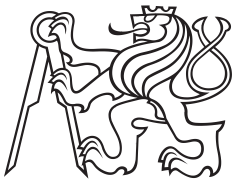
Bachelor Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Machine Learning for the Leptoquark Search Using CERN ATLAS Data

Lukáš Viceník

Supervisor: doc. Dr. André Sopczak
Field of study: Cybernetics and Robotics
May 2022

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Viceník Lukáš** Personal ID number: **483424**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Cybernetics and Robotics**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Machine Learning for the Leptoquark Search Using CERN ATLAS Data

Bachelor’s thesis title in Czech:

Hledání leptoquarků pomocí strojového učení v datech z CERN ATLAS experiment

Guidelines:

At the Large Hadron Collider (LHC) at CERN protons are collided and the collisions are recorded by the ATLAS detector. A motivation to construct the LHC has been the search for new particles. Such new particles could be Leptoquarks. Leptoquarks are predicted by several theories, but so far have not been detected in the rec-orded data. Their mass is unknown. Leptoquark signal events have been simulated, together with other events re-sulting from background reactions. The task is to recognize these simulated signal events automatically from other (background) events using the techniques of machine learning and possibly deep learning. Instructions:
1. Get familiar with the data and basic principles of searching for elementary particles in high-energy physics.
2. Get familiar with the existing implementation of classifiers to separate events of interest from the background, using high-level features and classical machine-learning techniques.
3. Design and implement a classifier based on high-level features using either classical machine learning or deep-learning or both.
4. Evaluate its performance on simulated data and determine the leptoquark sensitivity over a large mass range.
5. Compare and discuss the obtain sensitivity with previous results.
BONUS:
Study the effect of systematic uncertainties of the feature simulations on the performance of the machine learning results and the corresponding reduction in sensitivity.

Bibliography / sources:

[1] https://atlas.cern (“learn more”) An introduction to the ATLAS experiment for the public
[2] Dan Guest et al. Deep Learning and its application to LHC Physics. Annu. Rev. Nucl. Part. Sci. 2018, 68:1-22
[3] Pierre Baldil et al: Searching for Exotic Particles in High-Energy Physics. arXiv:1402.4735
[4] ATLAS Collaboration: Search for pair production of third-generation scalar leptoquarks decaying into a top quark and a tau-lepton in pp collisions at √s = 13 TeV with the ATLAS detector. JHEP 06 (2021) 179
[5] A.Sopczak, Searches for Leptoquarks with the ATLAS Detector, arXiv:2107.10094
[6] R.Duda, P.Hart, D.Stork: Pattern classification. Willey-Interscience, 2000
[7] Goodfellow, Bengio, Courville: Deep learning. MIT Press. 2016

Name and workplace of bachelor’s thesis supervisor:

doc. Dr. André Sopczak Institute of Experimental and Applied Physics CTU Prague

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **28.01.2022** Deadline for bachelor thesis submission: **20.05.2022**

Assignment valid until: **30.09.2023**

doc. Dr. André Sopczak Supervisor’s signature	prof. Ing. Tomáš Svoboda, Ph.D. Head of department’s signature	prof. Mgr. Petr Páta, Ph.D. Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

. _____ Date of assignment receipt

_____ Student's signature

Acknowledgements

Thank you to my supervisor, doc. André Sopczak, who has given me a lot of his time and advice and whose patience and optimism always motivated me to continue working under any circumstances. Thank you to my consultant, prof. Jan Kybic, who has given me valuable advice and remarks. And thank you to my family, for never ending support.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

In Prague, May 20, 2022

Abstract

In this thesis, we improve the cross-section limit for pair production of third-generation scalar leptoquark decaying into a top quark and a τ-lepton and design a method to predict its mass. Events are selected if they have two light leptons (electron or muon) of the same sign and exactly one hadronically decaying τ-lepton.
Algorithms from two machine learning categories widely used for tabular data classification, gradient boosting decision trees and deep neural networks, are deployed to analyze simulated data for leptoquark masses from 300 to 2000 GeV. The data for all available masses are combined to show that one universal classifier can be used for all Leptoquark mass cases. The dependence of the performance on the number of features and the size of the simulated data set is demonstrated.
Finally, the TRExFitter program developed in CERN is used to achieve reliable results for cross-section limit calculation. Additionally we study how to recognize Leptoquark mass using another connected classifier.

Keywords: CERN, ATLAS, Leptoquark, Machine learning, Deep learning, cross section, CatBoost, TabNet, LightGBM, XGBoost, MLP, Scikit-learn, Optuna, ROOT

Supervisor: doc. Dr. André Sopczak

Abstrakt

V této práci vylepšíme hodnotu cross-section limitu pro párovou produkci skalárních Leptokvarků třetí generace při rozpadu na top quark a τ-lepton. Událost je vybrána pokud obsahuje dva lehké leptony (elektron nebo muon) stejného znaménka nebo jeden hadronicky se rozpadající τ-lepton. Algoritmy ze dvou široce používaných kategorií strojového učení jsou nasazeny k analýze simulovaných dat pro hmotnosti Leptokvarků od 300 do 2000 GeV. V rámci klasifikace jsou data zkombinována za účelem dokázat, že je možné využít jednoho univerzálního klasifikátoru pro analýzu libovolné hmotnosti. Pro dva nejlépe predikující klasifikátory jsou provedeny experimenty snížení počtu vstupních příznaků a velikosti datasetu pro trénování. Na závěr je použit program TRExFitter vyvynutý v CERNu k dosažení věrohodných výsledků při výpočtu cross-section limitu. Navíc prověříme, jak odhadnout hmotnost Leptokvarků s použitím dalšího napojeného klasifikátoru.

Klíčová slova: CERN, ATLAS, Leptoquark, Machine learning, Deep learning, cross section, CatBoost, TabNet, LightGBM, XGBoost, MLP, Scikit-learn, Optuna, ROOT

Překlad názvu: Hledání leptoquarků pomocí strojového učení v datech z CERN ATLAS experiment

Contents

Figures

Tables

Chapter 1

Introduction

A new idea is that there could be a possibility to reinterpret results from Higgs boson searches as limit for Leptoquarks [1]. A difference is that while the Higgs boson of the Standard Model has a known mass, for Leptoquarks that are beyond the Standard Model of particle physics which means have not yet been discovered, the whole range of masses has to be analyzed. For every mass we have to delimit the cross-section which is proportional to the number of expected particles in the detector. The cross-section is established with precision of 2σ significance. In this thesis, we consider only particles decaying into the channel $2lSS + 1\tau$. This means two light leptons (electron and muon) of the same-sign electric charge, and one hadronically decaying tau lepton. After this pre-selection, we apply a gradient boosted decision trees and neural networks to separate simulated Leptoquark events from other events. When the separation is completed, we have studied the Leptoquark mass determination possibilities.

Part I

Theory

Chapter 2

CERN

The European Organization for Nuclear Research, known as CERN is the largest particle physic laboratory in the world. Its name is derived from French *Conseil Européen pour la Recherche Nucléaire*. The main function of CERN is to provide the particle accelerators and other infrastructure needed for high-energy physics research [2].
The laboratory is located just outside of Geneva, Switzerland, it was established in 1954 as one of the first European joint projects that today has 23 member states.

Major successes were the discovery of particles called the W and Z boson in 1983, the invention of the World Wide Web in 1989 by development of the HTTP protocol. In 1995, atoms of hydrogen antimatter counterpart were created. In 2000, a new state of matter called quark-gloun plasma was discovered. The last major event was the observation of the Higgs boson in 2012, at that time the Large Hadron Collider was already deployed [3].

2.1 LHC

The Large Hadron Collider (LHC) is the main accelerator at CERN, located in France and Switzerland as shown in Figure 2.1. It was built between 1998 and 2008 as the world's largest and most powerful particle collider. The 27-kilometer LHC ring, located 100 m underground, is the last element on a succession of accelerators. The particle beams are accelerated just below

the speed of light before they are forced to collide with counter-circulating beams, reaching a total collision energy of 14 TeV [3].

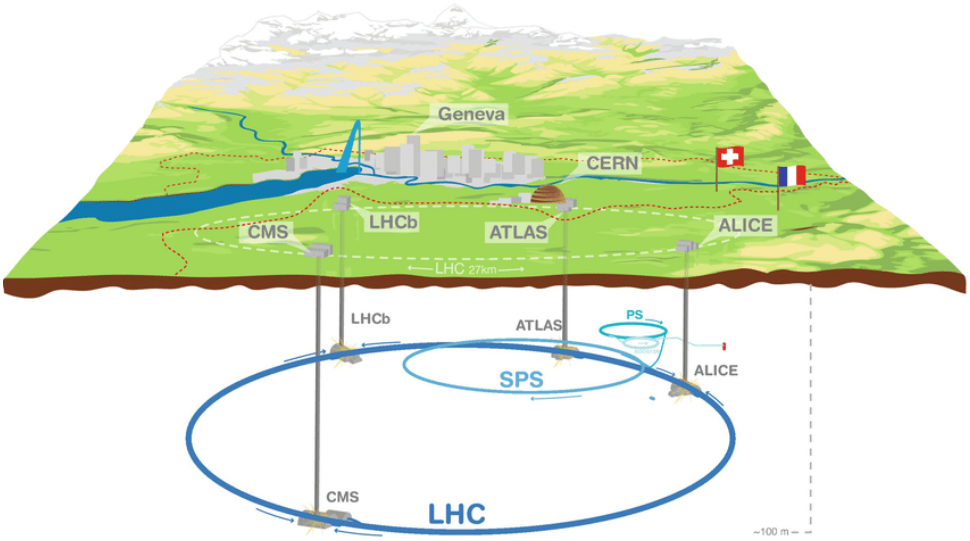


Figure 2.1: Overall view of the Large Hadron Collider, including the ATLAS, CMS, ALICE and LHCb experiments. Figure taken from [3].

2.2 ATLAS

ATLAS is the largest volume detector ever constructed for a particle collider. It has dimensions of a cylinder, is 46 m long and has 25 m in diameter. It is placed 100 meters underground and its weight is 7000 tonnes.
It consists of an inner tracking detector, which is surrounded by a superconducting solenoid providing an axial 2 T magnetic field, an electromagnetic and hadronic calorimeter, and a muon spectrometer. The detector model is shown in Figure 2.2. When proton-proton collisions occur, the produced particles are detected in sub-detectors according to their type. A trigger system selects 100 collisions per second out of 10^9 and sends them to the storage. Data are later analyzed with Offline computing methods [4].

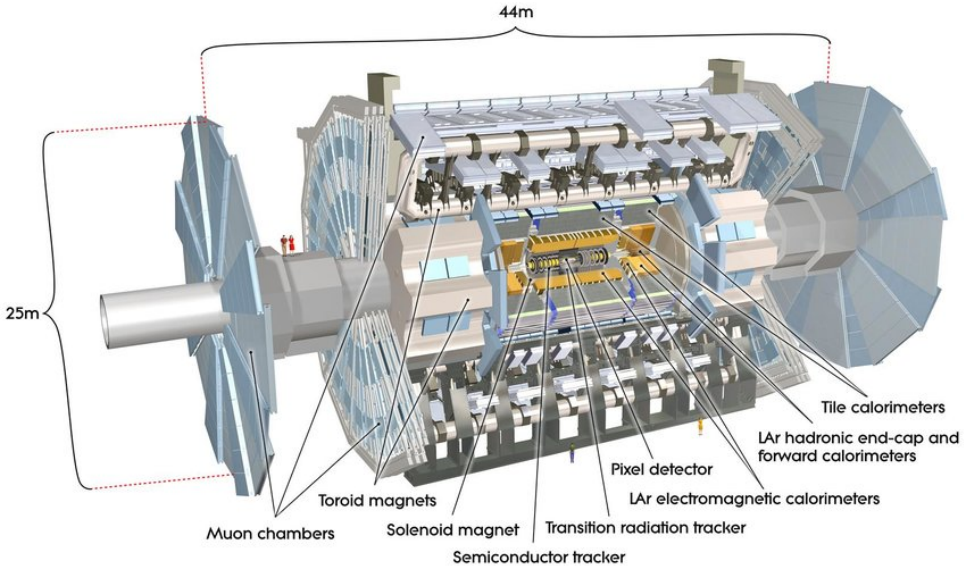


Figure 2.2: Computer generated image of the whole ATLAS detector. Figure taken from [3].

Chapter 3

Leptoquark theory

3.1 Cross-section

Given the particle approaching another particle, the cross-section for this process is the probability that the particles interact which each other.
Let us denote cross-section with a letter σ. The used units are barns, denoted b. For our use, fb or pb are the most suitable units. For better intuition, we can write

$$1 \text{ barn} = 10^{-24}\text{cm}^2. \tag{3.1}$$

Figure 3.1 shows the graphical interpretation of cross-section. The figure was taken from [5]. More information can also be found there.

3.2 Luminosity

The quantity that measures the ability of a particle accelerator to produce the required number of interactions is called the luminosity and is the proportionality factor between the number of events per second dR/dt and the cross-section σ. The equation

$$\frac{dR}{dt} = \mathcal{L}(t) \cdot \sigma \tag{3.2}$$

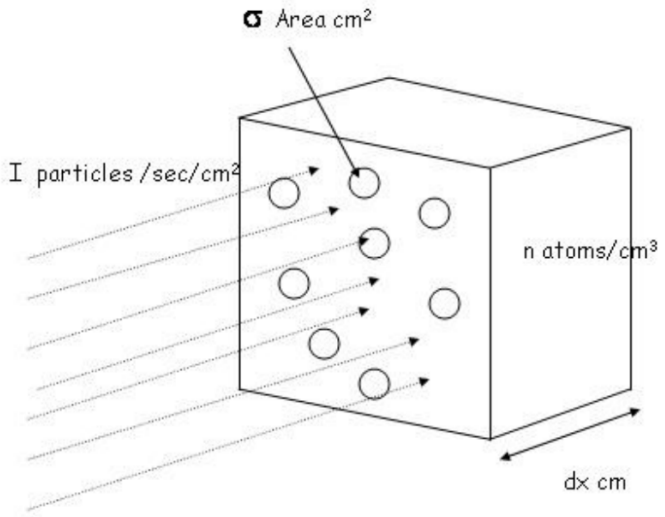


Figure 3.1: The concept of cross-section [5].

relates the number of events per second, luminosity and cross-section as mentioned. Therefore, the unit is $\mathrm{cm}^{-2}\mathrm{s}^{-1}$ [6]. In this thesis, integrated luminosity (L) is used, which we can express as

$$\mathrm{L} = \int_{t_1}^{t_2} \mathcal{L}(t)\, dt, \tag{3.3}$$

where we integrate over operational time. The final formula is

$$R = L \cdot \sigma. \tag{3.4}$$

The unit is cm^{-2} or b^{-1}.

3.2.1 Branching ratio

The branching ratio, BR or β, is the fraction of particles decaying to a particular final state [7].

3.3 Yukawa coupling

Yukawa coupling is an interaction between particles according to Yukawa potential [8]. This potential can be written as

$$V_{Yukawa}(r) = -g^2 \frac{e^{-\alpha m r}}{r}, \tag{3.5}$$

where g is a magnitude scaling constant, m is the mass of the particle, r is radial distance and α is scaling constant, so that $r \approx \frac{1}{\alpha m}$. If $m = 0$ Yukawa potential is reduced to Coulomb potential, and we can write,

$$V_{Coulomb}(r) = -g^2 \frac{1}{r}, \tag{3.6}$$

$$\text{where } g^2 = \frac{q_1 q_2}{4\pi\epsilon_0}, \tag{3.7}$$

which is well known equation for Coulomb potential.

3.4 Quarks

Quarks are elementary particles of the Standard Model [9]. There are six flavors of quarks with spin 1/2 specified and separated by generations in Table 3.1. Each of the six flavors of quarks can have three different colors [9]. Quarks form so-called color triplets with charge quantized into multiples of $1/3e$. They are invariant under rotations in color space, which implies $SU(3)$ color symmetry. Quarks can therefore form color singlets according to rules of $SU(3)$. It is believed that all observed particles are colorless. Therefore, quarks can not exist in a free state [10].

Quark	Symbol	Charge	Mass [MeV]
Up	u	+2/3	1.7-3.3
Down	d	-1/3	4.1-5.8
Charm	c	+2/3	1270
Strange	s	-1/3	101
Top	t	+2/3	172000
Bottom	b	-1/3	4190, or 4670

Table 3.1: Overview of lepton flavors with corresponding symbol, anti-particle and mass. Generations of particles are separated by lines.

3.5 Leptons

Leptons are particles of the Standard Model which do not undergo strong interaction. There are two classes of leptons, charged and neutral. For every lepton exists its antiparticle. Details are described in Table 3.2, where generations correspond to quark generations.

Lepton	Symbol	Anti-particle	Mass $[MeV/c^2]$
Electron	e^-	e^+	0.511
Neutrino	ν_e	$\bar{\nu_e}$	$7 \cdot 10^{-6}$
Muon	μ^-	μ^+	105.7
Neutrino (Muon)	ν_{mu}	$\bar{\nu_{mu}}$	0.27
Tau	τ^-	τ^+	1777
Neutrino (Tau)	ν_{tau}	$\bar{\nu_{tau}}$	0.31

Table 3.2: Overview of lepton flavors with corresponding symbol, charge and mass. Generations of particles are separated by line.

3.6 Leptoquarks

Leptoquarks are hypothetical particles placed beyond the Standard Model which combine properties of both leptons and quarks, particles of the Standard Model. They are supposed to be color triplet bosons with fractional charge that interact with quarks and leptons. Figure 3.2 shows the Leptoquark decay into a lepton and a quark. The Yukawa coupling is determined by two parameters: the branching ratio β, and a coupling parameter λ. For lepton, we have $\lambda_l = \sqrt{\beta}\lambda$ and for neutrino and quark $\lambda_\nu = \sqrt{1-\beta}\lambda$.

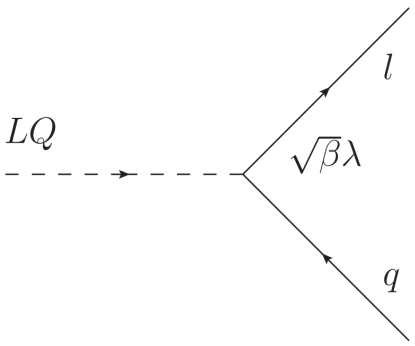


Figure 3.2: Feynamn diagram showing the Yukawa coupling $\lambda_l = \sqrt{\beta}\lambda$ between a Leptoquark, a lepton (l) and a quark (q).

There are three main production modes, pair-production, single-production and off-shell production. For our purpose, the first mentioned mode is relevant. It has the Yukawa coupling described with λ and has large cross-section. Figure 3.3 shows a Feynman diagram for Leptoquark production. We can see that two Leptoquarks are produced, and each decays into a quark and a lepton.

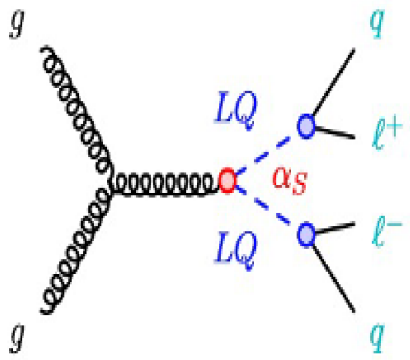


Figure 3.3: Feynman diagram of pair-production of Leptoquarks, taken from [1].

3.7 Higgs boson decay similarity

Recently, the possibility of reinterpretation of Higgs boson searches was noticed [1]. Figure 3.4 compares final states of Leptoquark pair and $t\bar{t}H$ production. We note that the final states are identical. The mentioned phenomenon is crucial for this work. The same methods as for the Higgs boson search can be used because of this remarkable identity.

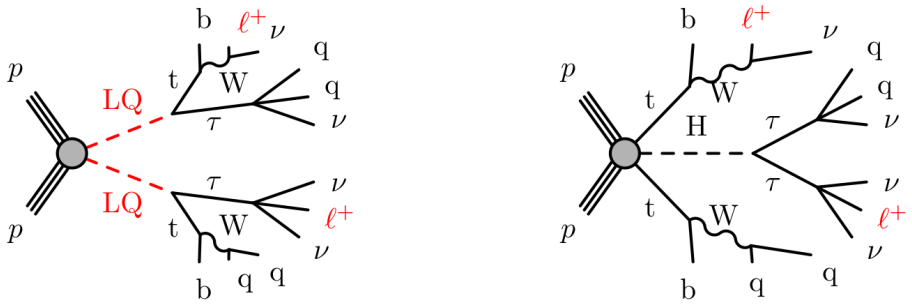


Figure 3.4: Left: Leptoquark decay after pair production, with final state identical to Higgs boson and two top quarks. Right: Production of Higgs boson and two top quarks.

3.8 State-of-the-art research

A search for pair production of third-generation scalar Leptoquark decaying into a top quark and $\tau - lepton$ was performed before [11]. The search is based on a dataset of pp collisions at $\sqrt{s} = 13$ TeV recorded with the ATLAS detector during Run 2 of the Large Hadron Collider, corresponding to an integrated luminosity of 139 fb^{-1}. Events are selected if they have one light lepton (electron or muon) and at least one hadronically decaying $\tau -$ lepton, or at least two light leptons. In addition, two or more jets, at least one of which must be identified as containing $b - hadrons$, are required. Six final states, defined by the multiplicity and flavor of lepton candidates, are considered in the analysis [11] and [12].
Six final states, termed channels were analyzed, defined by multiplicity and flavor of the lepton candidates:

- $1l+ \geq 1\tau$: one light lepton and at least one τ-had candidate.
- $2lOS+ \geq 1\tau$: two opposite-charge (denoted by OS, standing for opposite-sign) light leptons and at least one τ-had candidate.
- $2lSS/3l+ \geq 1\tau$ two same-charge (denoted by SS, standing for same-sign) light leptons or three light leptons, and at least one τ-had candidate.
- $2lOS/ + 0\tau$: two OS light leptons and no τ-had candidates.
- $2lSS + 0\tau$: two SS light leptons and no τ-had candidates.
- $3l + 0\tau$: three light leptons and no τ-had candidates.

In this thesis $2lSS + 1\tau$ is analyzed. The $2lSS/3l+ \geq 1\tau$ is the most similar channel, therefore we compare this channel result with our result.
For comparison, we need to determine the so-called expected cross-section (σ_e) and compare in with the cross-section given to us by theory, the co-called theoretical cross-section (σ_t). Figure 3.5 shows the expected cross-section (dashed line) and the observed cross-section (solid line) limit at 95% confidence level for $2lSSor3l+ \geq 1\tau_{had}$. A comprehensive description of how cross-section is computed will be given in the following text. Symbols for cross-section will be used only for equation purposes to avoid confusion with significance units.

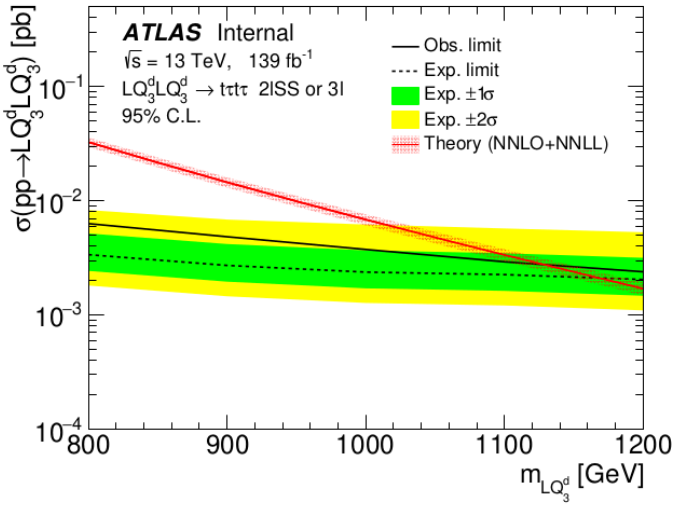


Figure 3.5: The 95% confidence level observed and expected upper limit in $2lSSor3+ \geq 1\tau_{had}$ channel after taking all systematics uncertainties into account [11][12](Figure 75).

Chapter 4

Machine learning algorithms

For this analysis, we decided to use algorithms from two families that are used the most for tabular data classification nowadays, gradient boosted decision trees (GBDT) and neural networks. Neural networks are a very frequently used algorithm to solve problems related to machine learning, but when we look at results of many kaggle competitions, GBDTs have the upper hand [13]. Therefore, we would like to verify this fact.

4.1 Boosted decision trees

As mentioned in the introduction part of this chapter, gradient boosted decision trees and neural networks are used for the analysis. At first, we focus on decision trees. Three most frequently used algorithms nowadays are XGBoost, CatBoost and LightGBM. We will start our explanation with XGBoost as basic model. Then we introduce improvements that are incorporated in CatBoost and LightGBM that have origins in XGboost.

4.1.1 Decision tree

A decision tree is a structure in which each internal node represents a test on an attribute, each branch represents the outcome of the test, and each

leaf node represents a class label. The paths from root to leaf represent classification rules. Figure 4.1 demonstrates the structure.

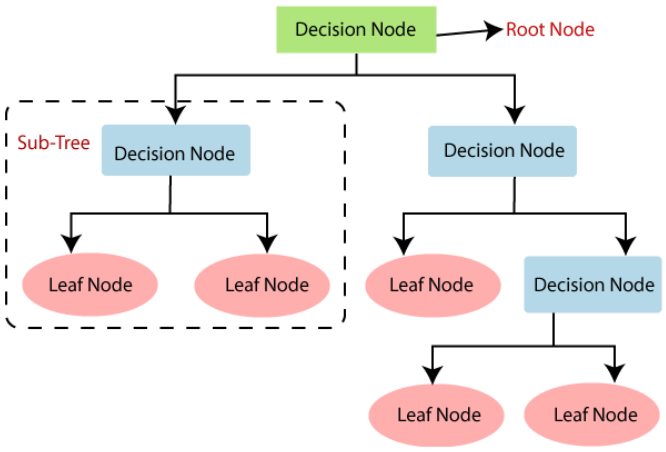


Figure 4.1: Green node corresponds to root, blue outcome of the test and red to leaves [14].

4.1.2 Ensemble learning

Ensemble learning is a machine learning paradigm where multiple models are trained to solve the same problem and combined to get better results. The main hypothesis is that when weak models are correctly combined, we can obtain more accurate models. The tree main classes of ensemble learning machines are bagging, stacking and boosting [15]. In our case, we will focus on boosting method.

4.1.3 Boosting

Boosting is an ensemble learning method where we sequentially fit multiple weak learners. Each model in the sequence is fitted, giving more importance to observations in the dataset that were badly handled by the previous models in the sequence. Each new model therefore focus its efforts on the most difficult observations to fit up to now, so that we obtain a better learner with lower bias.
The models are fit and added to the ensemble sequentially such that the

second model attempts to correct the predictions of the first model, the third corrects the second model, and so on 4.2.

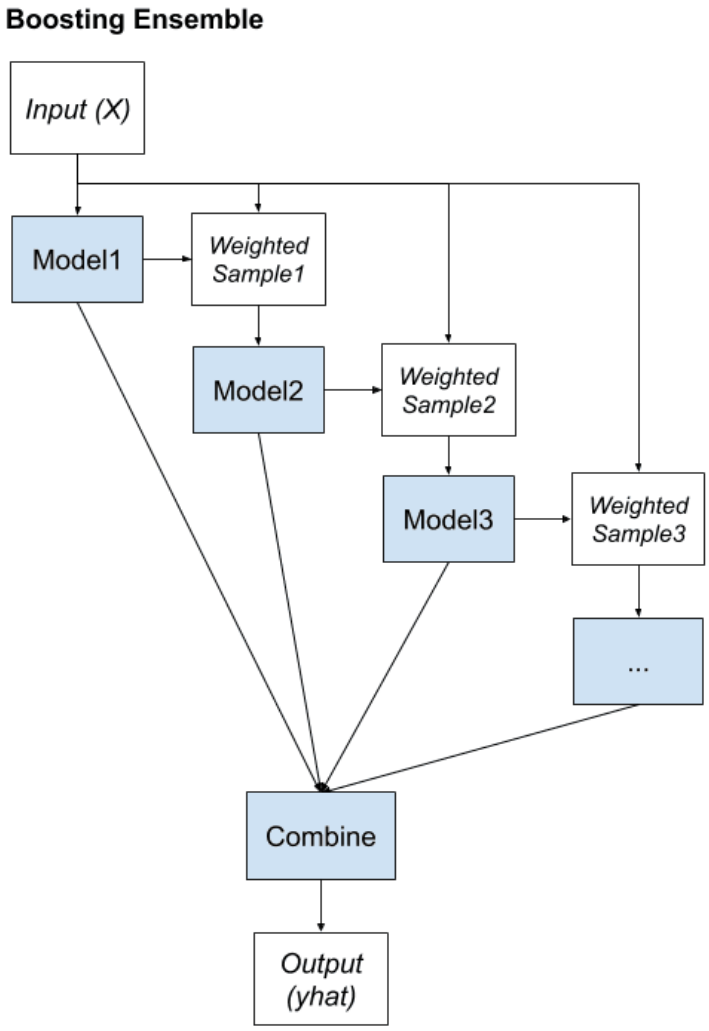


Figure 4.2: The original model is corrected by newly created successors, then are all learners combined [15].

4.1.4 Gradient boosting

Models are fit using any arbitrary differentiable loss function and gradient descent optimization algorithm. This gives the technique its name, as the loss gradient is minimized as the model is fit, much like a neural network. Let's take a look at how gradient boosting works in a more detail.

We are given some data $(\boldsymbol{x}_i, y_i)_{i=1}^n$ where $\boldsymbol{x}$ is vector of input variables and y is output variable. Then we need some differentiable loss function. Types of loss functions differ according to task we want to perform. For explanatory purpose, let us use the general term

$$\mathcal{L}(y, F(x)), \tag{4.1}$$

where y is observed and $F(x)$ is a function equivalent to predicted value. Then we want to minimize it as

$$F_0(x) = \arg\min_p \sum_{i=1}^{n} \mathcal{L}(y, p), \tag{4.2}$$

where $F_0(x)$ is initial model value and p corresponds to predicted value. To minimize the loos function, we compute

$$r_{im} = -\left[\frac{\partial \mathcal{L}(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \text{for } i = 1, ..., n, \tag{4.3}$$

where r_{im} is the pseudo-residual after derivative for sample i and tree m that we are building. This step corresponds approximately to subtracting of predicted value from observed value. We can also point out that this step is the gradient that Gradient Boost is named after.
Now we have to fit CART (Classification and Regression Trees) to pseudo-residuals and label terminal regions, R_{jm} where j is a number of regions that are leaves of the tree. To determine the output values, we compute,

$$p_{jm} = \arg\min_p \sum_{x_i \in R_{ij}} \mathcal{L}(y_i, F_{m-1}(x_i) + p), \tag{4.4}$$

which is a similar equation to 4.2. The difference is that previous prediction is now considered. To compute the output value for each leaf, we pick only y values that correspond to this region. Then we are looking for p that minimizes the equation.
The final step is to make a new prediction, as

$$F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} p_{jm} I(x \in R_{jm}), \tag{4.5}$$

where ν is a learning rate. This equation means that we take the original prediction and add values of corresponding leaves from our newly build tree multiplied by learning rate. Then, we continue with the next iteration. It is desirable to mention that this step represents boosting method described in 4.1.3. More information can be found in [16].

4.1.5 XGBoost

XGboost is a scalable end-to-end tree boosting system which allows to achieve state-of-the-art results on many machine learning task. Novel technics used are sparse-aware algorithm for sparse data and weighted quantile sketch for approximate tree learning. More insight on cache access patterns is achieved to build a scalable tree boosting system[17].

Let us briefly explain how XGBoost builds its trees and what are new ides in comparison to gradient boosting algorithm. We start with objective function

$$\begin{aligned} \mathcal{L}(\phi) &= \sum_i l(y_i, p_i) + \sum_k \Omega(f_t), \\ \text{where } \Omega(f) &= \gamma T + \frac{1}{2}\lambda ||w||^2. \end{aligned} \tag{4.6}$$

Here l is differentiable loss function, T is a number of leaves and Ω is regularization term that helps to prevent overfitting. The formula 4.6 can not be optimized by standard optimization methods, therefore we have to calculate its second-order Taylor approximation. When we remove constant values, we obtain

$$\mathcal{L}^{(t)} \approx \sum_1^n [g_i f_t(\boldsymbol{x}_i) + \frac{1}{2} h_i f_t^2(\boldsymbol{x}_i)] + \Omega(f_t), \tag{4.7}$$

where g is gradient ant h corresponds to hessian.
Define $I_j = \{i|q(\boldsymbol{x_i}) = j\}$, then we can write

$$\tilde{\mathcal{L}} = \sum_{j=1}^{T} [(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2] + \gamma T. \tag{4.8}$$

Here outer sum just iterates over all leaves. The first inner sum is an expansion of sum from 4.7, where we sum all the gradients belonging to one leaf and multiply them by the corresponding leaf output value w. The second sum is the same, except for λ the term that came from the expansion of Ω. When we minimize

$$(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2 \tag{4.9}$$

by standard method, we obtain

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}. \tag{4.10}$$

Now we just plug w_j^* in equation 4.8 and obtain

$$\tilde{\mathcal{L}}(q) = -\frac{1}{2} \sum_{j=1}^{T} \frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda} + \gamma T. \tag{4.11}$$

Let us just remind that g_i and h_i are gradients and hessians corresponding to chosen loss function l that we could substitute for some particular example as negative log-likelihood. Equation 4.11 is used to measure quality of a tree structure q. The greedy algorithm is then used to iteratively add nodes, aiming to reduce the loss. The algorithm is called greedy because when it is building trees, it does not take overall structure into consideration. The tree is therefore built in such a way that the condition is chosen according to the division that is made just by this condition, even though there could be better one if combined with another consequential condition.

In fact, XGBoost introduces an improved version of this paradigm. Even though the greedy approach increases speed, it is still slow if we had to choose a new threshold for every observation. Because of this difficulty, an improved version called Approximate Greedy Algorithm is used. It samples data into quantiles and then instead of trying to separate every observation, it builds conditions according to quantiles. The more quantiles the algorithm has, the more accurate it is.
To reach even better performance, XGBoost uses parallel learning which means that dataset is split into smaller ones, that can be processed by more computers at the same time. After that, the Weighted Quantile Sketch algorithm is used to merge data into an approximate histogram. Histogram is then divided into weighted quantiles, which means that confidence of predictions is combined to be more-less the same.

Another important improvement is used when some values in the dataset are missing. The algorithm is called Sparsity-Aware Split Finding. At the beginning, it creates a separate table for observations that contain missing values. When it is building a tree, it tries to add pseudo-residuals of these separated observations to both sides. A case with the better result is chosen. When we later on encounter another observation with more values missing but already processed is among them, it's pseudo-residual is placed to the same side.

XGBoost is also aware of hardware. Derivatives are very computationally demanding, therefore it is using a technique called Cache-Aware Access to store all gradients g and hessians h in the cache memory of the CPU, so they are computed quickly. Blocks for Out-of-Core Computation is used to compress in such a way that it is trying to minimize amount of data store on Hard Drive. Decompressing the data with CPU is still faster than reading them from Hard Drive. Sharding is used to read separate subsets of data at the same time from separate Hard Drives.

XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements Machine Learning

algorithms under the Gradient Boosting framework. It provides a parallel tree boosting to solve many data science problems in a fast and accurate way.

More information about XGBoost features is given in [16].

4.1.6 LightGBM

Similar to XGBoost, LightGBM is a distributed high-performance framework that uses decision trees for classification [18].
Now we take a look at some fundamental difference related to leaf growth, categorical feature handling, missing values handling and missing values handling.

Gradient-Based One-Side Sampling (GOSS) GOSS is a novel approach which down samples instances on base of training error. It means that gradient of every instance is observed. If the gradient is low, the instance is well-trained, if it is high, it is not trained well. LightGBM does not discard instances with small gradient to focus on instances with large gradient, it uses following steps:

- Sorts the instances according to absolute gradients in descending order.
- Selects some fraction $a/100$ of instances, where $a < 100$. These instances correspond to untrained data.
- Randomly samples $b/100$ instances from the rest of the data to reduce the contribution of well-trained instances.
- To maintain the original distribution, the algorithm increases the contribution of samples with small gradients by $(1 - a)/b$ to focus more on under trained instances. It puts more focus on under trained instances, but maintains distribution similar.

Exclusive Feature Bundling (EFB) Highly dimensional data very often contain features which are mutually exclusive, which means they never take zero values at the same time. Such features are identified and bundled into a single feature to reduce the complexity, it uses following steps.

- Constructs a graph with weighted edges which measures an overlapping between features.

- Sort the features in descending order by number of conflicts.
- Loop over the feature list and if a feature has threshold > conflict, add it to the bundle. Otherwise, create a new bundle.

To merge features, algorithm proceeds in the following way,

- Calculate the offset to be added to every feature in feature bundle.
- Iterate over every feature in every instance.
- Initialize the new bucket as zero for instances where all features are zero.
- Calculate the new bucket for every non-zero instance of a feature by adding the respective offset to the original bucket of that feature.

These are the most important improvements proposed by LightGBM, more details are given in [19] and [20].

4.1.7 CatBoost

CatBoost is the last GBDT algorithm that we mention in this thesis. According to [21] every implementation of gradient boosting face shift of the distribution which leads to prediction shift of the learned model because of boosting dependency on targets of all training examples. The same problem affect standard algorithms of preprocessing of categorical features. Mentioned issues are related to target leakage, which is the problem that should be diminished by usage of novel principles called Ordered boosting and Target-Based with prior.

Target-Based with prior (TBS) is a method inspired by online learning algorithms which get training examples sequentially in time. There is no time in processing tabular data, therefore "artifical time" has to be introduced, which is some permutation σ of the training examples. New feature values are computed in the following way

$$\hat{x}_k^i = \frac{\sum_{\boldsymbol{x}_j \in \mathcal{D}} \mathbb{1}_{\{x_j^i = x_j^i\}} \cdot y_j + ap}{\sum_{\boldsymbol{x}_j \in \mathcal{D}} \mathbb{1}_{\{x_j^i = x_j^i\}} \cdot y_j + a}, \tag{4.12}$$

where $\mathcal{D}_k = \{\boldsymbol{x}_j : \sigma(j) < \sigma(k)\}$ is all the available history, $\hat{x}_k^i$ is a new feature coding, y is target, p is prior and $a > 0$ is constant.

Ordered boosting Classic boosting algorithms are prone to overfitting on small or noisy datasets due to a problem known as prediction shift. In ordered boosting, random permutation of training examples is performed and n different supporting models is maintained. The model M_i is then trained using only the first i samples in the permutation. At each step, in order to obtain the residual for the step j, the model M_{j-1} is used. Model training and residual computation are therefore performed on different subsets, which prevents target leakage. More information is given in [22] and [23].

4.2 Neural networks

Neural networks are deep learning algorithms inspired by functionality of Biological neurons. Building blocks of network are node layers, where the first one is called input layer, then there is an arbitrary number of hidden layers and at the end there is output layer. Every neuron of a layer has its weight and threshold [24].

For purposes of our analysis, we will introduce and describe two neural networks algorithms. The first one is a multilayer perceptron, that was designed in PyTorch. The second one is a state-of-the-art deep learning model based on transformers called TabNet. We will follow the same path as in the description of gradient boosted decision trees, at first we explain the theory and then the implementation. In the following, a few basic terms are defined.

4.2.1 Perceptron

A perceptron is an algorithm for supervised learning of binary classifiers. This algorithm enables neurons to learn and processes elements in the training set one at a time.

Multiple input signals are weighted and summed. If the input signal exceeds a certain threshold, it outputs a signal, otherwise output is suppressed. This corresponds to a class prediction.

We can express perceptron as a function

$$f(x) = \begin{cases} 1 & \text{if } \boldsymbol{w} \cdot \boldsymbol{x} + b > 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.13}$$

where $\boldsymbol{w} \in \mathbb{R}^n$ are weights b is bias and $\boldsymbol{x}$ is input vector. Figure 4.3 shows the structure of a perceptron. More information can be found in reference [25].

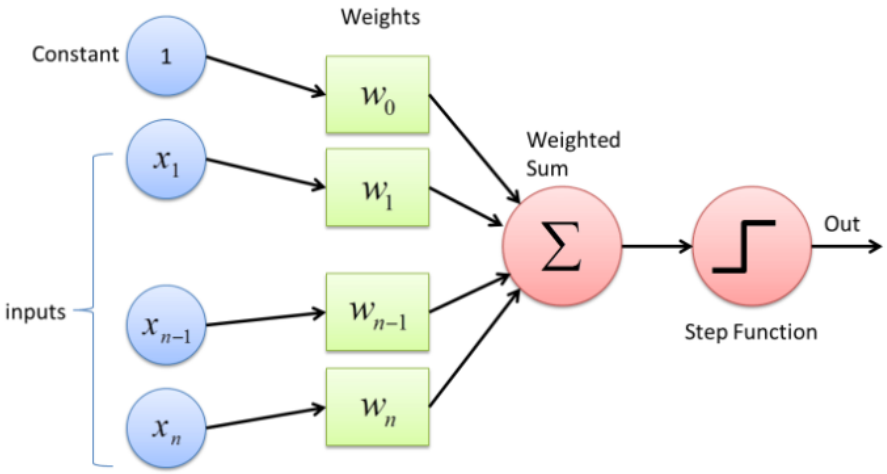


Figure 4.3: Model of perceptron where blue circles are inputs, green rectangles are weights, Σ is sum of weights and last block is activation function [26].

Perceptron uses gradient descent for learning of its weights,

$$\boldsymbol{w}^* = \boldsymbol{w} - \alpha \frac{\partial \mathcal{L}(\boldsymbol{x}, \boldsymbol{w})}{\partial \boldsymbol{w}}, \tag{4.14}$$

where $\boldsymbol{x}$ is input vector, $\boldsymbol{w}$ are weights and $\boldsymbol{w}^*$ are updated weights. Details are given in [27].

4.2.2 Multilayer perceptron (MLP)

A multilayer perceptron has input layer, output layer and arbitrary number of hidden layers in between. If a network contains more than one hidden layer, it is called a deep neural network. It differs from perceptron in a way that a linear combination of weights is propagated to the next layer. Algorithm of propagation of weights is called feed forward. When propagation

reaches the output layer, it has to propagate acquired information back to original weights. To achieve this, back propagation is used in such a way that backward gradients are computed through the network using chain rule. This calculation corresponds to 4.14. We also have to make sure that the activation function is differentiable. More information is given in [28]. A diagram of multilayer perceptron is shown in Figure 4.4.

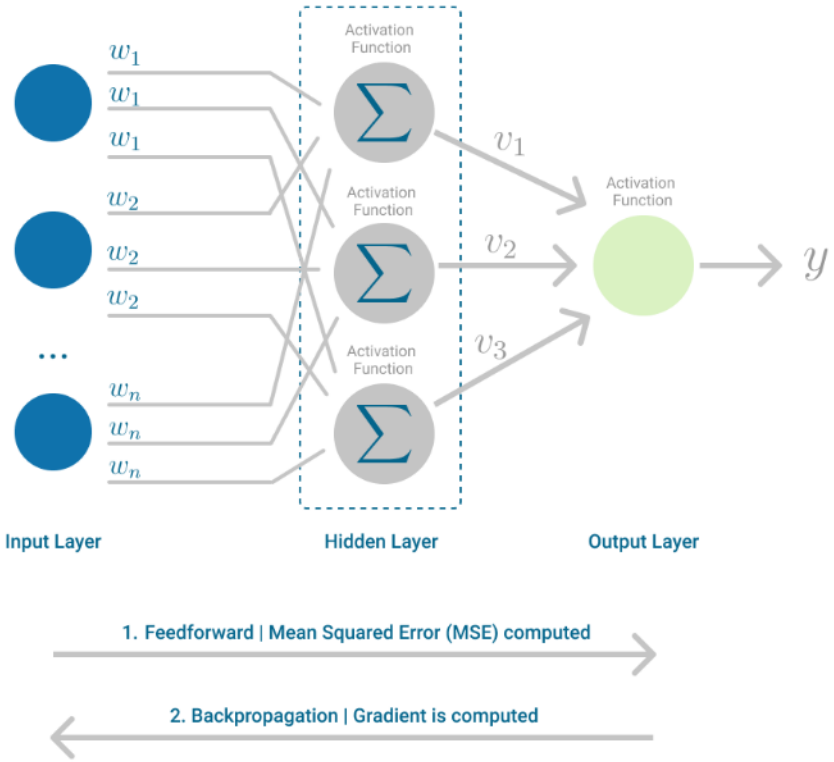


Figure 4.4: Depiction of multilayer perceptron [28].

4.2.3 TabNet

For the majority of data analysis task where tabular data are used, Gradient boosted decision trees algorithms are the best option, especially solution that is explained in the previous sections. On the other hand, deep neural networks are considered to be the best solution for variety of recognition tasks. TabNet neural network architecture stands somewhere in between and even though being a neural network it mimics certain aspects of GBDT [29].

Architecture

TabNet works with sequence of estimators similarly as boosting ensemble learning method uses a method called sequential attention. For single step, the following processes are present:

- Feature transformer: Four consecutive GLU decision blocks.
- Attentive transformer: uses sparse-matrix to give Sparse feature selection which enables interpretability and better learning as the capacity is used for the most important features.
- Mask: Used with the transformer to give out the decision parameters n(d) and n(a).

At the beginning before any feature transformation Batch normalization is performed. Then data goes to feature transformer, it passes through four activation functions called GLU and n(d) and n(a) are obtained. The n(d) outputs a decision from a particular step giving its prediction of continuous classes. The n(a) goes is an input for next attentive transformer.
After the attentive transformer, importance for every feature is acquired. Individual importances are then multiplied by the importance of the overall step and added with other steps to give feature importance for the overall model. A depiction of architecture is shown in figure 4.5

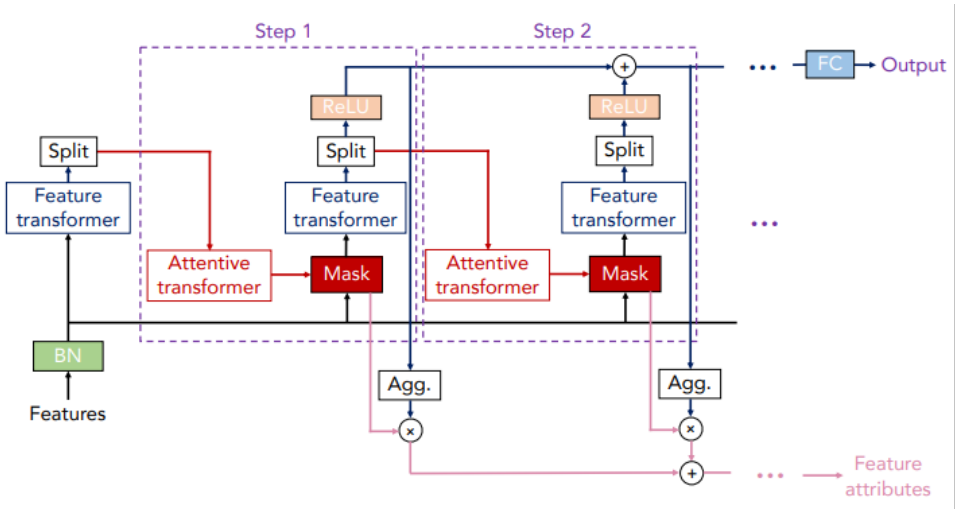


Figure 4.5: Schematic depiction of TabNet architecture [30].

Feature transformer

The feature transformer consists of four blocks. Fully connected layer, Batch Normalization and GLU, where GLU stands for Gated linear unit and is computed as sigmoid multiplied by x can be written as

$$GLU = S(x) \cdot x. \tag{4.15}$$

For robust learning, the layers are shared across two steps. Normalization $\sqrt{0.5}$ is present to stabilize learning. Outputs are n(d) and n(a). The scheme is visible in Figure 4.6.

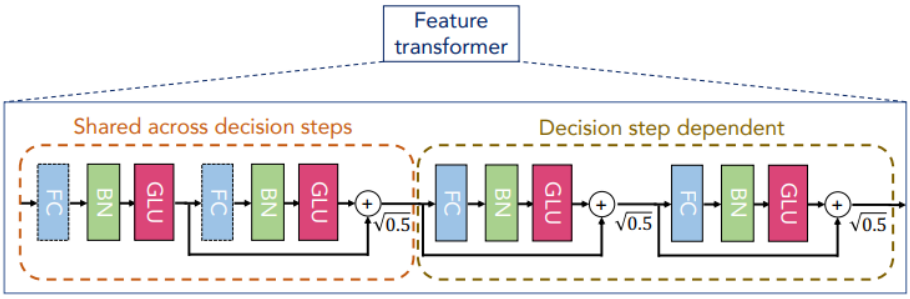


Figure 4.6: Schematic depiction of feature transformer [30].

Attentive transformer

Attentive transformer consists of FC layer, BN layer, Prior scales layer, and Sparsemax layer. At the beginning, n(a) goes through fully connected layer and Batch normalization [31]. Then it is multiplied by a Prior scale, which tells us how much we already know about the feature. The Sparse matrix is used which is like softmax function but instead of all features adding to 1, we have some features equal to 0 but the rest adds up to 1. This helps to make instance-wise feature selection when we can choose different features for every step, then output is fed into a mask layer. A depiction of attentive transformer can be seen in figure 4.7.

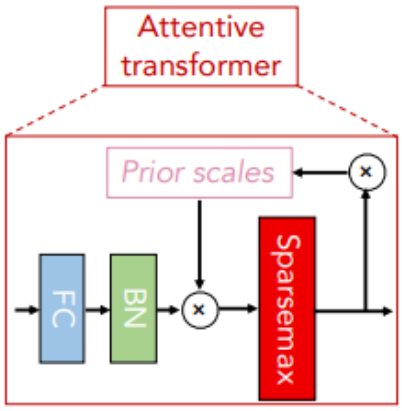


Figure 4.7: Schematic depiction of attentive transformer [30].

More comprehensive explanation and corresponding formulas are given in [29] and [30].

Chapter 5

Approach to analysis

5.1 Important concepts

In order to better outline how we will approach the problem, it is necessary to state the relevant tools and concepts. At first, we will describe in detail what will be needed for the analysis explanation and at the end of the chapter we will reveal the whole picture.

5.1.1 Confidence level

Confidence Level is a statistical measure of the percentage of test results that can be expected within a specific range. Confidence level 95% means that the result of the action will probably meet expectations 95% of the time [32].

5.1.2 Weights

For every single event, we can calculate its weight. It tells us the relative frequency of events, therefore we can make an appropriate prediction how many events of certain class to expect actually appear in the ATLAS detector.

For the weight calculation, we use the following formula.

$$w = \frac{Y(e_{f_0}) \cdot e_{f_1} \cdot e_{f_2} \cdot e_{f_3} \cdot e_{f_4} \cdot e_{f_5} \cdot e_{f_6} \cdot e_{f_7} \cdot e_{f_8} \cdot e_{f_9}}{e_{f_7}}, \tag{5.1}$$

where w are computed weights and Y is luminosity corresponding to the year of production defined as

$$Y(e_{f_0}) = \begin{cases} 36205.66, & \text{if } e_{f_0} = 2015 \ \vee e_{f_0} = 2016 \\ 44307, & \text{if } e_{f_0} = 2017 \\ 58450, & \text{if } e_{f_0} = 2018. \end{cases} \tag{5.2}$$

Particular factors e_{f_i} from equation 5.1 are described in Table 5.1. The purpose of mentioning this equation should be understood in such a way that certain properties from root n-tuples are used to calculate weights. These weights improve the agreement between simulated and recorded data.

Index	Feature name
f_0	RunYear
f_1	custTrigSF_LooseID_FCLooseIso_DLT
f_2	weight_pileup
f_3	jvtSF_customOR
f_4	bTagSF_weight_DL1r_70
f_5	weight_mc
f_6	xs
f_7	totalEventsWeighted
f_8	lep_SF_CombinedTight_0
f_9	lep_SF_CombinedTight_1

Table 5.1: Feature names corresponding to the IDs from the formula 5.1.

5.1.3 Significance

Especially in particle physics, a specific metric called significance is used to measure the outcome. For the approach called general approximation formula we have

$$\eta = \frac{S}{\sqrt{S+B}} \tag{5.3}$$

and for simplified version

$$\eta = \frac{S}{\sqrt{B}}, \tag{5.4}$$

where η is the symbol for significance. S is the number of signal events which means all the true positives and B is the number of background which means all the false positives [33]. True positive means predicted as signal and being signal, and true negative means predicted as signal but being background. Let us assume that we split data into two parts, expressible as x for testing and $1 - x$ for training. At this point we work with only x for significance calculation, therefore we have to multiply S and B by a factor $1/(1 - x)$ to obtain a result for total data, thus the

$$\eta = \frac{1}{1-x} \cdot \frac{S}{\sqrt{\frac{1}{1-x} \cdot B}}. \tag{5.5}$$

5.1.4 Leptoquark scaling factor

Let us assume that data related to Leptoquarks have calculated weights. After classification, we obtain the significance as a result, but we would like to adjust it to a certain value to set upper limit for cross-section with required confidence level. To fulfil such goal, we need to multiply data weights by a factor. Let us denote this factor as ξ. We can rewrite equation 3.4 to

$$R = w \cdot \xi, \tag{5.6}$$

where w is given in 5.1 and ξ is the Leptoquark scaling factor. This scaling factor is important if we want to calculate a limit for expected cross-section. In the last chapter, we will introduce software called TRExFitter that calculates the expected cross-section without explicit scaling factor tuning. It is important to state that the scaling factor is applied before training of the classifier to have possibility to observe the effect of changing weights on the loss function of the classifier, otherwise it would not be required to retrain the classifier using a feedback loop.

5.2 Analysis diagram

To better understand the overall process, let us introduce the diagram show in Figure 5.1 that encapsulates the major logically distinctive parts of code to separate blocks. At this point, our ambition is not to comprehensively describe every block, just to gently indicate the meaning of regions with the same color, which indicates similar functionality:

- Blocks painted blue correspond to code that is responsible for data pre-processing.
- Green blocks are related to data classification.
- Pink blocks indicate some kind of decision-making or recalculation.
- For the end and beginning cyan color was chosen.

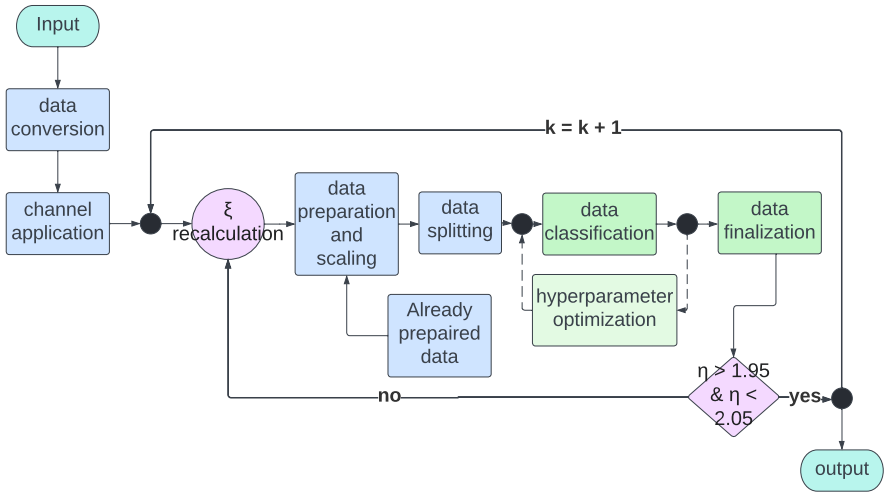


Figure 5.1: Diagram describing the whole process of analysis. Blue blocks are related to pre-processing, green are related to classification and pink ones indicate some recalculation.

All the blocks and especially blue and green ones will be described in detail in the following chapters, for now let us just introduce the meaning of pink blocks and feedback loops
When the finalization block is complete, we obtain the results. One of the results is significance, denoted as η. At this point, its value is completely arbitrary, but would like to set its value as close as possible to 2, which is our required significance level of 95%. Therefore, we propagate the value of significance back to the pre-processing section, where the value is used to recalculate the Leptoquark scaling factor ξ to find optimal weights. This process is repeated until the correct ξ is found. This calculation can be avoided by use of TRExFitter program, related results will be shown in the last chapter. Diagram 5.2 illustrates how we can substitute the mentioned process by specialized software.

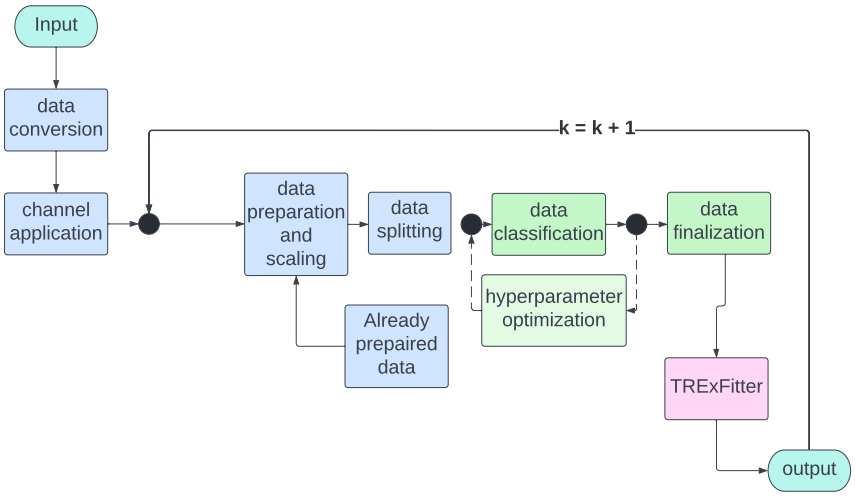


Figure 5.2: Substitution by specialized software. The feed-back loop for scaling factor ξ is replaced by TRExFitter program, therefore manual recalculation is not needed anymore.

5.3 The aim of the analysis

Leptoquarks are particles not belonging to the Standard Model, therefore we do not know their mass. Thus, multiple distinguished masses are used to analyze the entire mass region. For these masses, we are given a theoretical cross-section and want to calculate the expected one as given in 3.5. We aim to move the expected cross-section as low as possible. It means that when we look at the intersection of the mentioned cross-sections, we would observe larger segment of expected cross-section curve being under the curve of the theoretical one. This intersection is called the exclusion limit, and the mentioned interval is marked as excluded. It means that Leptoquarks within this mass region would be detected with 95% CL or excluded with 95% CL.

Part II

Implementation

Chapter 6

Data and pre-processing

6.1 Code

Scripts written for the overall analysis can be found on the web address https://gitlab.fel.cvut.cz/vicenluk/leptoquark.

6.2 Introduction

In an early stages of the analysis, real data is not used. To acquire data for training, it is needed to run Monte-Carlo simulations that generates data according to the known theory. Data that are produced in Monte Carlo simulation are latter stored in specialized files serving exactly for this purpose. Files are then converted by us to csv file format for better access to particular events. After conversion, data are separated to folders according to their class ($LQ, t\bar{t}H, t\bar{t}W, t\bar{t}Z, t\bar{t}, VV, Other$). In this stage, we pre-process the data and prepare them for classification. This part of the work was inspired by the master's thesis of Jakub Malý [34].

6.3 Important Tools

This chapter describes how data are pre-processed before they are used as input of a classifier. Let us at first introduce crucial tools that we will reference in further explanations.

6.3.1 ROOT

ROOT is an open-source framework built for analyzing high energy physics data in CERN [35]. Data from Monte-Carlo simulations are stored in files constructed by this framework. Thanks to ROOT, it is very straightforward to access particular features and plot corresponding histograms. For faster data manipulation and analysis, a python interface called PyROOT was introduced [36]. In this work, described tool is mainly used for manipulation with root files. Our data are stored in root n-tuples. N-tuples are root files structured similarly to algorithmic structures called trees.

6.3.2 NumPy

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more [37].

6.3.3 Pandas

Pandas is a Python open sources library mediating flexible and easy data manipulation and analysis. Pandas is built on top of NumPy, which provides support for multidimensional arrays. The most used data structure used in this thesis is Pandas DataFrame, which suits as very convenient representation for tabular data [38].

6.4 Data structure

Data from Monte-Carlo simulation are stored in specific files with tree-like structure called ROOT n-tuples. N-tuples are created with the help of ROOT framework that extends classical C++ programming language.
If we extend our idea about tree-like structure of an n-tuple, branches correspond to properties called variable or features for classification purposes. Every feature is represented by a histogram that contains values for every event with respect to the feature we are about to examine.

Every file carries its ID, which is a five-digit number identifying the particular file. Every single file includes only events corresponding to one class. Classes are the following

- LQ: Leptoquark events.
- $t\bar{t}H$: top quark, anti-top quark, Higgs boson.
- $t\bar{t}Z$: top quark, anti-top quark, Z boson.
- $t\bar{t}W$: top quark, anti-top quark, W boson.
- $t\bar{t}$: top quark, anti-top quark.
- VV: W or Z boson, W or Z boson.
- $Other$: Aditional reactions that could occur.

For our purposes we will use LQ as the signal and the rest as the background. This means that the events belonging to LQ are the ones we are looking for. In this work, we will use two LQ datasets, which differ by LQ model factor β corresponding to branching ratio. Table 6.1 and 6.2 show the number of events passing several selections. If we focus our attention on the Table 6.2, we realize that the number of events that remain is too small for the classification process, and therefore we will use only the first dataset for this analysis.

6.5 Data conversion

Now that we know how the data is distributed among the files, we can proceed to its conversion to more suitable format. For this purpose, a

LQ mass	Σ bef. select.	Σ aft. select.	Σ aft. pre-select.	%
500	206504	78806	2889	3.666
600	279192	64795	2562	3.954
700	176098	30914	1236	3.998
800	212365	83293	3376	4.053
900	213017	83232	3304	3.970
1000	160280	27301	1074	3.934
1100	213738	80948	3071	3.794
1200	214342	47055	1742	3.702
1300	107047	39532	1423	3.600
1400	106954	39134	1400	3.577
1500	106732	38210	1296	3.392
1600	115607	31073	1076	3.463
sum	2111876	644293	24449	3.795

Table 6.1: First dataset: Number of events before n-tuple selection, after n-tuple selection and number of events that remain after pre-selecting restrictions are applied. The last column is the percentage of remaining events after pre-selection.

script *data_convert_new.py* was written. It is necessary, at first, to place ROOT files in folders with appropriate names so that the name contains the corresponding class, then the content of the folder is loaded, and every file included is converted into the array. At this point, data are transformed into a table where columns correspond to features and rows correspond to events. For this purpose, Pandas DataFrame structure is used [39]. The final step of the conversion is to save the DataFrame as a csv file with an appropriate name containing the currently processed class.

6.6 Weights and pre-selection

For the purpose of the weights calculation and the pre-selection application script *data_weights_new.py* is used. The following description outlines its functionality.

At this point, we can easily manipulate created csv files. For the purpose of classification, it is needed to calculate the so-called weights. Weights are obtained for every single event in such a way that we calculate a specific linear combination of the features and additional temporary variables as year of detector run.

They are then used in the loss function of the classifier. For this purpose, significance on the output is always propagated back to have updated scaling at our disposal. We also use them to scale a number of simulated events to

LQ mass	Σ bef. select.	Σ aft. select.	Σ aft. pre-select.	%
300	92422	8143	122	1.498
500	341131	45238	1229	2.717
600	121854	16810	493	2.933
700	126643	17995	501	2.784
800	131077	18531	495	2.671
850	92953	13531	354	2.702
900	320584	45113	1260	2.793
950	94833	13241	339	2.560
1000	94930	13295	356	2.678
1050	95434	13474	359	2.664
1100	96291	13383	344	2.570
1150	96577	13335	365	2.737
1250	46595	6269	171	2.728
1300	260247	34897	904	2.590
1350	23463	3196	84	2.628
1400	23269	3099	72	2.323
1450	23652	3144	89	2.831
1500	23450	3037	77	2.535
1550	14228	1857	55	2.962
1600	14119	1755	34	1.937
1700	66286	12770	258	2.020
1800	14308	1689	36	2.131
1900	14117	1723	30	1.741
2000	14227	1677	40	2.385
sum	2242690	306774	8067	2.630

Table 6.2: Second dataset: Number of events before n-tuple selection, after n-tuple selection and number of events that remains after pre-selecting restrictions are applied. The last column is percentage of remaining events.

the number of events that are actually detected.

In order to distinguish the events that belong only to the selected channel, we apply the so-called pre-selection. By pre-selection, we mean the application of a filter that passes only events belonging to the chosen channel. As mentioned in previous text, our channel is $2lSS + 1\tau$. Details of the channel definition are given in the Appendix A1.

6.7 Data preparation

Before the classification, it is necessary to convert the data into a suitable format for this task. The script *data_prepare_new.py* serves this purpose. At this point, we need to remove the string names and replace them for numerical variables more suitable for classification. Every event was so far identified by its class name. We use script *constants.py* containing required dictionaries to substitute strings with corresponding integer values. Then we can produce the input matrix $\mathbf{X}$ and the truth label vector $\mathbf{y}$. The weights are multiplied by a scaling factor.

6.7.1 LQ scaling factor

Each event has its own weight, where the theoretical cross-section from n-tuples is used, because the scale of weights for LQ signal is unknown, it is necessary to scale the given cross-section to a new value called expected cross-section. Scaling consequently affects the weights. At this point, we introduce the iterative process of how final scaling for cross-section is obtained. It is important to mention that iteration is needed only if we want to update weights that are used in the loss function of classifier, otherwise we would just rescale the significance obtained after first iteration and used this scaling for the cross-section limit determination.
We would like to recalculate the scaling factor in such a way that it corresponds to 2σ significance in every iteration. Let us assume that we are in $n-th$ iteration. At this point we know the value of the scaling factor and the significance from the previous iteration. Let us denote the scaling factor as ξ_{n-1} and the significance as η_{n-1}. Now we calculate the current value of the scaling factor as

$$\xi_n = 2 \cdot \frac{\xi_{n-1}}{\eta_{n-1}}. \tag{6.1}$$

Now if

$$\eta_n \in [1.95, 2.05], \tag{6.2}$$

we declare the given scaling factor final and the iteration process is terminated, otherwise we continue with the next iteration. It should be mentioned that the only reason for the iterations is the change of significance on the output of the classification process as consequence of rescaled weights used in the loss function of the classifier. Otherwise, formula 6.1 would be used just once.

6.8 Data split

In the thesis, a method called train-test split is used. It means that we split the dataset into training and test sub-datasets, and each of them is suitable representations of the problem domain. The main important parameter is the size of datasets, commonly expressed as percentage between 0 and 1. Our chosen ratio was 0.8 to 0.2 in favor of training dataset.

6.8.1 Splitting challenges

For our purposes, we need to be able to organize data in the following ways:

- Events with negative weights have to be removed from the training dataset to prevent classifier confusion.
- Possibility to train on different separated masses.
- Possibility to train on all the masses and test on one particular mass.

The script *data_split_new.py* serves this purpose.

6.8.2 Negative weights

At this point, data are loaded into NumPy data-frames. For the purpose of negative weights removal, we simply use condition $weights > 0$ for feature $weights$.

6.8.3 Cross training-test

To achieve correct results, we have to be very cautious. We want to swap just the LQ signal in the test part of the data, therefore if we had swapped entire test parts of two datasets, we would achieve data leakage because splits for datasets are different and contain different events for train and test datasets.

6.8.4 Mass combined for training

If it is needed to train on all the masses together, we need to have information about masses included in the DataFrame. This artificial feature is labeled as y_lq_all and contains mass that particular events belongs to. At this point, we just use a condition $y\_lq\_all == lq\_mass\_test$ which will ensure that only the mass we are testing on is present in the test set, but the training set contains all the masses. When the process is finished, artificial features are removed from the data table.

Chapter 7

Classification

7.1 Introduction

Classification is the most important part of the thesis. At this point we have our data converted, pre-processed and split in such a way that it is suitable for the experiment we are intended to perform.
To achieve better reliability of the results, we used two type of machine learning algorithms, gradient boosted decision trees and neural networks. When the classification is finished, we need to evaluate results. The metric that determines the correctness of the classification is called significance and was theoretically described in section 5.1.3. Now we will focus on the way in which it is calculated.

7.2 Tools and libraries

We will describe tools that were used for implementations of classifiers and their hyperparameter tuning.

7.2.1 Scikit-learn

Scikit-learn is a python language free software machine learning library. Functions from this library are used very frequently is this work. More details are given in [40].

7.2.2 Pytorch

PyTorch is an open source machine learning library based on the Torch library [41]. In this work, it is used for neural network algorithm implementation.

7.2.3 Optuna

Optuna is an automatic hyperparameter tuning software framework, particularly designed for Machine Learning, and one use it with other frameworks like PyTorch or Scikit-learn. Optuna uses something called define-by-run API, which helps the user to write highly modular code and dynamically construct the search spaces for the hyperparameters. Different samplers like grid search, Random, Bayesian and Evolutionary algorithms are used to automatically find the optimal parameter. Algorithms used are

- Grid Search: It searches the predetermined subset of the whole hyperparameter space of the target algorithm.
- Bayesian: This method uses a probability distribution to select a value for each of the hyperparameters.
- Random Search: As the name suggests, randomly samples the search space until the stopping criteria are met.
- Evolutionary Algorithms: The fitness function is used to find the values of the hyperparameters.

Important terminology used in the following text is

- Objective function: Function we want to minimize or maximize

- Study: The whole optimization process is based on an objective function. The study needs a function which it can optimize.
- Trial: A single execution of the optimization function is called a trial. Thus, the study is a collection of trials.

For Hyperparameter optimization, we use the following process:

- At the beginning we use a classifier without any hyperparameters to set certain threshold
- We start hyperparameters tuning by manual optimization. The hyperparameter documentations and machine learning competitions serve this purpose.
- Optuna is deployed to find more optimal solutions.

7.3 Algorithm hyperparameters

In chapter 4 we discussed the theory behind algorithms used in this thesis. Now it is time to use this theory in practice. In this section, we will describe which hyperparameters are important for particular algorithms and the theory behind them.
At the end of the sentence in the parentheses, there is the value of the hyperparameter that was used for the analysis.

7.3.1 XGBoost

For XGBoost we have chosen parameters described in the following text.

- n_estimators : Specifies the number of decision trees to be boosted. (2300)
- max_depth : It limits how deep each tree can grow. (6)
- min_child_weight : Minimum sum of instance weight (hessian) needed in a child. If the tree partition step results in a leaf node with the sum of

instance weight less than min_child_weight, then the building process will give up further partitioning. (0.043)

- reg_alpha : Is the L1 regularization parameter, increasing its value makes the model more conservative. (0.0036)
- gamma : Is the regularization parameter for tree pruning. It specifies the minimum loss reduction required to grow a tree. (0.00025)
- learning_rate : Is a regularization parameter that shrinks feature weights in each boosting step. (0.5)
- colsample_bytree : The algorithm will randomly select a subset of features on each iteration (tree). (1.0)
- subsample : It represents the subsample ratio of the training sample. (0.8)
- scale_pos_weight : This parameter is useful in case we have an imbalanced dataset, particularly in classification problems, where the proportion of one class is a small fraction of total observations. (10)

7.3.2 LightGBM

For LightGBM we have chosen parameters described in the following text. Some hyperparameters are the same as ones used for XGBoost, therefore we only refer to them.

- n_estimators: *. (2500)
- max_depth: *. (3)
- reg_alpha: *. (0.79)
- colsample_bytree: *. (0.5)
- reg_lambda: L2 regularization term on weights. Increasing this value will make model more conservative. (7.4)
- min_child_samples: Minimal number of data in one leaf. Can be used to deal with over-fitting. (42)
- min_data_per_group: Minimal number of data per categorical group. (54)

7.3.3 CatBoost

For CatBoost we have chosen parameters described in the following text. Some hyperparameters are the same as ones used for XGBoost, therefore we only refer to them.

- n_estimators: *. (1550)
- max_depth: *. (3)
- learning_rate: *. (0.5)
- subsample: *. (0.659)
- random_strength: The amount of randomness to use for scoring splits when the tree structure is selected. This parameter is used to avoid overfitting the model. (9.14)
- bagging_temperature: Use the Bayesian bootstrap to assign random weights to objects. (0.8)
- border_count:The value of this parameter significantly impacts the speed of training on GPU. The smaller the value, the faster the training is performed. (80)
- l2_leaf_reg: Coefficient at the L2 regularization term of the cost function. (0.32)

7.3.4 TabNet

For TabNet, features are very different from ones used for GBDT we describe them in following text.

- n_steps: Number of steps in the architecture. (3)
- gamma: This is the coefficient for feature reusage in the masks. A value close to 1 will make mask selection least correlated between layers. (1.3)
- n_shared: Number of shared Gated Linear Units at each step. (1)
- mask_type: Masking function to use for selecting features. (entmax)
- optimizer_fn: Used to choose the optimizer. (Adam)

- lr: A scalar used to determine gradient step in gradient descent training. (0.02)
- weight_decay: Decrease of weight values in time. ($3 \cdot 10^{-6}$)
- momentum: Momentum for batch normalization. (0.7)
- scheduler: Pytorch Scheduler to change learning rates during training. (-)
- patience: Number of consecutive epochs without improvement before performing early stopping. (9)

7.3.5 MLP

Multilayer perceptron was written with help of PyTorch framework. We will explain the important parts of implementation in the following sections.

Train

In this section, training loss is recomputed and back propagation is performed to update weights of the network. This process is repeated for every batch. At first, we load the data and the weights. A gradient is zeroed to obtain correct parameters update when back propagation is performed. When loss is calculated, we multiply it by $w/\sum w$ to help the training. Then, back propagation is performed and optimizer is called. We update overall loss and make prediction.

7.3.6 Validation and test

Validation is applied on a subset of the original data. It should provide an unbiased evaluation of a model fit on the training dataset while tuning hyperparameters. When hyperparameters are tuned, bias is incorporated into the validation set.
Validation is very similar to the train section, it only performs calculations on validation dataset.

For testing, we use the fact that test data are unbiased, and therefore we can evaluate the final model on them.

7.3.7 Hyperparameters and architecture

For neural network we use the learning rate, number of epochs and batch size as hyperparameters, because these hyperparameters were already described in the previous text, we will focus mainly on the neural network architecture. Table 7.1 illustrates which building blocks are used. The linear layer is a simple connection of neurons. Dropout represents the probability that a layer is zeroed during the process, which prevents overfitting. ReLU is the activation function which mean that it makes decision whether neuron should fire or not. It is defined as

$$f(x)_{ReLU} = \begin{cases} 0, & \text{if } x < 0 \\ x, & \text{if } x \geq 0. \end{cases} \tag{7.1}$$

The overall architecture is ended with LofSoftMax layer which is important for classification into multiple classes, which is defined as

$$f(x_i)_{SFM} = \log\left(\frac{\exp(x_i)}{\sum_j \exp x_j}\right). \tag{7.2}$$

Block	Value
Linear layer	7, 20
Dropout	0.2
ReLU	-
Linear layer	20, 50
Dropout	0.2
ReLU	-
Linear layer	50, 10
Dropout	0.2
ReLU	-
Linear layer	10, 7
Dropout	0.2
LogSoftmax	-

Table 7.1: Used neural network architecture.

7.4 Classification

In previous sections, we described algorithms that will be used for the classification process. For this purpose, a script *classify.py* was written. The script is divided into two functions, where the first one is responsible for training and the other for testing. There are separate classes *tabnet_agent.py* and *mlp_agent.py* for deep learning algorithms.

When we pick the classifier that we want to use for consequential analysis, at first data are normalized by $StandardScaler()$ a function that normalizes every column of the input matrix $\mathbf{X}$ so that for mean and standard deviation we have $\mu = 0$ and $\sigma = 1$ [42]. Then we just choose a library function for corresponding classifier and pass tuned hyperparameters. We can also choose to use weights to affect performance of loss function. When the model is trained, we save the information for later use.

For the second part of classification, the model and test input matrix X_t are loaded. At this point, we want to obtain predictions and prediction probabilities. Functions $predict(X_t)$ and $predict\_proba(X_t)$ are used for this purpose [43] and [44]. Then, we can compute the metrics. Implementation of metrics is inspired by [34].

7.4.1 Accuracy (Acc)

Accuracy is the fraction of predictions that our model got right. The formula is

$$Acc = \frac{TP + TN}{TP + TN + FP + FN}, \tag{7.3}$$

where TP stands for True Positive, TN is True Negative, FP is False Positive and FN is False Negative [45].

7.4.2 F1

The F1 score is a certain combination of metrics called Precision and Recall. Let us therefore define these two terms in advance. For Precision, we have

$$Precision = \frac{TP}{TP + FP} \tag{7.4}$$

and recall is calculated as

$$Recall = \frac{TP}{TP + FN}. \tag{7.5}$$

At this point we have everything we need to compute the F1 metric. It corresponds to the formula

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}. \tag{7.6}$$

More details are given in [46].

7.4.3 AUC

AUC stands for Area Under the Roc Curve. It measures the entire two-dimensional area underneath the entire ROC curve. It corresponds to the gray region in Figure 7.1.

7.4.4 ROC curve

An ROC curve stands for receiver operating characteristic curve and represents a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters, True positive on vertical line and False Positive rate on horizontal line, they are defined as

$$TPR = \frac{TP}{TP + FN} \qquad FPR = \frac{FP}{FP + TN}, \tag{7.7}$$

Where TPR is True Positive Rate and FPR is False Positive Rate. Figure 7.1 shows a simple ROC graph where dashed line corresponds to ROC curve [47].

In our analysis, we classified the data into seven classes (LQ, $t\bar{t}H$, $t\bar{t}W$, $t\bar{t}Z$, $t\bar{t}$, VV, $Other$), therefore we obtain seven ROC curves. Figure 7.2 shows the

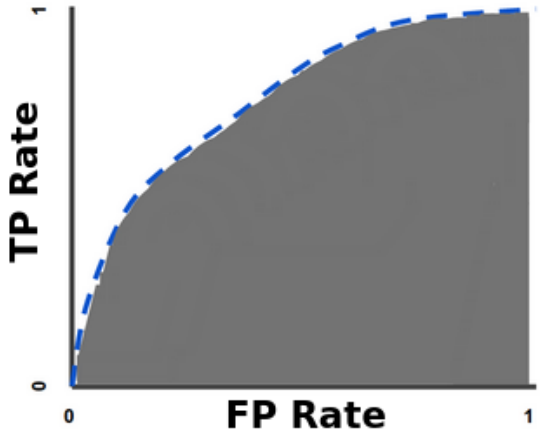


Figure 7.1: ROC curve with highlighted AUC region. On the horizontal axis is the False Positives rate. On the vertical axis is the True Positive rate.

ROC curves for testing on 500 and 1600 GeV masses, respectively. Metrics from previous sections are also computed and visualized in the legend. Figure 7.2 shows better separation for 1600 GeV mass, even though for both masses, the Leptoquark signal is remarkable. This is probably caused by higher the energy of heavier Leptoqurks which results in even larger differences from background.

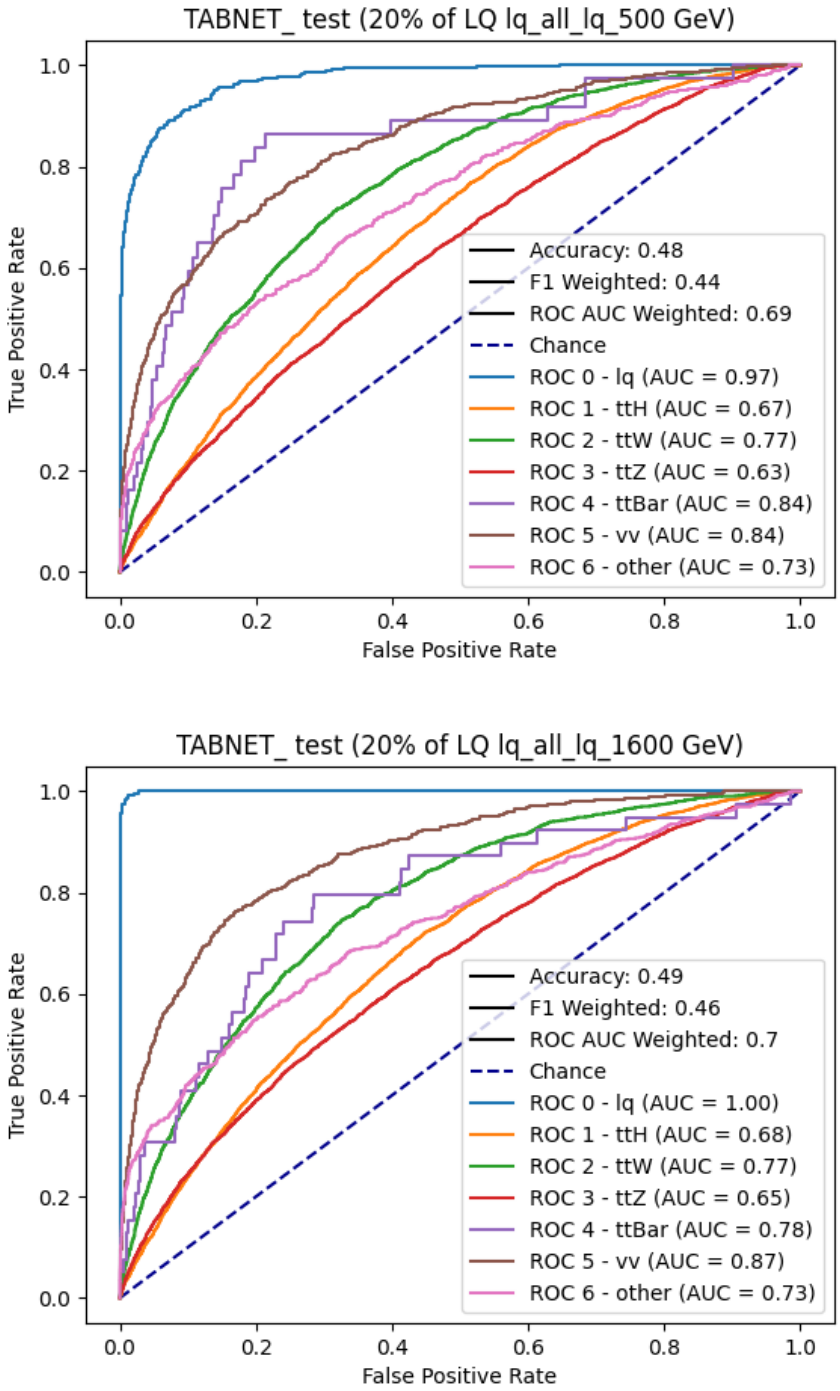


Figure 7.2: Top: ROC curve for TabNet trained on all the masses and tested on 500 GeV mass. Bottom: ROC curve for TabNet trained on all the masses and tested on 1600 GeV mass.

7.5 Finalization

At this point, the classification is completed, and we obtained our results from the classifier. To better understand how good is the separation, we need to find out the ratio between signal and background, compute significance and visualize the confusion matrix for the best threshold.

7.5.1 Signal versus Background

For every threshold, we determine the number of event belonging to signal and background. The threshold with the best ratio is chosen as classifier working point. Figure 7.3 serves as an example.

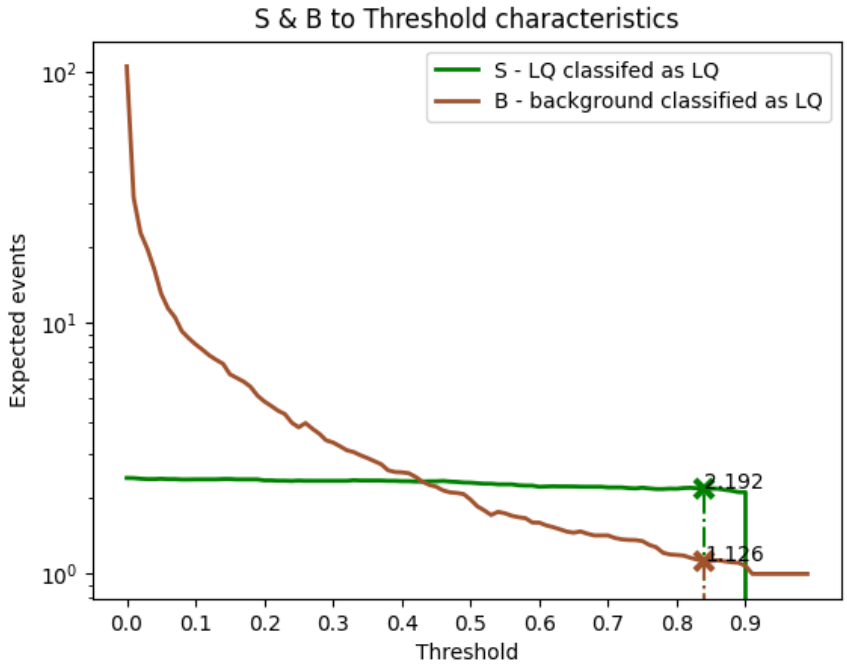


Figure 7.3: Green: Curve depicting signal. Brown: Background curve.

7.5.2 Significance

Description of significance classification is described in chapter 5.1.3. At this point, we just have to transform theory into practice, and use formula 5.4 to calculate the significance for the chosen thresholds, then determine the working point. This calculation is by definition of significance strongly connected to the previous chapter.
An example of plotted significance is shown in Figure 7.4. In this thesis, we use the definition of significance $S/\sqrt{B}$. For this definition, we can encounter problems when B is close to 0 which results in nonphysical results. For this reason, another two definitions are introduced. In the figure, a cut-off is applied. It is applied because no maximum could be found. For this reason, TRExFitter software is used later.

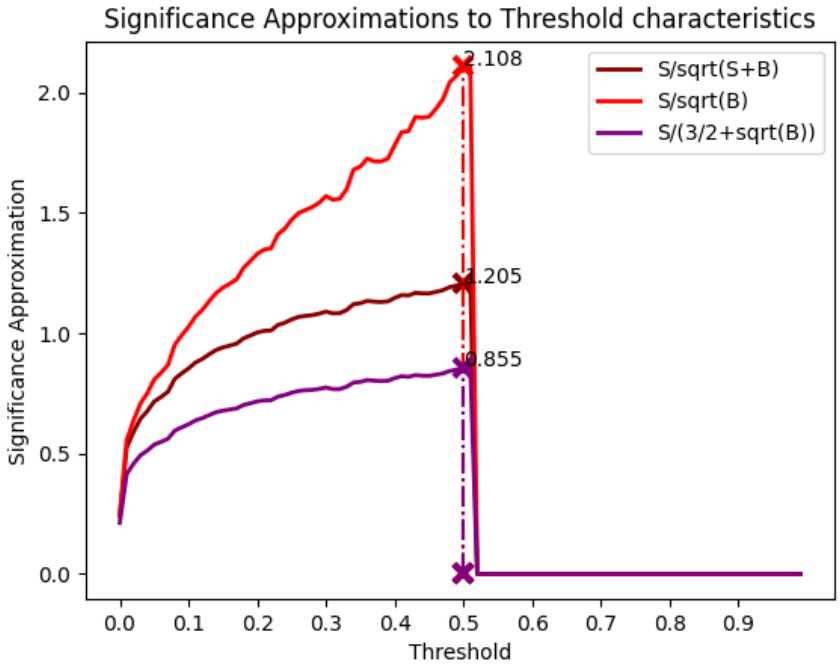


Figure 7.4: Significance plotted for all thresholds and chosen significance definitions.

7.5.3 Confusion matrix

Figure 7.5 illustrates how the confusion matrix is defined. On the x-axis there is the actual value and on the y-axis there is the corresponding prediction. It means, for example, when an observation is classified as true and its target is

also true, it belongs to TP. Then a matrix depicts the number of observations belonging to a certain category or their relative percentage.

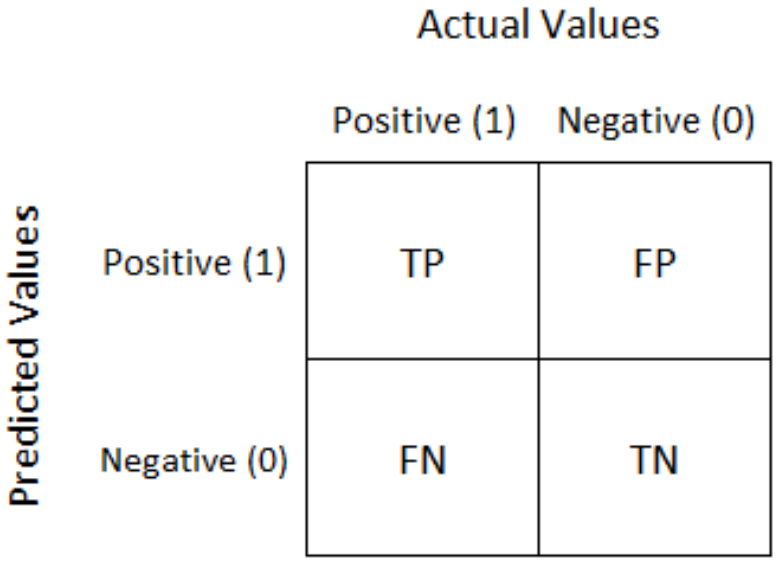


Figure 7.5: Simple schematic version of confusion matrix, taken from [48].

Let us take a look at a little more complicated version that is plotted for our purpose in Figure 7.6. Our matrix is a transposed definition from Figure 7.5. The first column is the most important one. The matrix value with coordinates $[0, 0]$ represents the signal that corresponds to True Positives, therefore the signal is classified correctly. Values on the first column under the first one are also important, they correspond to background. Background belongs to the region of False Positives, it means that events are classified as Leptoquarks even though they belong to some background class. The first column value over the rest of the column square root corresponds to the definition of significance.

Figure 7.6: Confusion matrix for the best threshold, mass 800 GeV.

Part III

Analysis results

Chapter 8

Analysis results

At this point, we would like to show what was achieved. We start with a comparison of classifiers to select two of them that are the most suitable. When classifiers are chosen, we show that by training only one classifier on all masses combined results in a versatile classifier with comparable capabilities to classifiers trained and tested on corresponding masses.
Another task is related to data shrinkage. By reducing the size of the dataset by a certain percentage, we demonstrate that results tend to be more dependent on data split and the average performance is decreased. Then, we reduce the number of features used for training and study if the performance of classifier is reduced.
When auxiliary experiments are explained, we proceed to the final stage of the analysis. For one classifier that performed the best, we compute the expected cross-section limit with the software called TRExFitter and compare results with the ones from a previous publication.
In addition, the precision of the mass determinations is studied. We use separated data from previous step as input for another classifier. This classifier is trained on all masses and is based on this experience determines the mass of the Leptoquark.

It is important to mention that for the purpose of experiments related to relative comparisons we set the theoretical cross-section $\sigma_t = 1$ pb, and the scaling factor $\xi = 1$, which results in high values of significance η. This has been done because the TRExFitter program uses its own cross-section and therefore the one from n-tuples is redundant. For the by hand determination of the cross-section limit, σ_t from the n-tuples is used with appropriate scaling factor ξ.

8.1 Comparison of classifiers

8.1.1 Description

We have five classifiers at our disposal, their hyperparameters are given in section 7.3. Every classifier is trained on four masses from $500 - 1600$ GeV interval. At the beginning of every training there is a random split that slightly changes the structure of the data, experiments are therefore repeated ten-times for every mass then we also plot the standard deviation for every classifier.

8.1.2 Results

Figure 8.1 presents results of the performance test. We can observe that it is very difficult to determine a clear winner. Let us point out that high number of significance appeared because cross-section parameter in weights was set to 1 pb. At first five experiments were run and then an additional five. High value of standard deviation remained similar. Table 8.1 shows how long it took for every algorithm to finish the task, which is testing on four masses and repeated ten times. High value for LightGBM is surprising but if we take into consideration that algorithms were optimized for performance it happens very often that solutions with high number of estimators prevail which results in high training time.

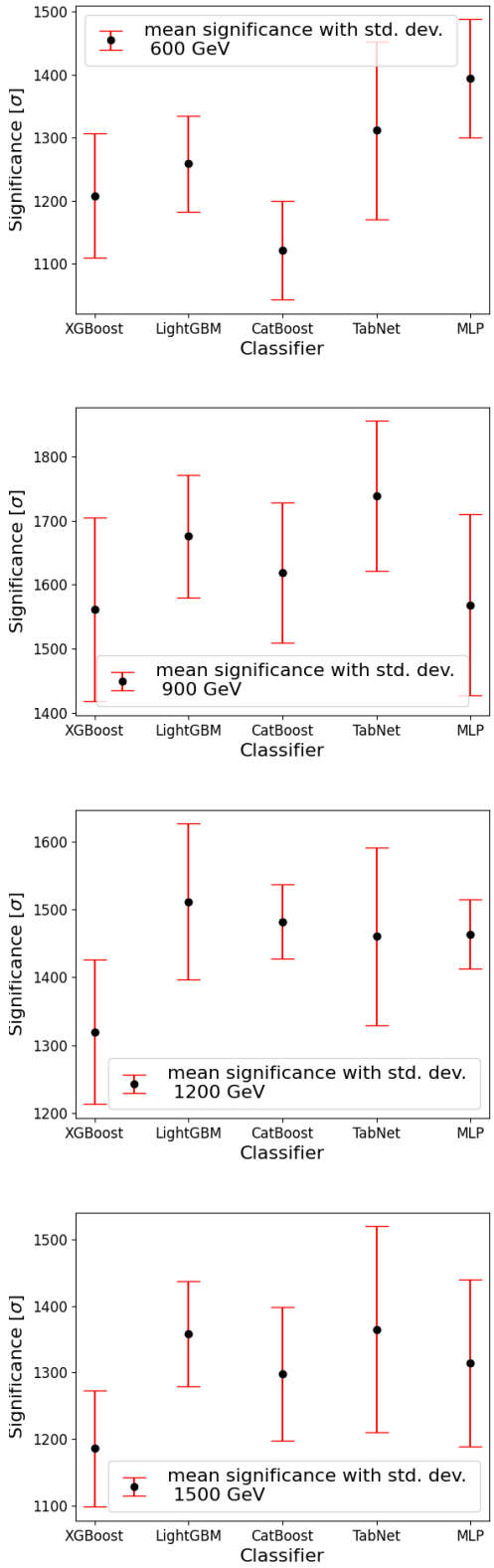


Figure 8.1: Mean results for all five algorithms tested in this thesis, XGBoost, LightGBM, CatBoost, TabNet and MLP with standard deviation after ten repetitions.

XGBoost	LightGBM	CatBoost	TabNet	MLP
53 m 42 s	89 m 11s	11 m 55 s	38 m 41 s	40 m 52 s

Table 8.1: Comparison of training speed for XGBoost, LightGBM, CatBoost, TabNet and MLP, Events = 81830, Features = 89, runs = 40.

8.1.3 Conclusion

The results for algorithms are very similar. To choose the two best, we also take their speed into account. According to this competition, CatBoost and TabNet are chosen for following experiments.

8.2 All masses combined versus separate masses

8.2.1 Description

This auxiliary experiment shows that it is not needed to use n individual classifiers to separate signal from background for n masses. We choose three masses from the interval $500 - 1600$ GeV and use them for the analysis. We train classifiers on all the masses and on chosen separate masses, then test on masses that corresponds to separate ones used for training. Training is performed 5 times to achieve a better reliability.

8.2.2 Results

We chose 600, 1000 and 1500 GeV masses, for this experiment. Therefore, we train on 600, 1000 and 1500 GeV and all the masses combined and test on 600, 1000 and 1500 GeV. Figure 8.2 shows training on all the simulated data. It is visible that for every chosen mass we acquired good sensitivity.

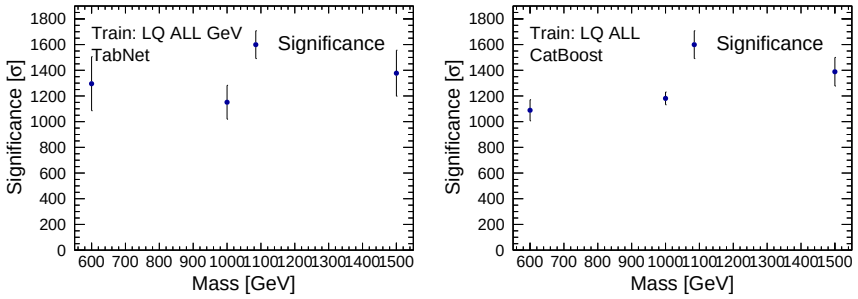


Figure 8.2: CatBoost and TabNet trained on all masses available.

Figure 8.3 shows results of training on one separate mass. we can observe that for majority of masses, results are good only when tested on corresponding mass. For CatBoost trained 1500 GeV results are poor even for corresponding mass.

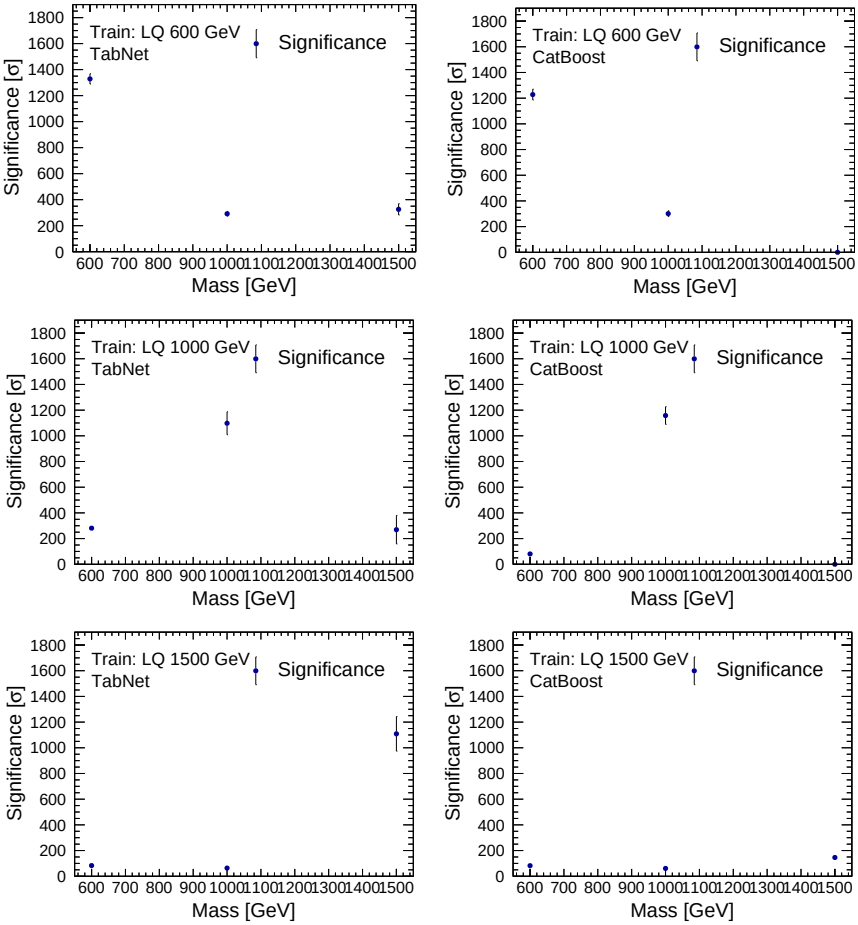


Figure 8.3: CatBoost and TabNet trained on Top: 600 GeV, Middle: 1000 GeV, Bottom: 1500 GeV.

8.2.3 Conclusion

From previous section results, we can conclude that training on all masses offers similar and sufficient performance for every testing mass, whereas training just on a separate mass is sometimes unstable for some classifiers even for the corresponding mass. Therefore, we can substitute 12 separate classifiers by an universal one and achieve stable results for signal and background separation.

8.3 Effect of the dataset size

8.3.1 Description

We would like to study how the dataset size affects the performance of the classifier. This experiment is approached in such a way that we start with 20% of available data and the size is increased by 20% every step until we reach 100%. We repeat every run 10 times to obtain reliable results. The experiment is performed for two masses from $500 - 1600$ GeV.

8.3.2 Results

Figure 8.4 shows how TabNet and CatBoost perform on reduced size of dataset. The experiment is repeated 10 times for higher reliability, thus we tested only on 600 and 1500 GeV masses.

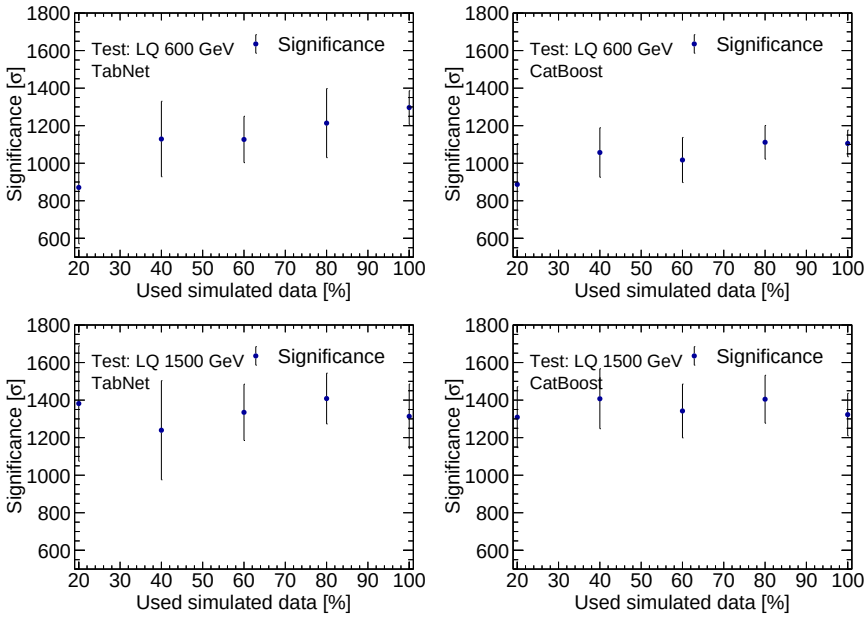


Figure 8.4: CatBoost and TabNet tested on 600 and 1500 GeV with dataset size reduction from 100% to 20% with 20% steps.

8.3.3 Conclusion

We can see from our results that if we would increase size of the dataset, we would certainly consequently increase stability of results. We can also conclude that slightly higher significance would be achieved.

8.4 Feature selection

8.4.1 Description

For this analysis, we use n-tuples that contain hundreds of features, during the pre-selection process when data are prepared for classification this number is reduced to 89. In this experiment, we perform a similar task as previously, but the number of features is reduced instead of the dataset size. We train on three masses from interval $500 - 1600$ GeV, every feature and mass combination is done 5 times.

Then we determine the four most important features and plot their histograms to see how much their distribution differs.

8.4.2 Results

We trained on 600, 1000 and 1500 GeV. Figure 8.5 shows the results.

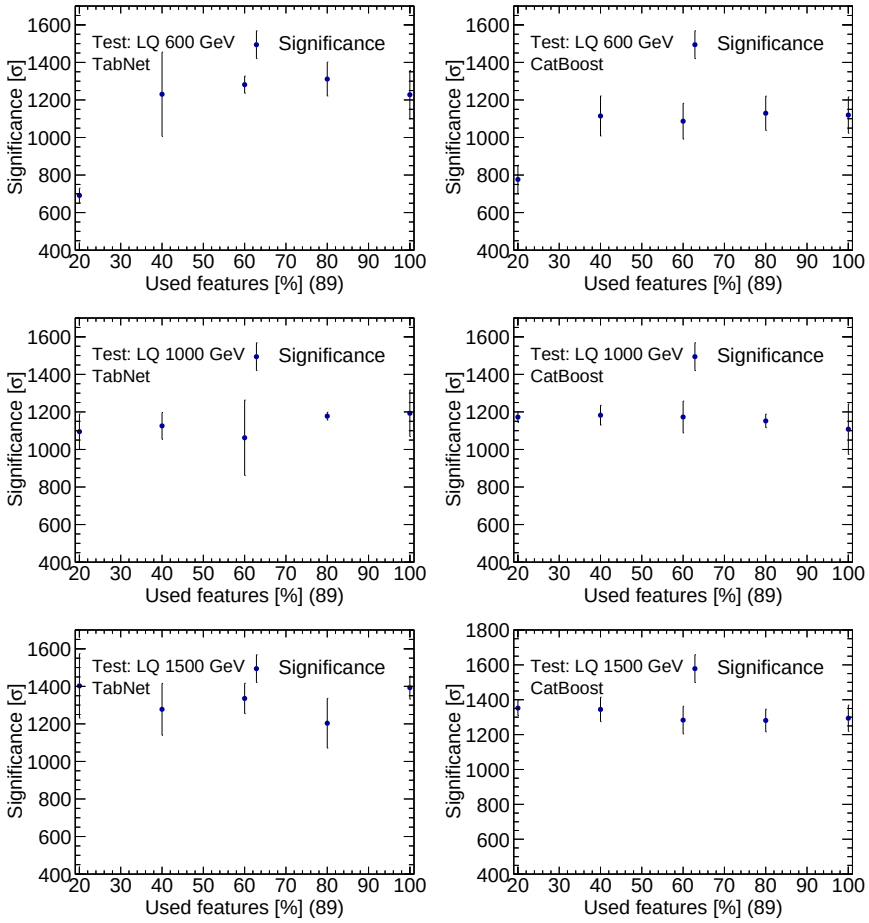


Figure 8.5: CatBoost and TabNet trained on Top: 600 GeV, Middle: 1000 GeV, Bottom: 1500 GeV.

The feature importance was changing significantly after every data shuffle, therefore we decided to use the following method to score the feature. An experiment of feature importance was performed 20 times and the order was noted. Then we summed the positions for every feature. The most important features according to this experiment are given in Table 8.2.
Then, we plot the feature histograms as shown in Figure 8.6. Cross-sections

Feature	Value
sumPsbtag	43
MtLepMet	90
lep_Pt_1	128
nJets_OR_TauOR	149
taus_pt_0	151
taus_RNNJetScore_0	219
met_met	235
taus_JetBDTSigMedium_0	246
DeltaR_min_lep_jet_fwd	293
minDeltaR_LJ_0	345
taus_decayMode_0	380
jet_pt0	408
Mll01	470
Ptll01	497
total_leptons	511
best_Z_other_MtLepMet	520
lep_E_1	550
lep_ID_0	587
jet_pt1	601
minOSMll	605

Table 8.2: Feature importance results for twenty the most important features, based on 20 experiments.

for particular masses were set to values that correspond to approximately significance of 2σ. They are shown in Table 8.3.

Mass [GeV]	Cross-section [pb]
500	0.0035295
600	0.00202356
700	0.00214232
800	0.0012804
900	0.00102528
1000	0.00148372
1100	0.0012529
1200	0.0011235
1300	0.00131376
1400	0.00140148
1500	0.00147178
1600	0.00158574

Table 8.3: Cross-section chosen for the most important feature plots. They correspond to significance of 2σ expected cross-section calculation.

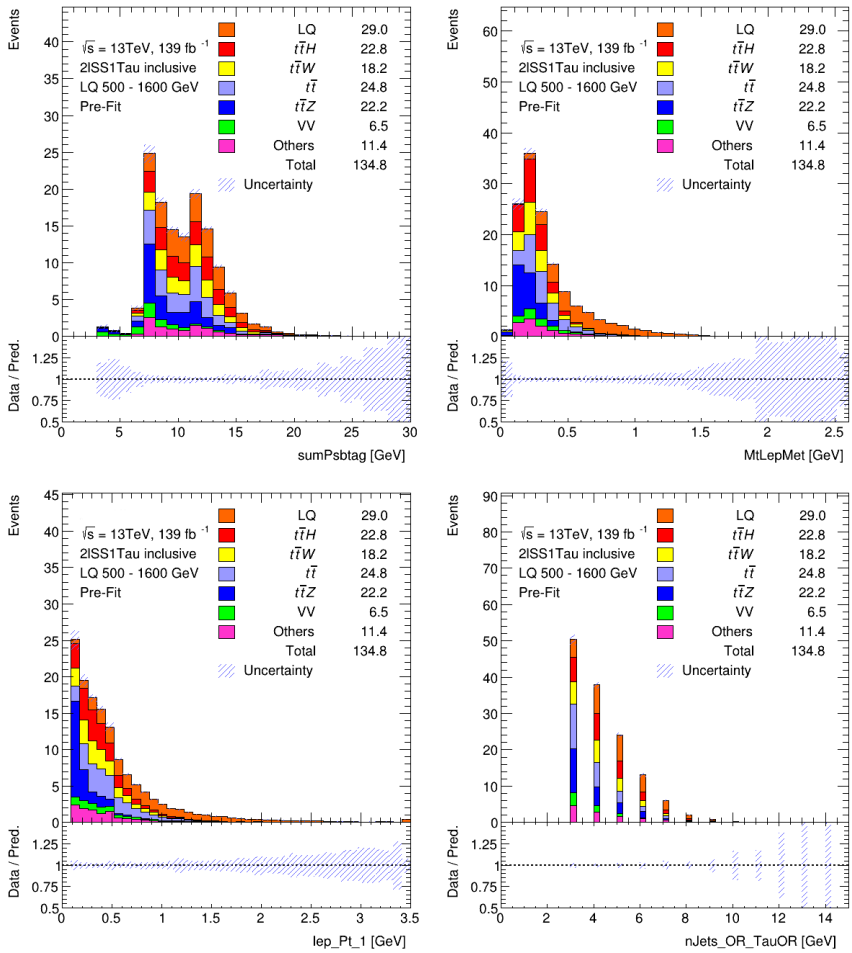


Figure 8.6: Histogram for features with the best score.

8.4.3 Conclusion

It is visible that after feature selection, performance is affected for lower masses and small number of features. For testing on 600 GeV and feature number reduced to 20%, we can observe such behavior. Difference in deviations are probably caused by data shuffle and feature correlations.

8.5 Cross-section limit

8.5.1 Description

Based on results from the previous experiments, the TabNet algorithm was chosen as the most suitable for this task. For every mass from interval, $800 - 1300$ GeV this experiment is performed and repeated 5 times. When separation results are obtained from the classifier, we use these results as input for the TRExFitter program. Result are values for expected cross-section σ_e at 95% CL are obtained. The results from [11] and [12] were also acquired using TRExFitter thus they can be compared directly with our results.

8.5.2 TRExFitter

TRExFitter is a framework used by many ATLAS physics analyses for statistical inference via profile likelihood fits [49]. It is also used to generate publication-level pre-fit and post-fit plots and tables.

TRExFitter needs data in root n-tuple format and a configuration file for its functionality. We already described the structure of n-tuples, therefore we focus on the configuration file. Important are the following sections [50].

- Job: Block that defines general options as shown in Figure 8.7.

```
Job: "LQ_2lSS1Tau"
  CmeLabel: "13 TeV"
  POI: "mu_lq"
  ReadFrom: NTUP
  NtuplePaths: "."
  NtupleName: nominal
  LumiLabel: "138.3 fb^{-1}"
  Lumi: XXX_LUMI % pb^-1
  PlotOptions: YIELDS,CHI2
  GetChi2: TRUE
  DebugLevel: 2
  HistoChecks: NOCRASH
  ImageFormat: "png","eps"
  Selection: XXX_SELECTION
  ReplacementFile: lq_replacement.txt
  SplitHistoFiles: TRUE
  BlindingThreshold: 0.15
```

Figure 8.7: Example of Job section code.

- Fit: Block specifies details of the fit model, as shown in Figure 8.8.

```
Fit: fit
  FitType: SPLUSB
  FitRegion: CRSR
  FitBlind: TRUE
  POIAsimov: 1
  UseMinos: "mu_lq"

NormFactor: "mu_lq"
  Title: "#mu lq"
  Min: 0
  Max: 10
  Nominal: 1
  Constant: FALSE
  Samples: lq_300
```

Figure 8.8: Example of Fit section code.

- Region: Blocks define the distributions considered in the fit, as shown in Figure 8.9.

```
Region: l21tau_lepPt0
 Label: "LABEL"
 Type: CONTROL
 DataType: DATA
 Variable: "lep_Pt_0/1000",60,0,300
 VariableTitle: "lep_Pt_0 [GeV]"
```

Figure 8.9: Example of Region section code.

- Sample: Blocks define the samples considered, as shown in Figure 8.10.

```
Sample: "lq_300"
 Type: SIGNAL
 Title: "#lq 300"
 TexTitle: "$lq 300"
 FillColor: 632
 LineColor: 1
 NtupleFiles: lq
 MCweight: "XXX_MCWEIGHT"
```

Figure 8.10: Example of Sample section code.

- Systematic: blocks specify systematic uncertainties.

The TRExFitter program was run by *trex-fitter* command within a specialized Docker container [51]. Then ROOT n-tuples, config and replacement file are needed. A root n-tuple contains required data. A config file contains all the sections mentioned above. The value of some parameters in the config file are too long or complicated and placing them explicitly in the file would complicate readability, therefore a replacement file is used to store these more complicated parts. The values are then referenced from the config file.
The following commands are used to produce particular results with the TRExFitter.

- trex-fitter n config_file: loads ROOT n-tuples.
- trex-fitter d config_file: draws pre-fit plots.
- trex-fitter ndwl config_file: additional w and l options are used to produce cross-section limit for given mass.

8.5.3 Results

Figure 8.11 shows cross-section limits from plots of our and previous results [11] and [12](Figure 75). The tendency is similar. Let us remind, theoretical cross-section is σ_t, expected cross-section is σ_e and observed one is σ_o.

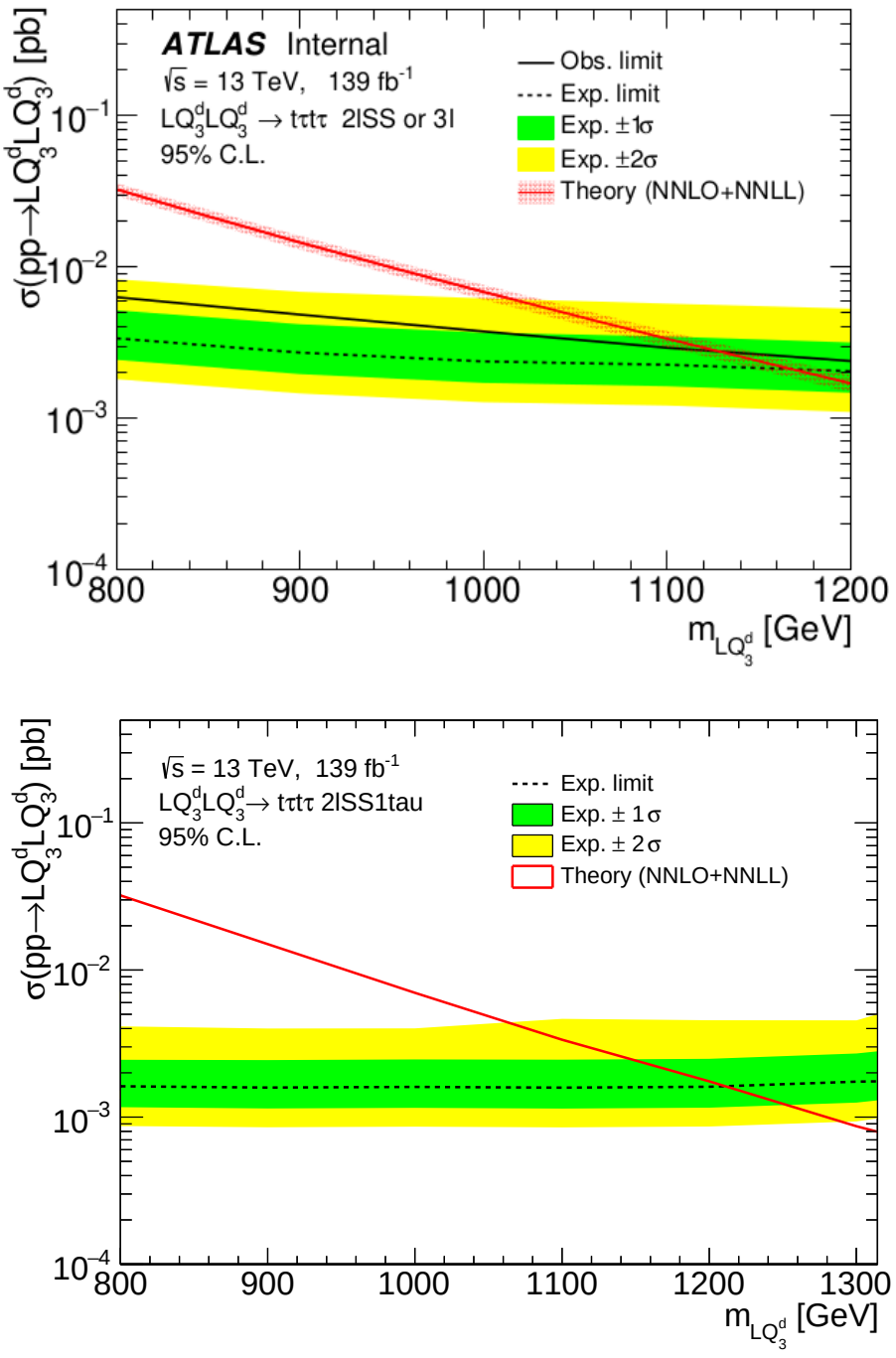


Figure 8.11: Top: Limit from original paper [11] and [12](Figure 75). Bottom: Result of our analysis. Red line indicates σ_t, solid line σ_o and dashed line is σ_e

Figure 8.12 is a direct comparison of σ_e from Figure 8.11. It is visible that the new result moves the σ_e line slightly down, which indicates improvement.

Note that the new results considered statistical uncertainties only

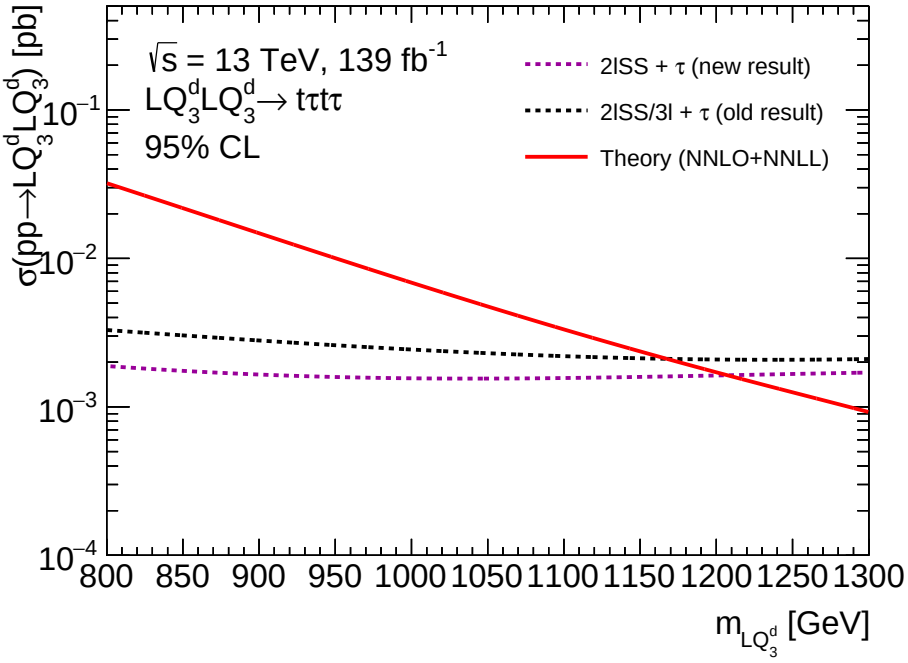


Figure 8.12: Comparison of two σ_e. Magenta: new result. Black: previous result [11] and [12](Figure 75).

8.5.4 Conclusion

Based on the results plotted with TRExFitter we can state that we managed to achieve a slight improvement in comparison to the previous publication [11] and [12]. The expected mass limit for the Leptoquark at the 95% CL was increased from about 1160 GeV to 1210 GeV. No detailed analysis of statistical and systematic uncertainties are included in this comparison.

8.6 Mass prediction

8.6.1 Description

The goal of the thesis has been to separate signal (Leptoquarks) and background (LQ, $t\bar{t}H$, $t\bar{t}W$, $t\bar{t}Z$, $t\bar{t}$, VV, *Other*) and calculate σ_e for our results. Therefore, at this point we can separate Leptoquarks from background but for real data, we can not determine their mass. For this purpose, we designed a second classifier that takes a selected LQ signal as input and predicts its mass.
For this experiment, it is important to use the same random data split as for the first classifier to prevent data leakage. Let us explain the overall process.

We focus on the first classifier. The original dataset contains all data remaining after pre-selection. We train on all LQ masses plus background and test on one LQ mass plus background. As a result, we obtain separated data. Events that are assigned as signal are used as input for the second classifier. For testing purposes we do not need to use separated data, the original 20% of LQ data is sufficient.
The second classifier is trained only on LQ masses. At this point we want to recognize which mass is on the input, therefore we classify into 12 classes, one for every mass. As a result, we obtain the probability that the event on the input belongs to certain LQ mass class. Diagram 8.13 shows the procedure.

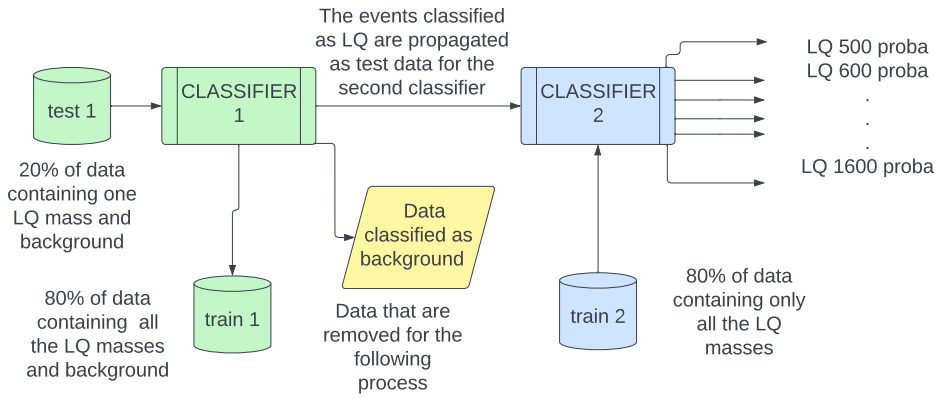


Figure 8.13: Green: Blocks corresponding to data separation. Blue: Blocks corresponding to mass prediction. Yellow: Data that are no longer used.

8.6.2 Results

Figure 8.14 shows on the horizontal axis, the mass present on the input of the classifier. On the vertical line, there is the predicted mass. To calculate the predicted mass, we use the output predictions from the classifier. Therefore, we know which event was classified as belonging to a certain mass. The predicted value is calculated as following

$$m_p = \frac{\sum_{i=1}^{12} n^i \cdot m_e^i}{N}, \tag{8.1}$$

where m_p is the predicted mass, n is a number of events, m_e is the expected mass and N is an overall number of events. The formula is the weighted mean over mass classes with the corresponding number of events.

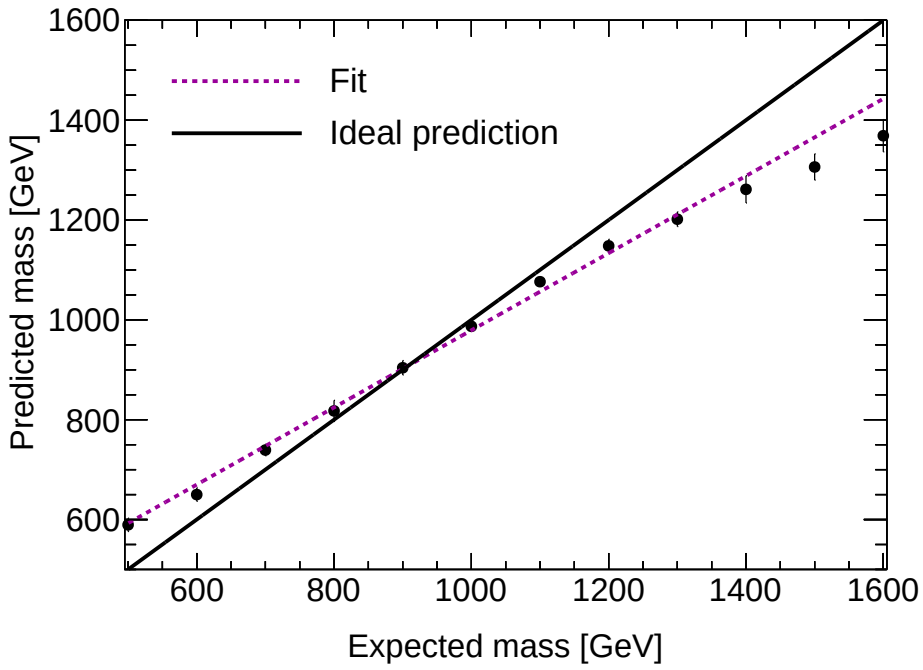


Figure 8.14: On the horizontal line is the mass that we would expect, on the vertical line is the predicted mass, then we fit the results to stress the tendency. The solid line is the diagonal for comparison. The error bars indicate the statistical uncertainty on the predicted mass, calculated as $\sqrt{variance}$.

8.6.3 Conclusion

Figure 8.14 shows that there is a good linear relation between the expected mass and the predicted mass. Above 1400 GeV there is a deviation from the

linear dependence beyond the statistical expectation. The linear dependence allows applying a scaling factor to obtain the predicted mass to be close to the expected mass.

Chapter 9

Conclusions

A method for cross-section limit calculation considering Leptoquark mass region from 500 to 1600 GeV belonging to $2lSS + 1\tau_{had}$ was developed. It included five state-of-the-art approaches for tabular data classification. XGBoost, LightGBM, CatBoost belong to Gradient boosting decision tree family of algorithms, whereas TabNet and PyTorch implementation correspond to neural networks. Classifiers were trained and tested on Monte-Carlo simulated data.

The classifiers were compared and two most suitable candidates are CatBoost and TabNet. At first, the performance of the classifiers was compared when trained on one separate mass and all the masses combined. The mass combination outperformed the single mass training in significance and reliability. Then, the effect of dataset size and feature number was studied. It is concluded that with larger dataset size and higher number of features, results are better and have smaller uncertainties. The 20 most important features were determined.

To set cross-section limits, TRExFitter software was used to produce limits for comparison with previous results. The TabNet algorithm was chosen. In comparison with previous results the cross-section limit was improved, corresponding to an extended Leptoquark mass range sensitivity from 1160 GeV to 1210 GeV. This is a significant improvement in comparison with previous results however, systematic uncertainties were not studied.

All tasks of the thesis were fulfilled. In addition, we implemented a Leptoquark mass determination method. A linear relation between the predicted and expected mass was observed and thus this allows to determine the Leptoquark mass, if it is discovered.

Bibliography

[1] A. Sopczak. *Searches for Leptoquarks with the ATLAS Detector*. [Online] https://arxiv.org/abs/2107.10094. 2021. DOI: 10.48550/ARXIV.2107.10094.

[2] *CERN*. [Online] https://en.wikipedia.org/wiki/CERN.

[3] C. Perkins. *CERN: Everything you need to know*. [Online] https://www.sciencefocus.com/science/cern/. Apr. 2022.

[4] ATLAS Outreach. *ATLAS Fact Sheet*. [Online] https://cds.cern.ch/record/1457044/files/ATLAS%20fact%20sheet.pdf. 2010.

[5] P. Grafström. "Lifetime, cross-sections and activation". In: (2007). [Online] https://cds.cern.ch/record/1047067, 14 p. DOI: 10.5170/CERN-2007-003.231.

[6] W. Herr and B Muratori. *Concept of luminosity*. [Online] https://cds.cern.ch/record/941318/files/p361.pdf.

[7] V. J. Martin. *Subatomic Physics: Particle Physics Lecture 3*. [Online] https://www2.ph.ed.ac.uk/~vjm/Lectures/ParticlePhysics2010_files/Particle3-2Nov.pdf.

[8] J. Zamastil and J. Benda. *Kvantová mechanika a elektrodynamika*. Univerzita Karlova v Praze, nakladatelství Karolinum, 2016. ISBN: 978-80-246-3223-0.

[9] R. Nave. *Quarks*. [Online] http://hyperphysics.phy-astr.gsu.edu/hbase/Particles/quark.html.

[10] M. A. Thomson. *Particle Physics, Handout 8 : Quantum Chromodynamics*. [Online] https://www.hep.phy.cam.ac.uk/~thomson/lectures/partIIIparticles/Handout8_2009.pdf.

[11] The ATLAS Collaboration. *Search for pair production of third-generation scalar leptoquarks decaying into a top quark and a τ-lepton in* pp *collisions at $\sqrt{s}$ = 13 TeV with the ATLAS detector*. [Online] https://cds.cern.ch/record/2750498. Jan. 2021. DOI: 10.1007/JHEP06(2021)179. arXiv: 2101.11582.

[12] M. Haleema et al. *Search for leptoquark pair production decaying to $t\tau$ $t\tau$*. ANA-EXOT-2019-15-INT1, ATLAS note. June 2020.

[13] R. Shwartz-Ziv and A. Armon. *Tabular Data Deep Learning is Not All You Need*. [Online] https://arxiv.org/pdf/2106.03253.pdf. Nov. 2021.

[14] javaTpoint. *Decision Tree Classification Algorithm*. https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm. (accessed April 17, 2022).

[15] J. Brownlee. *A Gentle Introduction to Ensemble Learning Algorithms*. [Online]https://machinelearningmastery.com/tour-of-ensemble-learning-algorithms/. April 27, 2021(accessed April 17, 2022).

[16] J. Starmer. *STATQUEST!!! An epic journey through statistics and machine learning*. [Online] https://statquest.org/video-index/.

[17] T. Chen and C. Guestrin. "XGBoost". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. [Online] https://doi.org/10.1145%2F2939672.2939785. ACM, Aug. 2016. DOI: 10.1145/2939672.2939785.

[18] S. Saha. *XGBoost vs LightGBM: How Are They Different*. [Online] https://neptune.ai/blog/xgboost-vs-lightgbm. Feb. 2022.

[19] A. Sharma. *What makes LightGBM lightning fast?* [Online] https://towardsdatascience.com/what-makes-lightgbm-lightning-fast-a27cf0d9785e. Oct. 2018.

[20] Guolin K. et al. "LightGBM: A Highly Efficient Gradient Boosting Decision Tree". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. [Online] https://proceedings.neurips.cc/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf. Curran Associates, Inc., 2017.

[21] L. Prokhorenkova et al. *CatBoost: unbiased boosting with categorical features*. https://arxiv.org/abs/1706.09516. 2017. DOI: 10.48550/ARXIV.1706.09516.

[22] B. John. *When to Choose CatBoost Over XGBoost or LightGBM [Practical Guide]*. [Online] https://neptune.ai/blog/when-to-choose-catboost-over-xgboost-or-lightgbm.

[23] T. Peretz. *Mastering The New Generation of Gradient Boosting*. [Online] https://www.kdnuggets.com/2018/11/mastering-new-generation-gradient-boosting.html.

[24] IBM Cloud Education. *Neural Networks*. [Online] https://www.ibm.com/cloud/learn/neural-networks. Aug. 2020.

[25] Simplilearn. *What is Perceptron: A Beginners Guide for Perceptron.* [Online] https://www.simplilearn.com/tutorials/deep-learning-tutorial/perceptron. Feb. 2022.

[26] S. Sharma. *What the Hell is Percpetron?* [Online] https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53. Sept. 2017.

[27] sethneha. *Is Gradient Descent sufficient for Neural Network?* [Online] https://www.analyticsvidhya.com/blog/2021/04/is-gradient-descent-sufficient-for-neural-network/. Apr. 2021.

[28] C. Bento. *Multilayer Perceptron Explained with a Real-Life Example and Python Code: Sentiment Analysis.* [Online] https://towardsdatascience.com/multilayer-perceptron-explained-with-a-real-life-example-and-python-code-sentiment-analysis-cb408ee93141. Sept. 2021.

[29] A. Rubert. *Classification with TabNet: Deep Dive.* [Online] https://syslog.ravelin.com/classification-with-tabnet-deep-dive-49a0dcc8f7e8. Jan. 2022.

[30] S. O. Arik and T. Pfister. *TabNet: Attentive Interpretable Tabular Learning.* [Online] https://arxiv.org/abs/1908.07442. 2019. DOI: 10.48550/ARXIV.1908.07442.

[31] V. Ilango. *Tabnet — Deep Learning for Tabular data: Architecture Overview.* [Online] https://vigneshwarilango.medium.com/tabnet-deep-learning-for-tabular-data-architecture-overview-448ced8f8cfc. Apr. 2021.

[32] K. Cranmer. *Confidence Intervals (Limits).* [Online] https://indico.cern.ch/event/208901/contributions/1501047/attachments/323314/450930/Cranmer_L3.pdf. 2013.

[33] X. Wang Y. Gao L. Lu. *Significance Calculation and a New Analysis Method in Searching for New Physics at the LHC.* [Online] https://cds.cern.ch/record/896115/files/com-phys-2005-052.pdf. Sept. 2005.

[34] J. Maly. "Automatic event recognition for Higgs boson detection". Presented 25 Jun 2020. May 2020. URL: https://cds.cern.ch/record/2722145.

[35] R Brun and F Rademakers. *ROOT - An Object Oriented Data Analysis Framework.* Version latest. [Online] 10.5281/zenodo.6408044. Apr. 2022. DOI: 10.5281/zenodo.6408044.

[36] *Python interface: PyROOT.* [Online] https://root.cern/manual/python/.

[37] Ch. R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020). [Online] https://doi.org/10.1038/s41586-020-2649-2, pp. 357–362. DOI: 10.1038/s41586-020-2649-2.

[38] The pandas development team. *pandas-dev/pandas: Pandas*. Version latest. [Online] https://doi.org/10.5281/zenodo.3509134. Feb. 2020. DOI: 10.5281/zenodo.3509134.

[39] pandas development team. *pandas.DataFrame*. [Online] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html.

[40] L. Buitinck et al. "API design for machine learning software: experiences from the scikit-learn project". In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 2013, pp. 108–122.

[41] A. Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf. Curran Associates, Inc., 2019, pp. 8024–8035.

[42] *sklearn.preprocessing.StandardScaler*. [Online] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html.

[43] *Supervised learning: predicting an output variable from high-dimensional observations*. [Online] https://scikit-learn.org/stable/tutorial/statistical_inference/supervised_learning.html.

[44] *What Is The Difference Between predict() and predict_proba() in scikit-learn?* [Online] https://towardsdatascience.com/predict-vs-predict-proba-scikit-learn-bdc45daa5972.

[45] *Classification: Accuracy*. [Online] https://developers.google.com/machine-learning/crash-course/classification/accuracy.

[46] J. Korstanje. *The F1 score*. [Online] https://towardsdatascience.com/the-f1-score-bec2bbc38aa6. Aug. 2021.

[47] *Classification: ROC Curve and AUC*. [Online] https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc. 2020.

[48] S. Narkhede. *Understanding Confusion Matrix*. [Online] https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62. May 2018.

[49] K. Choi et al. "Towards Real-World Applications of ServiceX, an Analysis Data Transformation System". In: *EPJ Web of Conferences* 251 (2021). Ed. by C. Biscarat et al. [Online] https://doi.org/10.1051%2Fepjconf%2F202125102053, p. 02053. DOI: 10.1051/epjconf/202125102053.

[50] A. Held. *Template fits:TRExFitter et al.* [Online] https://indico.cern.ch/event/822074/contributions/3471458/attachments/1865561/3067487/20190619_TRExFitter_AS.pdf. June 2019.

[51] D Merkel. “Docker: lightweight linux containers for consistent development and deployment”. In: *Linux journal* 2014.239 (2014). [Online] `https://www.docker.com/`, p. 2.

Appendices

Appendix A

Pre-selection formula

```
(((abs(df.lep_ID_0)==13) & (df.lep_isMedium_0>0) & (df.lep_isolationFCLoose_0>0)&
(df.passPLIVVeryTight_0>0)) | ((abs(df.lep_ID_0)==11) & (df.lep_isolationFCLoose_0>0)&
(df.lep_isTightLH_0>0) & (df.lep_chargeIDBDTResult_recalc_rel207_tight_0>0.7) &
(df.passPLIVVeryTight_0>0)))&(((abs(df.lep_ID_1)==13) & (df.lep_isMedium_1>0) &
(df.lep_isolationFCLoose_1>0) & (df.passPLIVVeryTight_1>0)) | ((abs(df.lep_ID_1)==11) &
(df.lep_isolationFCLoose_1>0) & (df.lep_isTightLH_1>0) &
((df.lep_chargeIDBDTResult_recalc_rel207_tight_1>0.7)) &
(df.passPLIVVeryTight_1>0)))&((((abs(df.lep_ID_0) == 13)) | ( (abs( df.lep_ID_0 ) == 11)&
(df.lep_ambiguityType_0 == 0) & (~(((df.lep_Mtrktrk_atPV_CO_0<0.1) &
(df.lep_Mtrktrk_atPV_CO_0>0))& ~((df.lep_RadiusCO_0>20)  &((df.lep_Mtrktrk_atConvV_CO_0<0.1)&
(df.lep_Mtrktrk_atConvV_CO_0>0))))&~((df.lep_RadiusCO_0>20)&((df.lep_Mtrktrk_atConvV_CO_0<0.1)&
(df.lep_Mtrktrk_atConvV_CO_0>0)))))) & (((abs( df.lep_ID_1 ) == 11) & (df.lep_ambiguityType_1 == 0)&~
(((df.lep_Mtrktrk_atPV_CO_1<0.1)&(df.lep_Mtrktrk_atPV_CO_1>0)) & ~((df.lep_RadiusCO_1>20)&
((df.lep_Mtrktrk_atConvV_CO_1<0.1)&(df.lep_Mtrktrk_atConvV_CO_1>0))))&~((df.lep_RadiusCO_1>20)&
((df.lep_Mtrktrk_atConvV_CO_1<0.1)&(df.lep_Mtrktrk_atConvV_CO_1>0))))|((abs(df.lep_ID_1) == 13))))&\
(df.nTaus_OR==1)&((df.nJets_OR_TauOR>2) &
(df.nJets_OR_DL1r_70>0))&((df.dilep_type>0) & ((df.lep_ID_0*df.lep_ID_1)>0))
```

Figure A.1: The full version of the pre-selection $2lSS + 1\tau$ in Python code.