisation of tH(bb) Signal and Background Separation Using Machine Learning.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2023.

Abstrakt

Studium Higgsova boson a jeho interakcí s ostatními částicemi bylo v posledních letech jedním z hlavních témat částicové fyziky. Od jeho objevu v roce 2012 byla vykonána řada experimentů za účelem bližšího porozumění jeho fyzikálních vlastností.

Tato práce se zaměřuje na interakci Higgsova boson s top quarkem, pravděpodobně nejhmotnější elementární částicí Standardního modelu částicové fyziky. Jsou využity machine-learningové algoritmy k odfiltrování všech nežádoucích procesů zaznamenaných detektorem, aby bylo dosaženo vyšší senzitivity cílového procesu.

K tomuto účelu je optimalizována řada machine-learningových modelů pomocí různých optimalizačních strategií. Poté co je nalezen model s nejlepšími kvalitami, je provedena řada statistických testů pomocí TRExFitter frameworku.

Střední hodnota mediánu intenzity signál procesu se zahrnutím statistických nejistot, byla 3.86. Po zahrnutí systematických nejistot byla střední hodnota mediánu 6.35.

Klíčová slova Machine-learning, CERN, Higgsův boson

Abstract

The study of Higgs boson and its interaction with other particles has been one of the main topics of particles physics in the last years. Since its discovery in 2012, many experiments have been carried out in order to understand its physical properties in detail.

This thesis focuses on an interaction of a Higgs boson with a single top quark, possibly the heaviest elementary particle of the Standard Model of particle physics. It uses machine-learning algorithms to filter out all unwanted processes recorded by detector to get the highest sensitivity of our target process.

For this purpose, multiple machine learning models are optimized with different optimization strategies. After the best quality model is obtained, a serie of statistical tests is performed with the TRExFitter framework.

The expected median value of signal strength obtained in this thesis with inclusion of statistical uncertainties was 3.89. After the inclusion of systematic uncertainties, the resulting expected median value was 6.35.

Keywords Machine-learning, CERN, Higgs boson

Contents

List of Figures

List of Tables

Introduction

The European Organization for Nuclear Research, better known as CERN [1] (derived from *Conseil Européen pour la Recherche Nucléaire*, translated as European Council for Nuclear Research in French) is one of the biggest scientific workplaces in the world situated on the Franco–Swiss border northwest of Geneva. It was founded in 1954 for the purpose of answering some of the most fundamental questions of physics, like what is the universe made of and how it works.

One of CERN's most famous facilities is the Large Hadron Collider [2] (LHC), which is the largest and the highest-energy particle accelerator in the world. It is built up to 175 meters below the ground and consists of a circular pipe with a circumference of 26659 meters where protons are accelerated to speeds close to the speed of light and then collide inside one of the detectors, where physical data from the collisions are collected.

The LHC's biggest detector is called ATLAS [3]. It is a general-purpose detector that investigates a wide range of physics, from the search for the Higgs boson to particles that could make up dark matter.

In 2012 [3], one of the biggest accomplishments of modern physics was made using ATLAS when a particle with properties corresponding to the Higgs boson was observed. Since then, further experiments have been done to investigate the properties of the Higgs boson to see whether it behaves as it is described in the Standard Model of physics or if the Standard Model needs to be extended.

This thesis focuses on the production of Higgs boson inside of the *tH(bb)* process. It uses machine-learning algorithms to separate this process from so-called background processes, that are not the subject of this analysis and which make up unwanted noise in the recorded data.

Chapter 1

Standard Model of Particle Physics

One of the first things we are taught in physics lessons at school is that, everything around us consists of very tiny particles called atoms, and that every atom consists of even smaller particles called protons, neutrons and electrons. Another concept that we learnt about is the presence of fundamental forces that surround us in our everyday lives: gravity, which attracts everything around us to the center of the earth and which holds all the planets in our solar system together; and electromagnetism, which we can experience when we for example put two magnets together or when we create an electric charge while combing our hair.

As our physics knowledge started to grow, we learnt that these electromagnetic forces are involved in much more - that these are the forces, which hold everything around us together - molecules in solid objects, atoms inside the molecules and also electrons and protons to form an atom.

This was some of the basic knowledge we obtained from elementary school or high school. However, in modern physics, it is already well-known that there are many more elementary particles to bear in mind. All of them are described in the Standard Model[4] [5] of Particle Physics, which has been developed over the second half of 20th century through the collaboration and hard work of a large number of scientists from different parts of the world. Main components of this model are summarized in figure 1.1.

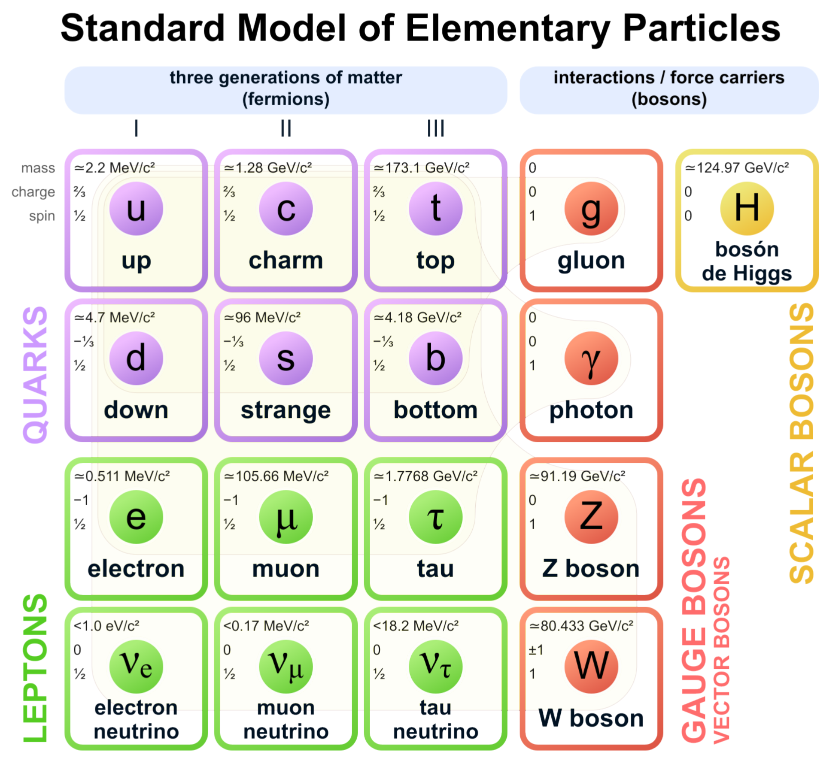


Figure 1.1: Scheme of elementary particles.

In the Standard Model of Particle Physics, there are two groups of elementary particles: quarks and leptons. Each group consists of six particles, which are divided in pairs into three categories (also called generations), based on their weight. The first generation consists of the lightest and the most stable particles which make up all of the physical world around us. The second and the third generation consist of particles that are heavier and less stable. These particles decay very quickly into more stable ones.

Quarks in the first generation are called Up quarks and Down quarks. These quarks are combined into protons and neutrons with so called strong nuclear forces and make up most of the nucleus weight.

Quarks of the second generation are called Charm quark and Strange quark. These particles can be found in hadrons, which are subatomic particles made of quarks.

Quarks of the third generation are called Top quark and Bottom quark. Top quark is the most massive of elementary particles and also stays in the center of our analysis.

Leptons are particles which do not undergo strong nuclear force and therefore exist as single entities. First generation leptons are electrons, which have negative electromagnetic charge and make up an electron shell of atom. The other particle from the first generation is called a neutrino, which, as its name suggests, is electrically neutral.

Leptons of the second generation are called Muon and Muon neutrino. Leptons of the third generation are called Tauon and Tauon neutrino.

The Standard Model also describes two more elementary forces, besides well-known gravity and electromagnetic forces - weak and strong nuclear force. The strong nuclear is very important, because not only that it holds quarks together to form protons and neutrons, but it also bounds protons and neutrons to form atoms nucleus. On the other hand, weak force is involved in many radioactive effects and also nuclear fusion, which takes place in the Sun.

Another important knowledge is that the weak force, strong and electromagnetic forces result from an exchange of force-carrying particles called bosons. These are considered as another group of elementary particles.

The weak nucleus force is carried by W and Z bosons, the strong nucleus power is carried by gluons and the electromagnetic force is carried by photons.

The forth fundamental force, gravity, should be carried by gravitons, however this particle has not been found yet. For particle physicists, it is still a remaining challenge to mathematically fit gravity into the Standard Model.

1.1 Higgs boson

The Higgs boson [6] is a particle form of the Higgs field, which is a fundamental field from which other elementary particles gain mass. The Higgs field exists in every point of the entire universe and every particle interacts with the Higgs field differently. The stronger is the interaction of a particle with the Higgs field, the heavier the particle is. For example, photon does not interact with Higgs boson at all and therefore has no mass, while the heaviest elementary particle, the Top quark, has a strong interaction with the Higgs field. Because of that, physicists believe that by studying its interaction with the Top quark, more can be learnt about the properties of the Higgs boson.

Chapter 2

Experimental and simulated data

2.1 Signal and background events

At CERN, any collision of two protons, which is either simulated or detected in a real experiment, is called *event*. From machine learning perspective, one event just simply corresponds to one row in a dataframe.

Among large number of different processes, there are two main categories we separate them into: *signal* and *background* processes. Signals are the processes we are interested in and which we want to separate from background processes. Backgrounds are processes that we want to reduce as much as possible. In other words, signals correspond to positive class and backgrounds to negative class in our analysis. Typically, the number of signal events is much smaller than the number of background events.

There can be also multiple processes considered as signals/backgrounds. In the case of this thesis, there is only one signal process called *tH(bb)* (abbreviated to *tH*) and multiple background processes.

2.2 Experiments at LHC

At the LHC, experiments are conducted in such a way, that two beams of protons are accelerated to the speeds close to the speed of light, both in the opposite directions. When a sufficient speed is achieved, these protons collide at an interaction point situated in the ATLAS detector.

The beams in fact are not one fluent line of protons, but instead are separated into small bunches of protons, which collide every every 25 ns. Number of proton-proton interactions per one bunch crossing is about 50.

When protons collide, they are destroyed and many new particles and antiparticles are created. There is a large number of particles that can be produced after the proton collision and the products can be even heavier than the original protons. Nonetheless, as was mentioned in the previous section, these newly created heavy particles decay very quickly into lighter particles and eventually into conventional stable particles, that can be typically observed in a form of jets, which are a narrow cone of hadrons and other particles (hadrons are compounds of two or more quarks held together by a strong force 1). There are also certain particles, that do not dissolve into the stable state before they are recorded, such as muons and antimuons.

These decay products are then recorded in the detector, followed by a so-called event reconstruction, which aims to reconstruct the whole sequence of created and then dissolved particles since the protons collided. This is a very difficult task, since after the collision, very high energetic particle is almost immediately dissolved into approximately 15 heavy particles, that are eventually decomposed into about 5000 charged particles visible by a detector.

2.3 Monte Carlo event simulations

For a given experiment at the LHC, there is typically an expected outcome based on Standard Model predictions. For example, we can estimate how many proton-proton collisions occur and what will be the frequencies (yields) of different processes.

So before every experiment, there is a Monte Carlo simulation based on the Standard Model, which simulates the whole experiment and shows, what outcome we should expect from an experiment with given parameters. This includes all the properties of final products that are recorded in detector, frequencies of different signal/background types etc.

After the experiment is performed, we can compare the results we obtained with results that were predicted by the simulation. For example, as we reconstruct our events and find out what were the frequencies of different signal/background types, we can make a comparison with what frequencies were predicted by Standard Model. If there is a big disagreement between the Standard Model and our results from the experiment, we can assume that the Standard Model does not describe the process correctly and certain modifications needs to be done. Such discovered disagreements then attribute to what is called physics Beyond the Standard Model.

It is also important to note, that because we are working with a simulator, we can of course generate more samples for the event we are interested in. This can be useful for the machine-learning model, which typically needs to work with as many samples as possible in order to have good qualities. To compensate the fact, that we generated more events than the Standard Model predicts, we take advantage of the so called event-weighting, where we evaluate the performance of our model on weighted samples.

This will be also the case in our analysis. Because the expected number of our signal events is only 73.2, this would not be sufficient amount to train a good machine-learning model. For $tt+b$ jets process, which correspond to production of top quark-antiquark pair, that are eventually dissolved into so-called b-tagged jets (jets originating from bottom quarks), the expected number of events is 59696, which is about 815 times more than for signal samples. More details on the event weighting are explained in section 3.2.

Chapter **3**

tH(bb) process

Since the top quark is the heaviest elementary particle, almost 50% heavier than Higgs bosson, it is expected to have a strong interaction with the Higgs field. This interaction with the Higgs boson is also called *Yukawa coupling* and has been one of the main research subjects for physicists in recent years.

One of the biggest accomplishments in the Higgs boson research since the discovery of the Higgs boson was an observation of the top quark pair coupling with the Higgs boson - the *ttH* process. One of the reasons why this discovery was so significant is the fact, that this process is very difficult to be observed.

In fact, the *ttH* coupling with a Higgs boson is very rare and accounts only for about 1% of all Higgs productions. Considering that the Higgs boson production itself is already very rare and was observed just recently in 2012, it is easy to imagine, how difficult it is to observe this process.

However, the double top pair is not the only production mode of Higgs boson associated with top quark. There is also a process, when a single top quark interacts with the Higgs boson - the *tH* process.

Beside the presence of top quark and the Higgs boson, the *tH* event topology is characterised also by an occurrence of either the down or up quark (called spectator quark), and an extra bottom quark [7]. Feynman diagrams of this process are shown in figure 3.1. Note that the x-axis of this diagram corresponds to time and the y-axis to space.

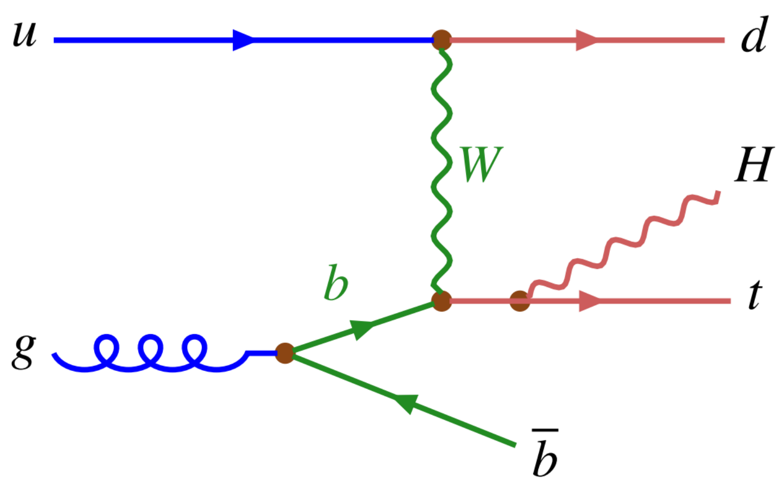


Figure 3.1: Feynman diagram of the tH(bb) process.

There are different channels of the *tH* production depending on the Higgs boson decay products. Each channel is studied by different scientific groups.

For example, if the Higgs boson decays into bosons or heavy leptons, which afterward decay into light leptons, it is the case of the *tH* multilepton channel, also abbreviated to *tH(ML)* [7].

If the Higgs boson decays into the pair of bottom quark-antiquark ($H \to bb$), we talk about the *tH(bb)* process, which is the main focus of this analysis.

3.1 Event selection

Because in the real experiment there are typically billions of recorded events and our signal process makes up only a tiny portion of them, it is necessary to filter out events, which have a very low probability to be a signal.

For this purpose, multiple selection criteria are used. They are typically applied in a different phase of ntuple production, however, each one of them should be applied before the machine-learning analysis.

Table 3.1 lists the selection criteria used in the *tH(bb)* analysis.

Pre-selection cuts
Exactly 1 tight (PLImprovedVTight) trigger-matched lepton with $p_T > 27$ GeV
No reconstructed hadronic τ
At least 3 b-tagged jets (DL1r tagger at 70% W.P.)
$E_T^{miss} > 25$ GeV
Veto for events with ≥ 5 jets and ≥ 4 b-tagged jets (a.k.a., ttHbb veto)

Table 3.1: Summary of the cuts applied to define the pre-selection region.

The first condition means that there needs to be at least one lepton recorded with a high quality. It is based on the fact, that the top quark always dissolves into W-boson, which then dissolves into neutrino and lepton, that can be recorded in the detector.

The second condition puts a veto on events, that are subject of research for different scientific group that investigates the *tH* multilepton process (*tH(ML)*), so there is no overlap between both analyses.

The third condition means that there needs to be at least 3 recorded jets coming from bottom quark dissolution. It is because the top quark is dissolved into a bottom quark, from which b-tagged jets are created.

E_T^{miss} is a minimum transverse energy expected for the process.

Similarly to the second condition, the fifth condition is used to ensure orthogonality with the ttH(bb) analysis, which is investigated by a different scientific group.

3.2 Weight calculation

Because the data we use for training are generated by a simulation, they do not reflect the proportions of signal and background events in the actual experiment. In reality, there is an expected number of events for each signal/background type. These expected values can be very different for each process, because some interactions have a high probability to take place and some very low. For this reason, event weights are used for evaluation of the model.

For event weight calculation, following formula is applied:

$$W_E = \frac{LX_{sec}\prod w_i}{\mathrm{w_T}} \tag{3.1}$$

The meaning of the used variables is following:

- L (luminosity) - number of colliding protons.
- X_{sec} (cross section) - theoretical probability of protons collision.
- w_1 ... w_5 - correspond to parameters. *weight_bTagSF_DL1r_Continuous*, *weight_pileup*, *weight_jvt*, *weight_forwardjvt*, *weight_leptonSF*. These are additional weight parameters which need to be taken into account.
- w_T (*totalEventsWeighted*) - normalization factor derived from total number of simulated events in the ntuple.

3.3 Process yields for the analysis

Table 3.2 summarizes all processes used in the analysis - number of samples before the preselection, number of samples after the preselection, yields and ntuple ID for each process:

Type	Dataset id	MC samples	Yields
tH	346676	42956	73.2
tt+b jets	410470	495366	59696.8
tt+c jets	410470	266707	33320.7
tt+light jets	410470	848249	105452.9
ttH	346343	1702043	1586.6
ttZ	410156	140704	840.6
ttW	410155	91813	259.0
tZq	410560	18855	169.2
tWZ	412119	2048	2.5
single t+W	410647	36948	5803.4
single t+t	410644	73972	3197.5
single t+s	410645	9668	257.7
WZ	3641XX	82141	4479.9
VV	36425X	9266	283.1
non-prompt	–	275569	5084.9

Table 3.2: Yields, Monte-Carlo samples and dataset id for each process type.

The majority of background is made of tt production. This makes sense, because the top quark in the tt process has exactly the same decay products as the top quark in the tH process, so they both pass the selection criteria, that are based on top quark decay products.

It is also obvious, how difficult it is to identify the tH process, as it makes up only a small portion of the top quark production. And even if we ignore the other top quark productions, there would still be different types of processes that can be interchanged with the tH process.

That is why the machine-learning algorithms are used for further background filtration.

3.4 Features used for the analysis

The list below shows, which variables are used for training and evaluation of machine-learning model. Each variable name correspond to a name in the root file, from which it is read:

- *njets_CBT5*
- *nnonbjets*
- *sphericity*
- *aplanarity*
- *nonbjets_eta*
- *rapgap_top_fwdjet*
- *fwdjets_pt*
- *chi2_min_DeltaEta_tH*
- *tagnonb_eta*
- *tagnonb_topb_m*
- *nfwdjets*
- *chi2_min_tophad_m_ttAll*
- *rapgap_maxptjet*
- *inv3jets*
- *nbjets*

- *chi2_min_toplep_pt*
- *nonbjets_pt*
- *chi2_min_deltaRq1q2*
- *chi2_min_Whad_m_ttAll*
- *leptons_charge*
- *foxWolfram_2_momentum*
- *chi2_min_Imvmass_tH*
- *chi2_min_bbnonbjet_m*
- *chi2_min_higgs_m*

Chapter 4

Machine-learning

4.1 Bias-variance tradeoff

In machine-learning, there is a famous mathematical formulation of expected error on test set which is called ”bias-variance tradeoff” [8][9]. This formula applies for any model and helps us better understand what is the source of inaccuracies we get on our test data.

Let us assume that we use mean of squares for error calculation. Then the formula is following [10]:

$$\underbrace{E_{\mathbf{x},y,D}\left[(h_D(\mathbf{x}) - y)^2\right]}_{\text{Expected Test Error}} = \text{Variance} + \text{Noise} + \text{Bias}^2$$

$$\text{Variance} = E_{\mathbf{x},D}\left[\left(h_D(\mathbf{x}) - \bar{h}(\mathbf{x})\right)^2\right]$$

$$\text{Noise} = E_{\mathbf{x},y}\left[(\bar{y}(\mathbf{x}) - y)^2\right]$$

$$\text{Bias}^2 = E_{\mathbf{x}}\left[\left(\bar{h}(\mathbf{x}) - \bar{y}(\mathbf{x})\right)^2\right]$$

$h_D(x)$ - output of our machine-learning algorithm h trained on dataset D

$\bar{h}$ - expected classifier. This is a theoretical model whose ouput and expected output of our machine-learning algirthm trained on all possible datasets drawn from a distribution of our input variables. Even though this model assumes averaging infinitely many classifiers across infinitely many training sets, it can be estimated by taking large number of instances of our machine-learning model and training each of them on different subset of a training dataset, that should be large enough to represent the real distribution input variables well.

$\bar{y}$ - expected value of y. Since the expected value is computed from the real distribution of variables, in this case the variables from our training dataset, we typically do not know what is the value of $\bar{y}$. This can be however also estimated by averaging outputs for each datapoint in a dataset which is large and representative enough to reflect the real distribution of input data.

So we can see, that we can decompose the error on training into 3 components. First component is called bias, second is called variance and the third one is called noise.

Bias is an inherent error, which would be present even with we had infinite number of training data. Bias reflects how flexible the model is and how complex decision regions it is able to create.

Variance reflects how much the classifier changes for different training datatets. In other words, it shows how sensitive the model is to changes in training dataset.

Noise reperesents the deviations in training data from expected values. Because typically we are not able to take into account everyting, that has an impact on target variable, this error is present regardless of how good our machine-learning model.

So the main conclusion of this section is, that the error on our training data can always have a different cause. When a model has high variance, it typically learns the training points too precisely, so it has low generalization ability and therefore a high test data error. This problem is called overfitting. When a model has a high bias, it is not robust enough to properly learn all the important information that is in data and make good predictions. This problem is called underfitting.

The relationship between bias and variance is, that when we try to decrease bias the variance increases and vice versa. For this reason, in most of the machine-learning tasks we are trying to find the best trade-off between bias and variance. So we want to get a model, which has a sufficient complexity to learn information from data, but also which does not have stick to the training data too much and doesn't lose an ability to generalize. For this goal, there are two different approaches described in the following sections: bagging and boosting.

4.2 Decision trees

Decision tree is very simple machine-learning algorithm. It is basically a sequence of binary conditions, based on which data points are classified.

Each branch splits on a certain feature and certain threshold which, that are determined during training. Because finding the best tree minimizing train error is NP-complete problem, greedy algorithms are most often used for training instead. One of the most popular, ID3 algorithm, is based on choosing the feature and the threshold, which after splitting minimize the entropy of target values in resulting sets (for classification).

Decision tree is an example of a model with high variance problem. Training on two different datasets typically gives two totally different trees. For this reason, decision tree is more often used as base algorithm for ensemble methods which are described in next sections.

4.3 Bagging

Bagging is an example of ensemble method, which uses certain number of models with high variance and tries to combine them to create a low variance model. As a base model, decision trees are chosen most often, because as it was mentioned in the previous section, it is a typical example of a model with very high variance. In this case we speak about model called random forest.

In the formula 4.1 we see that in order to reduce variance, we want our model to be as close to the average model as possible. Because average model returns expected output of the same classifier trained on infinitely many different training samples, we can simulate this by creating large number of instances of the same model (in our case trees) and train each one of them on a different subset of our training data.

In the datasets creation, there are two randomization techniques used: first is randomly choosing samples from original dataset with replacement (bootstrapping) and the second is randomly selecting certain number of features for each tree. Most common number of features for each tree is $\sqrt{2}$.

When random forest is succesfully trained, we make prediction of the random forest simply by averaging outputs of all trees.

4.4 Gradient boosting

In gradient boosting, the effort is to make a strong learner from combination of weak learners. Weak learner is a model whose predictions are a little bit better than random guessing. This means the task is opposite than in the case of bagging - we have a very biased model from which we want to build a model whose bias is very small.

4.5 XGBoost

In XGBoost, the goal is to minimize error on train set with a serie of CART trees. In each step, new tree f_t is constructed to improve prediction previous prediction $\hat{y}_i^{(t-1)}$ and minimize the loss over all training samples for loss function l. In addition to that, regularization term depending on the structure of tree f_t is added:

$$\text{obj}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \omega(f_t) + \text{constant}$$

In order to get formula suitable for optimization, Taylor expansion of the loss function up to the second order is used:

$$\text{obj}^{(t)} = \sum_{i=1}^{n} [l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \omega(f_t) + \text{constant}$$

where g_i and h_i are:

$$g_i = \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$$
$$h_i = \partial^2_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$$

Since first and third terms in formula are constants, they do not change a location of the minimum of the formula, so they can be removed. Then the objective function has following form:

$$\sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \omega(f_t)$$

Since f_t is a tree, it can be written in following form ($q(x)$ is a function assigning a leaf index to data point x):

$$f_t(x) = w_{q(x)}$$

In XGBoost, regularization term has following form (T is the number of leaves in a tree):

$$\omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

With a new definitions of a tree f_t and regularization terms, the objective can be reformulated as:

$$\begin{aligned} \text{obj}^{(t)} &\approx \sum_{i=1}^{n} [g_i w_{q(x_i)} + \frac{1}{2} h_i w_{q(x_i)}^2] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \\ &= \sum_{j=1}^{T} [(\sum_{i \in I_j} g_i) w_j + \frac{1}{2}(\sum_{i \in I_j} h_i + \lambda) w_j^2] + \gamma T \end{aligned}$$

where $I_j = \{i | q(x_i) = j\}$ is the set of indices of data points assigned to the j-th leaf.

If we define $G_j = \sum_{i \in I_j} g_i$ and $H_j = \sum_{i \in I_j} h_i$, then formula (ref) will have following compressed form:

$$\text{obj}^{(t)} = \sum_{j=1}^{T} [G_j w_j + \frac{1}{2}(H_j + \lambda) w_j^2] + \gamma T$$

To minimize this formula, we need to find an optimal value for each w_j in the sum. This can be done for each addend independently in following way:

$$w_j^* = -\frac{G_j}{H_j + \lambda}$$

Now we know, how to setup the weights of a tree, so the loss is reduced the most. Remaining task is to find an optimal leaf index for each datapoint.

Theoretically, we could evaluate all possible splits and see, for which one we had the highest loss reduction. This, however, is very computationally expensive. Instead, the tree is built from the lowest level, where each split is evaluated with formula(ref) with follwing formula:

$$Gain = \frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right] - \gamma$$

First component of the formula is a score of the left leaf, second one is a score on the right leaf and the third one is a score on current leaf. γ is hyper-parameter, which is used for regularization. The higher this parameter is, the more conservative is the algorithm.

4.6 AdaBoost

In this thesis, two boosting algorithms from scikit-learn library [11] are used - AdaBoost [12] and GradientBoostingClassifier, which implementation is in-detail described in [**?**].

AdaBoost is based on iterative reweighing of training samples based on the previous prediction weak classifier, which is typically a decission tree.

At the beginning of the algorithm, weight of each training sample is set to $\frac{1}{N}$, where N is a total number of samples. Then a decision tree is trained on these weighted samples, after which it assigns a prediction to each one. Then the weights of these samples are reweighed - if they were incorrectly classified, their weight is increased, otherwise their weight is reduced. This procedure continues, until the stopping criteria is met. The final prediction of a trained is then made by the weighted sum of all weak learners.

4.7 Neural network

Neural networks [13] is a very large category of machine-learning models, that can be used for many different purposes, from image classification, to music generation. For classification of tabular data, multi-layer perceptron is one of the most popular models.

The main entity of multi-layer perceptron is called artificial neuron. It typically has multiple number of inputs x_i with associated weight w_i, and one output. Formula for output of a neuron is:

$$\text{out} = f(\sum(w_i x_i) + w_0)$$

f of is called an activation function. The most commong choices of activation function are sigmoid, tanh and relu.

Neurons are organized into layers. MLP has 3 types of layers - input layer, hidden layer/s and output layer. Input layer accepts input features and pass them into hidden layers. Final outputs are then obtained from output layer. Neurons between each layers are fully connected - that means that the output of each neuron from previous layer serves as an input for neurons in next layers.

Neural networks use specific training procedure called backpropagation. Whenever a set of training samples are evaluated and loss function computed, then the gradient for each weight is computed from output layer backwards to input layer with the usage of chain rule, which makes the gradient computation very easy.

Chapter 5

Frameworks

5.1 ROOT Framework

ROOT [14] is a dataprocessing framework created at CERN for the purpose of high-energy physics research. It is daily used by scientists from all over the world for discoveries which may help us understand the most fundamental principles of the nature.

In ROOT, data are stored in binary compressed format called ROOT file, which is also efficient for parallel applications. ROOT also provides many statistical and mathematical tools for data analysis. It also has its own machine-learning framework called TMVA, which is described in the next section.

5.1.1 Transformation of ROOT files to csv files

Even though ROOT provides many useful tools for data analysis, it can not be used together with some other machine-learning libraries, like sci-kit learn or Optuna, which might be for certain tasks more useful. To be able to analyze data with both ROOT and other machine-learning frameworks, it was necessary to find a way how to convert ROOT files to csv format.

The basic entity for data storing in ROOT is called *tree*, which is basically a dataset stored in a ROOT file. Each tree contains structures called *branches*, which correspond to the column of a dataset. In one ROOT file, there can be multiple trees and also multiple directories where trees are stored, so it is always necessary to access the right tree before we start working with it. Functions for opening root files and accessing trees are: `TFile::Open` and `TFile::Get)`.

After we access the tree it is necessary to load data from it so we can then store it in a csv file. To access data in a tree, we first need create an array of variables, where each variable will be linked to each branch. Also each variable data type need to correspond to branch data type. Some branches can also contain vectors instead of single number per entry, so for these branches, we need to create an array of vectors. Finally, when we know which branches we want read from and create variables of the correct type, we link each branch to the corresponding variable with `TTree::SetBranchAddress`.

After linking variables with branches, we can access data from each row by `tree->GetEntry(i)`. This function loads data from entry i and stores them in variables we defined in previous step. In this way, we can loop over the whole dataset and save the values from the tree into a csv file.

5.2 TMVA

TMVA (Toolkit for Multivariate Data Analysis) [15] is a framework built in ROOT created for data preprocessing and machine-learning tasks. It offers large variaty of machine-learning algorithms for both classification and regression, like decision trees/forests, SVM, neural networks and much more. Beside these build-in methods, it is also able to work with Keras neural network models.

TMVA also provides many algorithms for transformation of input data. Beside the standard ones, like PCA and normalisation, there are also transformations that are not so common, like Uniformization and Gaussination. Those transformations are described below.

After finishing the analysis, TMVA also provides many graphical outputs, which include ROC curve of the classificator, correlation matrix of the input variables, distribution of data after transformations and more.

Since the work my thesis is based on uses TMVA for data analysis, my first task was understand how TMVA in the provided code works and how it could be improved for better classification performance.

5.2.1 TMVA workflow

In TMVA, first a `TMVA::Factory` object is created, which defines the name of our analysis, output file and what type of analysis we want to do.

The next we define which variables we want to work with. In the provided code, this task was done by first creating `DataLoader` object, where we first defined train and test trees, which were previously loaded from root files. Then we define all the variables we want use with `AddVariable` method. Since a single tree may contain multiple variables which are not useful for the analysis, this step in necessary.

After this, test and train trees are defined. With `DataLoader` object created in the previous step, this can be easily done with `PrepareTrainingAndTestTree`.

The next step is to book methods which we want to use for the analysis. This is done with facory. In `BookMethod` method we define what model we want to use for the analysis and what data we want to work with. We can also define which transformations we want to apply for the data.

The final step is to train, test and evaluate all the booked methods. This is simply done by methods `factory.TrainAllMethods`, `factory.TestAllMethods` and `factory.EvaluateAllMethods`.

The next section gives a detailed description of most important functions which were used in the work.

5.3 Optuna

For the purpose of finding hyperparameters of neural networks, framework called Optuna citeoptuna was used. Optuna is an open source optimization tool which can be used for any kind of parameters for most of machine-learning models - from logistic regression to complicated ensemle models. In order to do so, it uses the following algorithms for hyperparameters sampling:

- TPE sampler - bayesian optimization based on kernel fitting
- NSGA - multiobjective evolutionary algorithm
- CMA-ES - meta-heuristics algorithm for continuous space
- PartialFixedSampler

5.4 TRExFitter framework

TRExFitter is a plotting framework widely used in ATLAS analyses. Its main feature is a stacked histogram, which provide clear overview of background composition in a given experiment with associated uncertaintity for each bin. Another helpful functionality is a calculation of the number of expected events for each signal/background type based on a given preselection criteria and per-event weight formula. Its input file format is a root ntuple.

Futhermore, it allows to calculate the significancefor a given signal and background set based on the machine-learning model output distribution.

It also calculates the expected uncertaintities of the measurement with respect to the Standard Model expectation.

The TRExFitter script is organized into sections parts:

- Job
- Fit
- Limit
- Sample
- Region
- Systematic
- Norm factor

Job, Fit, Sample and Region sections are mandatory Trex-fitter run, while systematic and Norm Factor sections are optional.

In the following chapters each section and its most important parameters are explained. The first parameter is always the name of the block.

5.4.1 Job

In the Job block, parameters for the whole TRExFitter run are defined.

Most important parameters:

- InputFolder: location of the folder with root files
- NtupleName: name of the tree to read from a root file
- ReplacementFile: name and location of the replacement file. Replacement file contatins list of placeholders, which can be used inside certain config file parameters (for example in Selection parameter) and which are evaluated during the run of the script based on the replacement file definition. Placeholders always start with XXX
- Selection: basic selection criteria used for every event
- MCweight: per-event weight formula
- BlindingThreshold: maximum signal/background ratio for which real experiment data points are displayed. This ratio applies for each bin. If this value is excedeed, hashed grey area over whole bin is displayed.

5.4.2 Samples

This section defines, which root files should be loaded into the TRExFitter. This section typically contains multiple blocks, each starting with ”Sample” parameter, where each block represents one data file and how it should be used.

Parameters:

- Type: defines what kind of data block defines. Most typical values are SIGNAL, BACKGROUND and DATA for signal, background and real experiment data.
- Title: title shown in the plots for data loaded in current block.
- NtuplePaths: specific path for root file
- NtupleFiles: names of root files to be loaded. For one block, multiple root files can be loaded.
- Selection: additional selection criteria for loaded data

5.4.3 Regions

Each block of this section defines one final plot.

Parameters:

- Variable: name of the feature to be plotted. This parameter corresponds to the name of a branch in a root file. Range and number of bins is also defined in this parameter.
- Selection: additional event selection criteria. If we want to apply model output as threshold for events, it is typically used here.
- LogScale: boolean value determining if the Y axis should be logarithmically scaled

5.4.4 Running TRExFitter

The format or TRExFitter running command is following:

```
trex-fitter <action(s)> <config file> <options>
```

- action - sequence of actions which should be executed. For the purpose of this thesis, only sequence ”nd” was used, where ”n” stands for reading input root files and ”d” stands for drawing pre-fit plots.
- config file - name of the config to be used
- options - aditional options for plotting, can be left blank

5.4.5 Docker

For easier setup, TRExFitter was not provided in a form of source files for compilation, but is available as a docker image in Docker registry. The advantage of this implementation is that it is not needed to install all TRExFitter dependencies and spend time with compilation, instead it can be easily run as standalon container with all dependencies already installed.

For obtaining the docker image, it was necessary to login into into the repository with docker login command and than pull image with docker pull command. Then I ran the image with docker run command.

5.5 Python and other libraries

The whole machine-learning analysis was done in a Jupyter notebook environment with standard machine-learning libraries, like numpy, scipy, pandas, sci-kit-learn, etc. For the work with neural networks, the TensorFlow 2 library was used. For the XGBoost library, the last stable release was used over the whole scope of the thesis.

Deployment of the model into the tHbbskimmer framework was done in C++ language. The whole module was installed with CMake. For the loading and evaluation of trained models, XGBoost and TensorFlow C APIs were used. For easier use of TensorFlow C API, the CppFlow wrapper was used. The whole extension of the framework was carried out with the usage of the Git version control system.

Chapter 6

Significance

Significance is an evaluation metric specific to high energy physics. It is based on Gaussian approximation of Poisson distribution and is closely related to statistical proving of signal presence. The higher our significance is, the more likely we prove a signal presence in the statistical tests, that come as a next steps of our analysis.

6.1 Poisson distribution

Poisson distribution is a mathematical model, which describes the probability of number of k occurencies for event with given frequency μ. Real life example of Poisson model usage could be the calculation of probability, that we say ”Hello” k times during the day, if we know that we typically say ”Hello” with frequency μ.

Formula of Poisson distribution is:

$$P_k = \frac{\mu^k}{k!}e^{-\mu}$$

Poisson model is one of the most important probability models for this particle physics, because it can describe many types of physical quantities. It can for example describe the number of produced particles from certain interactions, frequency of energy ranges of detected particles etc.

It is also important to note that the sum of independent Poisson distributions with parameters $\mu_1, \mu_2, ...\mu_n$ also has Poisson distribution with parameter $\mu = \sum_i \mu_i$.

6.2 Central limit theorem and its consequences on Poisson distribution

Let X_1, X_2, ... X_n be sequence of independent and identically distributed (i.i.d.) random variables with finite expected value μ and finite variance $\sigma^2 > 0$. Let's also put $S_n = \sum_i = X_i$, then for $n \to \infty$:

$$\frac{S_n - n\mu}{\sqrt{n\sigma^2}} \sim G(0,1)$$

where G(0,1) is a Gaussian distribution with mean equal to 0 and variance equal to 1 (also called standard normal distribution).

For the mean and variance of lineary transformed random variable Z with finite expected value, following rules apply:

$$E(aX + b) = aE(X) + b$$

$$var(aX + b) = a^2 var(X)$$

This means that for n high enough, we can approximate S_n in following way:

$$S_n \sim \sqrt{n\sigma^2} G(0,1) + n\mu \sim G(n\mu, n\sigma^2)$$

In other words, sum of n i.i.d. random variables with finite mean μ and finite variance σ can be approximated with Gaussian distribution with mean $n\mu$ and variance $n\sigma^2$ (for sufficiently large n).

We already know, that Poisson distribution with given parameter μ can be understood as sum of Poisson distributions. It can be also considered as a sum of Poisson distributions with parameter $\mu_i = 1$ for every i. This means that for any natural number μ, we can approximate the Poisson $P(\mu)$ distribution with Gaussian distribution $G(\mu, \mu)$. This simplification will be very useful for significance calculation in the next section.

In practice, this convergence is obvious even for quite small cases of n. Even for n=5, the Poisson distribution already has a shape similar to Gaussian distribution. In the coppies of n = 100, Poisson distribution basically copies the graph of Gaussian density function.

6.3 Meaning of significance and its estimation

In high energy physics, the standard methodology of proving new discoveries is through hypothesis testing. In principle, we test if the hypothesis H_{0sb} of signal presence in our samples is more relevant than hypothesis H_{0b} of background only scenario. This means we compare two Poisson models - one with parameter μ_B (background only distribution) and the other with μ_{S+B} (signal + background distribution).

The comparison of models is done with the usage of p-value of an observed quantity n_o - if the probability of this observation under H_{0sb} is more likely than in the case H_{0b} hypothesis, we incline more to the hypothesis of signal presence.

This basically means, that when we apply the threshold of our machine-learning model and get number of signal and background events that are based on the Standard Model expectation, we want to have the composition of signal and background events such that we reject H_{0b} hypothesis on sufficiently high significance level.

The standard units for significance are percents. However in practice, significance is often measured as "number of standard deviations σs". This originate from the property of normal distribution, that distance of a given number of standard deviations from the mean is approximately equivalent to certain confidence interval for significance α.

We have already shown, that the Poisson distribution $P(\mu_B)$, which describes the background events, can approximated with Gaussian distribution $G(\mu_B, \mu_B)$. If we consider the signal events to be the main source of deviation from the expected value, we can calculate the significance with very simple formula:

$$\sigma = \frac{s}{\sqrt{b}}$$

where s is number of signal events and b is number of background events (understood as μ_B - expected number of background events according to Standard model) after the threshold cut is performed.

Naturally, the distance from the mean value using standard deviation gives an estimation of significance for two-tailed test, even though in our case we would be more interested in significance level of one-tailed test for maximum possible event number, because the number of signal events in our samples is always a positive value. This however is not a big complication, because in our case the critical values of one-tailed tests for maximum values are always less than critical values of two-tailed test, so the actual significance level we estimate is actually higher for one-tailed test, which is always the better case, than if our estimate would be too optimistic.

6.4 Evaluation of the model

After training the model on a training set, the task was to find an optimal threshold on which we would get the highest significance and at the same had a sufficient number of samples.

This was done by gradually reducing the number of samples based on model threshold *thr*. For each threshold *thr*, only samples greater than *thr* were considered. This was performed for the full range of thresholds from (0.0) to (1.0), with a step size of 0.005. On each threshold, the significance was calculated using $s/\sqrt{b}$ formula.

After the significance was calculated on the whole range, the highest value was used for comparison with other models.

To avoid statistical fluctuations, only the values of significance that was calculated on more than 1000 samples were considered.

Chapter **7**

Systematical uncertainties

7.1 Tests for the presence of signals

If we want to test the presence of a signal in our samples[16], we typically construct two Poisson models - one with parameter λ_b and the other with parameter $\lambda_b + \lambda_s$, where λ_b is expected background frequency and λ_s is signal frequency. Then for an observed value q, we calculate the p-value p_b of this observation for null hypothesis H_{0b}, that q comes from distribution P(λ_b). In the same way we construct the hypothesis H_{0sb} for P($\lambda_s + \lambda_b$) and calculate the p-value p_{sb}. If p_b is small enough that we can reject H_{0b} on sufficiently large confidence level α_1 and at the same time we do not reject the H_{0sb} hypothesis for predefined α_2, we assume that signal is present in our sample. The choice of α_1 and α_2 depends on experiment, for example for the Higgs boson discovery, 5σ was chosen as sufficient significance level.

7.2 Systematic uncertainties for tHq(bb) process

Beside the statistical uncertaintity, which arises from the imperfect measurements of physical quantity and which is present in all physical experiments, in LHC experiments there are usually other sources of uncertaintity which need to be taken into account called systematic uncertaintities. These are related to the reconstruction of physics objects in the detector, the techniques used to determine the expected size of the backgrounds and modelling of the signal and background process [7].

In the case of our experiment, the sources of uncertaintity are for example:

- Luminosity uncertainty - relative uncertantity on luminosity for 2015-2018 experiments is estimated to be 1.7%. This uncertantity applies for all processes modelled by Monte Carlo simulation.

- Electron and muon efficiency uncertainty - these uncertaintities arise from the reconstruction, identification and isolation efficiencies of the detector.
- Uncertainty related to Next Leading Order (NLO) corrections - this uncertainty comes from the fact, that the theory describing given process does not give accurate predictions, but only an approximation.
- Final State Radiation (FSR) uncertainty - after the collision of two protons and production of new particles, there is a chance that one of the particles radiates a different particle and thus the process does not correspond to casual schema of the process. Such and effect is reflected in FSR uncertainty
- Initial State Radiation (ISR) uncertainty - similar like in previous case, the difference is that the radiation occurs before the electrons collide and because of that, protons do not collide under energy they are expected

The first two cases are examples of uncertainties related to the properties of the detector, the rest of the uncertainties are related to modelling of the processes.

In our systematic uncertainties evaluation, only data coming from simulation are used, thus the detector uncertainties are not taken into account.

There is also another type of uncertainty that are denoted γ(bin x), that is associated with low statistics in a given bin due to the low number of samples. This uncertainty can be removed by rebinning the histogram, so all bins contain sufficient number of samples.

7.3 Binned profile likelihood

To be able perform statistical testing, we to need have a good estimate on the signal strength λ_s described at the beginning of this chapter. This estimate should also take into account all known uncertaintity sources, that are considered as nuisance parameters (NP).

For this purpose, TRExFitter provides binned profile likelihood functionality, which finds the best parameters based on the observed events using the log-likelihood minimization method. The likelihood is optimized over each bin of the input histogram. The likelihood function for this problem is following [17]:

$$L(\mathbf{n}, \boldsymbol{\theta}^0 | \mu, \boldsymbol{\theta}) = \prod_{i \in bins} P(n_i | \mu S(\boldsymbol{\theta}) + B(\boldsymbol{\theta})) \times \prod_{j \in n.p.} G(\theta_j^0 | \theta_j)$$

where μ is called the parameter of interest (POI) and expresses the multiplication factor for observed data, which correspond to standard model expectation. θ is the nuisance parameter and G is a Gauss distribution.

Chapter 8

Previous and current analyzes

In the next sections, the results of previous and ongoing analyzes are described, from the oldest to the newest ones. It is however important to note, that the results of some presented metrics are not directly comparable,

8.1 TMVA analysis with the neural network usage

This analysis was conducted by a 2021 CERN Summer Project attendant [18], whose goal was to improve prediction accuracy of signal data by testing different neural network configurations.

The chosen framework for this task was Keras combined with TMVA. The parameters that were used for the optimization were number of layers, number of neurons in each layer, number of epochs, batch size, activation function and optimization algorithms.

The performance comparison was done in following way: for each neural network architecture, AUC for different batch sizes and number of epochs was compared. The batch size values were following: 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000. The number of epochs for each batch size were: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20.

Some network architectures were also tested with different optimization algorithms (Adam and SGD) and activation functions (Relu, sigmoid and tanh).

For each training procedure, 7000 signal and 7000 background samples were used. These training samples were selected randomly with the same random seed, so there were the same data some for every run. Before each training loop, both signal and background samples were transformed in TMVA using the Gaussinization. The output of the network consisted of two output neurons followed by softmax function.

8.2 Compound model analysis

This analysis [19] was the main focus of Joint Institute for Nuclear Research physicist, who investigated the overall separation power with combination of multiple machine-learning models.

The methodology used in the project was based on the usage of separated neural networks, that were pretrained on some subset of variables for each signal/background type. These pretrained neural networks then served as an input for another neural network or some other machine-learning algorithm (for example Boosted decision trees), which provided the final output.

The advantage of this approach was that different feature subsets for each signal/background type could be used. This approach is generally useful in cases, when some processes do not share the same set of variables. Besides that, such a compound machine-learning model can have some interesting characteristics that can be to worthed study.

8.3 Current analysis

Current ongoing analysis [20] has been conducted by a scientific group in CERN and is the main subject of comparison for this thesis. It uses XG-Boost as a classification model and the foam algorithm for finding the optimal threshold.

Besides determining the AUC and significance, a full analysis with the usage of TRExFitter is performed. This includes calculation of statistical and systematic uncertainties, getting the fit results, ranking of systematic uncertainties etc.

Chapter 9

Thesis workflow

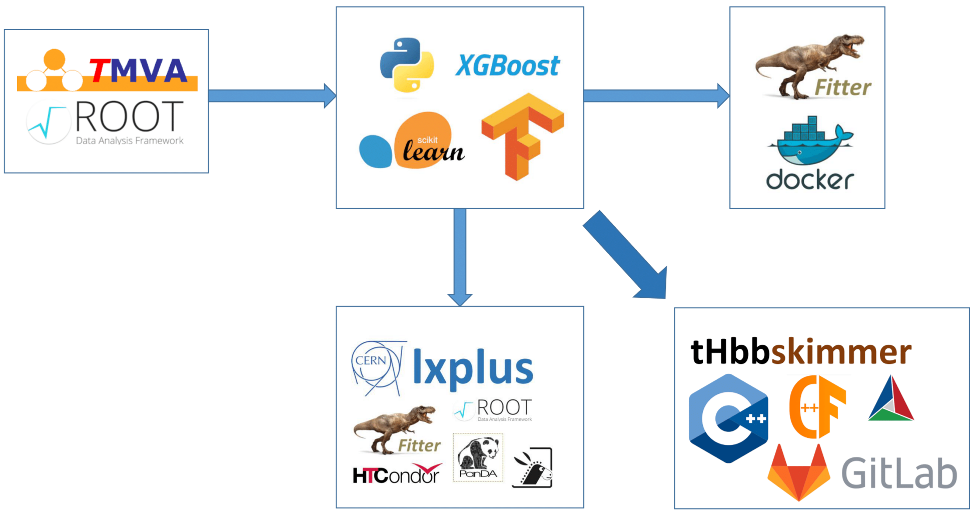


Figure 9.1: Scheme of different thesis phases.

Because the base analysis of the tH process was made in TMVA using Neural Networks, the first step was to utilize the provided code and study, how succesful are the other TMVA methods.

The TMVA framework however offered very small flexibility, so the next step was to convert the root files with both simulated and real-experiment data into csv format, so it would be possible to perform analysis also in other frameworks.

After the ntuples were converted into csv files, they could be easily loaded in Jupyter notebook environment using the pandas package and the analysis using Python libraries could be performed. This included: outliers removal, hyperparameters optimization, performance comparison of different models and all the other steps described in chapter 10.

After the model with best performance was found, it was time to to evaluate its outputs with TRExFitter framework. This could be carried out with two different procedures, one with inclusion of systematic uncertainties and the other without it.

Without the inclusion of systematic uncertainties, it was easier and faster to evaluate the model on a local computer. This procedure is described in chapter 11.

However, in order to get the complete analysis results and study how useful our model is for thorough tH analysis, it was necessary to also include the systematic uncertainties to the analysis. In this case, it was not so efficient to do the whole computation on local computer for various reasons. One of them was the size of systematic ntuples which was more than 300 GB. Instead, it was easier to work remotely on the CERN cluster called lxplus. This approach is described in detail in chapter 12.

As a final step, after we found an optimal model and explored how useful it is for *tH* analysis with TRExFitter framework, it was time to put it into production. This means deploy it into program called tHbbskimmer, which has been developed by a scientific team involved in tH process over the last years and which serves for a full tH analysis with inclusion of all systematic uncertainties and all background processes on the most recent data. This step is described in chapter 13.

Chapter **10**

Model selection

10.1 Dataset preparation

From table 3.2 it is obvious, that the number of simulated events for each process is much higher than the yields of each process that correspond to number of events that actually take place in a real experiment. Beside that, the ratio between the yields and the number of simulated events is very different for each process. For example, for *ttH* process there is 1702043 simulated events, even though the yields of *ttH* are 1586.6, so the ratio between yields and simulated events is $1702043/1586.6 = 1073.17$. On the other hand, the ratio between yields and simulated events for *ttL*, which is the most important background, is only $848249/105452.9 = 8.04$. Also the total number of simulated events is lower than for *ttH*, even when *ttH* is just a minor background.

For machine-learning model this means, that the main focus is on the backgrounds with a high number of simulated events, even when their frequency in real experiment is not necessarily high. It is because correct separation of these backgrounds decreases the total loss by the biggest amount, which is what most machine-learning algorithms aim for.

Therefore, it can be beneficial to somehow take into account the actual yields for training. First, the Monte-Carlo weights are used, from which the yields are calculated. This however brings some unpleasant effects.

Firstly, Monte-Carlo weights are related to properties of detector and some of them have negative values, which can not be used for training, only for testing. Since 36% of signal samples contain negative weights, it can not be ignored.

Another disadvantage is associated with the training itself and the resulting model. When Monte-Carlo weights are used, the chosen model usually has difficulties to start converging in early phase of the training. This is inconvenient if for example we want to set a criteria, such that the training should stop if there was no improvement in the last number of training epochs.

Then, after the training is successfully finished, we face another issue. For Monte-Carlo weighted training, most of the samples have a model output very close to zero.

Figure 10.1 shows graphs for an XGBoost model, which was trained with Monte-Carlo weights. The number of estimators was 500, maximum tree depth 1000 and learning rate 0.1. You can see that the signal and background events are rapidly reduced at a very small threshold values, which makes very unstable and spiky significance plot, which is the result of low statistics on these thresholds. In fact, if we wanted to apply condition, that there must be at least 800 background samples for significance calculation, we could work only on the interval [0.0, 0.01]

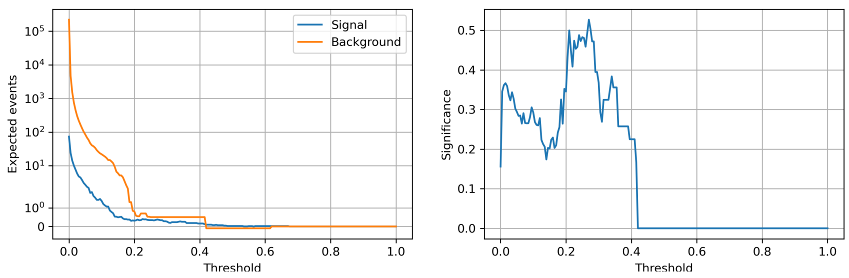


Figure 10.1: Signal/background reductions and significance plot for a model trained with Monte-Carlo weights.

Figure 10.2 shows the same plot for XGBoost model with the same parameters trained without weights. Signal and background events are reduced much more slowly and the significance curve with good statistics has a much wider range. In the previous case, we could of course transform the model output, so the samples are reduced more slowly, however in general it seems, that training without weights gives more stable results.

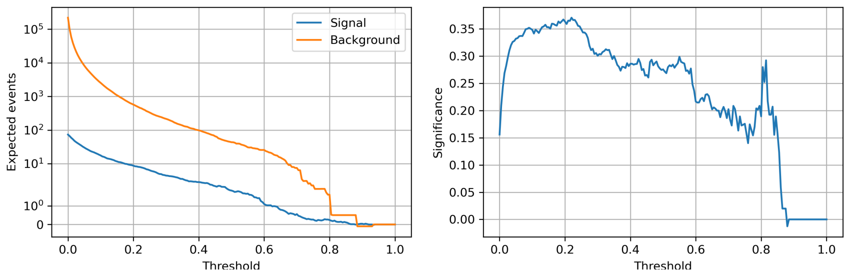


Figure 10.2: Signal/background reductions and significance plot for a model trained without weights.

The AUC for both models is 0.832. Best significance for first model is 0.361 on threshold 0.01. Best significance for second model is 0.358 on threshold 0.14. This means that the models trained with/without weights have a very similar performance.

10.2 Training and testing dataset preparation

Even though the training with weights seemed to have many disadvantages with a zero contribution to performance, using an information about the the frequency of events in the real experiment in some way still seemed to be a good idea. Instead of using weights, the decision was to create a dataset, which contains the events of different processes in the same ratio as their yields. This means that for a given process type t, the number of samples for training n_t was:

$$t_n = T\frac{t_y}{S_y} \tag{10.1}$$

where T is the size of training set, t_y are the yields for a given type and S_y is the sum of all yields used in the analysis. The only exception was for the tH process, which quantity was set separately in order to enhance the number of signal samples for training, which otherwise would be very small if calculated with this formula. A dataset created with this methodology is denoted as D_y. This notation is also used also in the next section.

The quality of training on such dataset was then compared to a more simple approach, where training samples are just randomly selected from all generated Monte-Carlo events. Training dataset created with this methodology is in the following sections denoted as D_{rand}.

When events are randomly sampled from all generated events, the probability of selecting a sample of type t simply corresponds to the ratio of generated Monte-Carlo samples for this type t_{MC} and sum S_{MC} of generated MC samples each type:

$$P(t) = \frac{t_{mc}}{S_{mc}} \tag{10.2}$$

For a large training dataset of size T, the number samples n_t for a given process t roughly corresponds to probability of selecting such a type multiplied by the size of training dataset:

$$n_t \sim TP(t) \tag{10.3}$$

In order to consistently have the same number of events for each type and also to avoid statistical fluctuations for low T, instead of random sampling, the number of samples for each type were calculated deterministically with the formula:

$$n_t = T\frac{t_{mc}}{S_{mc}} = TP(t) \tag{10.4}$$

Table 10.1 lists the number of samples and corresponding weighted sum of events for each process for D_y and D_{rand} datasets, with number of background events set to 100 000 and number of signal events is set to 20 000. The quality of training on these datasets is described in following sections.

	D_y		D_{rand}	
type	samples	weighted	samples	weighted
tH	20000	33.9	20000	33.9
ttb	27257	3289.5	19645	2372.4
ttc	15214	1902.6	10577	1322.0
ttL	48150	5975.6	33640	4177.2
ttH	72	0.1	6750	6.2
ttZ	383	2.2	5580	31.9
ttW	118	0.3	3641	10.2
tZq	77	0.7	747	6.7
tWZ	1	0.0	81	0.2
tW	2649	417.1	1465	228.7
single top-t	1460	64.3	2933	128.1
single top-s	117	3.1	383	10.2
WZ	2045	115.2	3257	189.4
VV	129	4.0	367	11.7
nonp	2321	45.0	10928	181.1

Table 10.1: Number of samples for D_y and D_{rand} datasets.

10.3 Hyperparameters optimization with Grid Search

One of the first steps of the analysis was a comparison of different machine-learning algorithms and exploration of their properties with a Grid Search algorithm.

Grid Search is one of the simplest methods for hyperparameters optimization. It trains and evaluates model for every defined combination of hyperparameters. It is very useful for initial investigation of model qualities, and also to have some intuition, which hyperparameters are important for training a good model and in which range their values should be.

Because the Grid optimization has already been done for neural network in previous analysis, only tree-based algorithms were analyzed with this method. Specifically, the following models were used:

- XGBoost
- GBDT - Gradient Boosted Decision Trees (scikit-learn implementation)
- AdaBoost (scikit-learn implementation)
- Random Forest (scikit-learn implementation)

All of these models were trained for binary classification problem, because training of binary classifier was much faster than training of multiclassifier. The performance comparison with multiclassifier is described in the next section.

The hyperparameters chosen for the optimization and their values were following:

- `n_estimators` $\in [50, 100, 200, 500]$ - number of trees built by the model
- `max_depth` $\in [1, 3, 5, 7, 15]$ - maximum depth of the trees
- `learning_rate` $\in [1.0, 0.5, 0.1, 0.05, 0.01, 0.001, 0.0001]$ (only for boosting algorithms)
- `criterion` $\in [gini, entropy, log\_loss]$ - function used to measure the quality of a split (only for Random forest)

The whole Grid optimization was done for 4 different strategies for training and testing dataset creations:

- Training on dataset containing 70% of all events created by method described by 10.4 (30 069 signal events, 1 765 057 background events). Testing on remaining samples (30% of all events - 12 887 signal samples and 756 463 background samples).
- Training on dataset containing 120 000 events created by method described in 10.4 (D_{rand} - 20 000 signal events, 100 000 background events). Testing on remaining samples (97% of all events - 22 956 signal samples and 2 421 527 background samples).
- Training on dataset containing 120 000 events created by method described in 10.1 (D_y - 20 000 signal events, 100 000 background events). Testing on remaining samples (97% of all events - 22 956 signal samples and 2 421 527 background samples).

The first method corresponds to a standard way of training and testing set preparation and is also widely used in many analyzes at CERN. The disadvantage of this approach however is that the training on such a large set takes a very long time and is not optimal for hyperparameters optimization, where model is trained multiples times. Because of this, only XGBoost model was trained and tested for all setups, as the training took much less time compared to scikit-learn models.

The motivation for training and testing on different datasets was for example to compare the performance of a models trained on diffently large datasets. Another interesting aspect was the comparison of evaluation metrics for model trained on D_y and D_{rand}

10.4 Performance of different models

Table 10.2 shows an AUC comparison of different models evaluated on the test set of either D_{rand} dataset (left side) or D_y dataset (right side) type. It shows the 7 best performances for each model sorted in descending order.

One can see that for boosting algorithms, an optimal maximum tree depth parameter is in the range of $[3, 7]$. Also an optimal learning rate is either 0.1 or 0.05. It is also obvious that there is a certain performance threshold, at which the performance is not significantly increased just by adding more trees to the model, as the performances for 200 trees and 500 trees models are very comparable.

For a random forest model, one can see that the best AUC was always achieved with a tree depth parameter set to maximum. Similarly as for the boosting algorithms, increasing number of trees to high values does not necessarily lead to a significant increase of the AUC.

D_{rand}				D_y			
XGBoost model							
ntrees	maxd	lr	AUC	ntrees	maxd	lr	AUC
500	5	0.05	0.836416	500	5	0.05	0.833855
500	3	0.1	0.835999	500	3	0.1	0.833778
200	5	0.1	0.835739	500	5	0.1	0.833333
200	7	0.05	0.835362	200	5	0.1	0.833331
500	7	0.05	0.835218	200	7	0.05	0.832606
500	5	0.1	0.835052	500	7	0.05	0.832526
200	7	0.1	0.834471	200	7	0.1	0.832303
GBDT model							
ntrees	maxd	lr	AUC	ntrees	maxd	lr	AUC
500	5	0.05	0.835651	500	3	0.1	0.833060
500	3	0.1	0.835392	500	5	0.05	0.832936
200	5	0.1	0.835008	200	5	0.1	0.832153
200	7	0.05	0.834509	500	7	0.05	0.831802
500	7	0.05	0.834327	500	5	0.1	0.831722
500	5	0.1	0.834266	200	7	0.05	0.831627
500	3	0.05	0.833877	500	3	0.05	0.831443
AdaBoost model							
ntrees	maxd	lr	AUC	ntrees	maxd	lr	AUC
500	3	0.05	0.832706	500	3	0.05	0.830223
200	3	0.1	0.832179	200	3	0.1	0.829952
200	3	0.05	0.829701	500	5	0.01	0.827762
500	5	0.01	0.829677	200	3	0.05	0.826964
100	3	0.1	0.828915	100	3	0.1	0.826661
100	5	0.05	0.827862	50	3	0.5	0.826574
50	5	0.1	0.827273	100	5	0.05	0.825652
Random forest model							
ntrees	maxd	criterion	AUC	ntrees	maxd	criterion	AUC
500	15	log_loss	0.827890	500	15	log_loss	0.825749
500	15	entropy	0.827890	500	15	entropy	0.825749
200	15	log_loss	0.827197	200	15	log_loss	0.824893
200	15	entropy	0.827197	200	15	entropy	0.824893
100	15	log_loss	0.825988	100	15	entropy	0.823756
100	15	entropy	0.825988	100	15	log_loss	0.823756
500	15	gini	0.824956	500	15	gini	0.822442

Table 10.2: Results of Grid Search optimization.

Another practical aspect to look at, when comparing machine models, is the time of training. Table 10.3 shows an average training time for all configurations where $ntrees = 500$. We can see, that XGBoost training is significantly faster than training of other models.

	XGB	GBDT	ADA	RF
time	56 s	1044 s	1007 s	149 s

Table 10.3: Average training times of different algorithms for $ntrees = 500$.

10.4.1 Source of performance differences

Another important observation is that the AUC for a model trained on D_{rand} dataset is always higher than for a model trained on D_y dataset, which is even more apparent from boxplot graph 10.3.

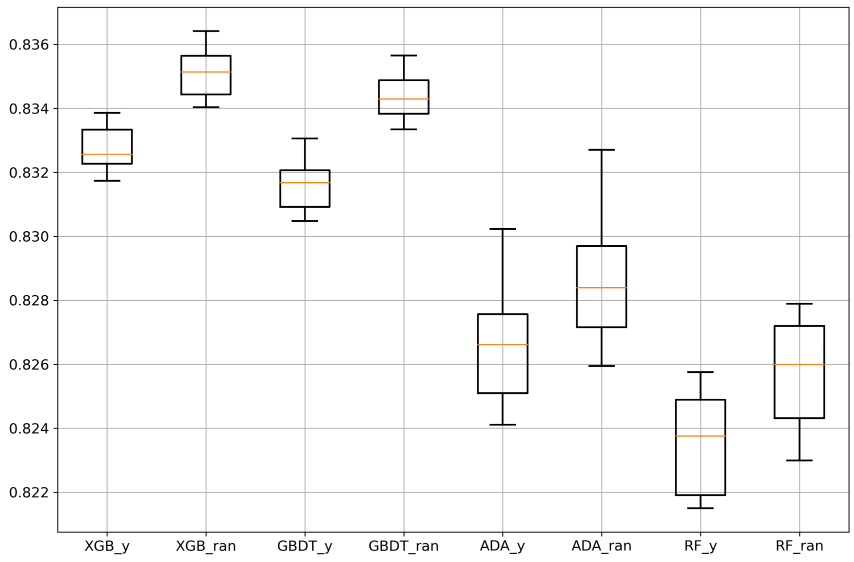


Figure 10.3: AUC differences for D_y and D_rand on test set.

This result makes sense, since the background types in D_{rand} train set are in the same proportion as background types in test set, because they both correspond to proportions of background events generated by Monte-Carlo simulation. This means that the loss minimization on train set is more correlated with the loss reduction on test set, compared to training and evaluation for D_y dataset, where the proportions of backgrounds correspond to the proportions of background yields. For example, if we would not reduce the size of ttH samples, they would be making up around 30% testing samples for both for both D_{rand} and D_y datasets. However, when the model was trained on D_y train set, ttH would be making up only 1% of all background samples, so the model would focus more on correct separation of different backgrounds and the AUC on the test set would not be as good as in the case of training on D_{rand} train set, where the ttH would also make up 30% of all background events.

Even though the number of ttH samples was reduced, the disproportion of the yields and generated background types is still present and projects into lower performance on test set for models trained on the D_y train set.

If we want to know, how well is model trained on D_y going to perform on data recorded from real experiment, we need to create a test set in the same way as D_y. Following plot shows AUC results for 9 best performing XGBoost models from the table 10.2 for test set created the same way as D_y are shown in figure 10.4. The test set is created from remaining samples after the train set is created, so no samples from the train set are contained in the test set.

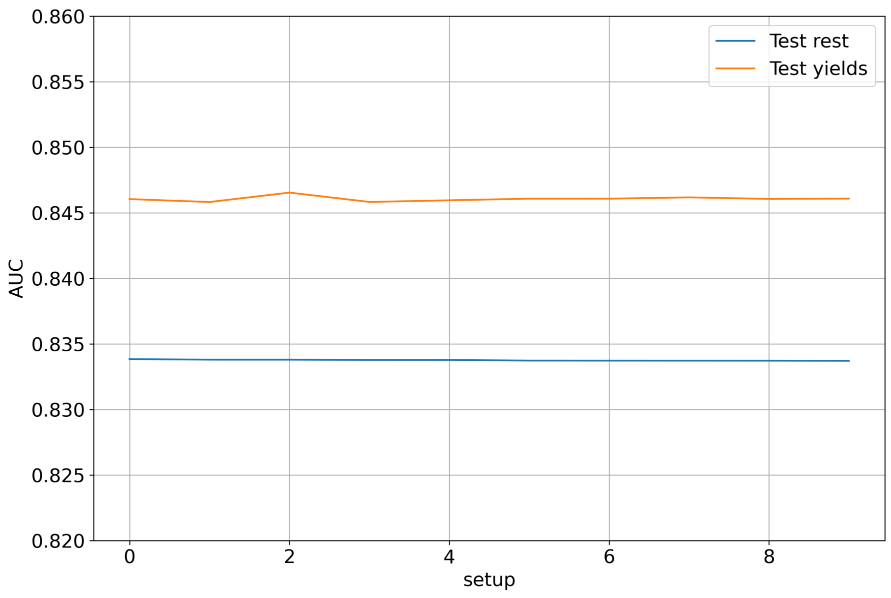


Figure 10.4: XGBoost evaluated on different test sets.

10.4.2 AUC and Significance correlation

In the previous sections, models were compared based on their AUC on test set, as it is a standard metric for comparison of machine-learning models. AUC however is not the most important metric in this analysis, because it does not have much of a physical meaning. The most important metric we want to optimize is the significance.

During the analysis, it became obvious, that models with high AUC tend to have high level of significance. So it might be interesting to look at the actual relationship between these values.

In the Grid search hyperparameters optimization, multiple metrics beside the AUC were evaluated, significance including. Figure 10.5 shows the values of AUC and significance for different XGBoost models, one data point corresponding to a trained model with different hyperparameters. On the left side of the figure are models trained on D_y, on the right side of the plot are models trained on D_{rand}.

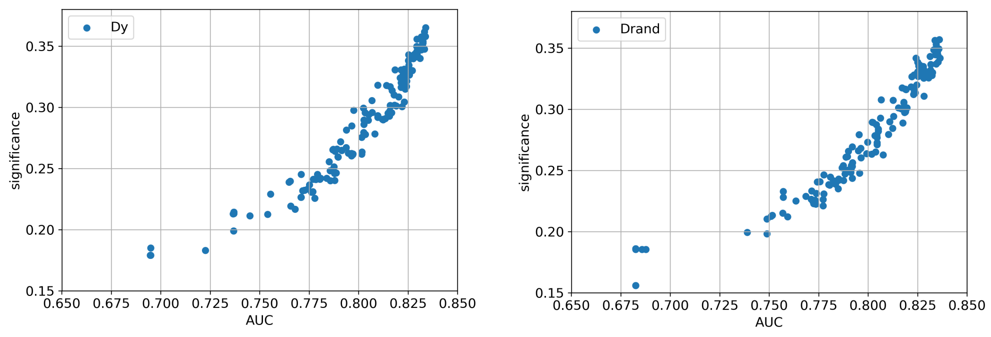


Figure 10.5: AUC and significance correlation for different models.

One can see that AUC and significance are very highly correlated. What is also obvious, is that for higher values of AUC, correlation is even stronger for models trained on D_y. This also makes sense, as the training is more focused on correct separation of background types with high yields, which is important for getting high significance values.This correlation pattern is obvious across all models used in Grid search.

An important outcome of this section is that with AUC maximization, we also maximize the significance of the model. This is very useful, because in the following optimization runs, we can focus mainly on maximization of AUC and then check, what is the resulting significance. An advantage of AUC maximization is that significance can sometimes have inconsistent values because of its definition in section 6. Also, in the next experiments, D_y is used as a primary dataset for model training because of high correlation of AUC and significance on test set, especially for high AUC values.

10.4.3 Training on large and small dataset

A standard procedure of training a machine-learning model is to take some percentage of all samples and use them for training and then test them on remaining samples. Standard ratios for training and testing are for example 50:50, 60:40, 70:30 etc.

In the previous sections however, the training took place on 120 000 samples, which make up only about 4% of all samples. The main motivation was the training time, which would take too long if models were trained on dataset containing 70% of all samples. For demonstration, in the table 10.4 are shown means of training times for different models, when training on a set containing 4% and 70% of all samples (*ntrees* parameter is set to 100). We can see that for example for XGBoost model, training takes almost 11 times longer if its done on 70% training set.

	XGB	GBDT	ADA	RF
4%	5.37	94.71	101.98	13.84
70%	62.28	1732.72	1977.25	284.67

Table 10.4: Means of training times in s for differently large datasets.

To have some overview, how much training on smaller dataset affects the performance, XGBoost model was trained on a set containing 70% of events for all grid search setups and and evaluated on 30% of all events. The results were then compared with the same model trained on D_y and tested on remaining samples (96% of all events).

Results are shown in the figure 10.6. Performances of XBoost models trained on D_x or sorted in ascending order and to each of them, performance XGBoost with same parameters trained on 70% training set is assigned. We can see that for the best performing models, the AUC on test set is very comparable regardless of how many training samples were used. This basically means, that 120 000 training samples in D_y dataset is sufficient for training a good model.

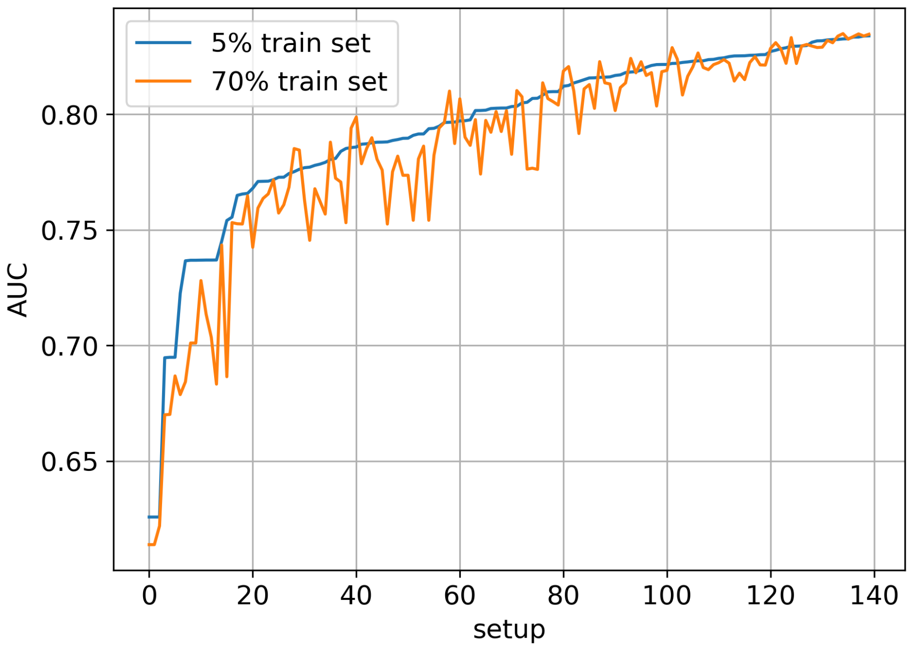


Figure 10.6: AUC differences for model trained on 70% of all samples and model trained on 4% of all samples.

This is a very important finding, because in the next steps of analysis, there will be thousands of procedures, which would take unmanageable amount of time, if they took place on a dataset containing 70% or even 50% of all samples. Now we know, that such a large training set is not necessary and we can find the best model by training on a dataset containing only 4% of all samples.

10.4.4 Binary versus multiclassifier performance

In previous analyzes, the whole task of signal separation was treated as a multiclassification problem, where all events were separated into 5 classes: *tH*, *ttb*, *ttc*, *ttL* and ”other” class containing all remaining background types. Advantage of this approach is that we can see, how well is each class separable from remaining samples. Disadvantage is that the training of such multiclassifier takes much longer, because for boosting algorithms training is typically done with One-Vs-The-Rest approach, where binary classifier for each class is trained. Evaluation is then done by normalizing predictions for each class and taking the one with highest value.

If we are not interested in AUC values for different backgrounds, it might be more beneficial to use binary classifier. It is however important to check, whether the One-Vs-The-Rest principle of multiclassifier does not improve the signal separation.

To verify that, following experiments were done - for 5 best performing models found with Grid Search, corresponding multiclassifier version of the same model was trained and evaluated on the same datasets. Events were put into 5 classes as described previously.

Results for each model are shown in the figure 10.7. Multiclassifier AUC for tH class is very comparable with AUC of binary classifier in all cases.

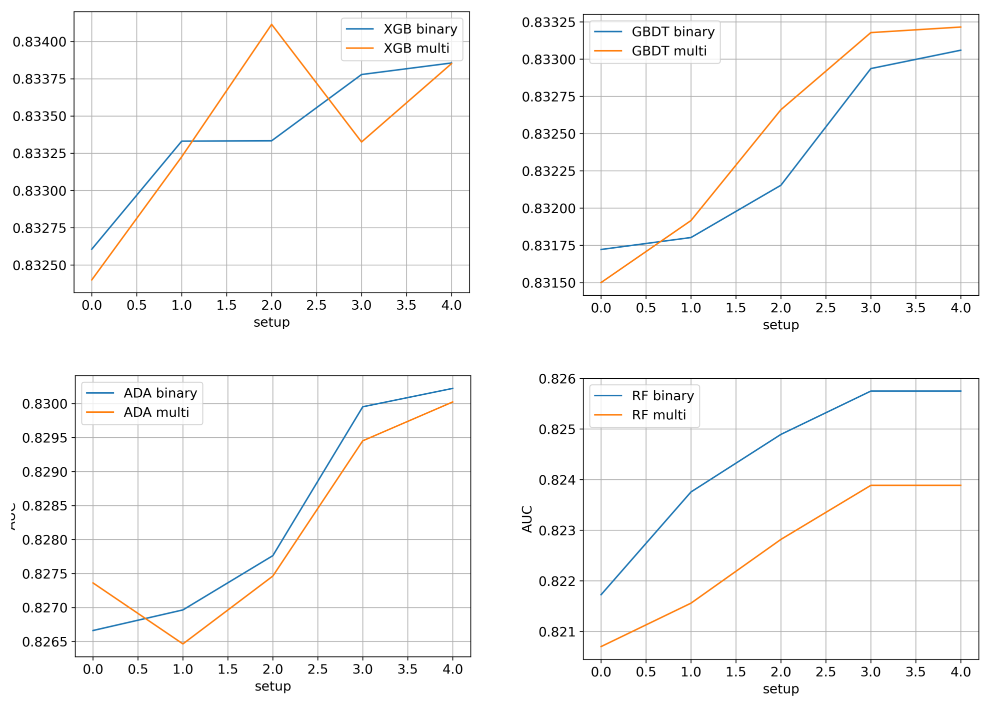


Figure 10.7: Comparison of binary and multiclassifier performances.

Because of the longer multiclassifier training time with no significant contribution to signal separation, only binary classifiers are considered in the next sections.

10.5 Hyperparameters optimization with Optuna framework

Grid Search algorithm gives a great overview of how different models perform and also what are reasonable values of hyperparameters used for the optimization.

To fine-tune the performance even further, Optuna framework is used as next step of hyperparamters optimization.

So far, the focus has been only on 3 hyperparameters - ntrees, maxdepth and learning rate. In the next sections, other hyperparameters are also taken into account. This however brings some new challenges, because the more hyperparameters we try to optimize, the more runs Optuna needs to converge and give an optimal values. Defining too many hyperparameters for optimization and hoping that Optuna find an optimal values in limited number of runs typically does not work. For this reason, the maximum of 6 hyperparameters with reasonable ranges were used in each Optuna run.

The following analysis focuses on hyperparameters optimization of XGBoost model, as it was the best performing model from Grid search analysis with lowest training time compared to other models.

As a train set, D_y is used because of the high correlation of AUC and significance on test set for a trained model. Validation and test sets are created the same way as train set, only the number of signal samples is 10 000 instead of 20 000. Also another type of test is created, contaning all samples beside those used in train set and test set. The first type of test set is in sections denoted as *Test D_y*, the other is denoted as *Test MC*.

The following section describes a special approach for stopping criteria, so there is no need to manually setup ntrees and learning rate hyperparametrs. In the next sections are described different Optuna runs with different hyperparameters and different optimization strategies.

10.5.1 Adaptive learning rate and stopping criteria with XGBoost callback functions

Because we want to minimize the number of hyperparaments that are used for the optimization, it would be good if some hyperparameters were set automatically, especially those, for which it is difficult to find default values, like learning and the number or number of trees.

A typical approach is to setup a learning rate and stopping criteria, which usually is the number of rounds, in which the evaluation metric (typically AUC) has not improved.

The disadvantage of this approach is that there is always a trade-off between the training time and performance. If we set the learning rate too high, then training stops after a few rounds, but the performance will not be great. If we set it too small, then the training will run for a very long time until the stopping criteria is met.

For this reason, the following approach is used in the Optuna analysis: At the start of the training, the learning rate is set to 0.1. If there has not been an improvement on validation set in last 5 rounds, then the learning rate is reduced and training continues. Training is stopped, if there was not any improvement in last 5 rounds and learning rate is lower than certain threshold.

This functionality is implemented with usage of *TrainingCallback* class from XGBoost callback package.

In most of the cases, training with adaptive learning gave better results than a training, in which the learning rate was set to a constant value and the training was terminated after no improvements on validation set in last n rounds.

10.5.2 Initial Optuna run

In the first Optuna run, the following hyperparameters were optimized:

- `booster` $\in [gbtree, gblinear, dart]$
- `max_depth` $\in [3, 7]$
- `scale_pos_weight` $\in [0.5, 1.0]$
- `alpha` $\in [0.0, 10.0]$
- `lambda` $\in [0.0, 10.0]$
- `gamma` $\in [0.0, 10.0]$

`booster` parameter defines, which model is built in each epoch for prediction improvement. Default value of this parameter is `gbtree`.

`max_depth` is a parameter we have already used in Grid optimization and stays for maximum tree depth. Since the best models from Grid Search had `max_depth` values in a range from 3 to 7, the same range is used for this parameter.

`scale_pos_weight` defines the scale factor for positive class. Since the proportion of signal and background samples for training is in ratio 1:5, range of $[1.0, 5.0]$ is chosen for this parameter.

`alpha`, `lambda` - L1 and L2 regularization term for weights. Default value for `alpha` is 1 for `gbtree` model and 0 for `gblinear` model. Default value for `lambda` is 0 for all models.

`gamma` - another parameter for overfitting prevention. Minimum loss reduction required to make a further partition on a leaf node of the tree.

For this study, TPE sampler with 1000 runs was used.

Table 10.5 shows the best AUC results sorted by an AUC on test set. Table 10.6 shows the means and standard deviations of parameters for 20 and 100 best models.

booster	max_d	s_p_w	lambda	alpha	gamma	Test Dy	Test MC
gbtree	5	1.0668	6.6946	9.2985	2.1685	0.8466	0.8340
gbtree	5	1.0790	4.6930	9.7976	1.9686	0.8466	0.8342
gbtree	5	1.0740	5.5907	9.9802	1.6195	0.8465	0.8341
gbtree	5	1.0813	4.9279	9.9993	0.8057	0.8465	0.8345
gbtree	5	1.0025	6.8698	9.4646	3.0867	0.8465	0.8340
gbtree	5	1.1216	5.4283	9.7549	1.3733	0.8465	0.8345
gbtree	5	1.0647	7.0618	8.4867	2.4968	0.8464	0.8340
gbtree	5	1.2731	4.7856	9.6456	2.3640	0.8464	0.8341
gbtree	5	1.1696	4.3730	9.5450	0.3648	0.8464	0.8342
gbtree	5	1.0028	6.8933	9.6063	2.1158	0.8464	0.8342

Table 10.5: 10 best performing models found with Optuna TPE sampler.

	max_depth	s_p_w	lambda	alpha	gamma
Mean 20	5.0	1.113	5.770	9.517	1.772
Std 20	0.0	0.124	0.932	0.412	0.691
Mean 100	5.0	1.123	5.500	9.512	1.680
Std 100	0.0	0.120	1.015	0.543	0.732

Table 10.6: Means and standard deviations of parameters for 20 and 100 best performing models.

We can see, that the mean of `scale_pos_weight` (`s_p_w`) is very close to its default value of 1.0 with quite a small standard deviation. The value of `max_depth` is for almost all results equal to 5. The booster for most of the best models is `gbtree`, which also correspond to default value of this parameter. Based on these observations, values of mentioned hyperparameters for the next runs are set to following values: `scale_pos_weight` = 1.0, `booster` = *gbtree*, `max_depth` = 5.

In an initial Optuna, we were able to find an optimal values of 3 hyperparameters and reduce the search space for the algorithm. Thanks to that, we can add some more hyperparameters for the next run:

- `colsample_bytree` $\in [0.5, 1.0]$
- `min_child_weight` $\in [0.0, 10.0]$

`colsample_bytree` parameters is the ratio of total number columns that are randomly selected and used for tree creation. `min_child_weight` is a minimum sum of instance weight needed in a child. Both of these parameters are used to reduce the effect of overfitting.

For the next run, ranges of previously used features were updated based on the previous results:

- `alpha` $\in [5.0, 20.0]$
- `lambda` $\in [0.0, 10.0]$
- `gamma` $\in [0.0, 5.0]$

All of these changes are based on the values of best performing models described in the table 10.6. Since values of parameter `alpha` are very close to the edge of defined interval for this parameter, the interval for this parameter is extended. On the contrary, the interval $[0.0, 10, 0]$ seemed to be too large for parameter `gamma`, as most of the optimal values were close to 1.5, so in the next run, range of this interval is reduced. Optimal values of parameter `lambda` are around the value of 5, so the same range as in the initial run is used.

In the next hyperparameters optimization run, 1000 trials study with TPE sampler is used. This study was done twice with two different objectives - on of them focusing on maximization of AUC on test set and the other on maximization of significance on test set.

The best obtained AUC on D_y test set across all TPE sampler runs was 0.848, with corresponding value of AUC on Monte-Carlo test set of 0.834. Best obtained singificance was 0.395.

10.5.3 Optimization with NSGA-II sampler

Another series of Optuna optimizations was done with NSGA-II sampler, which is based on genetic algorithms. Even though the main usage of NSGA-II algorithm is multi-objective optimization, it can be profitably used also for single objective purpose.

Hyperparameters used in NSGA-II analysis and their ranges are same as in the previous section. Also same as before, two types of optimizations were performed - one focusing maximization of AUC on test set and the other focusing on maximization of significance on test set.

Since NSGA algorithm typically requires a higher number of runs for good results compared to TPE sampler, the number of trials was set to 2500 for both objectives.

The best AUC on D_y was again 0.848, with corresponding value of AUC on Monte-Carlo test set of 0.8345. The best significance obtained with NSGA-II algorithm was 0.389.

10.6 Neural network optimization with Optuna framework

Since a Grid Search of neural network models has been done in a previous analysis (section 8.1), it was not subject of this thesis. Instead, optimization with Optuna framework was chosen as a more advanced tool for this task.

Optimized hyperparameters were following:

- `n_layers` $\in [1.0, 10.0]$
  - `n_units` $\in [1.0, 150.0]$
  - `activation` $\in [tanh, relu, sigmoid]$
  - `dropout` $\in [0.0, 0.5]$
- `learning_rate` $\in [1e^{-5}, 1e^{-1}]$
- `loss_function` $\in [mse, binary\_crossentropy, binary\_focal\_crossentropy]$

For each dense layer in `n_layers`, number of units and activation function were defined individually. Also each dense layer was followed by dropout layer with droupout rate defined by `dropout` parameter.

For optimization, both TPE and NSGA-II samplers with 1000 trials were used. Objective of the optimization was maximization of AUC on the test set. Both train and test set were created as D_y datasets, with an empty intersection between two sets (no sample from train set was contained in the test set).

The best AUC on D_y test set was 0.8438, with corresponding value of AUC on Monte-Carlo test set of 0.8337. The best significance obtained with neural networks was 0.389

Most of the best results were obtained with a neural network model with only 2 hidden layers. Mean value of neurons in the first hidden layer for 20 best performing was 123.8. For second hidden layer (before the output layer) it was 95.1.

For the neural network, most of the best results were obtained with NSGA-II sampler.

10.7 Optimization summary and the best model selection

From previous results, we can see that both XGBoost a neural network have very comparable performance, with XGBoost performing slightly better on both evaluation metrics. For this reason, XGBoost was chosen as a primary model in next chapters of this work.

Even though neural network had slightly lower performance compared to XGBoost model, it is one of the most common models in high energy physics. For a good comparison with other analyzes, the decision was to select best performing neural network and implement it to tHbb skimmer code (section 13) beside the XGBoost model. The selection of neural network model is described at the end of this section.

In the previous runs, there were multiple models with significance above 0.39. To find out, which model has the best significance stability, it was necessary to run them on different randomly sampled test sets and see, how significance changes for each set. Then we have a good picture, what significance we most likely get, when we use the model on data from real experiment.

For this purpose, 10 different test sets of D_y type were created. All of these sets were created from Monte Carlo samples after exclusion of events used for training. Boxplots in the figure 10.8 show how significance differs on these datasets for each model. All models are sorted by their significance performance in Optuna run.

We see that Model 0 has consistently highest level of significance. Before a final decision was made, it was also useful to examine the significance curves of the models.

Figure 10.9 shows the comparison of significance curves of Model 0 and Model 2 for all test datasets. We see that high significance of Model 0 is caused by sharp significance increase starting at threshold 0.71. On the other hand, Model 2 seems to have significance values much more uniform, which can be advantageous, because we can make threshold cut on different values depending on how many total events we want to work with, and still have a similar values of significance. This can be useful for example to see, how systematic plots change with different number of events, when the significance value is similar.

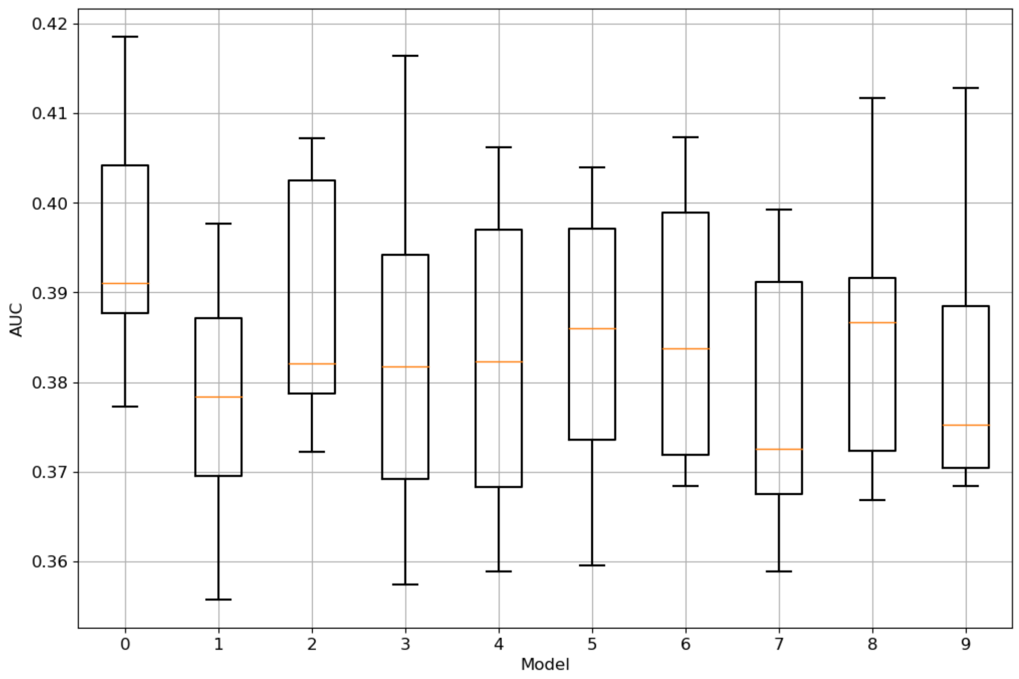


Figure 10.8: Significance boxplots of 10 best performing XGBoost models.

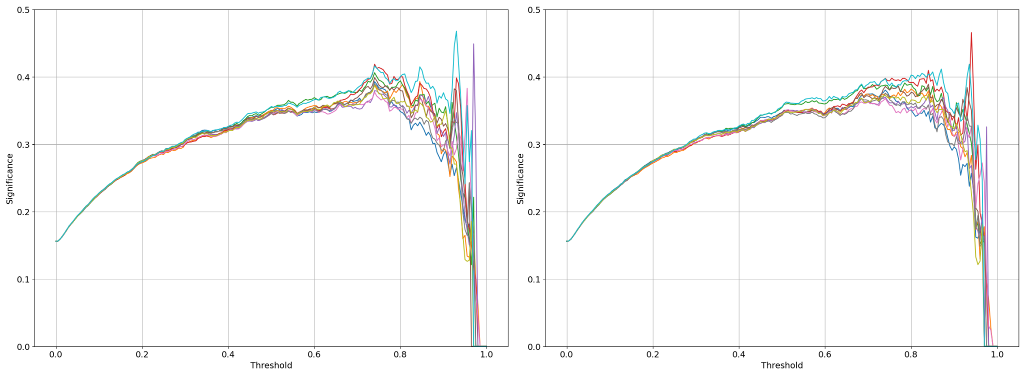


Figure 10.9: Significance plots on 10 different datasets for 2 different XGBoost models.

However, the significance levels before the sharp elevation for Model 0 is comparable for both models, so Model 0 was chosen as final model for next analysis, as it was the best performing model with consistently high values of significance.

For selection of the neural network model, the same procedure as for selection of XGBoost model was performed. Significance plots of best performing model on different on 10 different test sets are shown in the figure 10.10. AUCs on the test of selected XGBoost and neural network models are shown in the figure 10.11.

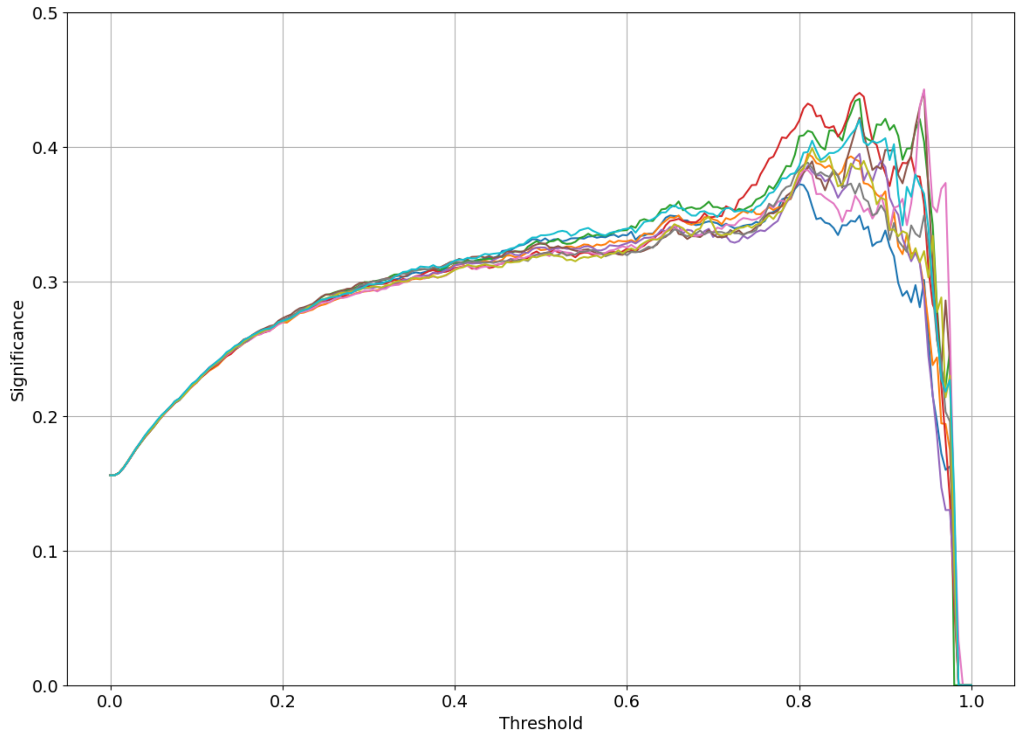


Figure 10.10: Significance plots on 10 different test sets for best performing neural network model.

10.8 Feature importance and correlation matrix

Figure 10.12 shows an importance of different features of the best performing XGBoost model. Feature importances are evaluated based on their average gain across all tree splits in XGBoost algorithm.

To see how features importance change for different models, average gains of 10 best performing XGBoost models trained on the same set were obtained and compared with boxplots in the figure 10.13.

The correlation matrix of 10 important variables are shown in the figure 12.4.

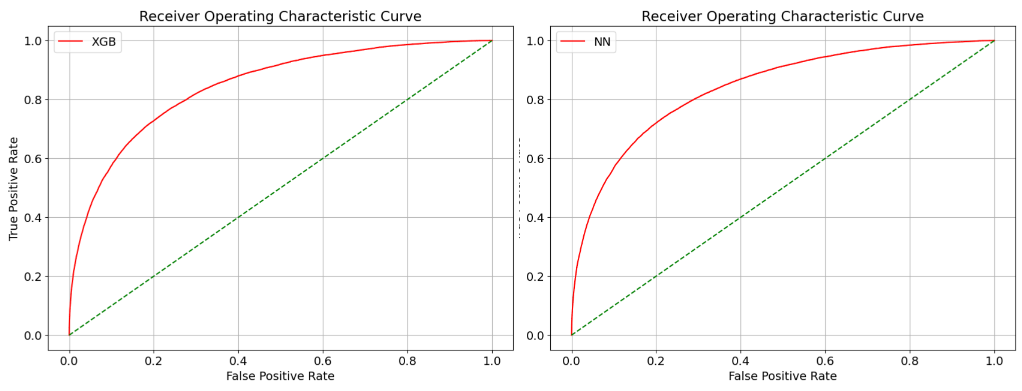


Figure 10.11: ROC curves for best performing Neural network and XGBoost model.

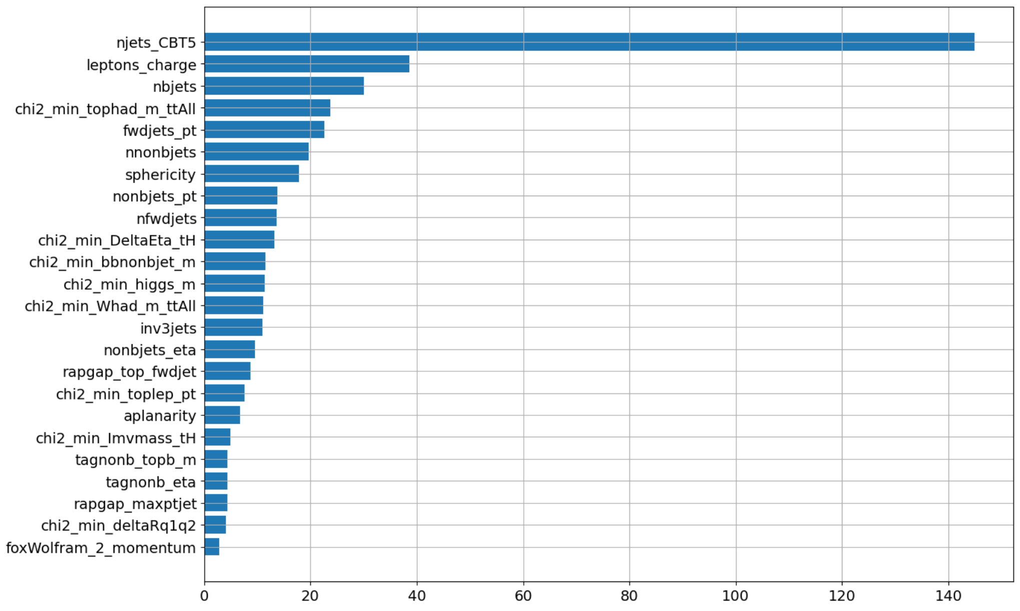


Figure 10.12: Feature importances of best performing XGBoost model.

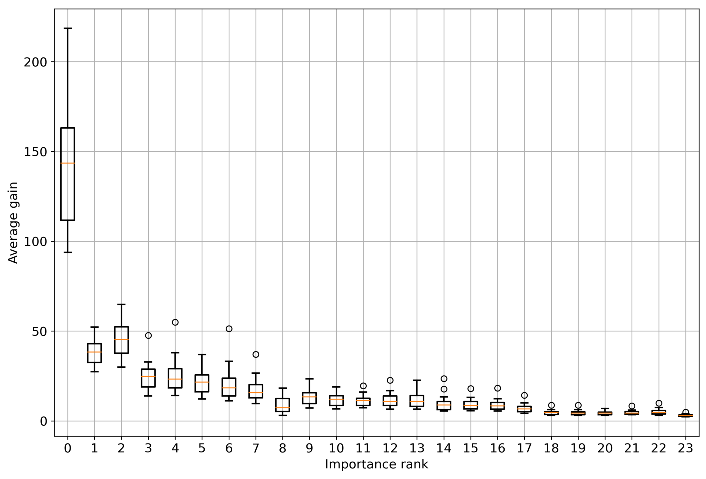


Figure 10.13: Boxplot of average gains for 10 best XGBoost performing models.

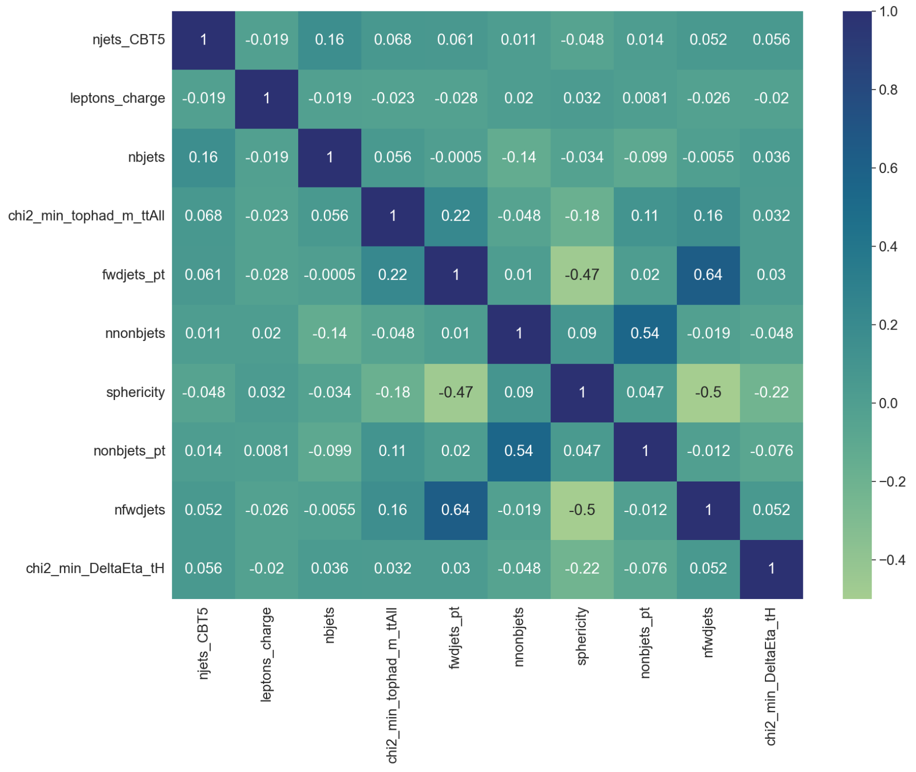
10.9 Training on a feature subset

In the previous section we were able to see the importance of different training variables. Normally, it would be interesting to know, how well the model performs if trained on a small subset of most important ones. However, recently it has been recently noted, that the modelling of *njets_CBT5* and *inv3jets* features might be inaccurate. Therefore, it is more interesting to see, how the model performs if trained without these variables, as *njets_CBT5* had by far the biggest separation power compared to other variables.

In order to do so, performances of 2 different XGBoost models trained on different training sets were compared. First XGBoost model is the one having best significance results in section 10.7. The other XGBoost had best values AUC of 0.848 on D_y test set across all Optuna runs.

AUC and significance comparison of these models for 10 different test sets are shown in figure 10.15.

An updated feature importance plot of the trained model with highest values of AUC and significance on test set is shown in figure 10.16.



	njets_CBT5	leptons_charge	nbjets	chi2_min_tophad_m_ttAll	fwdjets_pt	nnonbjets	sphericity	nonbjets_pt	nfwdjets	chi2_min_DeltaEta_tH
njets_CBT5	1	-0.019	0.16	0.068	0.061	0.011	-0.048	0.014	0.052	0.056
leptons_charge	-0.019	1	-0.019	-0.023	-0.028	0.02	0.032	0.0081	-0.026	-0.02
nbjets	0.16	-0.019	1	0.056	-0.0005	-0.14	-0.034	-0.099	-0.0055	0.036
chi2_min_tophad_m_ttAll	0.068	-0.023	0.056	1	0.22	-0.048	-0.18	0.11	0.16	0.032
fwdjets_pt	0.061	-0.028	-0.0005	0.22	1	0.01	-0.47	0.02	0.64	0.03
nnonbjets	0.011	0.02	-0.14	-0.048	0.01	1	0.09	0.54	-0.019	-0.048
sphericity	-0.048	0.032	-0.034	-0.18	-0.47	0.09	1	0.047	-0.5	-0.22
nonbjets_pt	0.014	0.0081	-0.099	0.11	0.02	0.54	0.047	1	-0.012	-0.076
nfwdjets	0.052	-0.026	-0.0055	0.16	0.64	-0.019	-0.5	-0.012	1	0.052
chi2_min_DeltaEta_tH	0.056	-0.02	0.036	0.032	0.03	-0.048	-0.22	-0.076	0.052	1

Figure 10.14: Correlation matrix of 10 most important variables.

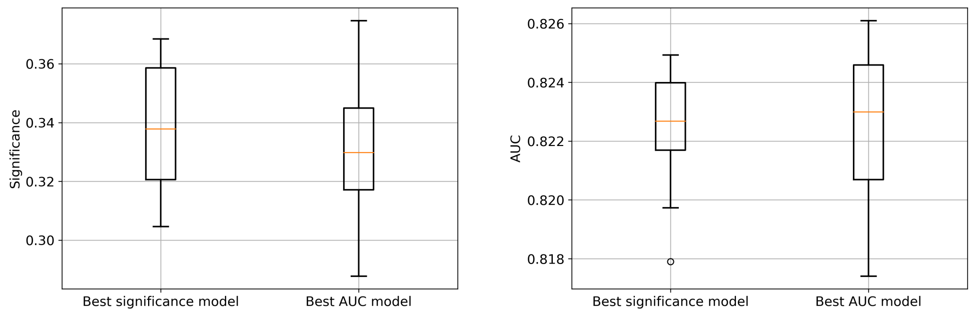


Figure 10.15: AUC and significance comparison of two best performing XG-Boost models trained on a subset of variables.

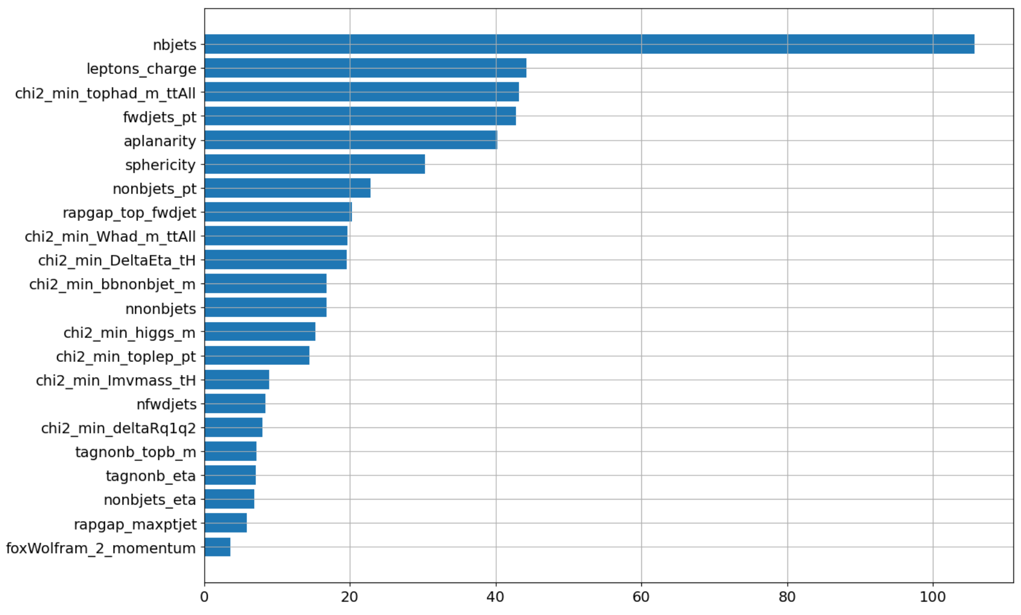


Figure 10.16: Feature importance of a model trained on the subset of features.

Chapter 11

TRExFitter evaluation without systematic uncertaintities

For the tasks such as creation of histograms of yields before and after cut, calculation of statistical uncertainties or calculation of upper limit, there was no need to work with systematic ntuples located on the CERN cluster. Instead, local run of TRExFitter framework inside a docker container was chosen as easier and faster approach.

11.1 Creation of root files from pandas dataframe

After the whole analysis from previous step is performed and the most suitable model for further analysis is found, it is necessary to convert the evaluated Pandas dataframe into root files, as this is the main data format TRExFitter works with.

This needs to be done also for data recorded from the real experiment, because they are crucial for further analysis.

First, the model is evaluated on all events that pass the preselection and the model output is written into a new column called *pred*.

After the evaluation is done, the dataframe is saved into a csv file, which is then converted into a root file by being loaded as a `RDataFrame` object and converted to the root file with `MakeCsvDataFrame` method.

These steps of conversion are performed for each signal/background type separately, because the TRExFitter reads each process type individually from associated root file.

11.2 Running TRExFitter

In the configuration file it is necessary to define all the regions that will be used in the produced plots and corresponding paths to ntuples. There can be also multiple input files defined per region, however one file should not be used for multiple regions to get correct results. It is also necessary to keep in mind, that the order of sample definitions correspond to the order in which they are plotted in stacked histogram, so the backgrounds with large number of samples should be defined as first and the small-size samples as last.

It is also useful to magnify the size of signal samples by adding the `NORMSIG` option to `PlotOptions` parameter to see the distribution of signal events, which otherwise would not be visible because of the small signal set size is compared to the background set.

A special type of background is the *non-prompt* background, which is calculated from the real experiment data and thus is read from this ntuple.

Also there is no need to define GHOST regions, as they are not needed for the stat-only calculation. It is however needed to properly define, which samples correspond to signal, background and data events.

Because in the produced root files, different process types are not mixed together, it is not necessary to use replacement file and its variables with `XXX` prefixes - this would be necessary if multiple process types were put in one root file and needed to be distinguished using these special variables.

After this, regions for plotting are defined, along with corresponding x and y axis ranges etc. Since the sizes of different signal/background types vary significantly, it is useful to use logarithmic scaling to have an overview of how a distribution for each type looks like.

Selection criteria can be defined on different levels. There can be defined a global selection criteria in the job block, selection criteria for each loaded ntuple is defined in Sample block and a selection criteria for each region is defined in Region blocks.

In the case of this analysis, it is necessary to apply selection criteria described in section 3.1, that is based on event preselection, and another selection criteria based on the model output. Both of them can be applied globally or individually per region. Using the criteria in a region block can be useful for example for comparison of the yields before and after the cut in one TRExFitter run.

Weight calculation for each event is defined in section 3.2. The `StatOnly` option needs to be set to `True`, so no systematic calculation is performed.

It is also not needed to define the Fit block, as the fit step is not performed in this section.

After all the mandatory fields are defined, TRExFitter is ready to run. The sequence of parameters used for local run is following: `ndwl`. The `n` parameter is used for ntuple preprocessing and the `d` parameter for plots production along with a table describing the yields and statistical uncertainties. `w` parameter is used for workspace creation and `l` is used for calculation of median signal strength value, when only statistical uncertainties are considered.

11.3 Docker usage

The whole run of the TRExFitter took place inside of the docker container. This form was chosen as it was the easiest way how to run TRExFitter locally without installation.

However, a disadvantage of using docker container is, that by default it provides only command-line outputs, which can not be used for viewing of the plots. To make the usage of TRExFitter inside the container more convenient, a bash script was made, which first copies modified config file into docker container, runs TRExFitter inside the container and then copies its outputs back to the working directory. In order to have no overlapping results from different runs, both the target folders in docker environment and local environment are deleted in each run.

11.4 TRExFitter outputs

After a succesful run, TRExFitter creates a folder with all defined region plots, and a table describing the yields and statistical uncertainties for each sample. Table 11.1 shows the number of events before and after an optimal cut of 0.74 is applied. All events are sorted by their yields, except for the signal process.

	Before cut	After cut
tH	73.21	14.07
tt+light jets	105453	448.77
tt+b jets	59696.8	524.95
tt+c jets	33320.7	128.57
non-prompt	5084.89	86.64
WZ + jets	4479.95	56.07
Wt channel	9258.6	153.61
tH	1586.61	12.35
t + Z	840.58	5.07
VV	283.06	2.69
t + W	258.95	0.34
tZq	169.19	13.69
tWZ	2.49	0.04
Total	220508	1446.92

Table 11.1: Yields of events before and after an optimal threshold cut is applied.

We can also calculate an expected median for signal strength μ by running TRExFitter with argument "`l`". This value before the cut was *3.89*. This is the median value of μ when only statistical uncertainties are taken into account. This value slightly declines to 3.76, if 20 bins are used instead of 10.

After the threshold cut is applied, this value increases to 4.56, which could be possibly lowered by more efficient binning.

The following figures show the stacked histograms for the model prediction and for the four most important variables. On the left, there are the distributions before the model threshold cut is applied, and on the right there are the distributions after it. The red dashed line shows the normalised signal distribution. Blinding applied for all plots is 0.1 (ratio of signal to background events in a bin).

The black dots in the histograms correspond to collected data values from the real experiments at LHC. There is a good agreement between the Monte-Carlo simulation and the recorded data, which is a good sign that the simulation describes well what is happening in the detector during the experiment.

The lower histogram shows the deviations of simulated data and data from the experiments. Values of statistical uncertainties for each bin are shown in this graph too. Note that the statistical uncertainties are much higher on the plots after the cut is performed, which is due to low statistics for these samples. Plots of the remaining regions are shown in Appendix A.

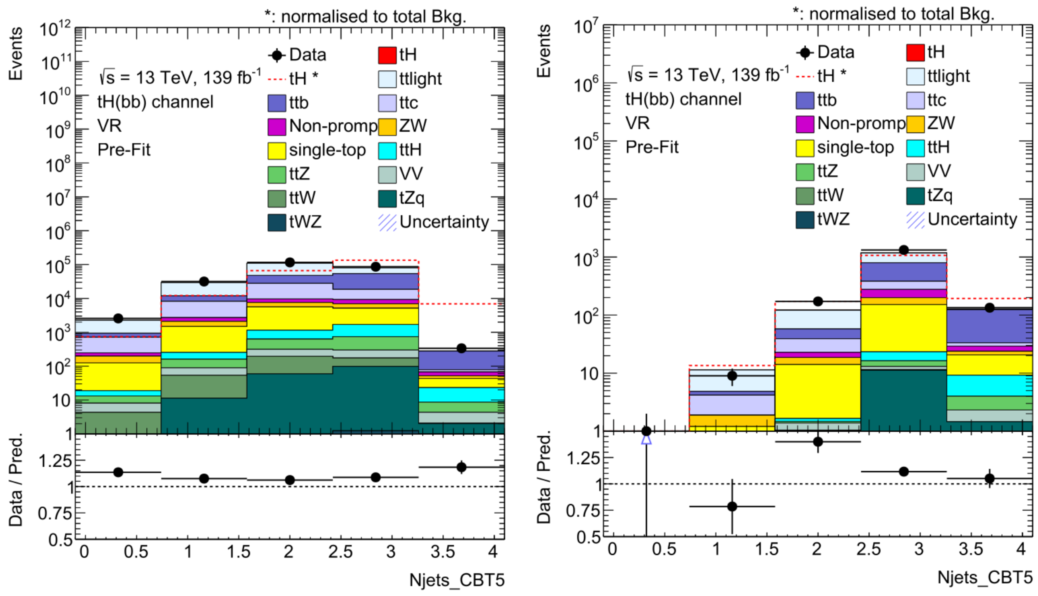


Figure 11.1: Model prediction distribution before and after the optimal threshold cut.

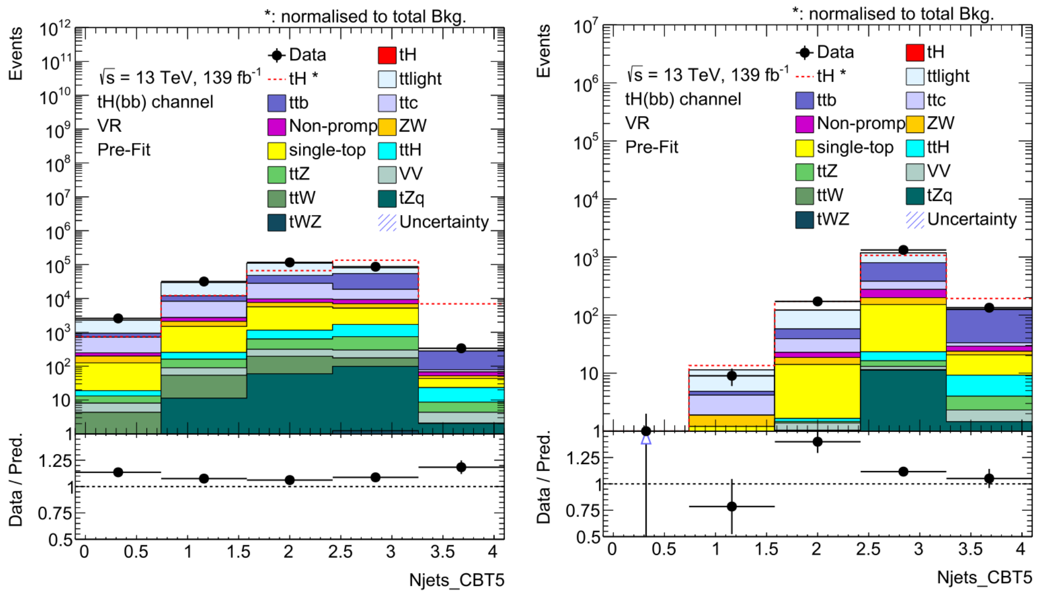


Figure 11.2: *Njets_CBT5* - distribution before and after the optimal threshold cut.

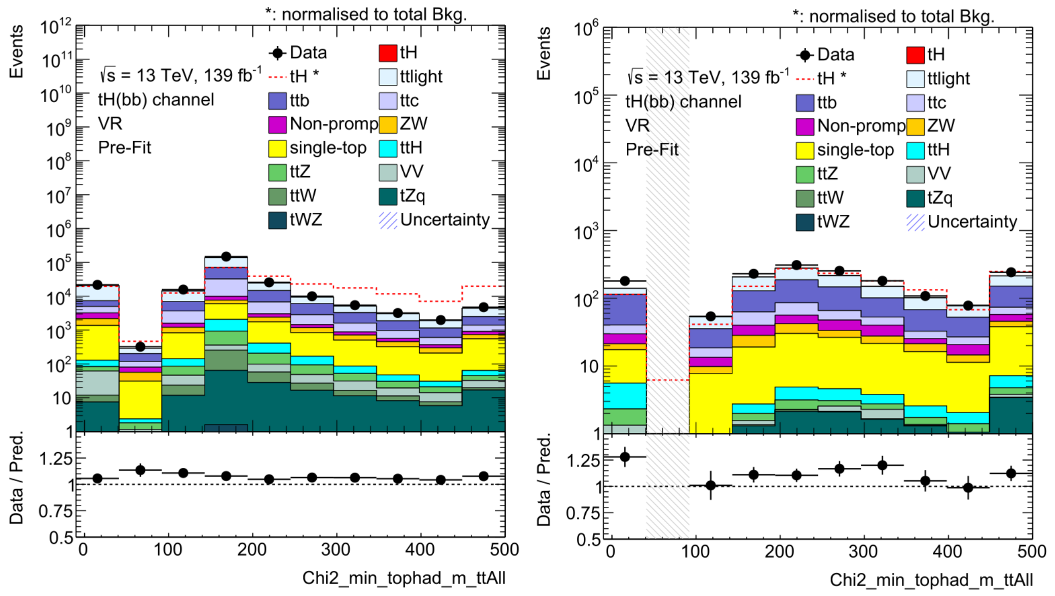


Figure 11.3: *chi2_min_tophad_m_ttAll* distribution before and after the optimal threshold cut.

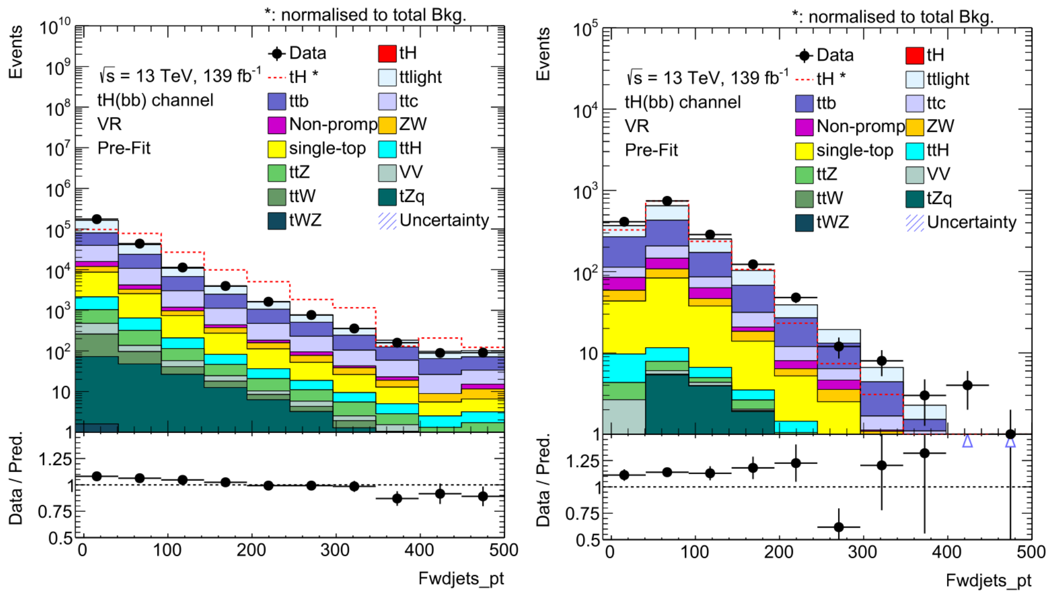


Figure 11.4: *fwdjets_pt* distribution before and after the optimal threshold cut.

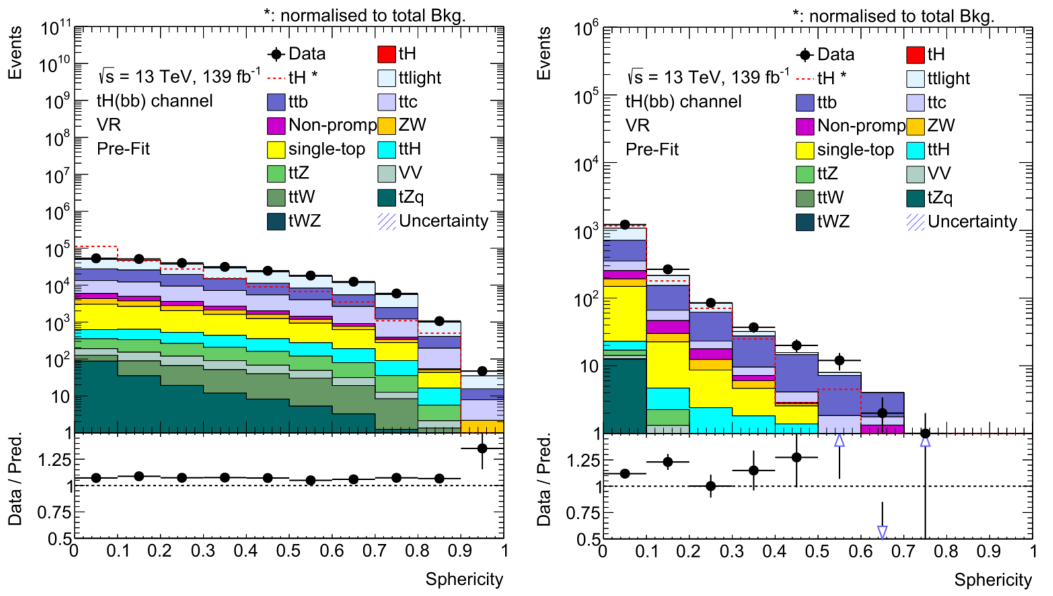


Figure 11.5: *sphericity* distribution before and after the optimal threshold cut.

Chapter 12

TRExFitter evaluation with systematic uncertainties

This chapter describes the procedure of obtaining the full analysis results which include systematic uncertainties. To be able to do that, we need to extend our configuration file for TRExFitter and also work with ntuples, that also include the systematic trees beside the nominal one. On the top of that, we need to include also samples that come from a different Monte Carlo simulation programs, which are defined in the sample blocks as GHOST. This step is done in order to compare the outputs of different simulation environments.

Since the systematic ntuples with all needed trees and the GHOST ntuples are all located on CERN cluster called lxplus, it is more convenient to work remotely on that cluster, instead of copying the files and work with them on local computers. One of the reasons is the size of the ntuples which exceeds 300 GB, and also that the subsequent calculations take a lot of time and computational power, so it is easier to use lxplus for it.

Another advantage of lxplus usage is, that it has many ROOT-related frameworks and other useful libraries already installed on it. Some of the useful tools preinstalled on lxplus is HTCondor [21], which can be useful for batching and parallelization.

Alternative tool, which can be used for this task, is Panda framework [22] in combination with Rucio [23] that provide interface to GRID, which is a huge network of interconnected computers around the world, that are managed by CERN and which are used for CERN-related computations.

12.1 Assigning model outputs to systematic ntuples

To be able to perform the whole analysis which includes systematic uncertainties, it is necessary to have a model prediction branch in each tree of systematic ntuples. This can be quite problematic, because the systematic ntuples are very large (over 300 GB) and assigning model output to each event takes a very long time. For the scope of this project, appropriate approximations have been chosen.

The alternative approach leverages the fact, that all the variations of an event can be found under the same event number in the variation trees. So instead of evaluation of the model for each systematic tree, the evaluation is performed just once for a nominal tree. Then a *predictions* csv file is created, which contains two columns called *eventNumber* and *pred*. The *pred* column corresponds to a prediction for given event, which is then assigned to all trees in systematic ntuple.

For this step a root macro was created, which does the following: loads a csv file and creats a hashmap from it, then all the events in every tree are iterated and assigned a prediction value into the newly created branch *pred* based on its event number. With the hashmap usage, the prediction assignment is almost instant, and the algorithm runs very fast, compared to the scenario in which we would be evaluating the XGBoost model for each event.

For faster run, the script uses `setBranchStatus` function, which keeps activated only the branches *eventNumber* and *pred* during the whole procedure.

It is important to note, that this approach can be used only for ntuples that come from the same production. This means it could not be used for ntuples that were produced by PP8 and HP7 simulators (GHOST samples in the configuration file), because they were produced in a separate run and there was no relationship between the event numbers from these ntuples and those from the predictions file.

For these ntuples, it was necessary to convert the nominal trees to a csv file as described in section 5.1.1, perform an evaluation of the model and after that create the predictions file and use the described macro. Since all POWHEG+HP7 and POWHEG+PP8 ntuples contained only nominal trees, this step was not too time-consuming.

12.2 Script for multiple ntuple assignment

Since each generated process is typically located in a different root file with specific id, that is often split into multiple files, it is convenient to use a shell script which assigns predictions to all defined ntuples, instead of running the root macro for each ntuple separately. For this purpose, a shell script *assign-Preds.sh* was created. Example of a script run:

Parameters of this script are:

- `file1` - text file defining all ntuple paths, one path on each line
- `file2` - file with event numbers and correspondning predictions
- `file3` - root macro described above

This script iterates all ntuples defined in file1. In each iteration it saves the trees from an ntuple into a text file, which is then loaded by root macro. The reason for this untypical method is, that there is no ROOT function, which would return content of a root file as an argument or loaded it into parameter. The only way of getting the content of a ROOT file is through the `ls()` function, which prints the contents on the standard output. This output needs to be then processed with shell commands in order to filter out unneeded information printed by the `ls()` function. It is also necessary to create a list of trees for each ntuple, because different ntuples may contain different variation trees. After this, when the tree list is created, the ROOT macro defined in previous section is executed.

Afer the whole setup is done, TRExFitter can be executed. The full sequence of run parameters to produce all necessary outputs is following:

```
trex-fitter ndwfplr tf.conf
```

Outputs of each option are presented and explained in following sections.

12.3 Production of prefit and systematic plots

As a first step, it is necessary to load all ntuples and turn them into histograms for further use within the framework. This is done with ”`n`” option when running TRExFitter. A new Histograms folder is created which contains everything we need for the next steps. Note that this is typically the longest step of the whole systematic uncertainty determination.

After "n" step finishes, we are ready to run "d", where similarly like in section 11.4, pre-fit plots of all defined regions are produced, including the yields of all processes and statistical uncertainty. Compared to previous results, only three main background sources are taken into account - *ttb*, *ttc* and *ttL*. The distribution of signal region is shown in figure 12.1 and the yields with corresponding uncertainties (with inclusion of systematic uncertainties) are shown in the table 12.1. Note the yields are a bit different than in figure 11.4, because the systematic ntuples come from different production.

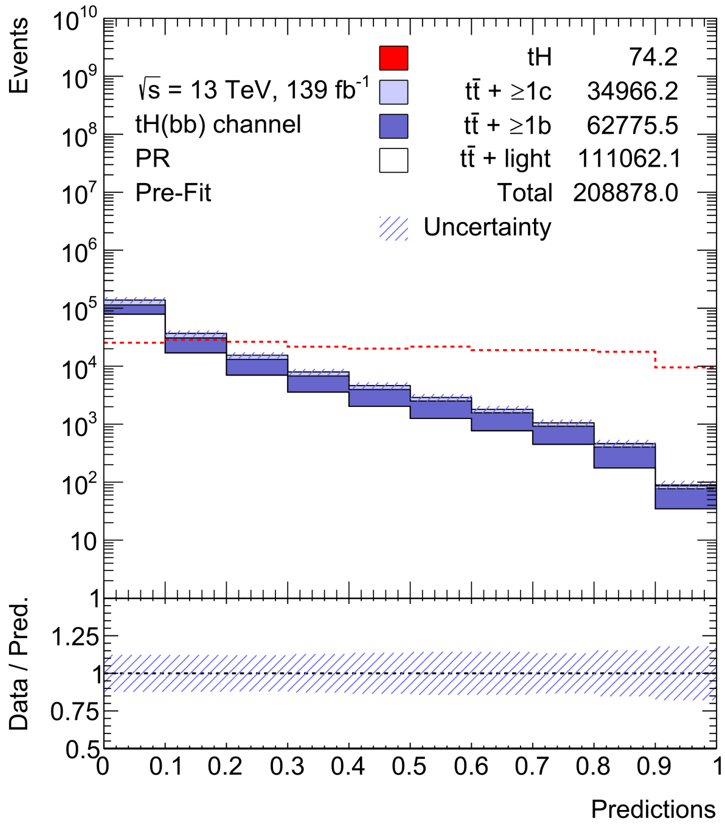


Figure 12.1: Distribution of machine-learning model output for signal and background. The signal (dashed line) is normalised to the background.

	Yields± Uncertainties
tH	74.2 ±1.3
$t\bar{t} + \geq 1c$	35000 ±19000
$t\bar{t} + \geq 1b$	63000 ±12000
$t\bar{t}$ + light	111000 ±11000
Total	209000 ±25000

Table 12.1: Yields of the processes for systematic study.

Also newly a Systematics folder with histograms is created. Each plot in this folder correspond to the systematics block defined in our config file. Compared to previous plots, a confidence interval of one σ size for the given uncertainty is shown. The bottom plot shows normalized deviations from the nominal value. Plots of 4 systematic uncertainties with highest impact are shown in the figure 12.2.

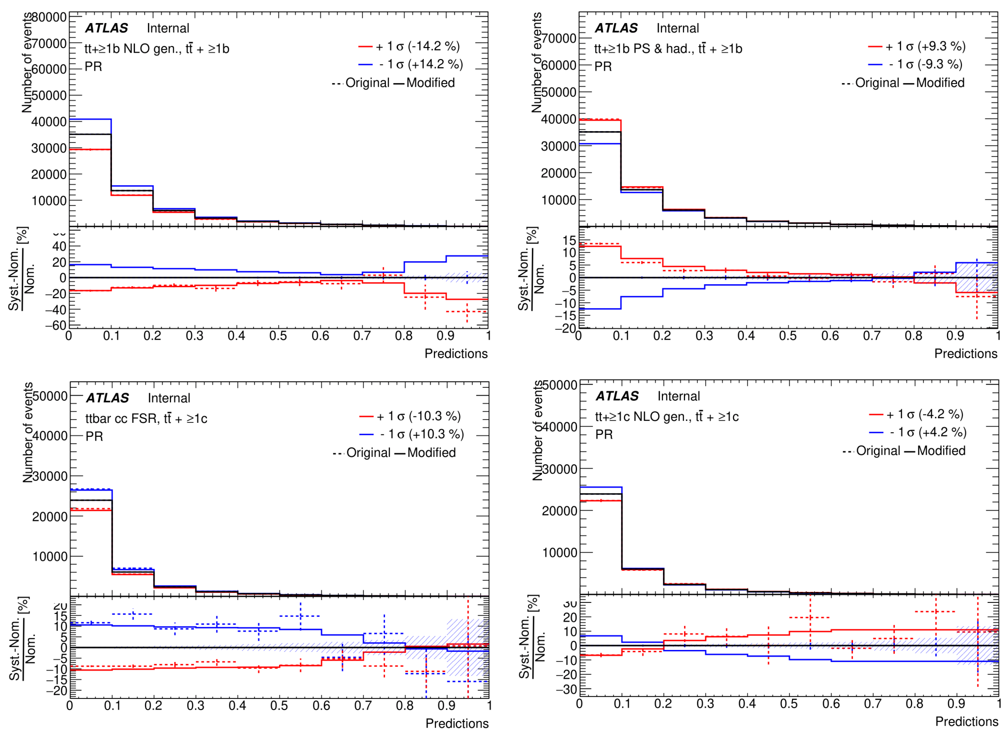


Figure 12.2: Four most impactful uncertainties.

12.4 Workspace creation and fit results

In the next step, the workspace containing the fit model is created. This step is performed by using ”`w`” argument in `trex-fitter` command. All the informations for building the fit model can be found in the the RooStats folder.

Also the *Pruning.png* plot was created. This plot shows the effect of systematics on each signal/background type. It also shows, whether the systematics survived pruning or not.

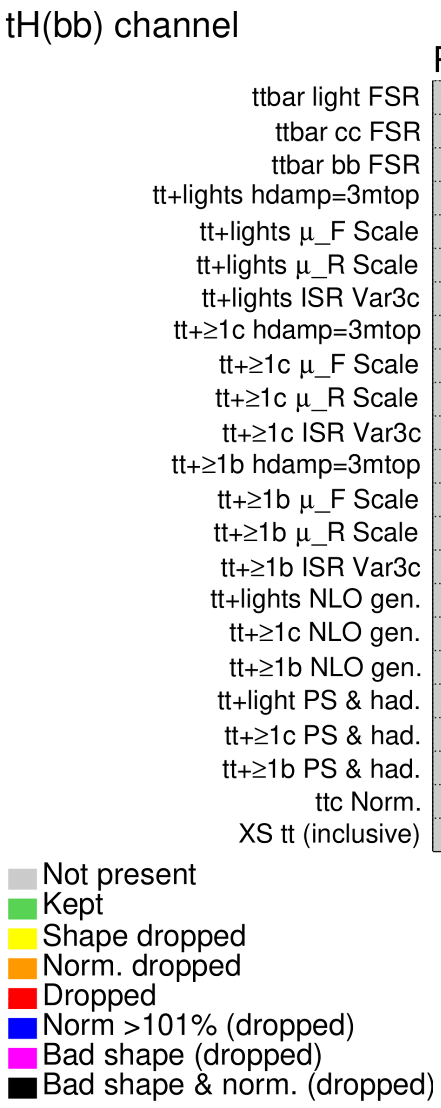
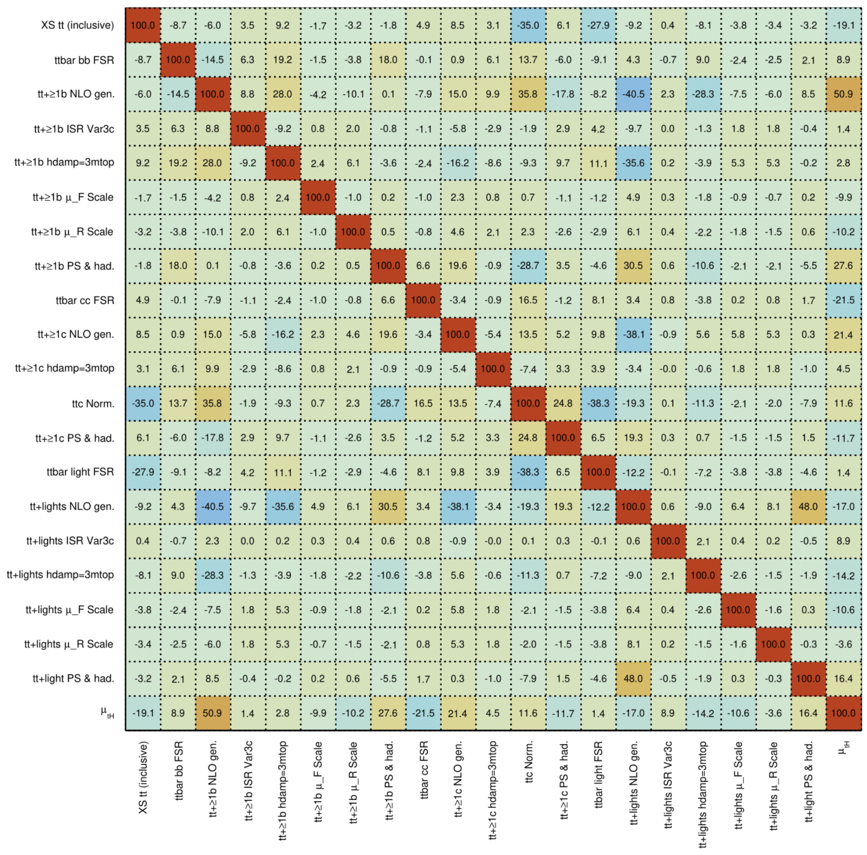


Figure 12.3: Pruning plot.

After the workspace is created, we are ready now to run a fit. This is done with "`f`" argument. In this step, following plots are produced:

- CorrMatrix which shows the correlations between nuisance parameters (fig. 12.4)
- Gammas showing the fit results for all gammas(fig 12.5)
- NuisPar shows the fit results for all nuisance parameters (fig 12.6)
- NormFactor shows the expected 95% confidence level limit for all Norm-Factors (typically includes the signal process, figure 12.7)



	XS tt (inclusive)	ttbar bb FSR	tt+≥1b NLO gen.	tt+≥1b ISR Var3c	tt+≥1b hdamp=3mtop	tt+≥1b µ_F Scale	tt+≥1b µ_R Scale	tt+≥1b PS & had.	ttbar cc FSR	tt+≥1c NLO gen.	tt+≥1c hdamp=3mtop	ttc Norm.	tt+≥1c PS & had.	ttbar light FSR	tt+lights NLO gen.	tt+lights ISR Var3c	tt+lights hdamp=3mtop	tt+lights µ_F Scale	tt+lights µ_R Scale	tt+light PS & had.	μ_{tH}
XS tt (inclusive)	100.0	-8.7	-6.0	3.5	9.2	-1.7	-3.2	-1.8	4.9	8.5	3.1	-35.0	6.1	-27.9	-9.2	0.4	-8.1	-3.8	-3.4	-3.2	-19.1
ttbar bb FSR	-8.7	100.0	-14.5	6.3	19.2	-1.5	-3.8	18.0	-0.1	0.9	6.1	13.7	-6.0	-9.1	4.3	-0.7	9.0	-2.4	-2.5	2.1	8.9
tt+≥1b NLO gen.	-6.0	-14.5	100.0	8.8	28.0	-4.2	-10.1	0.1	-7.9	15.0	9.9	35.8	-17.8	-8.2	-40.5	2.3	-28.3	-7.5	-6.0	8.5	50.9
tt+≥1b ISR Var3c	3.5	6.3	8.8	100.0	-9.2	0.8	2.0	-0.8	-1.1	-5.8	-2.9	-1.9	2.9	4.2	-9.7	0.0	-1.3	1.8	1.8	-0.4	1.4
tt+≥1b hdamp=3mtop	9.2	19.2	28.0	-9.2	100.0	2.4	6.1	-3.6	-2.4	-16.2	-8.6	-9.3	9.7	11.1	-35.6	0.2	-3.9	5.3	5.3	-0.2	2.8
tt+≥1b µ_F Scale	-1.7	-1.5	-4.2	0.8	2.4	100.0	-1.0	0.2	-1.0	2.3	0.8	0.7	-1.1	-1.2	4.9	0.3	-1.8	-0.9	-0.7	0.2	-9.9
tt+≥1b µ_R Scale	-3.2	-3.8	-10.1	2.0	6.1	-1.0	100.0	0.5	-0.8	4.6	2.1	2.3	-2.6	-2.9	6.1	0.4	-2.2	-1.8	-1.5	0.6	-10.2
tt+≥1b PS & had.	-1.8	18.0	0.1	-0.8	-3.6	0.2	0.5	100.0	6.6	19.6	-0.9	-28.7	3.5	-4.6	30.5	0.6	-10.6	-2.1	-2.1	-5.5	27.6
ttbar cc FSR	4.9	-0.1	-7.9	-1.1	-2.4	-1.0	-0.8	6.6	100.0	-3.4	-0.9	16.5	-1.2	8.1	3.4	0.8	-3.8	0.2	0.8	1.7	-21.5
tt+≥1c NLO gen.	8.5	0.9	15.0	-5.8	-16.2	2.3	4.6	19.6	-3.4	100.0	-5.4	13.5	5.2	9.8	-38.1	-0.9	5.6	5.8	5.3	0.3	21.4
tt+≥1c hdamp=3mtop	3.1	6.1	9.9	-2.9	-8.6	0.8	2.1	-0.9	-0.9	-5.4	100.0	-7.4	3.3	3.9	-3.4	-0.0	-0.6	1.8	1.8	-1.0	4.5
ttc Norm.	-35.0	13.7	35.8	-1.9	-9.3	0.7	2.3	-28.7	16.5	13.5	-7.4	100.0	24.8	-38.3	-19.3	0.1	-11.3	-2.1	-2.0	-7.9	11.6
tt+≥1c PS & had.	6.1	-6.0	-17.8	2.9	9.7	-1.1	-2.6	3.5	-1.2	5.2	3.3	24.8	100.0	6.5	19.3	0.3	0.7	-1.5	-1.5	1.5	-11.7
ttbar light FSR	-27.9	-9.1	-8.2	4.2	11.1	-1.2	-2.9	-4.6	8.1	9.8	3.9	-38.3	6.5	100.0	-12.2	-0.1	-7.2	-3.8	-3.8	-4.6	1.4
tt+lights NLO gen.	-9.2	4.3	-40.5	-9.7	-35.6	4.9	6.1	30.5	3.4	-38.1	-3.4	-19.3	19.3	-12.2	100.0	0.6	-9.0	6.4	8.1	48.0	-17.0
tt+lights ISR Var3c	0.4	-0.7	2.3	0.0	0.2	0.3	0.4	0.6	0.8	-0.9	-0.0	0.1	0.3	-0.1	0.6	100.0	2.1	0.4	0.2	-0.5	8.9
tt+lights hdamp=3mtop	-8.1	9.0	-28.3	-1.3	-3.9	-1.8	-2.2	-10.6	-3.8	5.6	-0.6	-11.3	0.7	-7.2	-9.0	2.1	100.0	-2.6	-1.5	-1.9	-14.2
tt+lights µ_F Scale	-3.8	-2.4	-7.5	1.8	5.3	-0.9	-1.8	-2.1	0.2	5.8	1.8	-2.1	-1.5	-3.8	6.4	0.4	-2.6	100.0	-1.6	0.3	-10.6
tt+lights µ_R Scale	-3.4	-2.5	-6.0	1.8	5.3	-0.7	-1.5	-2.1	0.8	5.3	1.8	-2.0	-1.5	-3.8	8.1	0.2	-1.5	-1.6	100.0	-0.3	-3.6
tt+light PS & had.	-3.2	2.1	8.5	-0.4	-0.2	0.2	0.6	-5.5	1.7	0.3	-1.0	-7.9	1.5	-4.6	48.0	-0.5	-1.9	0.3	-0.3	100.0	16.4
μ_{tH}	-19.1	8.9	50.9	1.4	2.8	-9.9	-10.2	27.6	-21.5	21.4	4.5	11.6	-11.7	1.4	-17.0	8.9	-14.2	-10.6	-3.6	16.4	100.0

Figure 12.4: Correlation matrix of nuisance parameters.

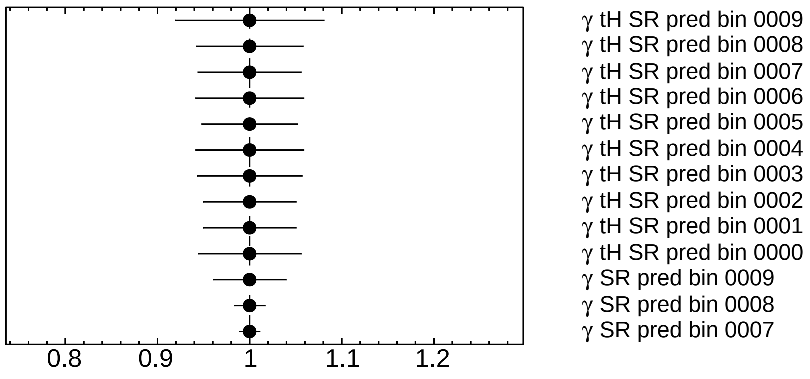


Figure 12.5: Fit results for all gammas.

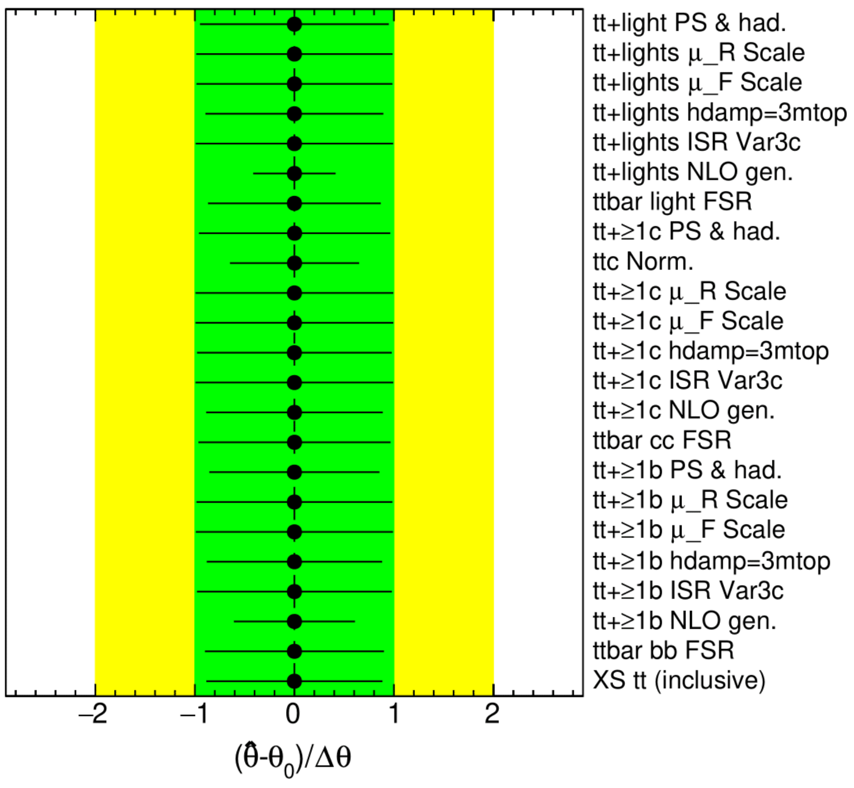


Figure 12.6: Fit results for nuisance parameters.

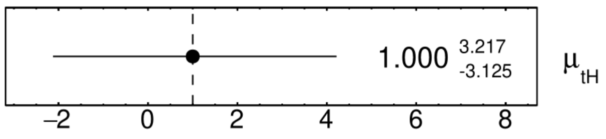


Figure 12.7: Signal strength after fit on Asimov dataset.

It is important to note that we performed a fit on an Asimov dataset [17]. This dataset is built from the exact prediction of signal and background in every bin, and it is very useful in evaluating our fit model before fitting data. For the Asimov dataset, the best-fit result for the normalization factors is just their nominal value. Fit on the Asimov dataset is done by setting `FitBlind: TRUE` in a Fit block.

12.5 Creation of ranking plot and calculation of expected median of signal strength

The last step is to create a ranking plot with ”`r`” option that shows, which nuisance parameter has the largest impact on the uncertainty of the signal strength μ.

For each nuisance parameter, four fits are performed. The specific nuisance parameter is fixed to one of these configurations per fit [17]:

- pre-fit value + pre-fit uncertainty
- pre-fit value - pre-fit uncertainty
- post-fit value + post-fit uncertainty
- post-fit value - post-fit uncertainty

Resulting ranking plot is shown in figure 12.8.

With ”`l`” option, we can also calculate the expected median value of signal strength μ. This very useful for comparison with other analyzes.

The median signal strength value with inclusion of statistical and systematic uncertainties was **6.346**.

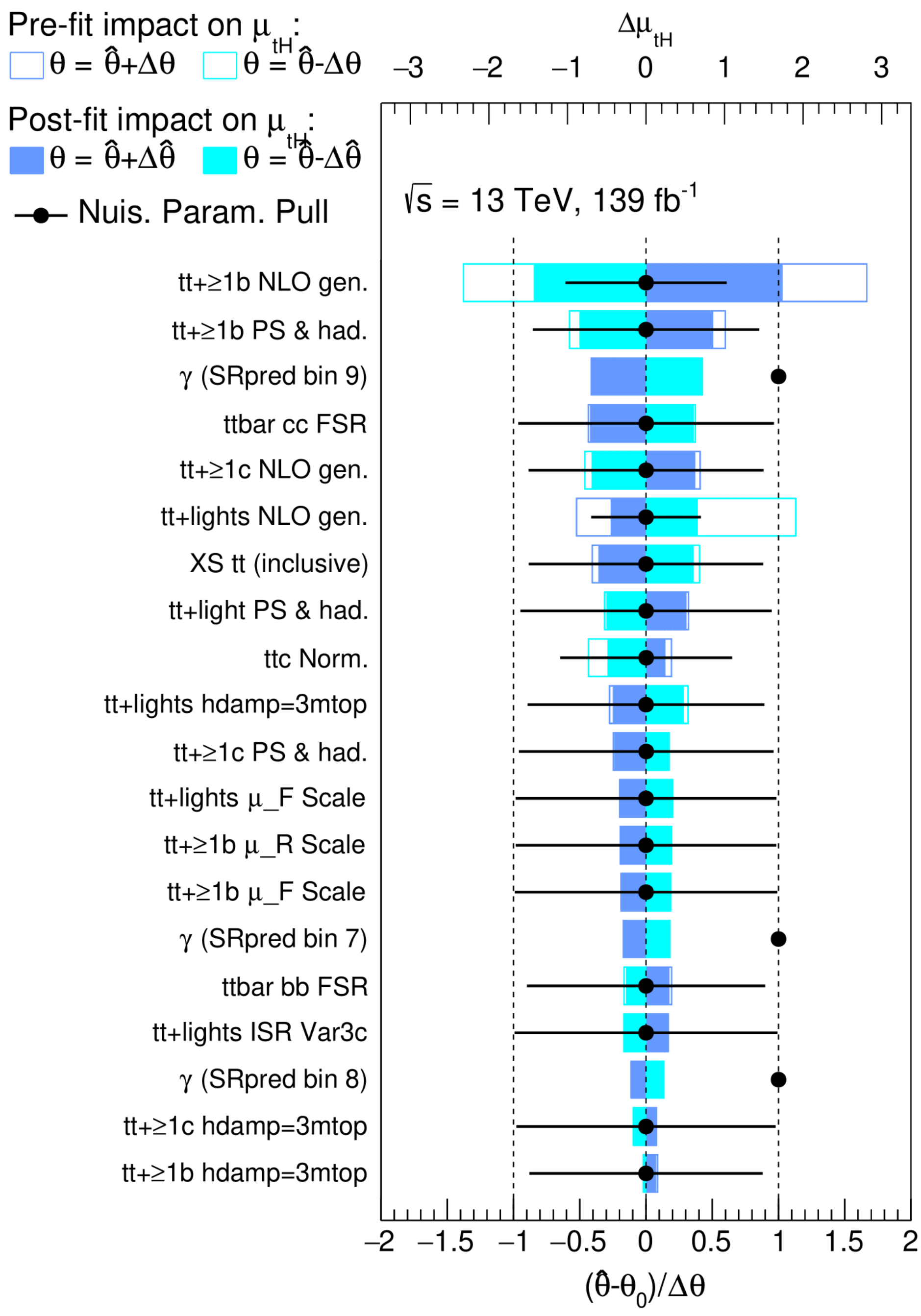


Figure 12.8: Ranking plot of nuisance parameters.

Chapter 13

Deployment

The final step of this thesis was to integrate trained models into a program called tHbbskimmer [24], which is one of main tools for the tH analysis. It has been developed in recent years by a scientific group involved in *tH* ntuple production research and is periodically used whenever a new Monte-Carlo or experiment data are produced. All ntuples which were used for training and evaluation in thesis were produced with tHbbskimmer.

The main purpose of tHbbskimmer is to process ntuples in raw form and create ntuples suitable for analysis. Raw ntuples are stored on GRID, which is a large site of interconnected computers around the world which are used for CERN data storaging and computing. So in the first phase, tHbbskimmer copies these raw files from GRID to lxplus for further processing. In this stage, first preselections and event filtering are already applied. All of this is realized with usage Panda [22] and Rucio [23] frameworks. Rucio is a framework for GRID data management and Panda is used for job submissions and monitoring.

After all preprocessed ntuples are succesfully transfered to lxplus, the second phase of ntuple skimming takes place. Here, multiple skimming jobs are submitted to HTCondor [21], which is another workload management framework widely used on lxplus. In each job, new branches important for *tH* analysis are created from existing ones and redundant branches are discarded. In this phase, also branches for machine-learning models outputs are created. Outputs of the models are mostly made from newly created branches.

After this stage, new ntuples with reduced size and new branches, including branches with machine-learning models output, are ready for further analysis. The next step is typically TRExFitter evaluation as described in previous chapters.

Most of the tHbbskimmer code is written in C++ for fast run of the program. Large part of the program is also written in Bash and Python.

Because of the large ntuple sizes and long processing time, complete run of tHbbskimmer typically takes several weeks. Because of this, any code modifications need to be properly tested before used in production.

13.1 Deployment of optimized XGBoost and Tensorflow models

Since XGBoost and Tensorflow are both widely used frameworks in CERN experiments, the decision was to include outputs of both of these models into the tHbbskimmer code.

The addition of machine-leargning models was done in second stage of ntuple skimming, where compiled C++ executable was operating on preprocessed ntuples transfered from GRID and creating a new ones from them. More specifically, in the C++ code it was necessary to create a new slots for models outputs and define a function, which accepts variables for evaluation as arguments and returns output of loaded XGBoost/Tensorflow model.

Since Tensorflow model was not used in previous versions of thbbskimmer, it was necessary to update CMake files and also the installation instructions. Tensorflow framework was included in the project in form of precompiled dynamically linked C API libraries, so no compilation was needed for its run. For an easier use of Tensorflow C API framework, wrapper called Cppflow [25] was used.

Before thbbskimmer is launched on lxplus, run environment is switched to Cent OS 6 environment containing LCG_99 package bundle. This environment also contains some older Tensorflow library files, so it is also necessary to update global variables, so the newly installed Tensorflow libraries are loaded instead of the outdated ones.

Before these changes could be incorporated into the project, it was necessary to properly install and test updated thbbskimmer code in lxplus environment, so the danger of bugs in production run is minimized.

The whole extension of thbbskimmer code was done using git version control system.

Conclusion

After series of optimizations and testings of different machine-learning models, we were able to get a model with mean value of significance over different testing set of 0.39 along with a reasonably low value of variance. The best performing model with these results was an XGBoost model.

Beside the XGBoost model, a neural network model with a very comparable performance was also successfully trained.

An expected median value of signal strength μ with inclusion of systematical and statistical uncertainties was 6.346. Median with inclusion of statistical uncertainties only was 3.89. All results were obtained with XGBoost model.

Bibliography

[1] Standard Model. Available from: https://home.cern/about

[2] LHC. Available from: https://www.space.com/large-hadron-collider-particle-accelerator

[3] Atlas. Available from: https://atlas.cern/

[4] Standard model, physics page. Available from: https://physics.info/standard/

[5] Standard Model. Available from: https://en.wikipedia.org/wiki/Standard_Model

[6] The Higgs boson. Available from: https://home.cern/science/physics/higgs-boson

[7] Vecchio, V.; Aly, M.; et al. Search for associated production of a Higgs boson and a single top quark in multi-lepton final states using *pp* collisions at 13 TeV with the ATLAS detector. Technical report, CERN, Geneva, 2022. Available from: https://cds.cern.ch/record/2802431

[8] Hastie, T.; Tibshirani, R.; et al. *The Elements of Statistical Learning - Data Mining, Inference, and Prediction.* 2017.

[9] Standard Model. Available from: https://home.cern/about

[10] Machine Learning course at Cornell university. Available from: http://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote19.html

[11] Scikit-learn ensemble models. Available from: https://scikit-learn.org/stable/modules/ensemble.html

[12] Machine-learning course at FIT, CTU. Available from: `https://courses.fit.cvut.cz/BI-ML1/`

[13] Development, Application and Representation of Algorithms for Discoveries with the ATLAS Forward Proton (AFP) Detector at CERN, Martin Vatrt. Available from: `https://dspace.cvut.cz/handle/10467/83229`

[14] ROOT framework. Available from: `https://root.cern/`

[15] TMVA documentation. Available from: `https://root.cern/manual/tmva/`

[16] Statistical testing in High energy physics. Available from: `https://atlas-stats-doc-dev.web.cern.ch/atlas-stats-doc-dev/statisticaltests/`

[17] Trex-fitter documentation. Available from: `https://trexfitter-docs.web.cern.ch/trexfitter-docs/settings/`

[18] Patzwahl, M. Performance Tests of tH(bb) Signal and Background Separation Using a Binary Classifier Neural Network. 2021. Available from: `https://cds.cern.ch/record/2789973`

[19] Koval, O. A.; Boyko, I. R.; et al. Higgs boson production in association with a single top quark at the LHC. *Proceedings of the 23rd International Scientific Conference of Young Scientists and Specialists (Ayss-2019)*, 2019, 10.1063/1.5130094.

[20] Vecchio, V.; Aly, M.; et al. Search for associated production of a Higgs boson and a single top quark using pp collisions at 13 TeV with the ATLAS detector in the tH(bb) final state. Technical report, CERN, Geneva, 2020. Available from: `https://cds.cern.ch/record/2713507`

[21] Work with HTCondor on lxplus. Available from: `https://batchdocs.web.cern.ch/local/submit.html`

[22] Panda documentation. Available from: `https://twiki.cern.ch/twiki/bin/view/PanDA/PanDA1`

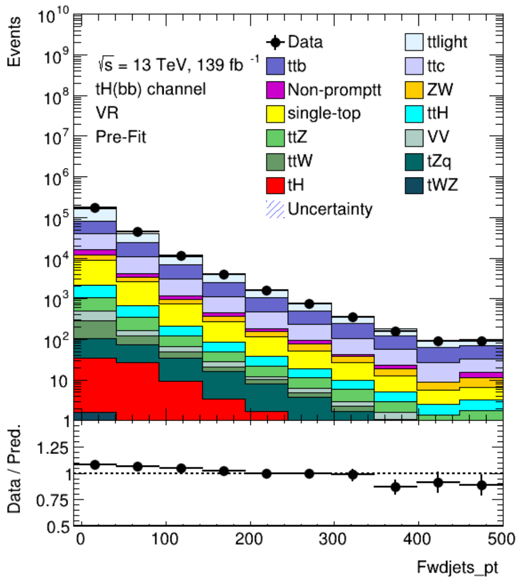
[23] Rucio documentation. Available from: `https://rucio.cern.ch/`

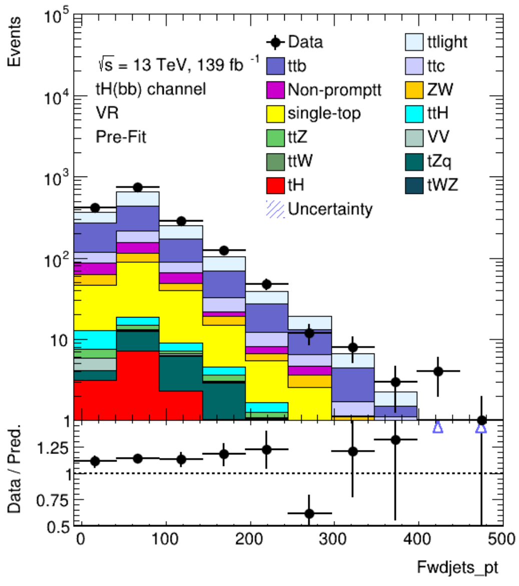
[24] tHbbSkimmer source page. Available from: `https://gitlab.cern.ch/atlasHTop/thbbskimmer`

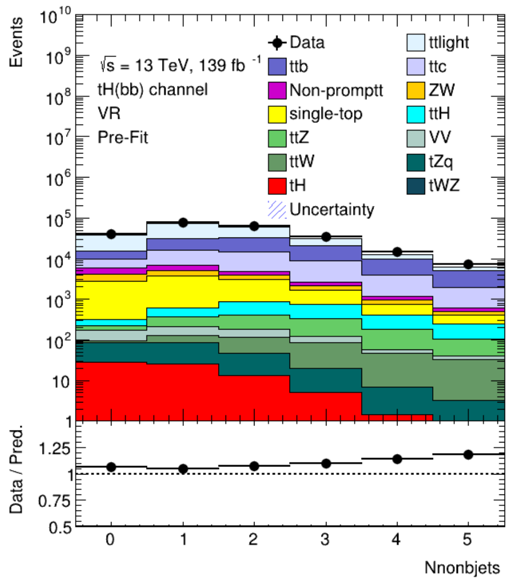
[25] CppFlow - C Api Wrapper source page. Available from: `https://github.com/serizba/cppflow`

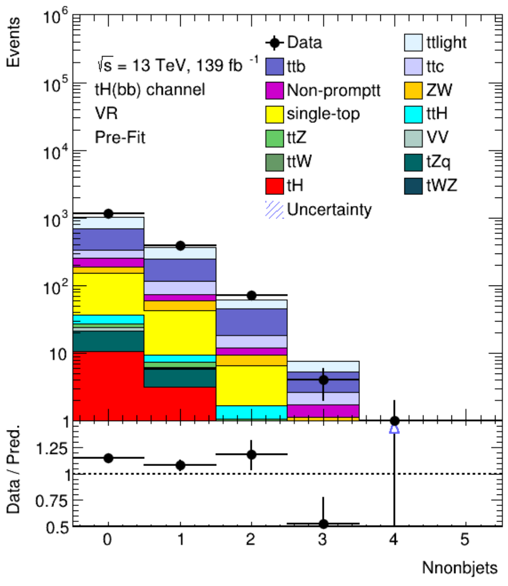
Appendix A

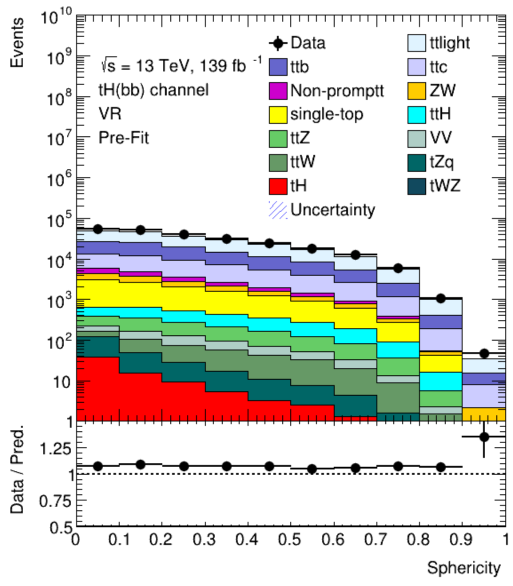
TRExFitter Stacked histograms

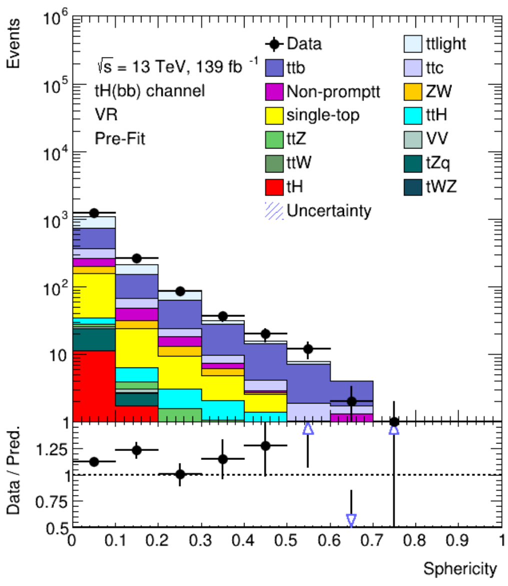


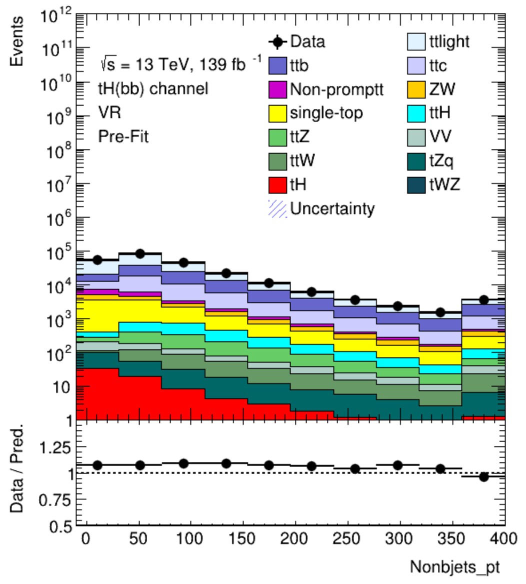



Events
Data / Pred.
√s = 13 TeV, 139 fb⁻¹
tH(bb) channel
VR
Pre-Fit
Data
ttb
Non-promptt
single-top
ttZ
ttW
tH
Uncertainty
ttlight
ttc
ZW
ttH
VV
tZq
tWZ
Nnonbjets

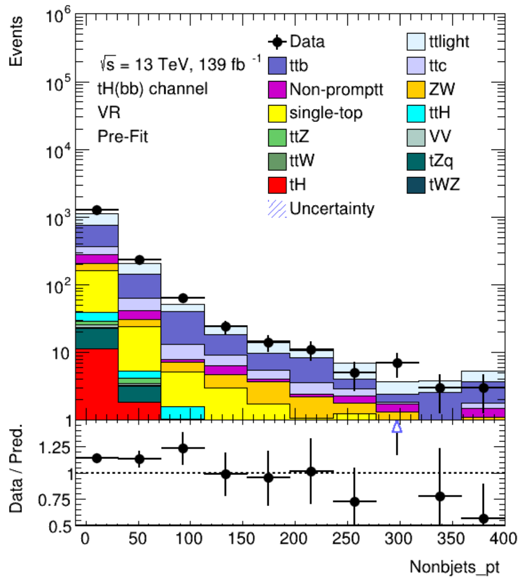

Events
Data / Pred.
√s = 13 TeV, 139 fb⁻¹
tH(bb) channel
VR
Pre-Fit
Data
ttb
Non-promptt
single-top
ttZ
ttW
tH
Uncertainty
ttlight
ttc
ZW
ttH
VV
tZq
tWZ
Nnonbjets


Events
Data / Pred.
√s = 13 TeV, 139 fb⁻¹
tH(bb) channel
VR
Pre-Fit
Data
ttb
Non-promptt
single-top
ttZ
ttW
tH
Uncertainty
ttlight
ttc
ZW
ttH
VV
tZq
tWZ
Sphericity

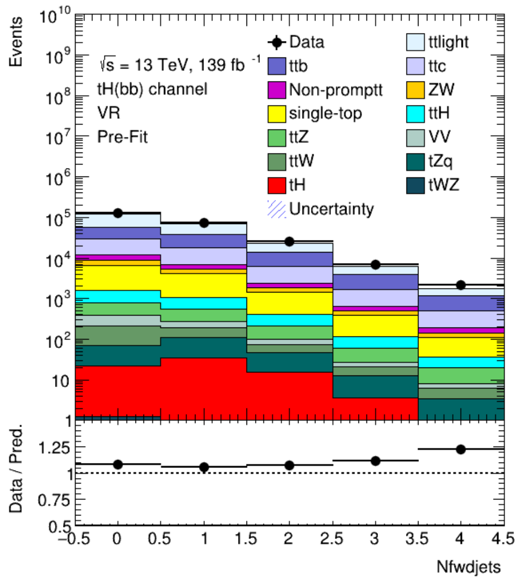

Events
Data / Pred.
√s = 13 TeV, 139 fb⁻¹
tH(bb) channel
VR
Pre-Fit
Data
ttb
Non-promptt
single-top
ttZ
ttW
tH
Uncertainty
ttlight
ttc
ZW
ttH
VV
tZq
tWZ
Sphericity



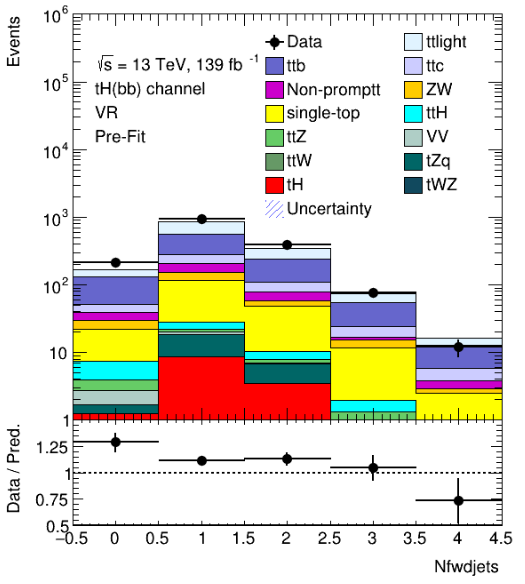
√s = 13 TeV, 139 fb⁻¹
tH(bb) channel
VR
Pre-Fit
Data
ttlight
ttb
ttc
Non-promptt
ZW
single-top
ttH
ttZ
VV
ttW
tZq
tH
tWZ
Uncertainty
Events
Data / Pred.
Nonbjets_pt



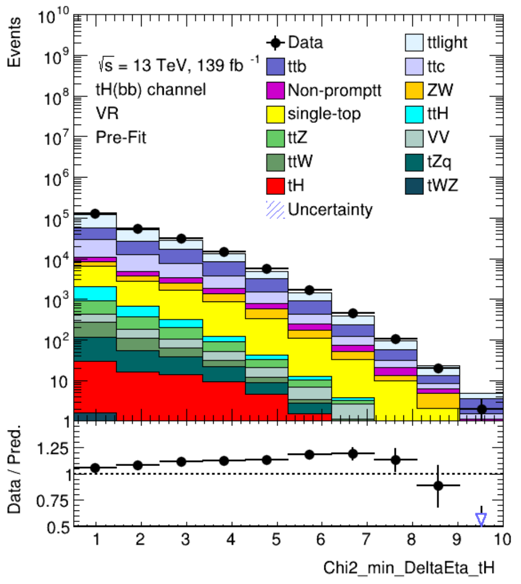
√s = 13 TeV, 139 fb⁻¹
tH(bb) channel
VR
Pre-Fit
Data
ttlight
ttb
ttc
Non-promptt
ZW
single-top
ttH
ttZ
VV
ttW
tZq
tH
tWZ
Uncertainty
Events
Data / Pred.
Nonbjets_pt

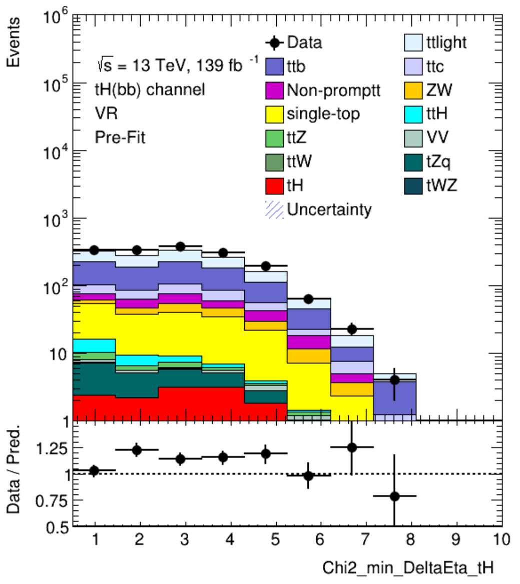


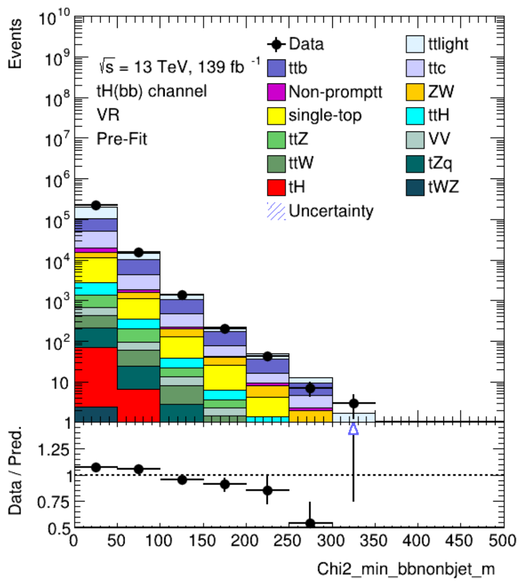
√s = 13 TeV, 139 fb⁻¹
tH(bb) channel
VR
Pre-Fit
Data
ttlight
ttb
ttc
Non-promptt
ZW
single-top
ttH
ttZ
VV
ttW
tZq
tH
tWZ
Uncertainty
Events
Data / Pred.
Nfwdjets

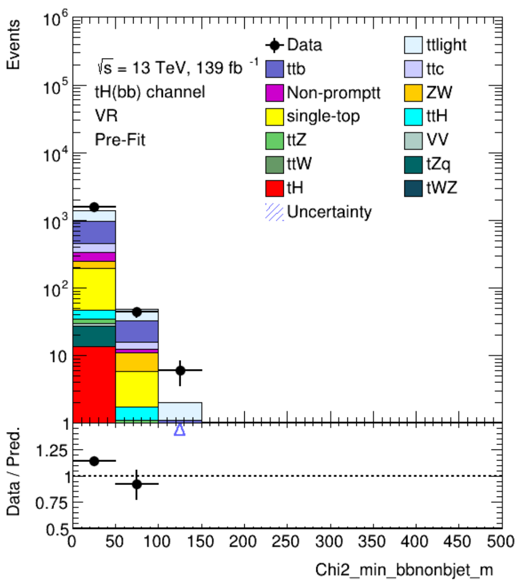


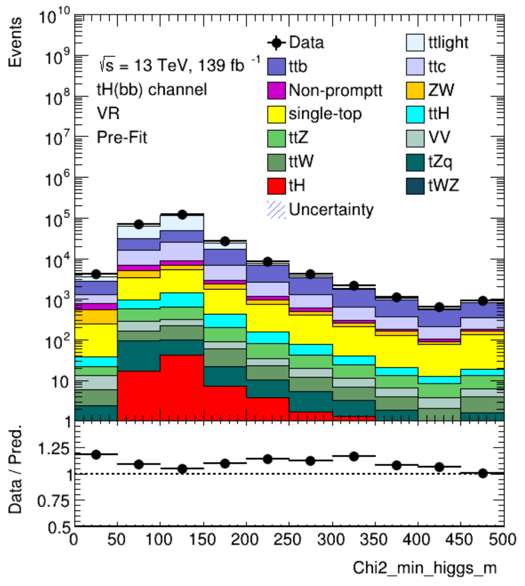
√s = 13 TeV, 139 fb⁻¹
tH(bb) channel
VR
Pre-Fit
Data
ttlight
ttb
ttc
Non-promptt
ZW
single-top
ttH
ttZ
VV
ttW
tZq
tH
tWZ
Uncertainty
Events
Data / Pred.
Nfwdjets


√s = 13 TeV, 139 fb⁻¹
tH(bb) channel
VR
Pre-Fit
Data
ttb
Non-promptt
single-top
ttZ
ttW
tH
Uncertainty
ttlight
ttc
ZW
ttH
VV
tZq
tWZ
Events
Data / Pred.
Chi2_min_DeltaEta_tH

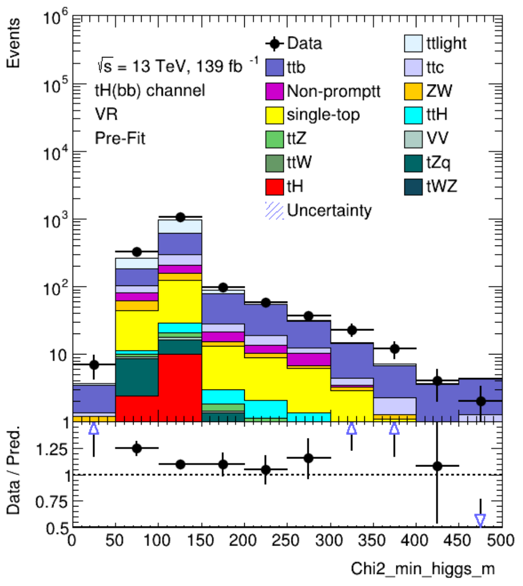

√s = 13 TeV, 139 fb⁻¹
tH(bb) channel
VR
Pre-Fit
Data
ttb
Non-promptt
single-top
ttZ
ttW
tH
Uncertainty
ttlight
ttc
ZW
ttH
VV
tZq
tWZ
Events
Data / Pred.
Chi2_min_DeltaEta_tH


√s = 13 TeV, 139 fb⁻¹
tH(bb) channel
VR
Pre-Fit
Data
ttb
Non-promptt
single-top
ttZ
ttW
tH
Uncertainty
ttlight
ttc
ZW
ttH
VV
tZq
tWZ
Events
Data / Pred.
Chi2_min_bbnonbjet_m

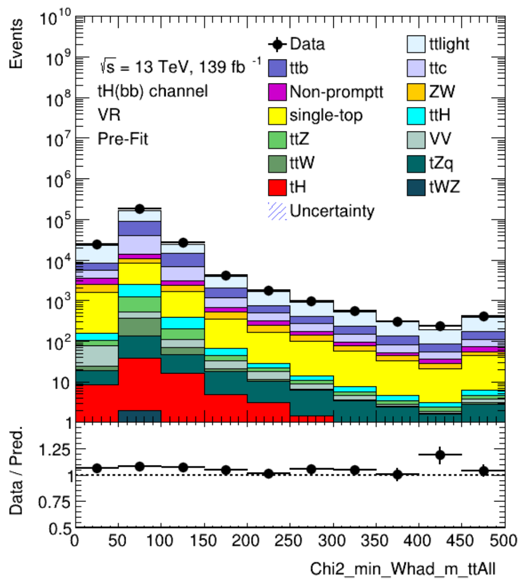

√s = 13 TeV, 139 fb⁻¹
tH(bb) channel
VR
Pre-Fit
Data
ttb
Non-promptt
single-top
ttZ
ttW
tH
Uncertainty
ttlight
ttc
ZW
ttH
VV
tZq
tWZ
Events
Data / Pred.
Chi2_min_bbnonbjet_m



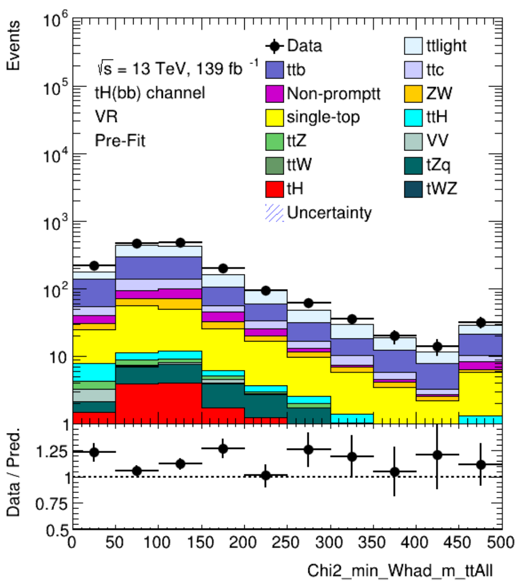
Events
√s = 13 TeV, 139 fb⁻¹
tH(bb) channel
VR
Pre-Fit
Data
ttlight
ttb
ttc
Non-promptt
ZW
single-top
ttH
ttZ
VV
ttW
tZq
tH
tWZ
Uncertainty
Data / Pred.
Chi2_min_higgs_m



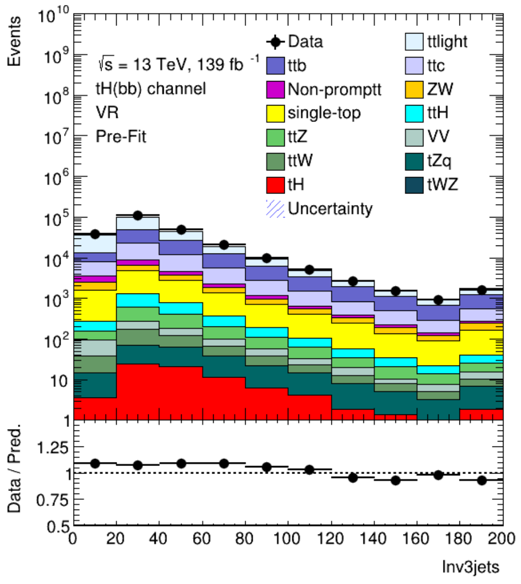
Events
√s = 13 TeV, 139 fb⁻¹
tH(bb) channel
VR
Pre-Fit
Data
ttlight
ttb
ttc
Non-promptt
ZW
single-top
ttH
ttZ
VV
ttW
tZq
tH
tWZ
Uncertainty
Data / Pred.
Chi2_min_higgs_m

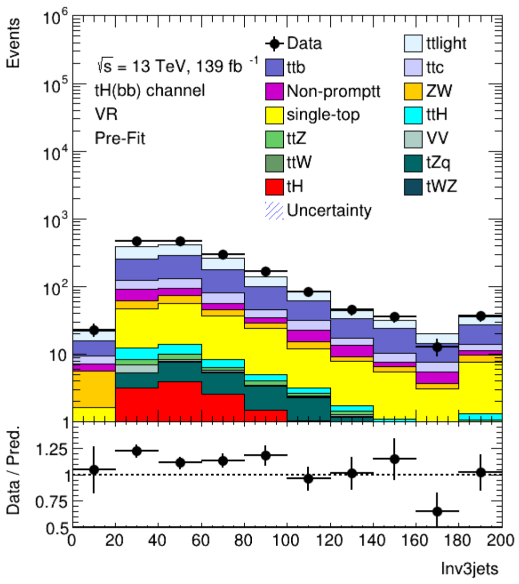


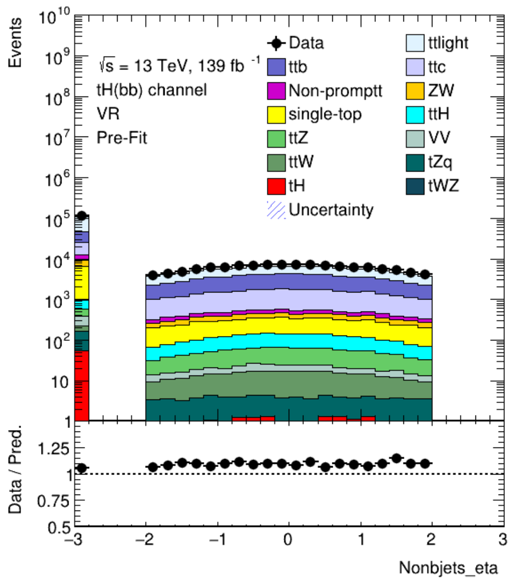
Events
√s = 13 TeV, 139 fb⁻¹
tH(bb) channel
VR
Pre-Fit
Data
ttlight
ttb
ttc
Non-promptt
ZW
single-top
ttH
ttZ
VV
ttW
tZq
tH
tWZ
Uncertainty
Data / Pred.
Chi2_min_Whad_m_ttAll

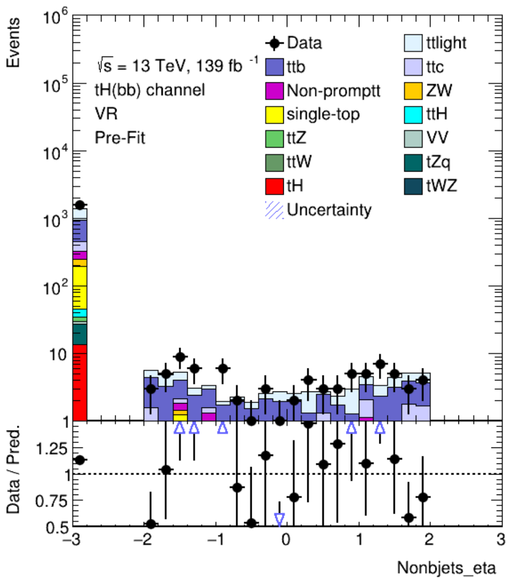


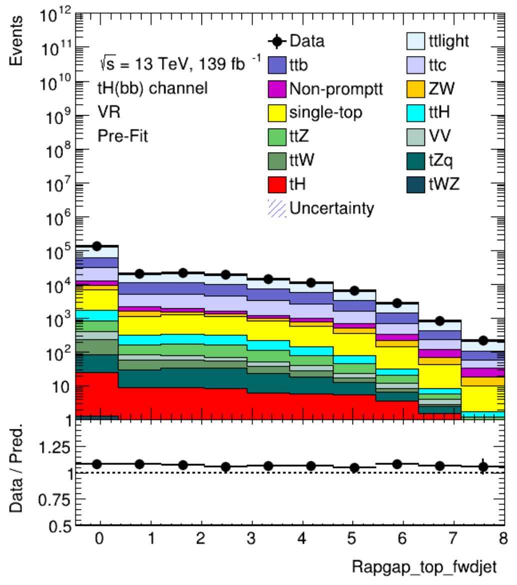
Events
√s = 13 TeV, 139 fb⁻¹
tH(bb) channel
VR
Pre-Fit
Data
ttlight
ttb
ttc
Non-promptt
ZW
single-top
ttH
ttZ
VV
ttW
tZq
tH
tWZ
Uncertainty
Data / Pred.
Chi2_min_Whad_m_ttAll


Events
Data Pred.
√s = 13 TeV, 139 fb⁻¹
tH(bb) channel
VR
Pre-Fit
Data
ttb
Non-promptt
single-top
ttZ
ttW
tH
Uncertainty
ttlight
ttc
ZW
ttH
VV
tZq
tWZ
Inv3jets

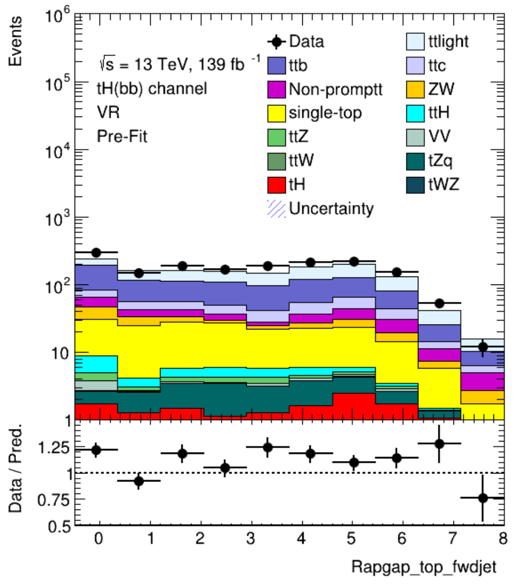

Events
Data / Pred.
√s = 13 TeV, 139 fb⁻¹
tH(bb) channel
VR
Pre-Fit
Data
ttb
Non-promptt
single-top
ttZ
ttW
tH
Uncertainty
ttlight
ttc
ZW
ttH
VV
tZq
tWZ
Inv3jets


Events
Data / Pred.
√s = 13 TeV, 139 fb⁻¹
tH(bb) channel
VR
Pre-Fit
Data
ttb
Non-promptt
single-top
ttZ
ttW
tH
Uncertainty
ttlight
ttc
ZW
ttH
VV
tZq
tWZ
Nonbjets_eta

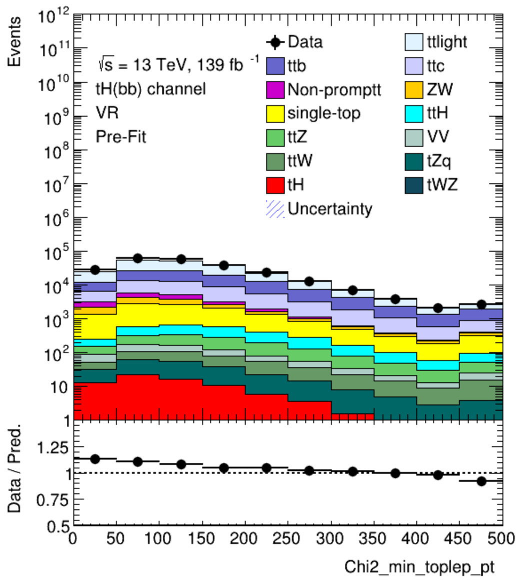

Events
Data / Pred.
√s = 13 TeV, 139 fb⁻¹
tH(bb) channel
VR
Pre-Fit
Data
ttb
Non-promptt
single-top
ttZ
ttW
tH
Uncertainty
ttlight
ttc
ZW
ttH
VV
tZq
tWZ
Nonbjets_eta



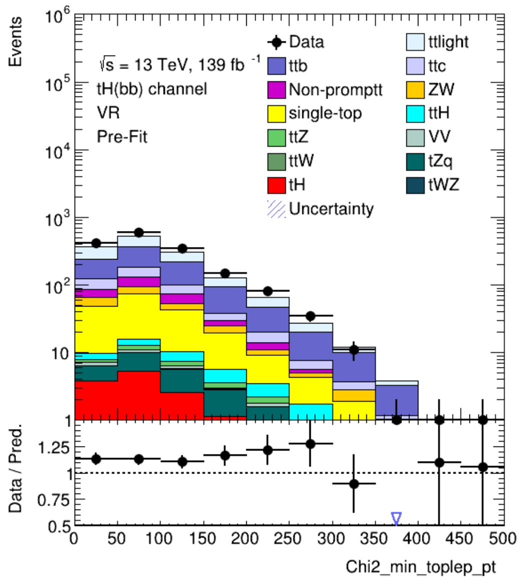
Events
Data / Pred.
√s = 13 TeV, 139 fb⁻¹
tH(bb) channel
VR
Pre-Fit
Data
ttlight
ttb
ttc
Non-promptt
ZW
single-top
ttH
ttZ
VV
ttW
tZq
tH
tWZ
Uncertainty
Rapgap_top_fwdjet



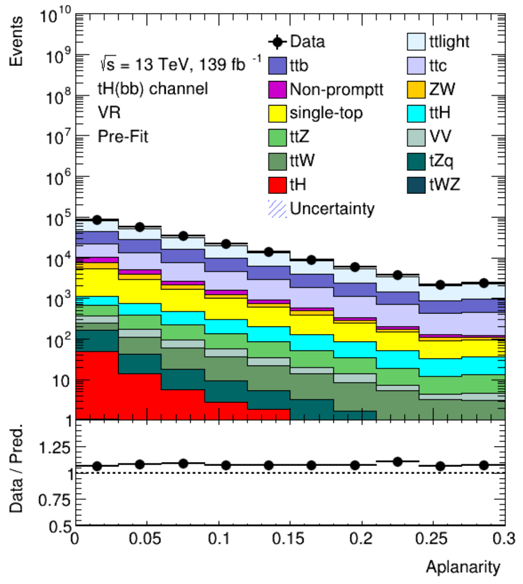
Events
Data / Pred.
√s = 13 TeV, 139 fb⁻¹
tH(bb) channel
VR
Pre-Fit
Data
ttlight
ttb
ttc
Non-promptt
ZW
single-top
ttH
ttZ
VV
ttW
tZq
tH
tWZ
Uncertainty
Rapgap_top_fwdjet

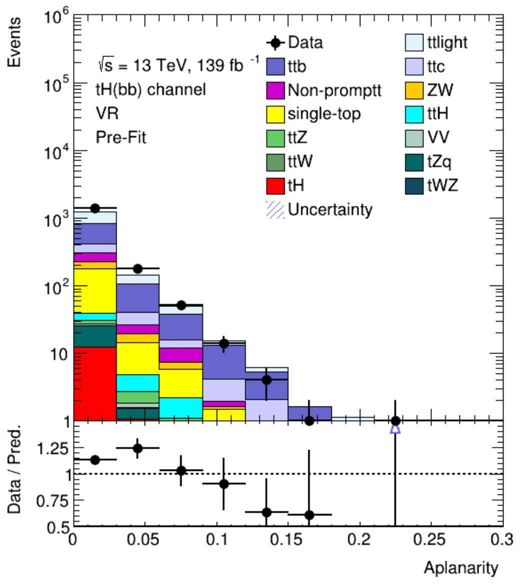


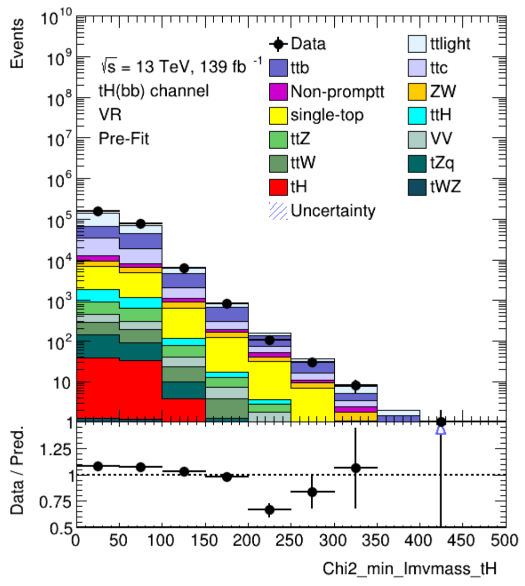
Events
Data / Pred.
√s = 13 TeV, 139 fb⁻¹
tH(bb) channel
VR
Pre-Fit
Data
ttlight
ttb
ttc
Non-promptt
ZW
single-top
ttH
ttZ
VV
ttW
tZq
tH
tWZ
Uncertainty
Chi2_min_toplep_pt

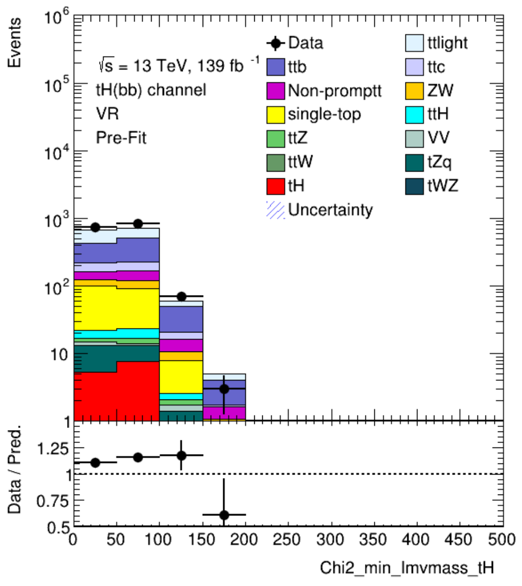


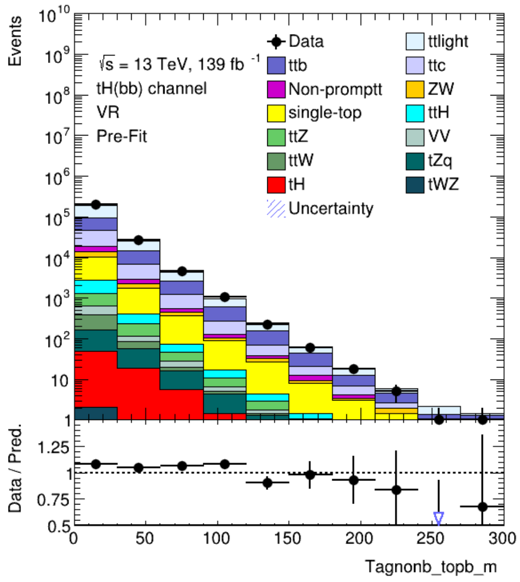
Events
Data / Pred.
√s = 13 TeV, 139 fb⁻¹
tH(bb) channel
VR
Pre-Fit
Data
ttlight
ttb
ttc
Non-promptt
ZW
single-top
ttH
ttZ
VV
ttW
tZq
tH
tWZ
Uncertainty
Chi2_min_toplep_pt

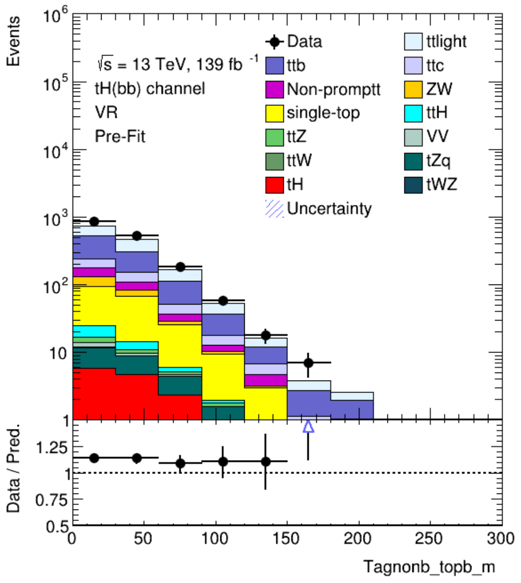

Events
√s = 13 TeV, 139 fb⁻¹
tH(bb) channel
VR
Pre-Fit
Data
ttb
Non-promptt
single-top
ttZ
ttW
tH
Uncertainty
ttlight
ttc
ZW
ttH
VV
tZq
tWZ
Data / Pred.
Aplanarity

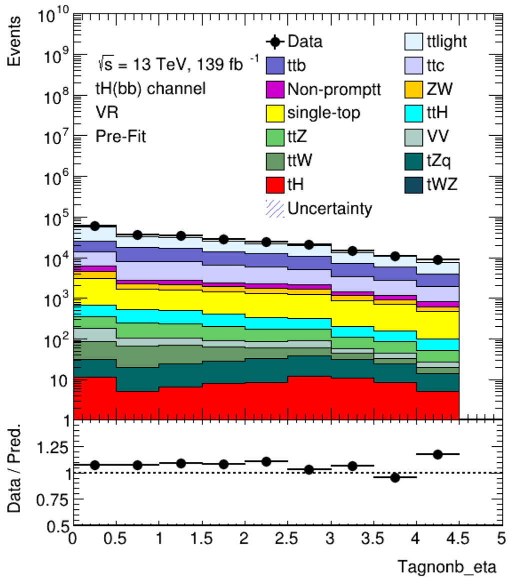

Events
√s = 13 TeV, 139 fb⁻¹
tH(bb) channel
VR
Pre-Fit
Data
ttb
Non-promptt
single-top
ttZ
ttW
tH
Uncertainty
ttlight
ttc
ZW
ttH
VV
tZq
tWZ
Data / Pred.
Aplanarity

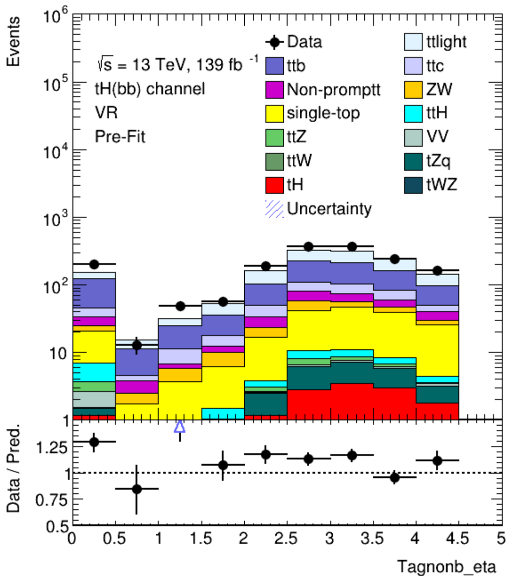

Events
√s = 13 TeV, 139 fb⁻¹
tH(bb) channel
VR
Pre-Fit
Data
ttb
Non-promptt
single-top
ttZ
ttW
tH
Uncertainty
ttlight
ttc
ZW
ttH
VV
tZq
tWZ
Data / Pred.
Chi2_min_Imvmass_tH

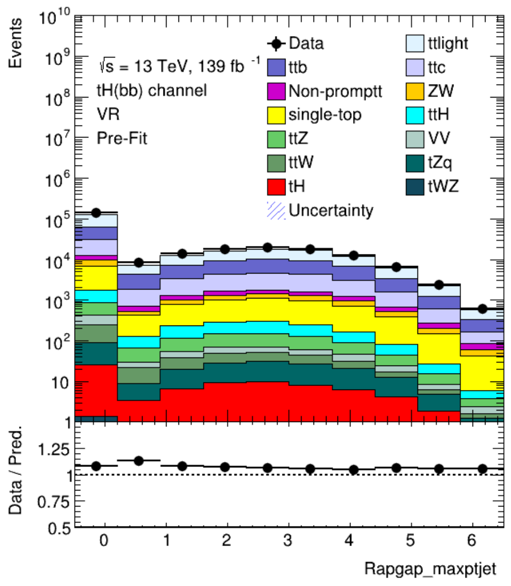

Events
√s = 13 TeV, 139 fb⁻¹
tH(bb) channel
VR
Pre-Fit
Data
ttb
Non-promptt
single-top
ttZ
ttW
tH
Uncertainty
ttlight
ttc
ZW
ttH
VV
tZq
tWZ
Data / Pred.
Chi2_min_Imvmass_tH

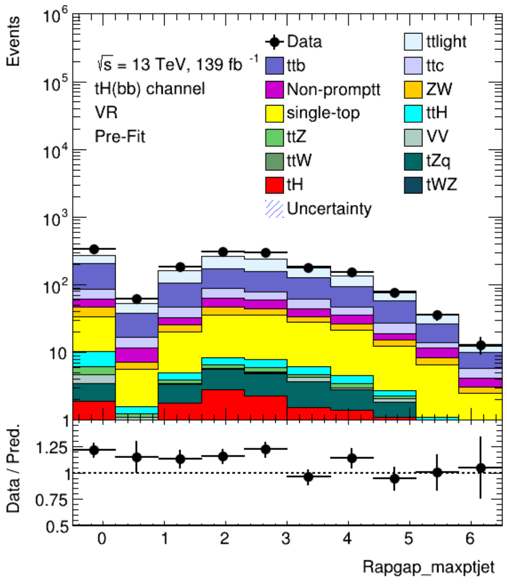

Events
√s = 13 TeV, 139 fb ⁻¹
tH(bb) channel
VR
Pre-Fit
Data
ttb
Non-promptt
single-top
ttZ
ttW
tH
Uncertainty
ttlight
ttc
ZW
ttH
VV
tZq
tWZ
Data / Pred.
Tagnonb_topb_m

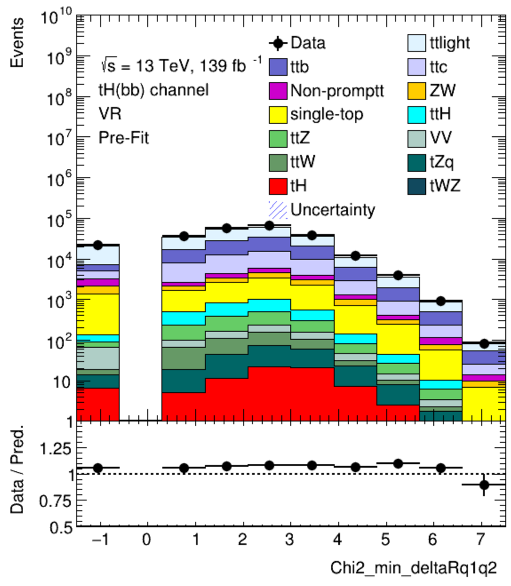

Events
√s = 13 TeV, 139 fb ⁻¹
tH(bb) channel
VR
Pre-Fit
Data
ttb
Non-promptt
single-top
ttZ
ttW
tH
Uncertainty
ttlight
ttc
ZW
ttH
VV
tZq
tWZ
Data / Pred.
Tagnonb_topb_m

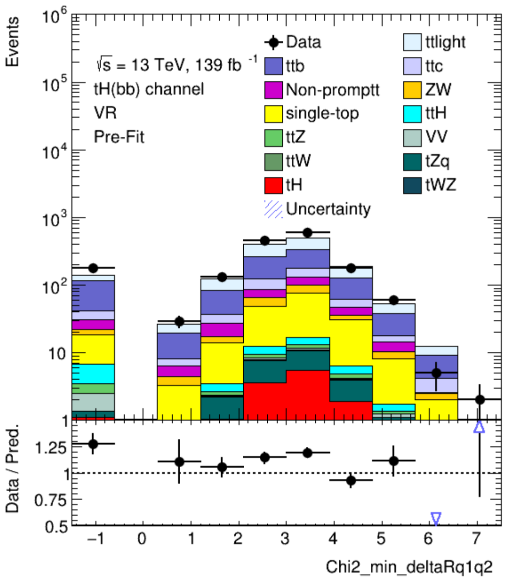

Events
√s = 13 TeV, 139 fb ⁻¹
tH(bb) channel
VR
Pre-Fit
Data
ttb
Non-promptt
single-top
ttZ
ttW
tH
Uncertainty
ttlight
ttc
ZW
ttH
VV
tZq
tWZ
Data / Pred.
Tagnonb_eta


Events
√s = 13 TeV, 139 fb ⁻¹
tH(bb) channel
VR
Pre-Fit
Data
ttb
Non-promptt
single-top
ttZ
ttW
tH
Uncertainty
ttlight
ttc
ZW
ttH
VV
tZq
tWZ
Data / Pred.
Tagnonb_eta


√s = 13 TeV, 139 fb⁻¹
tH(bb) channel
VR
Pre-Fit
Data
ttb
Non-promptt
single-top
ttZ
ttW
tH
Uncertainty
ttlight
ttc
ZW
ttH
VV
tZq
tWZ
Events
Data / Pred.
Rapgap_maxptjet

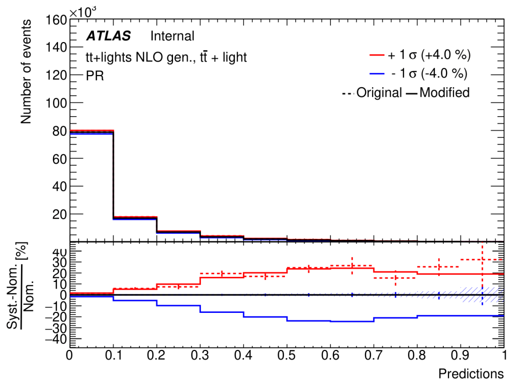

√s = 13 TeV, 139 fb⁻¹
tH(bb) channel
VR
Pre-Fit
Data
ttb
Non-promptt
single-top
ttZ
ttW
tH
Uncertainty
ttlight
ttc
ZW
ttH
VV
tZq
tWZ
Events
Data / Pred.
Rapgap_maxptjet

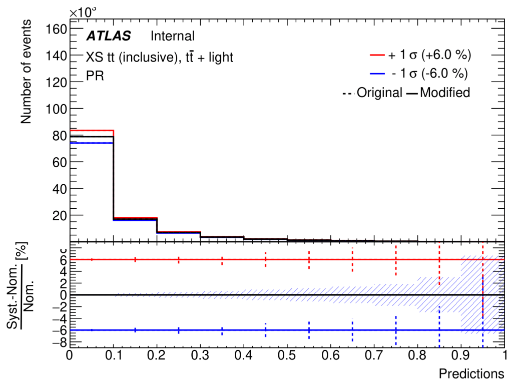

√s = 13 TeV, 139 fb⁻¹
tH(bb) channel
VR
Pre-Fit
Data
ttb
Non-promptt
single-top
ttZ
ttW
tH
Uncertainty
ttlight
ttc
ZW
ttH
VV
tZq
tWZ
Events
Data / Pred.
Chi2_min_deltaRq1q2

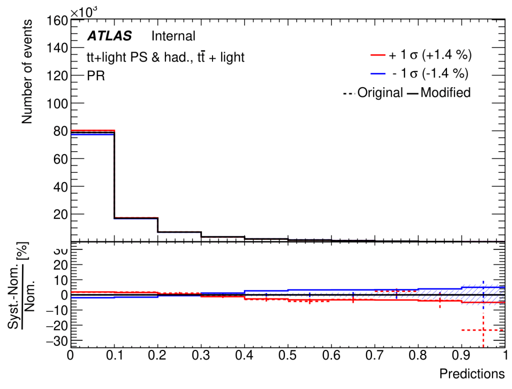

√s = 13 TeV, 139 fb⁻¹
tH(bb) channel
VR
Pre-Fit
Data
ttb
Non-promptt
single-top
ttZ
ttW
tH
Uncertainty
ttlight
ttc
ZW
ttH
VV
tZq
tWZ
Events
Data / Pred.
Chi2_min_deltaRq1q2

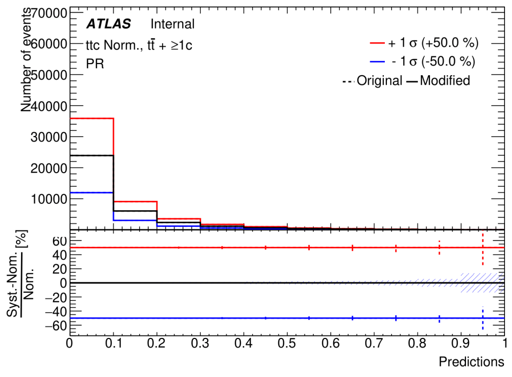
Appendix **B**

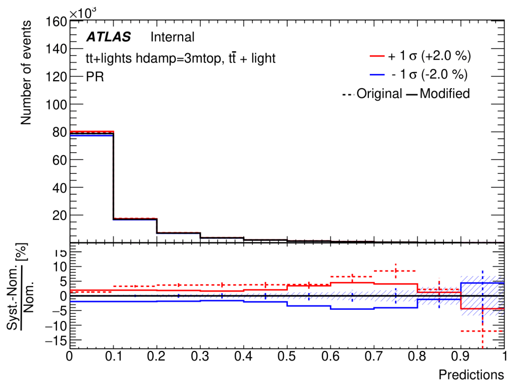
Plots of systematic uncertainties

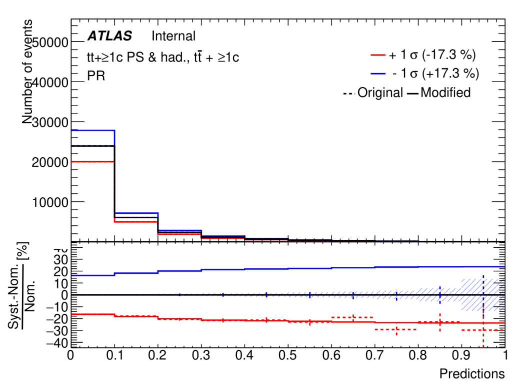


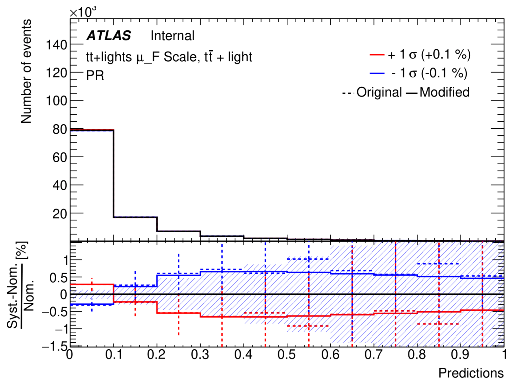


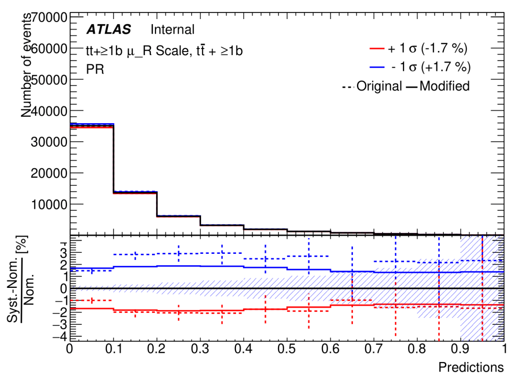


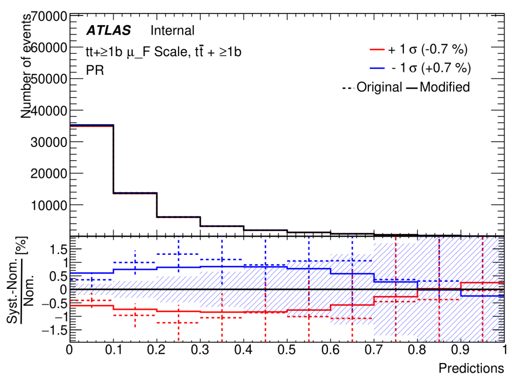


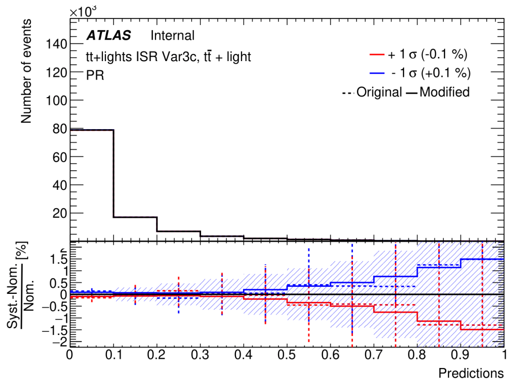

ATLAS Internal
tt+lights hdamp=3mtop, $t\bar{t}$ + light
PR
+ 1 σ (+2.0 %)
- 1 σ (-2.0 %)
-- Original — Modified
Number of events
×10³
Syst.-Nom. / Nom. [%]
Predictions

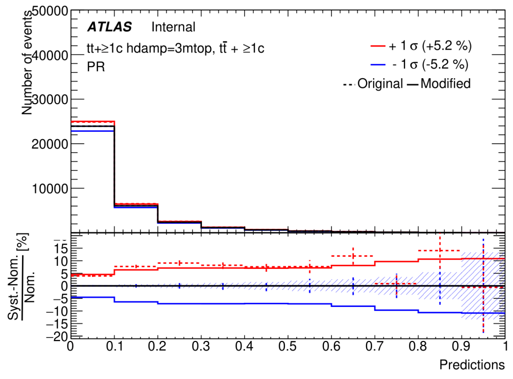

ATLAS Internal
tt+≥1c PS & had., $t\bar{t}$ + ≥1c
PR
+ 1 σ (-17.3 %)
- 1 σ (+17.3 %)
-- Original — Modified
Number of events
Syst.-Nom. / Nom. [%]
Predictions

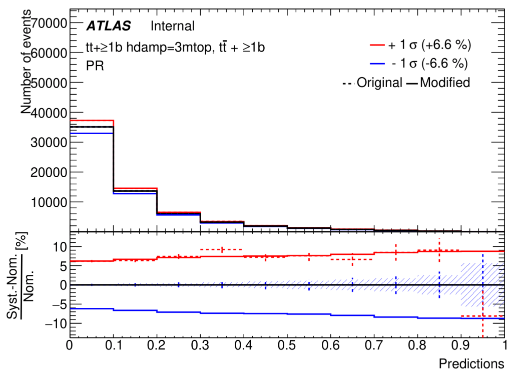

ATLAS Internal
tt+lights μ_F Scale, $t\bar{t}$ + light
PR
+ 1 σ (+0.1 %)
- 1 σ (-0.1 %)
-- Original — Modified
Number of events
×10³
Syst.-Nom. / Nom. [%]
Predictions

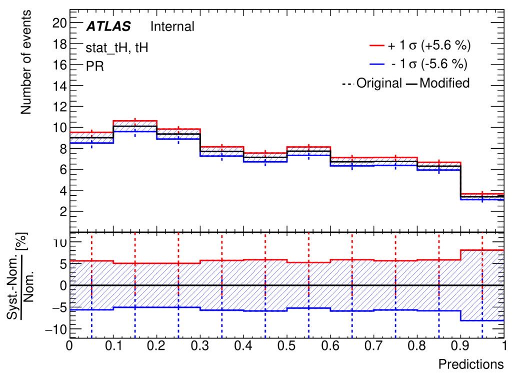

ATLAS Internal
tt+≥1b μ_R Scale, $t\bar{t}$ + ≥1b
PR
+ 1 σ (-1.7 %)
- 1 σ (+1.7 %)
-- Original — Modified
Number of events
Syst.-Nom. / Nom. [%]
Predictions

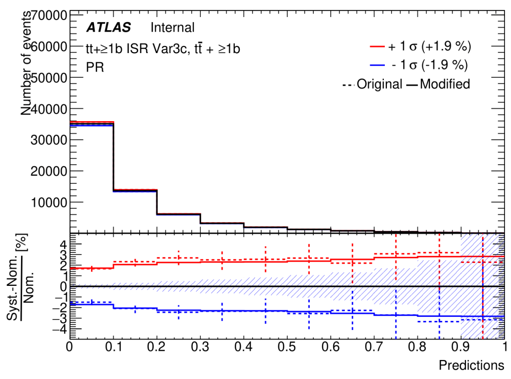

ATLAS Internal
tt+≥1b μ_F Scale, $t\bar{t}$ + ≥1b
PR
+ 1 σ (-0.7 %)
- 1 σ (+0.7 %)
-- Original — Modified
Number of events
Syst.-Nom. / Nom. [%]
Predictions

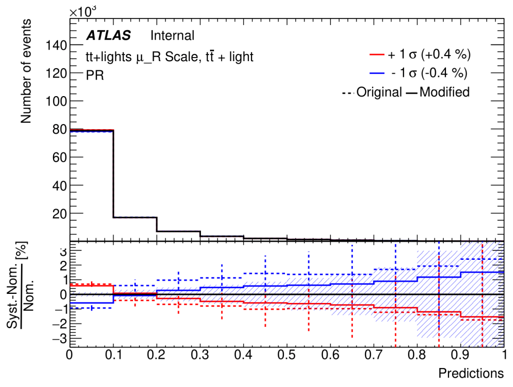

ATLAS Internal
tt+lights ISR Var3c, $t\bar{t}$ + light
PR
+ 1 σ (-0.1 %)
- 1 σ (+0.1 %)
-- Original — Modified
Number of events
×10³
Syst.-Nom. / Nom. [%]
Predictions

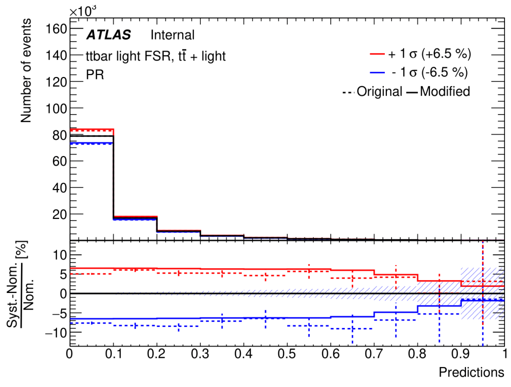

ATLAS Internal
tt+≥1c hdamp=3mtop, $t\bar{t}$ + ≥1c
PR
+ 1 σ (+5.2 %)
- 1 σ (-5.2 %)
-- Original — Modified
Number of events
Syst.-Nom. / Nom. [%]
Predictions

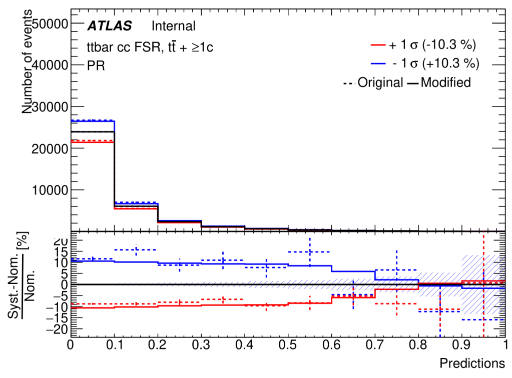

ATLAS Internal
tt+≥1b hdamp=3mtop, $t\bar{t}$ + ≥1b
PR
+ 1 σ (+6.6 %)
- 1 σ (-6.6 %)
-- Original — Modified
Number of events
Syst.-Nom. / Nom. [%]
Predictions

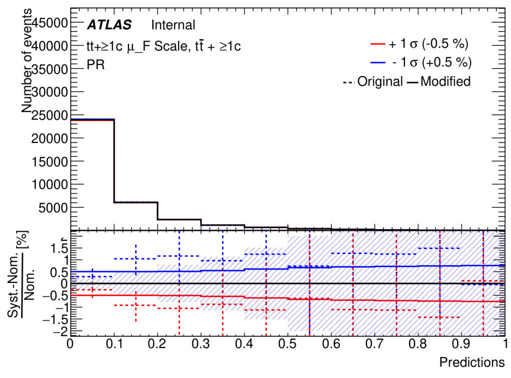

ATLAS Internal
stat_tH, tH
PR
+ 1 σ (+5.6 %)
- 1 σ (-5.6 %)
-- Original — Modified
Number of events
Syst.-Nom. / Nom. [%]
Predictions

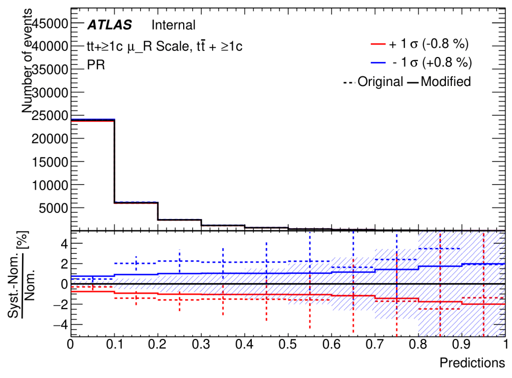

ATLAS Internal
tt+≥1b ISR Var3c, $t\bar{t}$ + ≥1b
PR
+ 1 σ (+1.9 %)
- 1 σ (-1.9 %)
-- Original — Modified
Number of events
Syst.-Nom. / Nom. [%]
Predictions

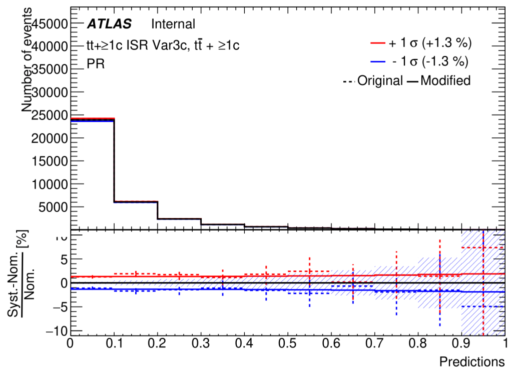

ATLAS Internal
tt+lights μ_R Scale, $t\bar{t}$ + light
PR
+ 1 σ (+0.4 %)
- 1 σ (-0.4 %)
-- Original — Modified
Number of events
Syst.-Nom. / Nom. [%]
Predictions


ATLAS Internal
ttbar light FSR, $t\bar{t}$ + light
PR
+ 1 σ (+6.5 %)
- 1 σ (-6.5 %)
-- Original — Modified
Number of events
Syst.-Nom. / Nom. [%]
Predictions


ATLAS Internal
ttbar cc FSR, tt̄ + ≥1c
PR
+ 1 σ (-10.3 %)
- 1 σ (+10.3 %)
-- Original — Modified
Number of events
Syst.-Nom./Nom. [%]
Predictions


ATLAS Internal
tt+≥1c μ_F Scale, tt̄ + ≥1c
PR
+ 1 σ (-0.5 %)
- 1 σ (+0.5 %)
-- Original — Modified
Number of events
Syst.-Nom./Nom. [%]
Predictions


ATLAS Internal
tt+≥1c μ_R Scale, tt̄ + ≥1c
PR
+ 1 σ (-0.8 %)
- 1 σ (+0.8 %)
-- Original — Modified
Number of events
Syst.-Nom./Nom. [%]
Predictions


ATLAS Internal
tt+≥1c ISR Var3c, tt̄ + ≥1c
PR
+ 1 σ (+1.3 %)
- 1 σ (-1.3 %)
-- Original — Modified
Number of events
Syst.-Nom./Nom. [%]
Predictions

Appendix C

Contents of enclosed CD

`README.txt`	
`thesis.pdf`	Text of the thesis
`TMVA`	Directory of TMVA analysis
`Python`	Directory of Python analysis
`TrexLocal`	Directory of TRExFitter local run
`TrexSyst`	Directory of TRExFitter systematic analysis
`tHbbSkimmer`	Directory changes pushed to tHbbskimmer project

.