People's Democratic Republic of Algeria

Ministry of Higher Education and Scientific Research

University Saad Dahlab of Blida 1

Faculty of Sciences

Physics Department

Master's Thesis in Theoretical Physics

Enhancing Higgs Boson Signal Through Machine Learning Methods

Prepared by: **Billal KHIDER** and **Cerine BENHAMMOUDA**

Defended on September 22nd, 2021 before the jury composed of:

B. SI LAKHAL	PROF.	Ecole National Polytechnique - Alger	President
M. OULD MOHAMED	MCB	University Saad Dahleb - Blida 1	Examiner
K. A. BOUTELDJA	MAA	University Saad Dahleb - Blida 1	Examiner
N. BOUAYED	MCA	University Saad Dahleb - Blida 1	Supervisor
A. YANALLAH	MCB	University Saad Dahleb - Blida 1	Co-Supervisor

2020 / 2021 - Blida - 1

Abstract

The *Higgs boson* discovery took place as late as 2012, after decades of long search. Its detection was challenging, since it is produced from very rare processes with tiny cross sections around $2.2fb$ to be covered with huge amounts of background noise, to instantly into new constituents. In this thesis, the main objective is to enhance that tiny signal, meaning finding the region in the phase space where the *Higgs signal* is more dominant. Thanks to datasets provided by ATLAS opendata, we are able to find an optimal model architecture to boost the *Higgs signal*, by training different optimization techniques, where we adjust their parameters and add non correlated variables. To do so we start by using traditional techniques, but since they are not that effective in increasing the signal, we are obligated to use machine learning methods in the form of artificial neural networks and boosted decision trees algorithms as they can adjust several parameters simultaneously. After multiple training sessions, we achieve optimal architectures to enhance the Higgs signal from the real data with reasonable value. $Keywords$: Higgs boson, Machine Learning, Artificial Neural Networks, Boosted Decision Trees.

ملخص

اكتشاف بوزون هيغز لم يتم إلا مؤخرا في ٢٠١٢، بعد عقود طويلة من البحث. كان اكتشافه صعبا نظرا الى أنه ينتج من عمليات تفكك نادرة جدا ذات مقاطع فعالة صغيرة تبلغ حوالي $2.2fb$ ، و أن إشارته تغطى بكميات هائلة من ضوضاء الخلفية ، وأيضا لسرعة تحلله فوريا إلى مكونات جديدة. الهدف الرئيسي من هذه الأطروحة يكمن في تعزيز تلك الإشارة الصغيرة وإظهارها مما يعني إيجاد المنطقة المثالية في فضاء الطور حيث تكون إشارة هيغز الاكثر هيمنة. بفضل مجموعات من البيانات التى يوفرها $ATLAS\ opendata$ ، يمكننا العثور على بنية نموذجية لتعزيز إشارة هيغز باستخدام تقنيات تحسين مختلفة حيث نقوم بتعديل معاملاتها وإضافة متغيرات غير مرتبطة. للقيام بذلك، نبدأ باستعمال تقنيات تقليدية لكن نظرا أنها ليست بتلك الكفاءة في رفع الإشارة، نحن ملزمون باستخدام أساليب التعلم الآلي في شكل خورزميات شبكات عصبية إصطناعية و شجرة القرار المعززة حيث يمكنها ضبط العديد من المعاملات في آن واحد. بعد عدة تجارب حققنا بنى مثالية التي تعزز إشارة هيغز بقيم معقولة.

كلمات مفتاحية: بوزون هيغز، التعلم الآلي، شبكات عصبية إصطناعية، شجرة القرار المعززة.

Remerciements

Nous tenons à remercier **Mr. Bouayed** notre promoteur à qui nous adressons notre plus profonde reconnaissance, pour son bon encadrement et pour les conseils fructueux qu'il n'a cessé de nous prodiguer.

Nous adressons par la même occasion **Mr. Yanallah** notre co-promoteur qui n'a épargné ni temps ni effort pour nous aider.

Nos remerciements les plus sincères s'adressent aux membres du jury, pour avoir accepté d'évaluer notre travail malgré les circonstances actuelles (Covid-19).

C. BENHAMMOUDA et **B. KHIDER**

Dedication

I dedicate this dissertation work to my big family my source of inspiration,

To my friends who supported me throughout the process,

Without whom non of may success would be possible.

- BILLAL KHIDER

To my dear parents who never doubted me and gave me a dignified education,

To my dear grandmothers who have always encouraged me,

To my brother and sister who have always supported me,

To all my family,

To my friends.

- CERINE BENHAMMOUDA

Contents

List of Figures

List of Tables

Introduction

One of the techniques used today to study physical phenomena is the collision of particles in particle accelerators, such as the LHC, high-level mathematical analysis methods are used to study the products of the collisions that can indicate signs of the presence of some stealthy particles like the $Higgs$ boson or particles beyond the standard model, like the graviton or the supersymmetric particles. The physics program of the large Hadron Collider (LHC) has the potential to address and to unravel many of the most fundamental questions in modern physics, such as: the origin of mass, the particle nature of dark matter, and the missing pieces associated with the comprehension of the theory of fundamental interactions. One of the main objectives of the LHC is to prove the experimental consistency of the Standard Model (SM), an essential aspect to understand the mechanism that breaks electroweak symmetry which gives mass to elementary particles.

The data generated by high energy physics (HEP) experiments from the proton-proton collisions at the center of mass energy $\sqrt{s} = 13$ TeV is very huge with high dimensions resulting from approximately 1 billion collision per second which would exceed today's technological storage capabilities by far [1]. The digitized summary is recorded as *collision events*. Since, it is not possible to read out all these events, a "$trigger$" system is used to filter the collected data and select the rare events with a very small cross section around $2.2fb$ $(2.2 * 10^{-43}\ m^2)$ for further analysis.

Traditional data analysis techniques in HEP use a sequence of boolean decisions followed by statistical analysis on the selected data [2]. The techniques that consisted of changing one parameter at the time used to work perfectly on small data and were able to study all the simple particles. But eventually, we had to go further as an attempt to detect particles resulting from rare processes, so big data needed to be introduced where these techniques became insufficient. Typically, both the individual cuts decisions and the subsequent statistical analysis are based on the distribution of a single observed quantity motivated by physics considerations, which is not easily extended to higher dimensions. Hence, all this complexity leads physicists to search for new methods to improve their analysis by employing machine learning algorithms that utilize multiple variables simultaneously. Even though, all this variety of machine learning techniques are applied in high energy physics until lately in 2012, it provides an important boost in the environments of data analysis and multivariate classifications, saving countless human-hours of design and analysis work.

In our thesis, we are interested in the $H \longrightarrow ZZ^* \longrightarrow 4l$ channel to *enhance the Higgs signal* from the simulated and experimental datasets. The procedure we follow to unravel the tiny signals like the Higgs', is to put several cuts on the phase space surrounding the region in which the Higgs signal is higher than the background noise and ignoring other regions. We started by applying traditional cuts on the data, before we concluded that those methods are no longer effective when facing a huge size of high dimensional data. The thing that takes us to the main aim of this master degree thesis is: to show how we can amplify the *Higgs* signal buried inside a huge background by applying machine learning methods (ANN and BDT). We then confront the problem of optimization and how to handle it for better results. The optimization (adjustment) of parameters works as magnifying glass as they both make objects appear bigger. Another target of our work is to evaluate machine learning models with the new unlabeled experimental data.

This thesis is divided into three chapters :

Chapter 1 We explain how the data is generated for the golden decay process in two part:
1- The experimental data from the LHC experiments, and particle identification and reconstruction.
2- The simulated data generated by *CERN* softwares with all the steps required in particle

identification and reconstruction.

Chapter 2 We start with a small generalization of concepts of machine learning, confront the optimization problems in machine learning with some solutions, and also explain how the machine learning models we use (ANN and BDT) work for the signal and background classification.

Chapter 3 The chapter is devoted to present and discuss the results we obtain from adding cuts and modifying some parameters in the machine learning models, in order to obtain the best architecture which leads to *enhance the Higgs signal over the background ratio*, while dealing with the data provided by ATLAS open source.

Chapter 1

Generating data for the $pp \longrightarrow H \longrightarrow ZZ \longrightarrow 4lep$ process

1.1 Experimental Data

1.1.1 Large Hadron Collider

The Large Hadron Collider (LHC) is the most powerful particle accelerator ever built. It was constructed by the European Council of Nuclear Research (CERN) between 1998 and 2008, in collaboration with over 10,000 scientists and hundreds of universities and laboratories in more than 100 countries, on the franco-swiss border near Geneva, Switzerland, inside a tunnel over 100 metres below CERN. Ever since, the LHC is pushing the limits of human knowledge, in cosmology, astrophysics and *high energy physics*, enabling physicists to go beyond the Standard Model, and understand the fundamental nature of the universe. Such as the famous discovery of the *Higgs* boson that was announced by CERN on 4^{th} of July 2012, at about 125 GeV, to be the first fundamental discovery at the LHC. In addition to that, the LHC is expected to have the key answers to some of the most controversial questions of the age: supersymmetry; dark matter; dark energy; extra dimensions.

Inside the LHC, proton-proton collisions are performed with a 13 TeV centre-of-mass energy, where bunches of protons (10^{11} each bunch) are accelerated every second at very high speed, nearly the speed of light, to collide into each other, where each single proton-proton collision can decay into 100 resulting particles or more, including electrons, photons, and less familiar particles such as muons. The new particles are detected by either the ATLAS detector or the CMS detector. These detectors study the trajectory of the particles and their energy, so that we can re-illustrate what happened in the collision, so that eventually we understand how physics works on the lowest level and to search for new particles. [3]

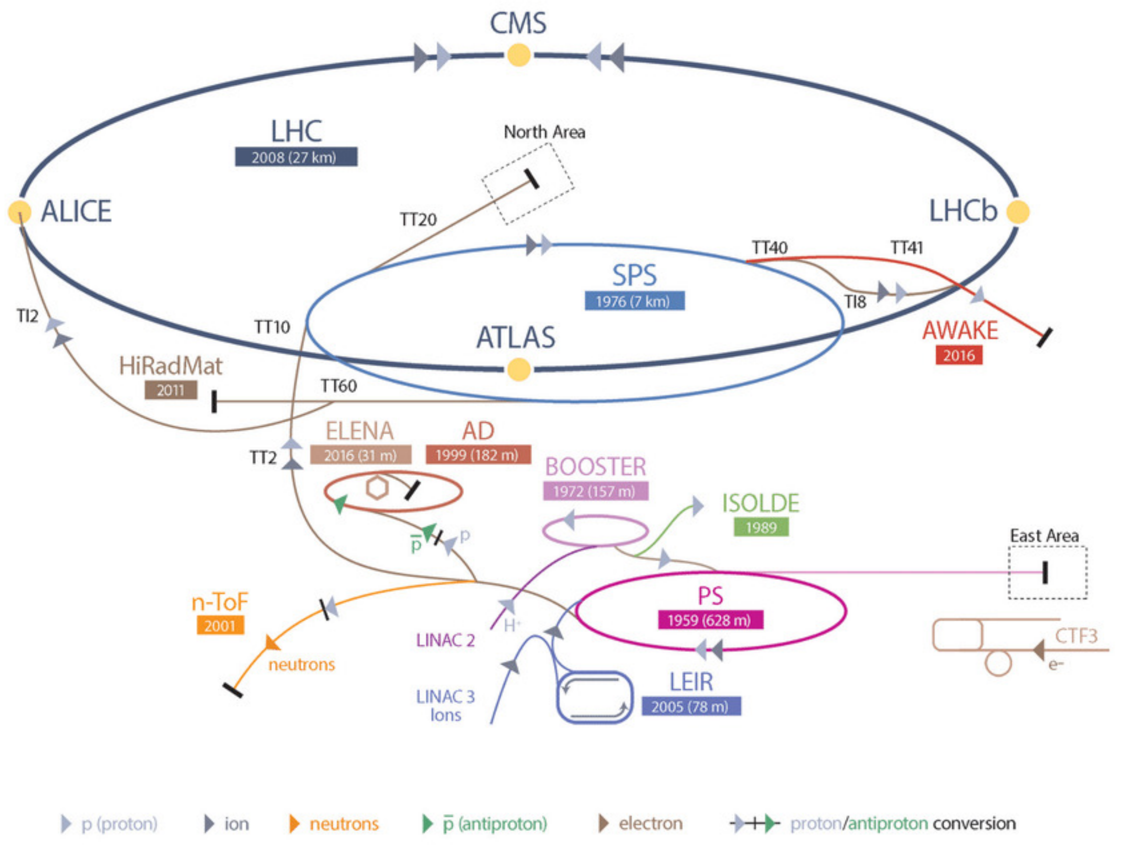


Figure 1.1: Schematic of CERN's LHC starring accelerator complex. Colors denote accelerated particle type. (*).

1.1.2 ATLAS Detector

ATLAS is one of two general-purpose detectors at the LHC. It has a cylindrical shape of 46 meters long, 25 m high and 25 m wide, the ATLAS detector weights over 7000 tonne and is the largest volume particle detector ever constructed [4]. Every second, the LHC sends beams of particles with high energy up to 6.5 TeV, or at 99.99999 % the speed of light, where one billion collision take place at the center of the ATLAS detector, the detector then, track and identify only one million collision, the ones that are considered interesting to help in the current scientific studies.

ATLAS is composed mainly out of six different detecting subsystems to detect the different particles, wrapped in layers around the collision point to record the trajectory, momentum, and energy of the particles, and identify them as precisely as possible. There is a huge magnet system bends the paths of the charged particles to measure their momenta. The inner tracking detector (ID) is made out of a silicon pixel detector, a silicon microstrip detector (SCT), and a straw tube transition radiation tracker (TRT). The ID is enclosed by a thin superconducting solenoid that provides a 2 T magnetic field, and by high granularity liquid Argon (LAr) sampling electromagnetic calorimetry. The electromagnetic calorimeter is divided into a central barrel with a pseudorapidity of $|\eta| < 1.475$, an end-cap regions on either end of the detector, the outer wheel $1.375 < |\eta| < 2.5$, and $2.5 < |\eta| < 3.2$ for the inner wheel. In the range paired to the ID ($|\eta| < 2.5$), it is radially divided into three layers, the first one has a fine sectionalisation in η to ease e / γ separation and to ameliorate the resolution of the shower position and direction measurements. In the region $|\eta| < 1.8$, the electromagnetic calorimeter is preceded by a presampler detector to correct for upstream energy losses. An iron-scintillator calorimeter gives hadronic coverage in the central rapidity range $|\eta| < 1.7$, while a LAr hadronic end-cap calorimeter provides coverage over $1.5 < |\eta| < 3.2$. The forward regions ($3.2 < |\eta| < 4.9$) are instrumented with LAr calorimeters for both electromagnetic and hadronic measurements. The muon spectrometer (MS) surrounds the calorimeters and consists of three large air-core superconducting magnets providing a toroidal field, each with eight coils, a system of precision tracking chambers, and fast detectors for triggering. The blend of all these systems provides charged particle measurements together with efficient and precise lepton and photon measurements in the pseudorapidity range $|\eta| < 2.5$. [5]

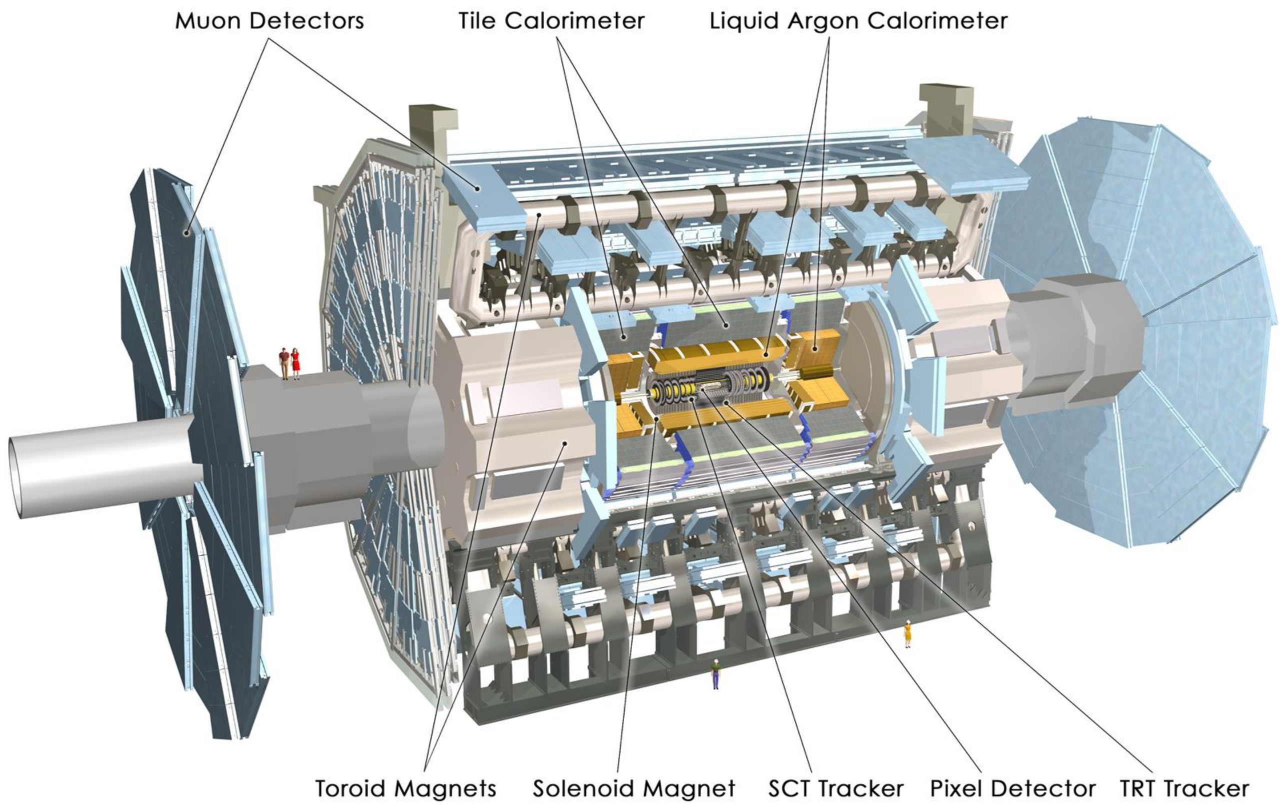


Figure 1.2: ATLAS detector and its sub-detectors (*).

1.1.3 Proton Proton collision

The goal of colliding particles is to answer questions such as what is all matter made of, and what creates the interactions of matter, in the most fundamental level. By discovering new particles and phenomenon we can find answers to these questions. Particle production, the concept of mass-energy relationship, has been always one of the selected topics to investigate in high energy nuclear reactions over several decades. This type of research probably started by the time scientists wanted to accelerate particles up to relativistic speeds and to smash them into other particles and see what may turn out. Passing over the techniques of acceleration and particle detection, scientists observed that in proton-proton (pp) collisions at relativistic energy, more particles came out than those went in. The extra came out particles were, principally, created pions and/or heavier particles at higher interaction energies. [6]

The protons which collide in LHC origin from hydrogen atom. A simple bottle of hydrogen gas provides H_2 molecules to a duoplasmatron (ion source in which a cathode filament emits electrons into a vacuum chamber) which uses electric field to strip hydrogen atoms of their electrons to obtain protons. These protons are then accelerated by the chain of accelerators before they are injected into the LHC where dedicated particle detectors may detect products of the collisions. The corresponding collision data represent very important experimental input for studying the structure and interaction of protons and also atoms, including mainly the simplest atom - the hydrogen from which the protons originated. [7]

In most proton collisions the quarks and gluons inside the two protons interact to form a wide array of low-energy, ordinary particles. Occasionally, heavier particles are produced, or energetic particles paired with their anti-particles. Very occasionally, these collisions produce new particles for us to find. Through this process of colliding particles, scientists discovered *top-quarks*, *bottom* and *charm-quarks*, and the W and Z bosons. Acceleration is necessary because the faster the particles are, the higher energy they obtain, opening up the possibility to produce previously unknown phenomena. Most recently, the *Higgs* boson, which holds the secret of the origin of mass, was discovered at the LHC. [8]

Higgs Boson Production at the LHC

This part shows diagrams of the four main *Higgs* production channels in pp collisions at the LHC: [9], [10], [11]

gg fusion $\longrightarrow$ H Resulting from direct process via the fusion of two gluons $pp \longrightarrow gg \longrightarrow Higgs$, this is the dominant mechanism for *Higgs* production at LHC, because on the experimental side, the gluon-fusion process has the biggest cross section. And on the theoretical side, it is the most challenging process to compute because of the top-loop.

Vector boson fusion (VBF) The vector boson fusion process is the second most dominant production mode at the LHC. It typically takes to 10% of the total *Higgs* cross section for low masses and up to 50% for the very *Higgs* boson fusion channel $pp \longrightarrow qq(q\bar{q}) \longrightarrow qqVV \longrightarrow qqH$, which is useful for the *Higgs* discovery over a large range of *Higgs* mass at the LHC.

Pair fusion associated production ($b\bar{b}, t\bar{t}$) A top anti-top (or a bottom antibottom) quark pair is emitted simultaneously by two gluons. The pair then annihilates to produce a *Higgs* boson $pp \longrightarrow ggt\bar{t} \longrightarrow Ht\bar{t}$. This mode has a cross section of two magnitude orders inferior to the direct production. Thus it's not easily exploitable unless there's a very high luminosity.

Electroweak boson associated production (WH, ZH) The associated production of a *Higgs* boson with a Z vector boson, pp $\longrightarrow$ ZH, also known as $Higgs - Strahlung$, is one of the most prominent paths towards an accurate understanding of the *Higgs* boson couplings.

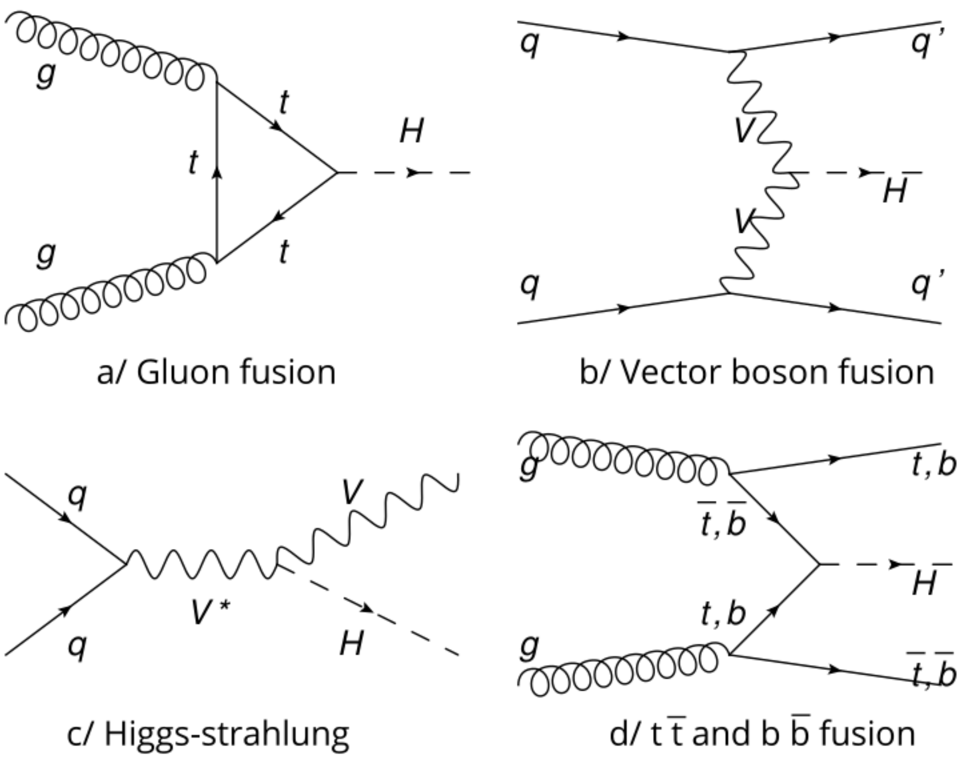


Figure 1.3: Feynman diagrams of the four major partonic *Higgs* production processes at the LHC (*).

1.1.4 Charged Particles Identification and Reconstruction

The physical processes of interest in high energy physics experiments occur on times scales too short to be observed directly by particle detectors, like the *Higgs* boson produced at the LHC will decay within approximately 10^{-22} seconds, thus decays essentially at the point of production. However, the decay products of the initial unstable particle, which are observed in the detector, can be used to deduce its properties. A better knowledge of the properties (for example type, energy, momentum, direction) of the decay products permits more accurate reconstruction of the initial physical process.

Particle identification Inside the LHC, both ATLAS and CMS detectors are based on the concept of cylindrical detection layers, interlaced around the beam axis, starting from the beam interaction region, particles resulting from the proton proton collisions first enter a tracker, in which charged particle trajectories (tracks) and origins (vertices) are reconstructed from signals (hits) in the sensitive layers. Particles are observed in a detector through the energy they deposit when traversing material, which is subsequently digitized and stored at CERN Data center. The traker is immersed in a magnetic field that bends the trajectories and allows the electric charges and momenta of charged particles to be measured. Electrons and photons are then absorbed in an electromagnetic calorimeter (ECAL), The corresponding electromagnetic showers are detected as clusters of energy recorded in neighbouring cells, from which the energy and direction of the particles can be determined. Charged and neutral hadrons may initiate a hadronic shower in the (ECAL) as well, which is subsequently fully absorbed in the hadron calorimeter (HCAL). The corresponding clusters are used to estimate their energies and directions, muon and neutrinos penetrate further inside the detector they traverse the calorimeters with little or non interactions, while neutrinos escape undetected, Muons produce hits in additional tracking layers called muon detectors (silicon tracker), located outside the calorimeter. [12], [13], [14].

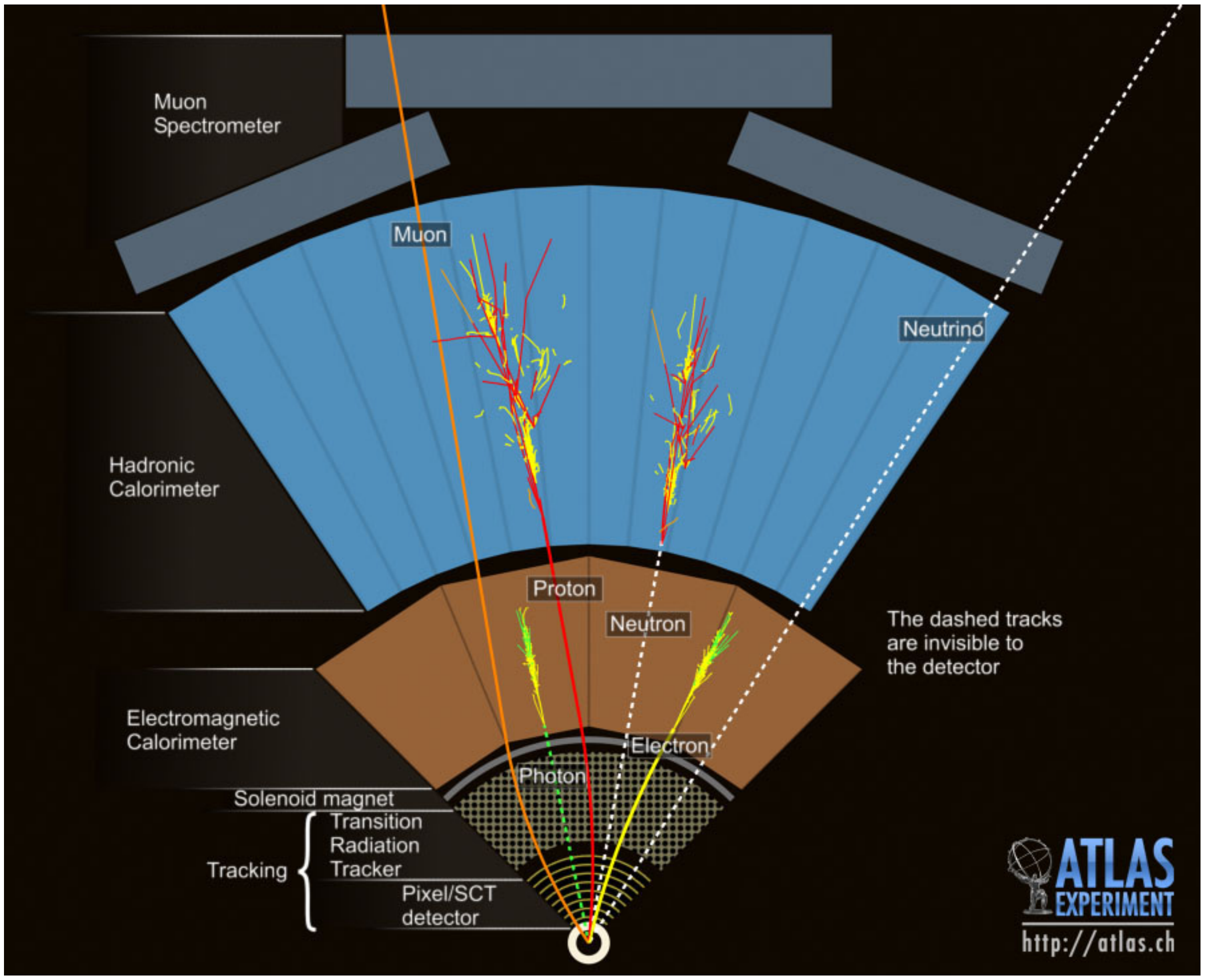


Figure 1.4: Particles identification at ATLAS detector (*).

A significantly improved event description can be achieved by exploits information from the experiment observables as :

- transverse momentum P_T
- pseudo rapidity η
- energy E_T
- azimuthal angle ϕ
- Leptons charge
- Leptons type

Particle reconstruction Reconstruction is the process of converting the raw digital signals in the detector into the physical properties of particles. Particle physics detectors are usually composed of several sub-detectors, each taking advantage of specific interaction mechanisms to detect the passage of a specific type of particle and measure its properties. Parameters extracted from the detectors' layers are gathered to identify each individual particle, called traditional identification method, and by combining the corresponding measurements to reconstruct the particle's properties on the basis of this identification.

Another method is invented by CERN called particle-flow (PF) reconstruction algorithm, this algorithm identifies and reconstructs each final state particle immediately. The algorithm attempts to estimate the energy, momentum, and the identity of each particle. Algorithm reconstruction or (PF) typically involves several steps that turn the data from the detector electronics raw measurements into higher level data objects corresponding to the physical particles that were detected (features) [15]:

• Feature Extraction : The signal from the passage of particles through a detector element, e.g calorimeter cell, is observed in the raw electronic output associated with the element. This signal is then characterized.

• Pattern Recognition : The pattern of signals in geometrically adjacent detector elements is associated with the passage of a signal or group of particles. In calorimeters, this step is commonly referred to as clustering.

• Object Characterization : Properties of the objects are measured in tracking detectors, this step means fitting a pattern of "hits" to a helix. In calorimeters, this step extracts the energy, location, and other properties of the cluster that for example characterize the shape of the cluster.

• Combined reconstruction : Objects in different detectors are associated together to create a refined particle candidate.

Electrons and Muons reconstruction and identification

Electrons and Muons are employed to reconstruct the *Higgs* decay final state in the $H \longrightarrow ZZ \longrightarrow 4lep$ channel. Good identification and reconstruction performance is decisive for this search and is achieved using the various features of the ATLAS detector systems. The Final state 4lep particles must satisfy some requirements. Electron reconstruction algorithms in the ATLAS detector start from energy deposits in the EM calorimeter (cluster energies) that are matched to reconstructed tracks of charged particles in the inner detector. Electrons with $P_T^e > 7$ GeV are reconstructed within the geometrical acceptance defined by a pseudorapidity $\eta < 2.5$, and Muons within the geometrical acceptance $\eta < 2.4$ and $p_T^\mu > 5$ GeV are reconstructed by combining information from silicon tracker and the muon system. By applying those requirements on the data collected at "data CERN center", physicists ensure that leptons (electrons and muons) coming from the decay of the golden channel are reconstructed with high quality, avoiding a huge fraction of background events. [16]

1.1.5 Expected level of Background for the $H \to ZZ^* \to 4lep$ process

The expected background yield is not distinguishable from the golden decay process. Therefore, it contribute by a huge fraction in the collected data :

- $pp \longrightarrow ZZ^*(\gamma) \longrightarrow 4lep$
- $pp \longrightarrow t\bar{t} \longrightarrow 4lep$
- $pp \longrightarrow Z + jets \longrightarrow 4lep$ (where $jets = c\ \bar{c}$ or $b\ \bar{b}$ quarks)

These background expectations within a mass window about the specific SM *Higgs* mass. For *Higgs* masses just above the LEP limit$(120-130)$ GeV. The $Z + jets$ contribute with a significant fraction, about 30% of the ZZ^* irreducible background. The $t\bar{t}$ channel is practically negligible. The reducible backgrounds $Z + jets$ and $t\bar{t}$ become negligible for masses above 150 GeV. [17]

NOTICE : ATLAS uses a right-handed coordinate system with its origin at the nominal interaction point (IP) in the centre of the detector and the z-axis along the beam pipe. The x-axis points from the IP to the centre of the LHC ring, and the y-axis points upward. Cylindrical coordinates (r,ϕ) are used in the transverse plane, ϕ being the azimuthal angle around the z-axis. The pseudorapidity is defined in terms of the polar angle Θ as $\eta = -lntan(\frac{\Theta}{2})$. The transverse momentum p_T and other transverse variables, are defined as the variables component in the

$x - y$ plane, the transverse energy E_T is defined as $\sqrt{m^2 + p_T^2}$, where m represents the mass of a considered object. The distance in the pseudorapidity-azimuthal-angle space is defined as $\Delta R = \sqrt{\Delta\eta^2 + \Delta\phi^2}$.

Tableau 1.1: Some particles properties detected by ATLAS sub-detectors.

Basic particles reconstructed in the ATLAS detector and their properties.	
Particle	Summary
Electron	Electrically charged, detected as a trajectory in the tracking system ending in the electromagnetic calorimeter.
Muon	Electrically charged and able to penetrate through the calorimeter system. Detected as a trajectory in the tracking system extending into the muon chambers.
Pion,kaon	Electrically charged, detected as trajectories in the tracking system ending in the hadronic calorimeter. These two particle types are distinguished by the hadron calorimeter (HCAL) to their trajectories and measuring the opening angle of the light cone.
Photon	Electrically neutral, identified by an energy deposit in the electromagnetic calorimeter with no nearby charged particle trajectory.

1.1.6 Max storage and Trigger system

At its design operation, the LHC delivers collision events to each of its experiments at a rate of up to 40 MHz. The amount of raw data, prior to any further data processing steps, is typically of the order of $\geq 1MB$ per recorded collision event. Therefore, if the readout systems allowed to record all collision events, the total amount of data to be stored would rise up to more than 40 TB per second, this would exceed today's technological capabilities by far. [18] For the purpose of studies, only a very tiny fraction of the initially collected data contains the rare signatures of events, which may be actually relevant to the *Higgs* process $pp \longrightarrow H \longrightarrow ZZ^* \longrightarrow 4lep$. Filtering the data and recording only such events of interest allows reducing the rate to manageable levels of a few hundred Hz. To this end, the Trigger and Data Acquisition (TDAQ system) is employed. This preprocessing system comprises three stages which are called level 1 (L1), and level 2 (L2) and Event filter (EF) as is illustrated in figure [1.5].

LEVEL 1 The hardware trigger : Level 1 of trigger is an extremely fast and automatic process that looks for simple signs of interesting physics, e.g particles with a large amount of energy or unusual combination. This is like a reader simply scanning the headlines of a newspaper to see if anything catches their eye , this level works on a subset of information from the calorimeter and muon detectors. The decision to keep the data from an event is made less than two-and-half microseconds after the event occurs. This is the way through which we select the best 100,000 events each second from the billion available.

LEVEL 2 The HLT CPUs analysis : Level 2 is based on the HLT large CPUs analysis which refines the previous analysis done by the level 1 trigger, it either takes information from sub-detector layers (calorimeters, trackers, muon detectors), it conducts a very detailed analysis. The HLT analysis can select about 1000 events per second, these events are passed on to a data storage system for further offline analysis.

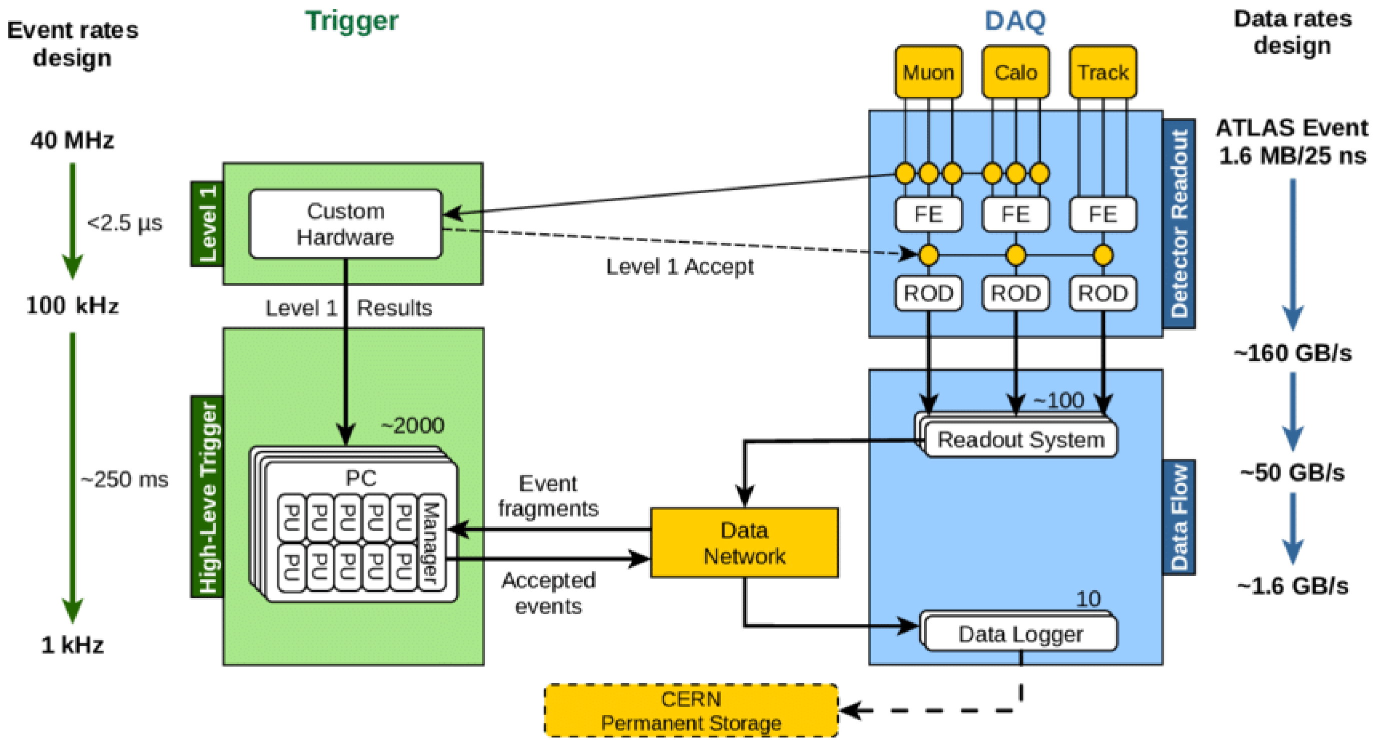


Figure 1.5: ATLAS Trigger and Data-Acquisition system (*).

1.2 Simulated Data

1.2.1 Proton Proton Simulated Collision

1. When the two beams of protons collide at the interaction points (vertex). Each resulted parton from the hadrons collisions carry a fraction of its momentum and follows Parton Distribution Functions (PDFs), which are models in terms of energy sharing and flavor [1.6].

2. The partons inside the colliding hadrons emit radiation, initiating a succession of decays $q \to qg, g \to q\bar{q}, g \to gg$. Since the strong coupling constant α_s has a large value, these branching processes have a high probability to happen.

3. At a momentum transfer scale Q^2, two partons enter the *hard interaction* process. The products of the hard scattering are the final state elementary particles, partons, leptons and bosons, that characterizes the event topology. Even though this hard process is not observable it determines the main properties of the collision event.

4. The resulting partons (quarks and gluons) start branching and initiate the final state for the hard process.

5. After branching in the initial and final state showers, the momentum scale decreases down to the cutoff scale $\Lambda_{QCD} \backsim 1$ GeV , where the perturbative theory is no longer valid.

6. Below Λ_{QCD}, the strong interaction confines the partons into colourless hadrons, which is followed by the decay of the unstable particles. Therefore, through fragmentation and decay, the parton cascades evolve into jets of stable and meta-stable particles which are observable in particle physics detectors. [19]

There is a main approach which is adopted in the Monte Carlo generators, in order to describe and reproduce the phenomenology at the hadronic collider. This approach is called factorization theorem which allows the independent treatment of perturbative and nonpertubative phenomena. For a proton proton collision $p_A p_B \longrightarrow X$, where X is a generic final state, this theorem can be expressed by the formula:

$$\sigma_{AB} = \int dx_a dx_b f_a(x_a, Q^2) f_b(x_b, Q^2).\bar{\sigma}_{ab\longrightarrow X} = \begin{cases} f_a(x_a, Q^2) f_b(x_b, Q^2) & \text{non perturbative factor} \\ \bar{\sigma}_{ab\longrightarrow X} & \text{perturbative factor} \end{cases} \tag{1.1}$$

Where σ_{AB} is the total cross-section. x_a and x_b are the fractions of the parton momentum carried by the two partons a and b involved in the interaction. and $\bar{\sigma}_{ab\longrightarrow X}$ is the hard partonic scattering cross section.

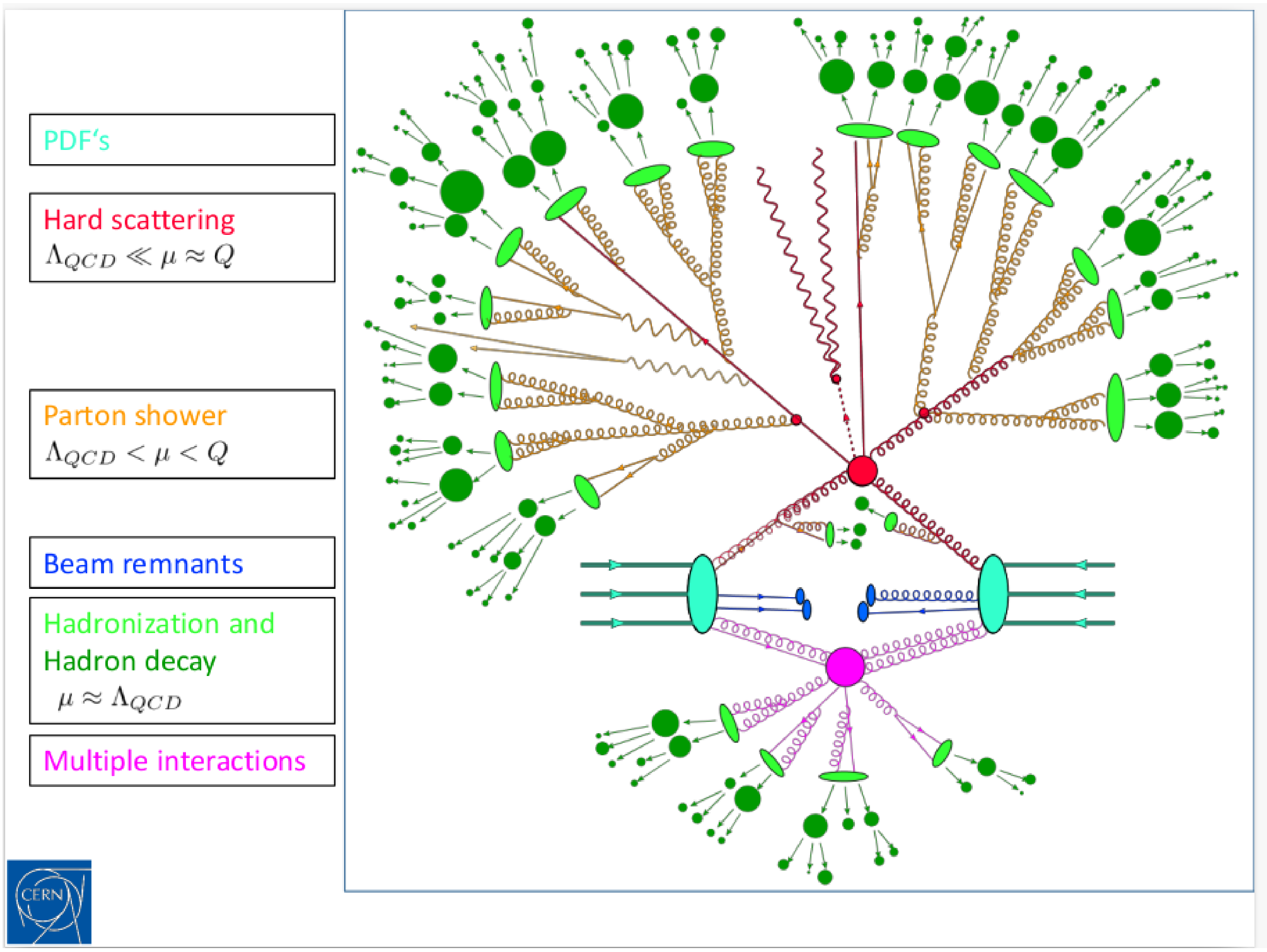


Figure 1.6: Schematic description of the Hard proton proton collision (*).

Parton shower distribution functions In the equation 1.1 we've seen the parton distribution functions (PDFs) describe the dynamics of the partons participating in the hard process in relation with the protons colliding They represent the probability of a parton to carry a fraction x of the proton momentum. The PDF estimations uncertainties need to be taken into consideration in any theoretical prediction. There are many PDF sets available, the most common ones : $QTEQ$ [20], $MSTW$ [21], $NNPDF$ [22].

1.2.2 Steps in Event generation process

Simulated data, commonly called Monte Carlo (MC), is a key feature of the LHC experiments, these events are simulated with machines using theoretical models and are used to compare theory with real data in order to search for new physics phenomena. [19] [23] The full simulation requires the following steps :

1. The compilation of the hard subprocess The first step in the event generation is the simulation of the hard process which is defined by the collision of two bunch of protons constituents at high momentum scale.

2. The parton shower and hadronization step After the hard collision a parton shower is used to evolve the event further. Due to the large momentum transfers during the hard subprocess step, the final state particles obtained have a high energy. During the parton shower, the involved partons lose energy.

3. Pile-up simulation In the same bunch crossing there are multiple pp interactions, most of them are not frontal and collide with low energy. Those interactions constitute the so-called pile-up. The pile-up is reproduced adding to the underlying event additional simulated events.

4. Detector Simulation Interaction of the generated particles inside the ATLAS detector is simulated, including the propagation of each particle in the different detector layers. For the active detector elements, the simulated energy deposits are processed through a simulation of readout electronics for each sub-detector, including effects such as simulated noise.

5. Digitization The detector response is derived from the particle interactions and it is written in a format compatible with the real output of the detector. In addition to that, because of the high rate of collisions in the LHC, digitised signals from several simulated events can be piled-up to create samples with realistic experimental background.

6. Reconstruction and identification Particle trajectories and energies from the detector are reconstructed by applying reconstruction algorithms which make use of the specific properties of different particle types. In the last step, the obtained information is gathered and the particle objects are identified as particles.

1.2.3 Generating data for $H \rightarrow ZZ^* \rightarrow 4lep$ process with Madgraph5

MadGraph is a Monte Carlo event generator. Nowadays, this software is widely used to simulate events at the LHC, and was used before the LHC era for obtaining future prediction in new physics models. This section shows how to generate collision evens with $MadGraph5/\_aMC@NLO$ for the golden decay $pp \longrightarrow H \longrightarrow ZZ^* \longrightarrow 4lep$ in order to compare the simulated data to the experimental data for deep studies. The software is free and available from the following website `https://launchpad.net/mg5amcnlo`. Madgraph allows us also to control the cross sections, the luminosity and the energy for the events we want to generate. It can be extended to incorporate several programs.

The required packages for generating the pp process are:

• **Delphes** : a $C++$ framework, performing a fast multipurpose detector response simulation.

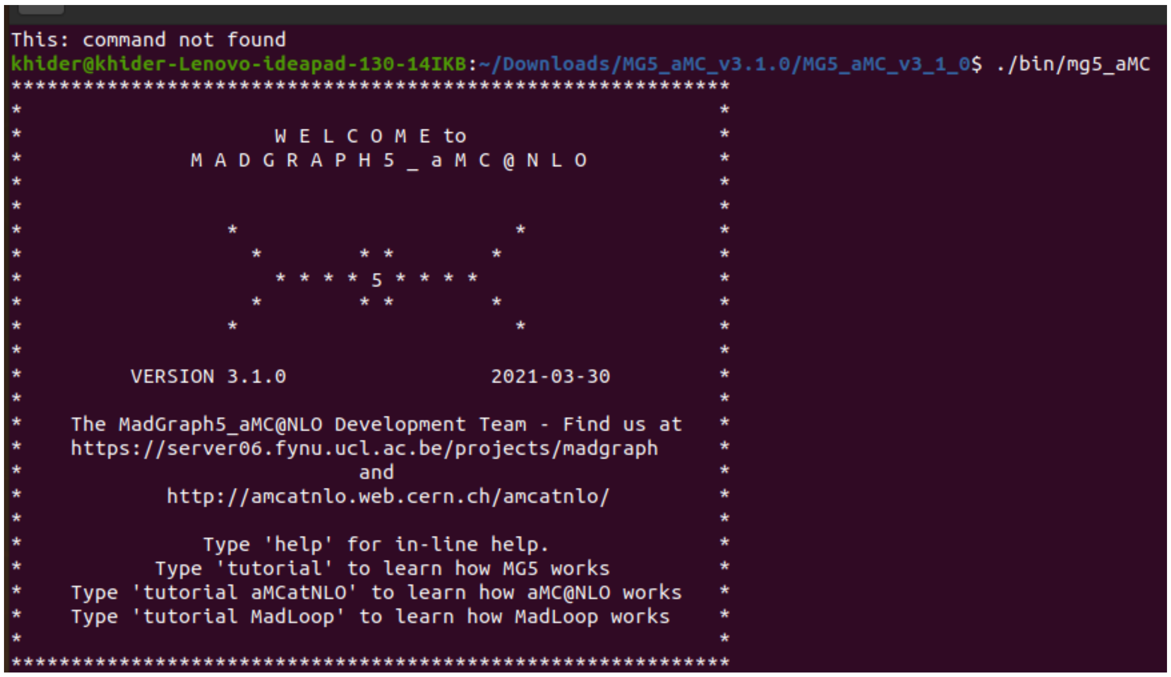
• **Pythia** : used for parton showering. The Pythia main program reads the file *pythia_card.dat* with Pythia run parameters, opens the LHE file *unweighted_events.lhe* and passes it to event simulation. The output is written in the form of *StdHep* files, which store all the Pythia event information in *HEPEVT* common block format.

• **MadAnalysis** : a Fortran based program, used for simple analysis of events and making plots.

• **ExRootAnalysis** : a package designed to simplify *ROOT* tree production and analysis.

After installing all required packages on Madgraph, we follow the next steps [24] :

• The first step is to open Madgraph software by $./bin/mg5\_aMC$ as shown in [1.7] :



```
This: command not found
khider@khider-Lenovo-ideapad-130-14IKB:~/Downloads/MG5_aMC_v3.1.0/MG5_aMC_v3_1_0$ ./bin/mg5_aMC
************************************************************
*                                                          *
*                     W E L C O M E  to                    *
*              M A D G R A P H 5 _ a M C @ N L O           *
*                                                          *
*                                                          *
*                 *                       *                *
*                   *        * *        *                  *
*                     * * * * 5 * * * *                    *
*                   *        * *        *                  *
*                 *                       *                *
*                                                          *
*         VERSION 3.1.0                2021-03-30          *
*                                                          *
*    The MadGraph5_aMC@NLO Development Team - Find us at   *
*    https://server06.fynu.ucl.ac.be/projects/madgraph     *
*                            and                           *
*            http://amcatnlo.web.cern.ch/amcatnlo/         *
*                                                          *
*               Type 'help' for in-line help.              *
*           Type 'tutorial' to learn how MG5 works         *
*    Type 'tutorial aMCatNLO' to learn how aMC@NLO works   *
*    Type 'tutorial MadLoop' to learn how MadLoop works    *
*                                                          *
************************************************************
```

Figure 1.7: Homepage of Madgraph

• The second step is to generate the following $H \longrightarrow ZZ^* \longrightarrow 4lep$ process by typing : generate $pp > h > l + l - l + l-$

• The third step is to create a folder to contain the generating process results by typing : *output h4lep*.

• The fourth step is to set all installed packages ON, with modification on the run card. The energy used is 8 TeV (4000 GeV for each beam), with a luminosity of 10 fb^{-1}.

• Finally plotting the histogram out of each delphes output file (signal, background, data).

The produced histogram with reconstructed *Higgs* mass in this events is illustrated in the figure[1.8] below :

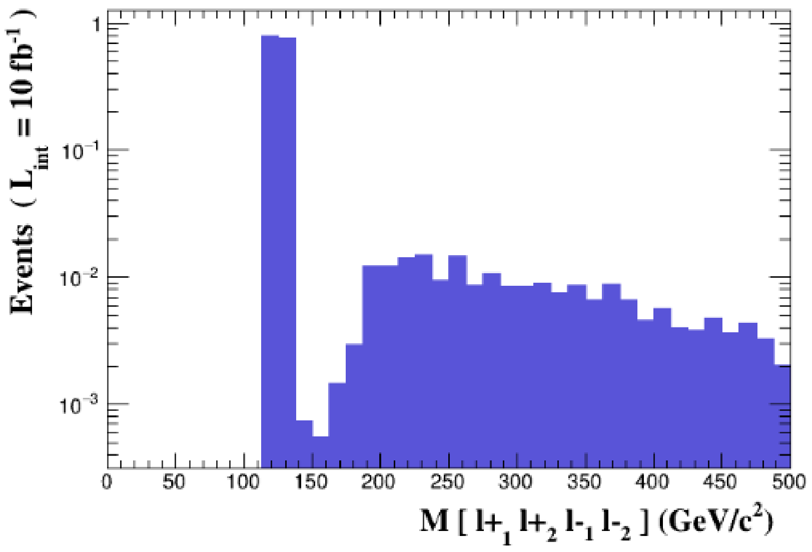


Figure 1.8: Histogram plotted with Madgraph with Higgs boson mass peak arround 126 GeV.

1.2.4 Systematic uncertainties

The uncertainties on the lepton reconstruction and identification efficiencies [16] and the momentum scale and resolution are determined using samples of W, Z and J/ψ decays as following:

Electon reconstruction and identification The relative uncertainty on the signal acceptance due to the uncertainty on the electron reconstruction and identification efficiency is $2.6\pm\%$ ($\pm1.7\%/\pm1.8\%$) for the 4e ($2e2\mu/2\mu2e$) channel for $m_{4l} = 600$ GeV and reaches $\pm8.0\%$ ($\pm2.3\%/\pm7.6\%$) for $m_{4l} = 115$ GeV respectively.

Muon reconstruction and identification The relative uncertainty on the signal acceptance due to the uncertainty on the muon reconstruction and identification efficiency is $\pm0.7\%$ ($\pm0.5\%/\pm0.5\%$) for the 4μ ($2e2\mu/2\mu2e$) channel for $m_{4l} = 600$ GeV and increase to $\pm0.9\%$ ($\pm0.8\%/\pm0.5\%$) for $m_{4l} = 115$ GeV respectively.

Electron energy scale The uncertainty on the electron energy scale results in an uncertainty of $\pm0.7\%$ ($\pm0.5\%/\pm0.2\%$) on the mass scale of the m_{4l} distribution for the 4e ($2e2\mu/2\mu2e$)

final state channel respectively.

Muon energy scale The uncertainty on the muon energy scale is $\pm 0.04\%$ ($\pm 0.015\% / \pm 0.02\%$) on the mass scale of the m_{4l} distribution for the 4e ($2e2\mu/2\mu 2e$) final state channel respectively.

Total uncertainty The uncertainties associated with the *Higgs* signal is estimated from the MC simulation is as following:

• The uncertainty on the total yield due to the QCD scale uncertainty is 5%, while the effect of the PDF and α_s uncertainty is 4% to 8% for processes associated initially by quarks (gluons).

• The numbers of expected *Higgs* signal (m_H = 125 GeV) and background events with the numbers of observed events in the data, in a window of size 50 GeV around 125 GeV is illustrated in the table below :

Tableau 1.2: The numbers of expected signal ($m_H = 125$ GeV) and background events, together with the numbers of observed events in the data, in a window of size 5 GeV around 125 GeV, for the combined $\sqrt{s} = 7$ TeV and $\sqrt{s} = 8$ TeV data.

	signal	ZZ^*	$Z + jets$,$t\bar{t}$	Observed
4μ	$2.09 \pm 0.30\%$	$1.12 \pm 0.05\%$	$0.13 \pm 0.04\%$	6
2e 2 μ / 2 μ 2e	$2.29 \pm 0.33\%$	$0.80 \pm 0.05\%$	$1.27 \pm 0.19\%$	5
$4e$	$0.90 \pm 0.14\%$	$0.44 \pm 0.04\%$	$1.09 \pm 0.20\%$	2

Chapter 2

Machine Learning methods for enhancing Higgs signal

2.1 General Machine Learning concepts

Machine learning (ML) integrates different technologies whose ultimate goal is to elicit meaningful information from a data sample. A machine learning algorithm (in the following multivariate analysis method or MVA method) is a mathematical model that contains parameters. Those parameters are optimized so that the model can describe the data efficiently. The phase in which the best values for the parameters are found is known as training : it is when the algorithm learns from the data. After the phase of learning, the algorithm can make predictions on unknown data and this phase is often referred to as application and is known as testing. [25]

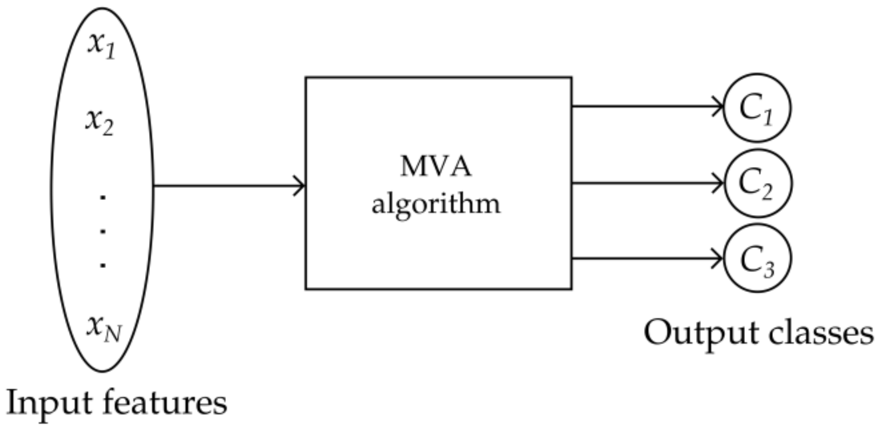


Figure 2.1: A schematic representation of a machine learning algorithm. N input features are provided, and the multivariate method assigns each event in the sample to one of three classes (*).

Algorithms take as inputs a set of variables for each data point, these variables are commonly referred to as features, they carry information about the properties of each data point. These features can be, for example, the energy and momenta of particles, where each data represents a single event detected at the LHC experiment.

The initial sub-division among machine learning algorithms is whether they implement supervised or unsupervised learning. The supervised algorithm requires each data point to be labeled with the desired output, which should produce that particular event once trained. The multivariate method learns by adjusting its parameters in order to minimise the error between its outputs and the labeled data (actual output). Unsupervised learning on the other hand, is used when there is no labeling available for the data, where events are grouped based on similarities in their features. Unsupervised learning methods are often used to preprocess the data before using supervised algorithms. These methods are sometimes referred to as clustering methods.

Machine learning can perform different tasks depending on the required outputs. The simplest one is when the data belong to two different classes like the data for the $pp \longrightarrow H \longrightarrow ZZ^* \longrightarrow 4lep$ (explained in chapter 1), the data is mixed between two classes *Higgs* signal and background. This case is called binary classification because we have two categories of features, and this is the case discussed in this master thesis for *Higgs* signal and background classification. Another task is to classify data according to more than two classes as is shown in figure [2.1] above. This kind of multi-class classification is not conceptually different from the binary classification since any k-class problem can be divided into k two-class problems. It is, however, generally simpler to directly implement a multi-class algorithm instead of relying on this theoretical fact. The third task a machine learning algorithm can perform is regression, when the output is continuous. Generally, the output of a classifier is also continuous, and can be interpreted as the probability of a data point to belong to a class. The user can then perform a cut on the output that maximises the performance of the classifier according to some criteria of choice. During our work, we will face optimization problems with some solutions discussed in the next section, since everything comes down to an optimization problem in the machine learning field.

2.2 What is Optimization ?

In machine learning techniques there is always the goal of coming up with mathematical models that best fit a set of observations, to then possibly be able to make principled predictions of similar outcomes. Given a model parameterized by weights w, and a data set of random variables $(x, y) \backsim P_{data}\ (x, y)$, learning is actually the task of maximizing or minimizing a function with respect to some of its parameters and this function is often called a “loss” or “cost” function. Optimality is defined in the context of updating the set values for the model parameters that occurs in the lowest possible value of the loss function or conversely in the highest value depending on the type of the problem. The loss is interpreted in terms of target labels in the supervised learning settings [26]. The typical formulation of an optimization problem can be phrased as follows :

$$\begin{cases} \text{Minimize } f(\theta).....(*) \\ \text{subject to w} \in \Theta \end{cases} \tag{2.1}$$

Where f is the loss function parametrized by a real vector of parameters θ that can take one value in the feasible set Θ. In the case where θ is a subset of $\mathbb{R}^d$, where d is the dimensionality of the parameter vector θ, our optimization problem is constrained. Such optimization problems can be solved with high precision even by hand if the number of parameters is very small. But, the majority of the problems are not analytically solvable. Fortunately, optimization offers an approximate solution.

Once the optimization problem has been interpreted into mathematical form, the types of algorithms that can be used to solve it are known as optimizers. These optimization methods can be divided into groups according to the level of used information and the complexity to calculate it. Zero-order methods only use the value of the fonction $f(\theta)$. First-order methods also require the value of the first derivative $f'(\theta)$. Second-order methods require the second derivative $f''(\theta)$. There are many ways for optimizing the parameters of a model, by minimizing the loss function with respect to the parameters, like the gradient based method and the maximum likelihood estimator algorithms that are discussed below:

2.2.1 Gradient based Optimization method

The gradient based or gradient descent, is one of the most popular algorithms to perform optimization, by minimizing an objective function parameterized by a model's parameter. For the problem described in equation (*) [2.1] above with a sub-differentiable function f. A solution can be found iteratively by initializing the weights to θ_0 and updating them at each iteration i according to the rule:

$$\theta_{i+1} = \theta_i - \lambda_i \nabla_\theta f(w_\theta) \tag{2.2}$$

where λ_i is the step size, or learning rate in Figure [2.2]. The learning rate can prevent convergence if too large, or slow it down if too small. Its magnitude can be held constant during the optimization procedure, or adjusted at each iteration using various methods.

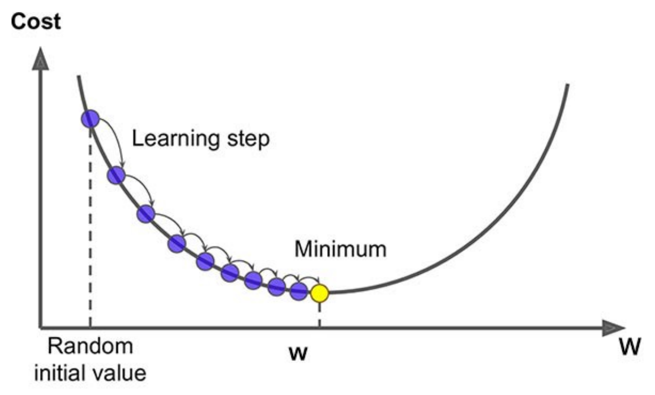


Figure 2.2: The way how gradient descent works, w in the graph is the parameter θ that we need to optimize it (*).

2.2.2 Maximum Likelihood Estimator (MLE)

Often when building statistical machine learning models, there are several parameters that we need to optimize, let's call them θ. The approach of optimizing θ is known as Maximum Likelihood Estimation (MLE), with a given sample $P_{data}(x)$, x is the vector of our sample inputs $x = x_1, x_2, ...x_n$. We construct a parametric model $P_{model}(x|\theta)$ and build the likelihood (LH):

$$L(\theta|x_1, ..., x_n) \tag{2.3}$$

$$L(\theta|x_1, ..., x_n) = p(x_1, ...x_n|\theta) = p(x_1|\theta)p(x_2|\theta)......p(x_n|\theta) \tag{2.4}$$

We can replace the probability distribution function PMF (p) (statistical term describes the probability distribution of the discrete random variable) by PDF (f) (statistical term describes the probability distribution of the continuous random variable) functions, since PDF is applicable for continues random variable while PMF is applicable for discrete random variable :

$$L(x) = p(a \leqslant x \leqslant b) = \int_a^b f(x)dx \tag{2.5}$$

$$L(\theta|x_1, ..., x_n) = \prod_{i=1}^{n} f(x_i|\theta) \tag{2.6}$$

The objective is to get the maximum value of likelihood, the higher it is, the more likely that the parameter value gives the data that we have observed. The parameter θ that corresponds to the higher LH function is called maximum likelihood estimator (MLE) as is illustrated in figure [2.3].

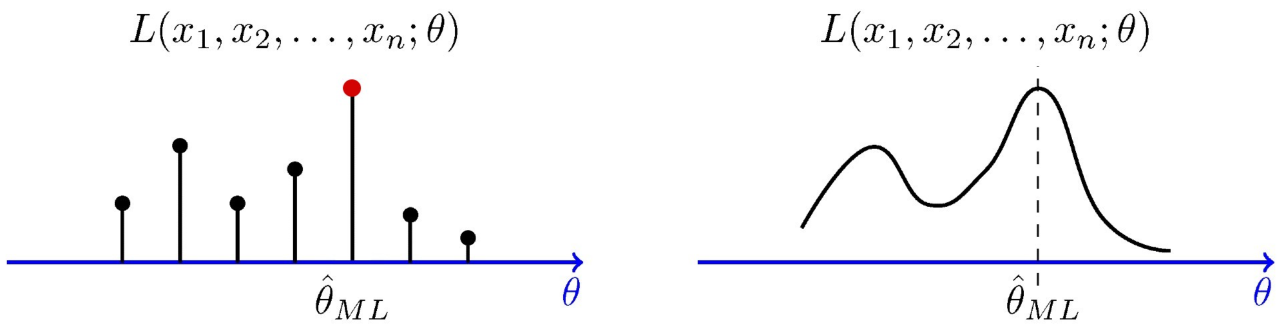


Figure 2.3: Maximum likelihood Estimator (MLE) (*).

In order to get this maximum likelihood estimator, we need to find the maximizer of our likelihood function. Fortunately, we know that the *logarithm* is a monotone function (function that preserves or reverses the given order), like the (LH). This means that maximizing the *log* function is similar to maximizing the LH function, as it is illustrated in the equation below:

$$log(\theta|x_1....x_n) = logL(\theta|x_1, ..., x_n) \tag{2.7}$$

$$\theta^* = argmax_\theta log(\theta|x_1....x_n) = argmax_\theta logL(\theta|x_1, ..., x_n) \tag{2.8}$$

The goal of the optimization is to maximize the likelihood to find the best values for parameters, therefore, the best machine learning model to classify our data. Since the product of several terms that are < 1 tends to be numerically unstable to compute, it is, instead preferred to minimize the negative log-likelihood function :

$$\theta^* = argmin_\theta(-\log L(\theta|x)) \tag{2.9}$$

2.3 Kinematic cuts on $pp \longrightarrow H \longrightarrow ZZ^{*} \longrightarrow 4lep$ Signal and background distribution and their optimization

2.3.1 Theoretical description

Let's take a signal to background distribution. Suppose we try to measure some parameter x which can be any discriminating variable that allows us to distinguish between signal (S) and background (B) events. The expected distribution for x to be S and B respectively, as it is shown in the figure [2.4] below :

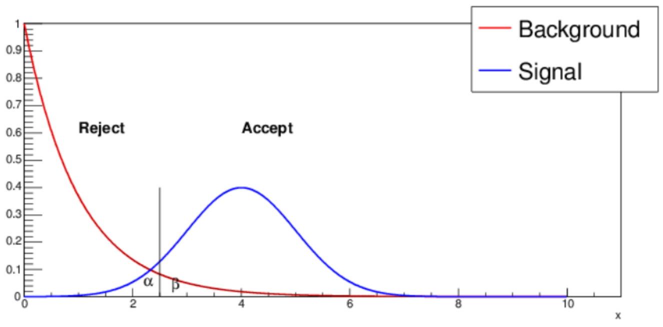


Figure 2.4: Schematic representation of a cut decision (*).

The objective here is to put the cut in the best region that allows it to take the highest quantity of the signal with just a tiny contribution of backgrounds, in order to maximize or enhance the *Higgs* signal to the background ratio. If we choose the cut as shown in the Figure [2.4], then we accept all events after $x = x_{cut}$ and we reject all those below. This means losing a fraction α of the *Higgs* signal, this is called type 1 error and α is known as the significance, and admitting a fraction of background β what is called the type 2 error and $1-\beta$ is the power. The actual problem is how we know exactly where to put the cut for minimizing the loss of the α fraction as possible inversely, in another hand, maximizing the loss of the background β, but unfortunately we can't know exactly where. Typically, the cut is placed in the region where the signal intersect with the background, in the simplest case like our example in the figure [2.4], we start moving the cut around the x axis in the direction where the signal is expected to be higher than the background, until we get the best region that gives $(S/B) > max$ or $\frac{\partial(1-\alpha)}{\partial\beta} > max$. This traditional method causes a huge loss of the signal. All these difficulties are only confronted when choosing one discriminating variable as a cut parameter. Imagine having to separately optimize more than one variable where each variable could change the distribution of another. In this case the optimization is impossible and even if it is possible it would take a very long time, here the ML is off rescue. [27]

2.3.2 Experimental description

In the largest hadron collider after the pp collisions we can measure a variety of experimental observables:

- Transverse momentum P_T
- Energy E
- Pseudo rapidity η
- Azimuthal angle ϕ
- Leptons type and leptons charge

And others for each particle resulting from the pp collision, which play the role of discriminating variables that allow distinguishing between signal and background events. The final state of the collected data with 4 leptons (4 Electrons or 4 Muons or 2 Electrons and 2 Muons) is constituted of signal events and background events. In order to reduce the contribution of the background for best signal to background ratio, we apply a variety of cuts on the collected data, for example:

In the golden process, the transverse momentum p_T for Electrons is required to be higher than 6 GeV and higher than 7 GeV for Muons, the pseudorapidity η is required to be lower than 2.47 for Electrons and 2.70 for Muons. Also, the data required to verify the sum of leptons charge is equal to zero. [5] When we apply these cuts criterions on the data a huge fraction of background (zz^*, $t\bar{t}$ and also $z + jets$) is thrown away, because the majority of background 4 leptons final state don't verify the required criterions, but even by applying a number of cuts there is always fractions of background contribute in the data. It is coming from the 4 leptons background process which verifies the criterions cuts.

2.3.3 The need of machine learning for powerful optimization

The data collected from the ATLAS detector is complex with high dimensions and the correlation is not visible. The traditional analysis techniques used by physicists is a sequence of boolean cut decisions followed by statistical analysis on the selected data. Typically, both the individual cuts decisions and the subsequent statistical analysis are based on the distribution of a single observed quantity motivated by physics consideration, which is not easily extended to higher dimensions. Fortunately, the ML algorithms, like Artificial Neural Network (ANN) and Boosted Decision tree (BDT), can optimize a set of variables at the same time that will give a better classification (*Higgs* signal or background events) results than individual traditional cuts. [28], [29]

2.4 Artificial Neural Network in High Energy Physics

Artificial neural networks are an important tool in high energy physics [30], playing a major role in data analysis. It performs complex tasks, learns and memorises all at the same time, as it is an attempt to imitate the human brain and its magnificent abilities. There are several types of neural networks that form the basis for most pre-trained models in deep learning:

- Artificial Neural Networks (ANN).
- Convolution Neural Networks (CNN).
- Recurrent Neural Networks (RNN).

Here, we will be discussing the ANN which is the simplest.

2.4.1 How does Artificial Neural Network work?

Artificial Neural Network is formed out of layers of artificial neurons, one input layer, one output layer and one or more hidden layers in-between. Each neuron is connected fully or partially to the neurons in the next layer, just like in the biological neurons. Two-class classification problems are among the tasks most frequently encountered in high energy physics data analysis, usually in the form of separating interesting data (signal) from unwanted noise (background). Each neuron in the network receives inputs from either the input variables (input layer) or from the previous layer, and provides an output either of the entire network (output layer) or it is used as an input to the next layer. Within a neuron, inputs are combined linearly with proper weights that are different for each of the neurons. Each output is then transformed using an activation function as the *logistic* function (*sigmoid*), with this function the output values are bound between 0 and 1 (see Figure [2.5]). We can also use the rectified linear activation function (*ReLU*) (see Figure [2.6]). [2]

$$Sigmoid(x) = \frac{1}{1+\exp(-x)}, \tag{2.10}$$

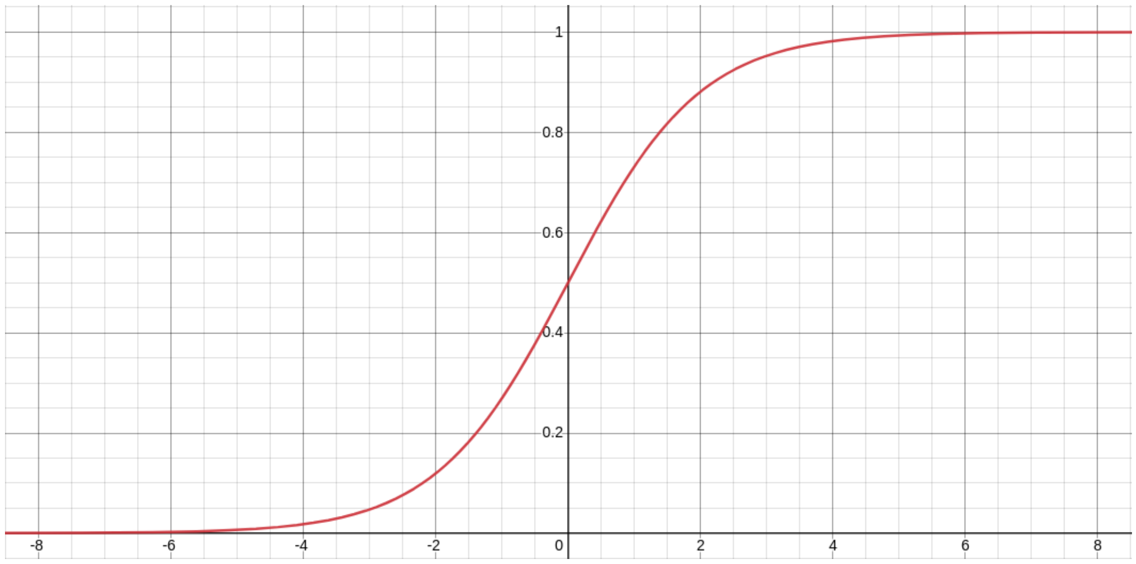


Figure 2.5: Graphical representation of the Sigmoid function (Logistic).

$$ReLU(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \tag{2.11}$$

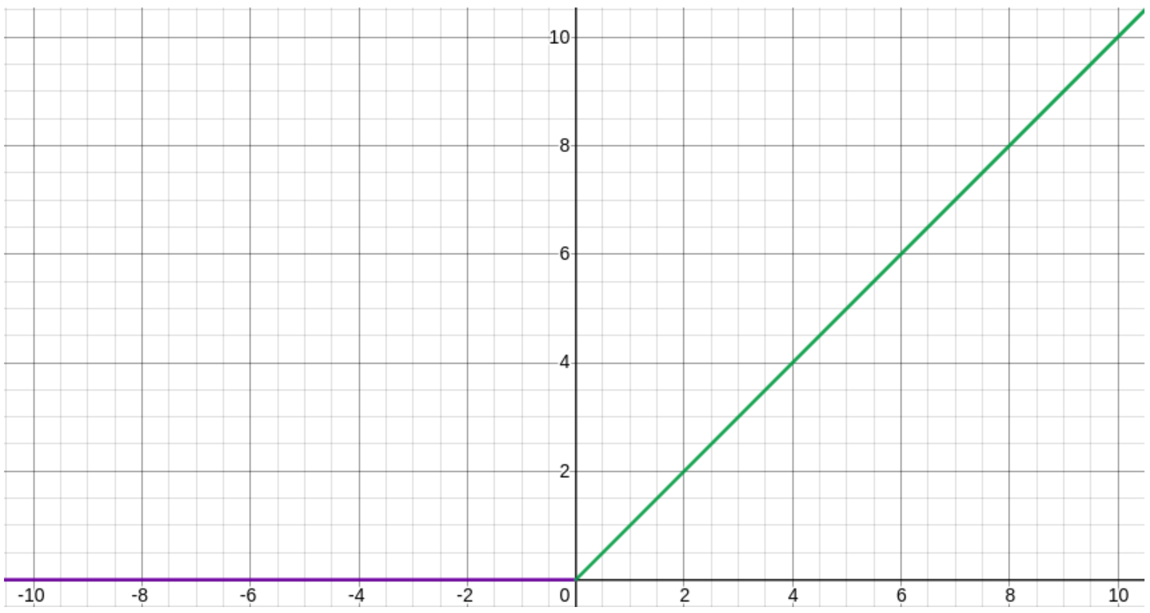


Figure 2.6: Graphical representation of the rectified linear activation function (ReLU).

An artificial neuron receives a set of inputs (x_1, x_2, ..., x_n). They can be any measured variables resulting from the golden proton proton collision that allows us to distinguish between the *Higgs* signal and background events. A weight w_i (i = 1,..., n) is associated with each input. If the weight is positive, then the signal is excited. Otherwise the signal is inhibited. Artificial Neurons collect all the input signals, calculate a net signal and transmit an output signal as it is illustrated in figures [2.7] and [2.8].

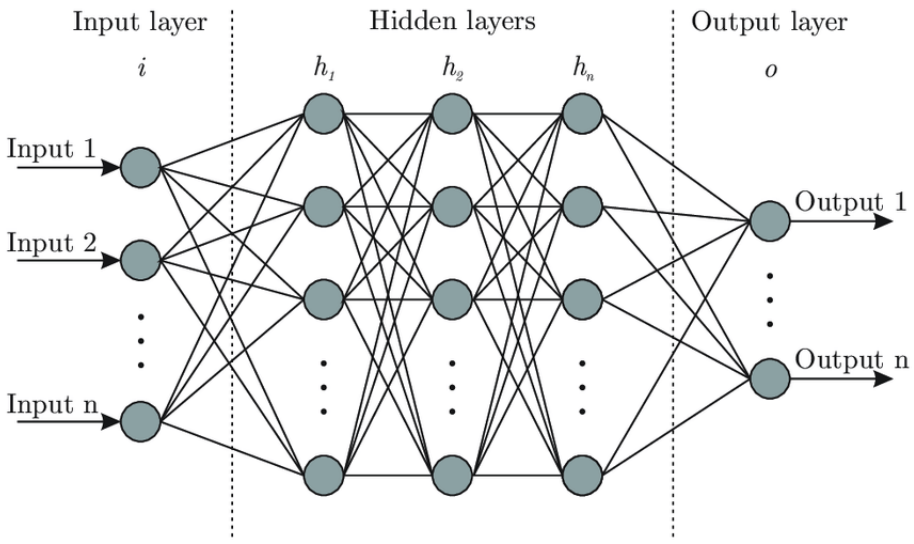


Figure 2.7: Schematic representation of an Artificial neural network with multiple hidden layers (*).

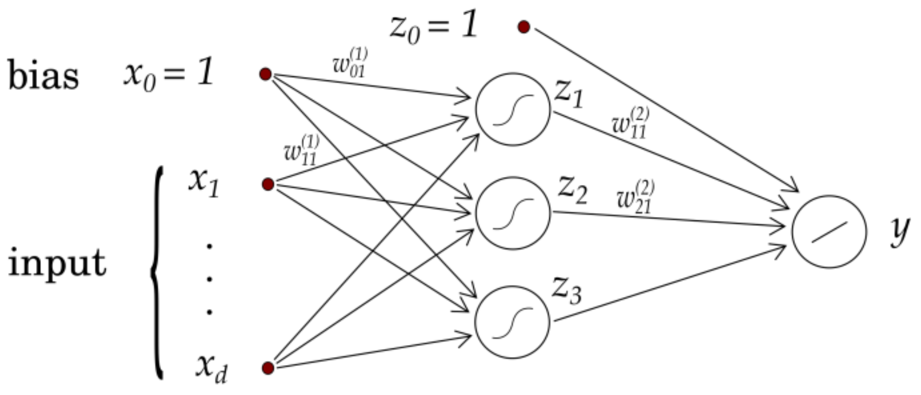


Figure 2.8: A multilayer perceptron with a hidden layer made up by 3 neurons and a bias node (*).

The outputs y resulting from the process of artificial neurons in the figure[2.8] is given by:

$$y = W^{(2)}_{01} + \sum_{j=1}^{3} W^{(2)}_{j1} z_j = W^{(2)}_{01} + \sum_{j=1}^{3} W^{(2)}_{j1} Sigmoid(w^{(1)}_{01} + \sum_{i=1}^{d} W^{(1)}_{ij} x_i) \qquad (2.12)$$

2.4.2 Training Neural Network

In a neural network, the space of functions searched is defined by the structure of the network, which defines a series of transformations. In a simple case, the first embedding is simply the input vector, and the final embedding is the output of the network. The elements of the matrix W are referred to as weights and those of vectors as biases. The general structure of these transformations, such as the dimensionality of each W and the choice of activation function is referred to as the network architecture, which, taken together with the training parameters constitute the hyperparameters of the network. The weights and biases of the network are initialized randomly. Finding the hyperparameters that minimize the loss function is done through an iterative process called training. Conceptually, this uses the labeled training examples and calculates the gradient of the loss function with respect to the model parameters. In practice, the calculations are done through a technique called backpropagation, which is an efficient process of computing this gradient.

Artificial neural networks start by assigning random values to the weights. The key for the ANN to perform its task correctly and accurately is to adjust these weights to the right numbers. But finding the right weights is not very easy, especially when you're dealing with multiple layers and thousands of neurons.

Basically, what happens during training is that the network adjusts itself to glean specific patterns from the data. In our case for example, the *Higgs* signal and background noise event classification, when you train the Artificial Intelligence (AI) model with a set of data, each layer detects a specific class of features. Further down the network, deeper layers will start to pick out more and more advanced features, for when you run a new data through the well-trained neural network, the adjusted weights of the neurons will be able to extract the right features and determine with accuracy to which output class the event belongs. One of the challenges of training neural networks is to find the right amount and quality of training examples. Also, training large AI models requires vast amounts of computing resources. To overcome this challenge, many engineers use transfer learning as a training technique where you take a pre-trained model and fine-tune it with new, domain-specific examples. Transfer learning is especially efficient when there's already an *AI* model that is close to your use case. Also, it is required to use GPU (Graphics processing unit) for parallel tasks, and a huge RAM (random access memory) for fast processes.

2.4.3 Backpropagation

Back-propagation is the essence of neural network training. It is the method of fine-tuning the weights of a neural network based on the error rate obtained in the previous epoch (i.e., iteration). In order to minimize the classification error. The mean square error is defined as :

$$E(x_1, x_2, -, x_N|W) = \sum_{a=1}^{N} E_a(x_a|W) = \frac{1}{2}\sum_{a=1}^{N}(y_a - y_a^*)^2 \tag{2.13}$$

Where the sum is taken over the whole training batch, with y_a and y_a^* are respectively the neural network output for the event x_a and its label. Through y_a, the error depends on the network's weights.

Once the whole dataset inputs, has passed through the network the weights are updated by minimizing the error (E) using gradient descent technique, which indicates the direction of steepest increase for E, the weights are then changed by a quantity defined as :

$$\Delta W = -\eta \nabla_W E \rightarrow \Delta W_i = -\eta \frac{\partial E}{\partial W_i} \tag{2.14}$$

This modification decreases the training error. η is called learning rate, it is one of the most important hyperparameters in neural networks. If it is too small, convergence to the global minimum as is illustrated in figure[2.9], might take too long or not happen at all. Conversely, if it is too large the convergence will be very fast but with bad predicted outputs, which leads to weak classification.

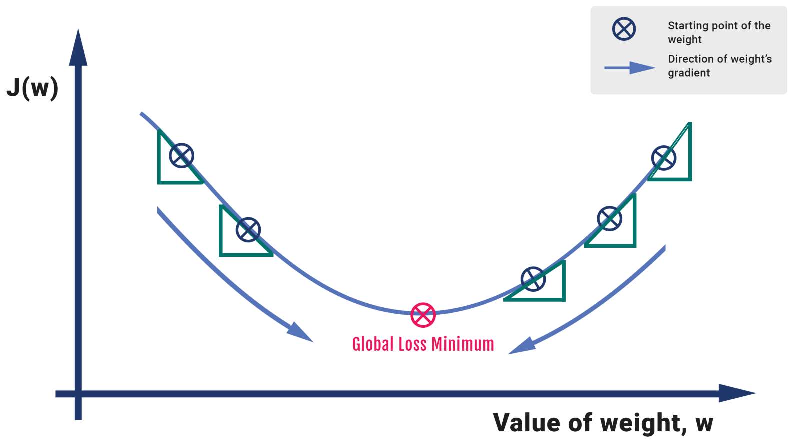


Figure 2.9: Illustration of Gradient Descent (where J is the error) (*).

The gradient of E is calculated through the chain rule. It is as the error propagates back from the outputs to the inputs, hence, the name *backpropagation*. As an example, the weights in the second layer are updated via :

$$\Delta W_{j1}^{(2)} = -\eta \sum_{a=1}^{N} \frac{\partial E_a}{\partial W_{j1}^{(2)}} = -\eta \sum_{a=1}^{N} (y_a - y_a^*) Z_{ja} \tag{2.15}$$

where z_{ja} is the output of the $j-th$ hidden neuron for event x_a. One more step back, and the update for the weights in the first layer is :

$$\Delta W_{ij}^{(1)} = -\eta \sum_{a=1}^{N} \frac{\partial E_a}{\partial W_{j1}^{(2)}} = -\eta \sum_{a=1}^{N} (y_a - y_a^*) W_{j1}^{(2)} Z_{ja} (1 - Z_{ja}) x_{ia} \tag{2.16}$$

With all the discussed steps of backpropagation, we will be able to minimize the error to the smallest value, given by that a very good classification which is similar to the actual classification data 1 for the *Higgs* signal and 0 for the background events.

2.4.4 ANN overtraining

The major challenge in training neural networks is how long to train them in order to obtain the best output possible after ending the training process. But the problem occurs when neural networks are too little trained. This results in a model that underfits the train and the test sets. Conversely, too much training will mean that the model overfit the training dataset and have poor performance on the dataset. Thus, having too many iterations may cause too much variance on training data. Fortunately, there is a widely used method to avoid this problem called "*early stopping*". The strategy behind this technique is simple. We need to split the training data from the validation dataset, this validation data is unseen data in the model training, and by checking accuracy against the validation dataset on every epoch. If train accuracy goes higher and the validation accuracy goes lower in this point we stop training in order to avoid the overfitting, as is illustrated in the figure [2.10] [31].

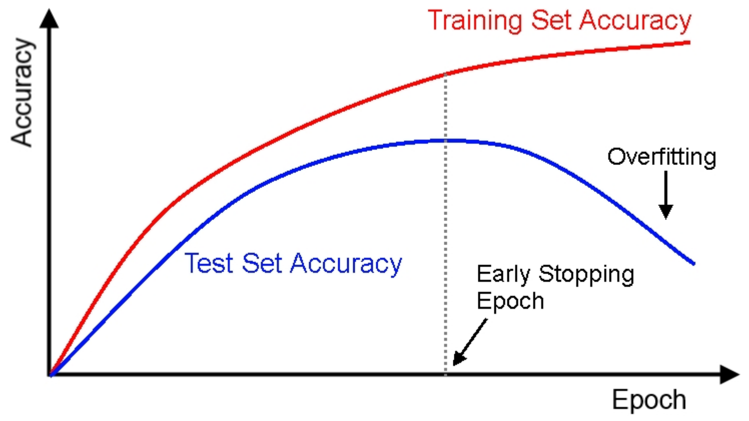


Figure 2.10: Illustration of neural network's early stopping (the blue curve represents vallidation dataset) (*).

2.5 Boosted Decision Tree

The boosting algorithm is one of the most powerful learning techniques for optimization, implemented in the software package TMVA, introduced during the past decade. As a tree it is high and it has branches and leaves. The decision tree is similar, it has branches and leaves. Decision tree is a generalization of rectangular cuts boosted by several algorithm techniques. The boosting algorithm is a procedure that combines many weak classifiers to achieve a final powerful classifier. [32], [33]

2.5.1 Working principle of a Decision Tree

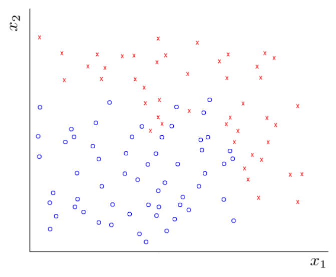
A decision tree works by repeatedly operating cuts into the dataset collected. Hence, separating it into smaller subsets that exhibit a better class separation than the original dataset. It is a supervised learning algorithm. At the end of the training process, the features space (dataset) is divided into smaller blocks, and previously unseen data are classified according to the block they fall in.

In a simple example with only two input features, x_1 and x_2, the data can be represented in a scatter plot as illustrated in Figure [2.11a]. The decision tree applies univariate cuts to the data. These cuts have the general expression.

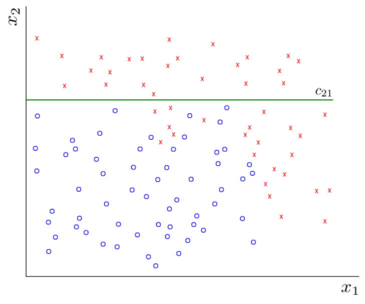
$$x_i - c_{ij} = 0 \tag{2.17}$$

Where x_i denotes the i-th feature and c_{ij} the j-th cut on x_i. In our two dimensional example, the applied cut is a straight line parallel to one of the axes, and at the first iteration the dataset is split into two subsets. In a simple decision tree with one node, for example with a single cut on x_2, the simulation might look like figue [2.11b], as we see the data points with x_2 larger than the cut value all belong to the same class, therefore, no more cuts are needed. For the other subset, the one that falls below c_{21}, the data is still heavily mixed between the two classes. Applying another cut on x_1, as illustrated in figure [2.11c] can significantly improve the separation.

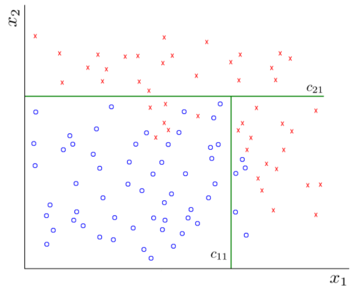
After these cuts, the resulting subsets are not mixed as before. In order to reach optimal separation on the data, one could continue cutting until each region only contains data belonging to the same class, but is not always desirable because it can lead to overtraining the model.



(a) Scatter plot of two class data in a two-dimention.



(b) The first cut on the dataset.



(c) The second cut on the subset.

Figure 2.11: Illustration of Boosted Decision Tree cuts (*).

2.5.2 Training and Optimisation Strategy

In our data for the process $pp \longrightarrow H \longrightarrow ZZ^{*} \longrightarrow 4lep$ the training monte-carlo samples, it consists of both *Higgs* signal that we want to optimize and background events. For each event, supposedly there is a set of PID (particle identification) variables or what is often called *discriminating variables*, useful for distinguishing between signal and background like (transverse momentum P_T, pseudo rapiditiy η, radial position ΔR, energy E_T^{miss}, hit multiplicity, and others......). Decision tree algorithms try to find the best cut on one of the discriminating variables that splits the phase space into two parts, one of which is enriched in signal and the other one in background. The algorithm loops over all discriminating variables and for each of them tests all possible cut values. Initially there is a sample of events at a "*root node*". Now there are two samples called "branches", for each branch we repeat the same splitting process until a given number of final pure branches, called leaves are obtained, or until each leaf is pure signal or pure background like is illustrated in figure [2.12].

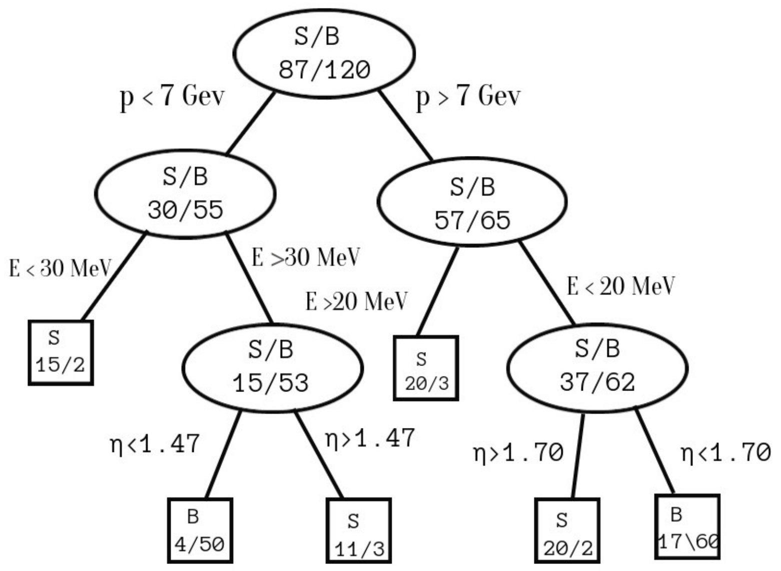


Figure 2.12: Schematic illustration of a decision tree (S for signal, B for background). Terminal nodes called leaves are shown in boxes. If signal events are dominant in one leaf, then this leaf is a signal leaf, otherwise, it is background leaf.

In figure [2.12], 3 discriminating variables are used for signal and background separation : Transverse momentum P_T, Energy E, and Pseudo rapidity η. [34]

The best splitting is defined on the basics of what is called "splitting index" as :

• **Gini index (GI)** : $Gini = (\sum_i^n W_i)P(1-P)$ where n is the number of events in Monte-Carlo sample and P is the purity.

• **Cross entopy** : $-Plog(P) - (1-P)log(1-P)$.

• **Miss classification error** : $1 - max(P, 1-P)$.

Every splitting index has to be a good measure of inequality because we want to measure the inequality of background populations in each block of phase space. We define the purity of the sample in a branch by $p = \frac{\sum_s W_s}{\sum_s W_s + \sum_b W_b}$ where $\sum_s$ is the sum over signal events, and $\sum_b$ is the sum over background events.

Let's pick the common one, "Gini index", the first split that decision tree start with it is the split that has the minimum value of Gini index, because the lower value of GI means high information gain (GI) with :

$$IG = 1 - \sum_i p_i GI \tag{2.18}$$

At the end, if a leaf has purity greater than $\frac{1}{2}$ then it is called a signal leaf, and if the purity is less than $\frac{1}{2}$ it is a background leaf. Events are classified as signals if they land on signal leaf and background if they land on background leaf. The resulting tree is a Decision Tree.

2.5.3 BDT overtraining

Decision tree would be highly overtained. There are many blocks containing just one event, as is mentioned above they are called leaves. To avoid splitting leaves, we need to define some stopping criterions that save a leaf block from split, the common criteria are :

• Require signal purity in the block: for example, if there are events just from one type.

• Minimum number of events in the block: for example, the block might be required to contain at least 5% of the total sample and this is what is called leaf node size in TMVA.

• Maximum number of cuts in the sequence defining the block: for example, no more than three branching are allowed, and this is called tree depth in TMVA.

But unfortunately the decision tree is very sensitive to statistical fluctuations in the training monte-carlo sample. If two discriminant variables have similar performance somewhere at the top of the tree, a statistical fluctuation might lead to a usage of one or the other. From that point however, the tree trained on the fluctuated sample will evolved completely different. As a result, its classification performance will be very weak, which leads to very weak classifiers. For solving this problem and increasing the efficiency of the tree, we need what is called boosting and AdaBoost techniques described below.

2.5.4 Combining multiple classifiers (boosting or bagging algorithm)

The root node contains a dataset. Boosting algorithm creates basic learners sequentially. Initially, a bunch or a subset from the data sample records (signal and background events) is picked at random form, then gets passed the first base learner that can be any model.

Typically, the base learner is a tree with one depth (one depth means the root block and two son blocks). Once it is trained, the algorithm checks where the incorrectly classified events are in the base learner and puts it in the second base learner with a new subset from the data, which is created sequentially in order to correct the incorrectly classified events. This operation is called *sampling with replacement*. What happens in the second base learner is also to check the incorrect classified events and put them in the third base learner. This is generally how boosting techniques work for enhancing the power of the decision tree, and the complet process is called "*Boosted Decision Tree*" [35]. If there is a total N_l base-learners the classification of a previously unseen data (signal and background events) is the average of the responses of the single classifiers :

$$y(x) = \frac{1}{N_l} \sum_{i=1}^{N_l} y_i(x) \tag{2.19}$$

where $y_i(x)$ is the output of the $i-th$ classifier and x is the average of the responses of the single classifiers. The score in some BDT leaves block is very high, so to get a binary answer (+1 for signal, -1 for background) in this case, it is required to normalize the BDT output score using the following formula :

$$\frac{N_B - N_S}{N_B + N_S} \tag{2.20}$$

For boosted decision tree :

$$y_i(x) = \begin{cases} +1 & \text{for signal} \\ -1 & \text{for background} \end{cases} \tag{2.21}$$

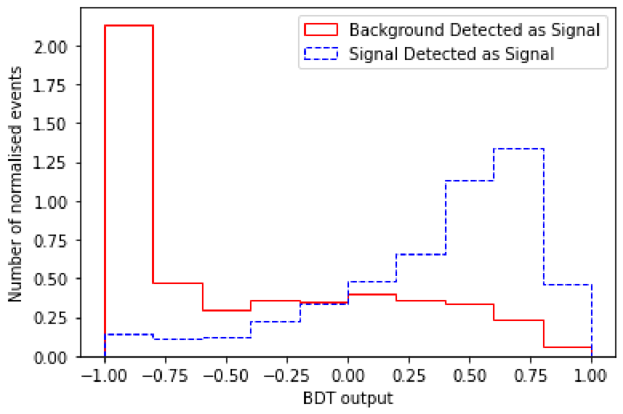


Figure 2.13: BDT score (response) distribution for an exemplary BDT training and testing.

The overall outputs or what is called the score of the Boosted decision tree will be close to +1 for signal events, and to -1 for background events. This process of averaging the response of multiple classifiers is known as voting. The better the classification, the more the background and *Higgs* signal distribution are separated as is illustrated in figure [2.13].

2.5.5 Adaboost algorithm

Adaboost is not much different than how boosting techniques work. However, there is something new in *Adaboost*: sample weight. Each event in the dataset has a weight value. Initially all events are basically assigned the same weights $\omega = \frac{1}{n}$ where n is the number of events recorded in the dataset, this is the first step. In the second step, we create sequentially a set of base learners starting from the first one. All these base learners are decision trees with one depth and we call each base learner a "*stump*". After the training, the first stump is classified correctly, some records form the first bunch of data set with some incorrectly classified events because it is a weak learner model. In this case we calculate the total error (TE) with :

$$TE = \frac{Number(incorrectly\_classified)}{Number(total\_Events)} \tag{2.22}$$

The third step is that we try to find the performance of the stump (PS), and that basically means how the stump has classified the $Higgs\ signal$ or $background$ events. In order to do that, we have the following formula :

$$PS = \frac{1}{2} log \frac{1 - Total\_Error}{Total\_Error} \tag{2.23}$$

We calculate the Total Error and the performance of the stump in order to update the weights of the incorrectly classified events, because only those wrongly classified events will be passed to the next stump. The fourth step is to increase the weights for the incorrect classified events so that the second stump take it in consideration and inversely decrease the weights for the correct classified events by using the simple formulas :

$$\begin{cases} \omega_{new} = \omega \exp^{performance} & \text{for incorrect classified events} \\ \omega_{new} = \omega \exp^{-performance} & \text{for correct classified events} \end{cases} \tag{2.24}$$

After applying the updating formula, we divide each single updated weight by the sum of all updated wights in order to normalize them.

In the next step, we will create a new bunch dataset for the second stump with the incorrect classified events from the first stump. Similarly, we apply all the previous steps on the second stump by updating the event weights for the incorrect classified events in the second stump, that will go on the third stump with another new bunch of data. This will be going continuously until we specify some numbers of stumps (in Figure [2.14]). In the boosting algorithm the score of this process will be the averaging response of the multiple weak classifiers stumps and is known as voting with +1 for *Higgs* signal and -1 for background.

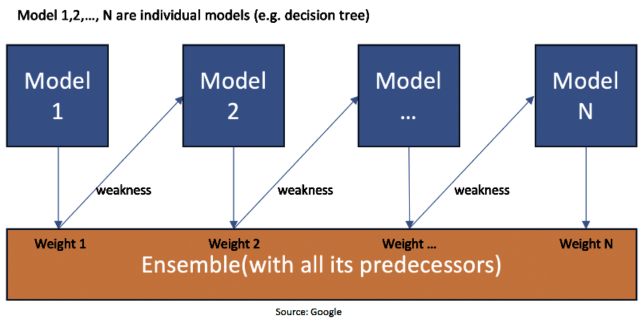


Figure 2.14: Illustration of Adaboost algorithm (Model means stump with depth 1) (*).

Chapter 3

Enhancing Higgs signal in ATLAS opendata

3.1 ATLAS opendata visualization and analysis

ATLAS Opendata [http://opendata.atlas.cern/] provides the public with data from the proton-proton collisions at the LHC (experimental data). The data is accompanied by MC simulated samples describing both several Standard Model processes and hypothetical Beyond Standard Model signal production for educational purposes. Associated computing tools are also provided to make the analysis of the dataset accessible. The general aim of releasing the datasets and the tools is to provide the user-friendly and straightforward interactive interface to replicate the procedure used by high energy physics researchers, and enable users to experience the analysis of particle physics data in an educational environment. The released samples are provided in a simplified data format, that contains just 4lep final state. The resulting format is a ROOT tuple with more than 80 branches.

3.1.1 Experimental sample

The experimental data is taken from the pp collision process at the LHC, and collected by the ATLAS detector at $\sqrt{s} = 13$ TeV during 2016, after applying quality criteria on the beam of the protons, the detector and the data. The publicly released data corresponds to an integrated luminosity of 10 fb^{-1}, contains approximately 270 million collision events [36], and it is of a manageable size with just 4 final state of leptons (4 Electrons or 4 Muons or 2 Electrons and 2 Muons). Experimental data doesn't have many events, because as we know the process $H \longrightarrow 4lep$ is very rare, so just a few hundred are detected and stored for further analysis.

The experimental sample contains the following observables:

• Transverse momentum with (MeV) unit for each lepton p_T (*lep_pt*)

• The pseudorapidity for each lepton η (*lep_eta*)

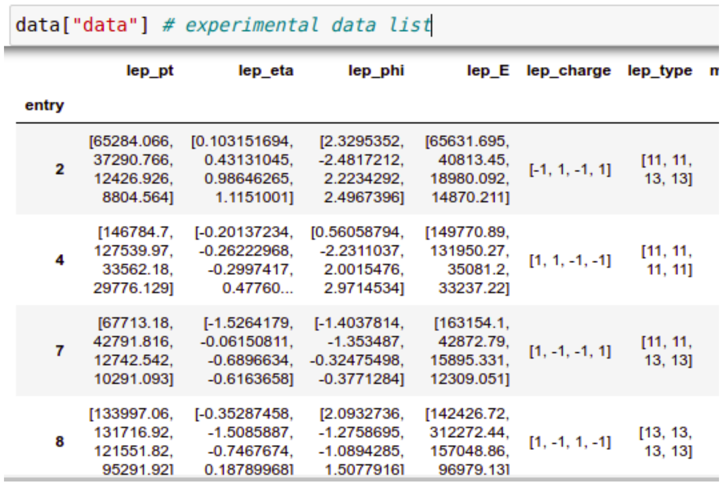
• The azimuthal angle for each lepton ϕ (*lep_phi*)

• The energy of the lepton E (*lep_E*)

• The type of each lepton (*lep_type*) (11 for Electron and 13 for Muon, this convention is taken by CERN collaboration)

• The charge of each lepton (*lep_charge*)

Each line from the list represents an event (Higgs signal or backgrounds) for the process $pp \longrightarrow H \longrightarrow ZZ^* \longrightarrow 4lep$, as shown in the figure[3.1] below :



```
data["data"] # experimental data list
```

entry	lep_pt	lep_eta	lep_phi	lep_E	lep_charge	lep_type	m
2	[65284.066, 37290.766, 12426.926, 8804.564]	[0.103151694, 0.43131045, 0.98646265, 1.1151001]	[2.3295352, -2.4817212, 2.2234292, 2.4967396]	[65631.695, 40813.45, 18980.092, 14870.211]	[-1, 1, -1, 1]	[11, 11, 13, 13]	
4	[146784.7, 127539.97, 33562.18, 29776.129]	[-0.20137234, -0.26222968, -0.2997417, 0.47760...	[0.56058794, -2.2311037, 2.0015476, 2.9714534]	[149770.89, 131950.27, 35081.2, 33237.22]	[1, 1, -1, -1]	[11, 11, 11, 11]	
7	[67713.18, 42791.816, 12742.542, 10291.093]	[-1.5264179, -0.06150811, -0.6896634, -0.6163658]	[-1.4037814, -1.353487, -0.32475498, -0.3771284]	[163154.1, 42872.79, 15895.331, 12309.051]	[1, -1, -1, 1]	[11, 11, 13, 13]	
8	[133997.06, 131716.92, 121551.82, 95291.92]	[-0.35287458, -1.5085887, -0.7467674, 0.18789968]	[2.0932736, -1.2758695, -1.0894285, 1.5077916]	[142426.72, 312272.44, 157048.86, 96979.13]	[1, -1, 1, -1]	[13, 13, 13, 13]	

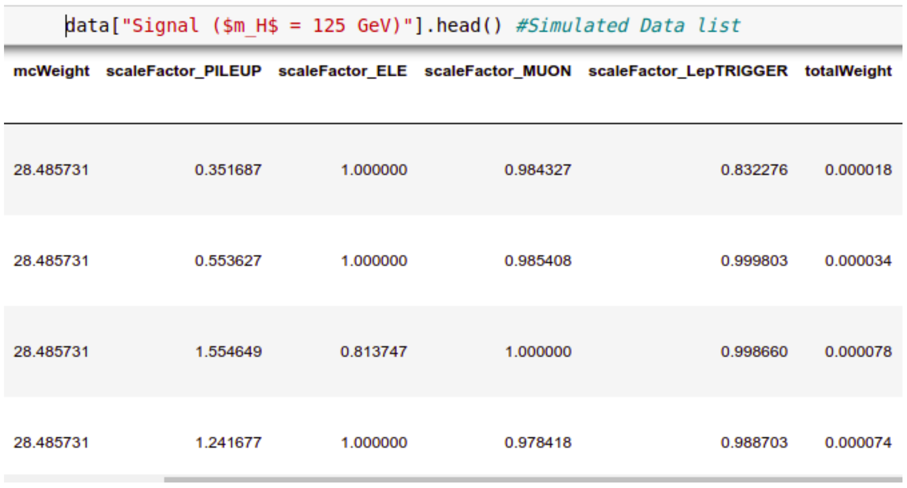
Figure 3.1: Experimental data provided by ATLAS opendata visualized with jupyter notebook.

3.1.2 Simulated sample

The Monte-Carlo simulated samples from ATLAS describe several SM processes, which are used to model the expected distributions of different signal and background events. For the process $pp \longrightarrow H \longrightarrow ZZ^* \longrightarrow 4lep$, the simulated data are generated using PYTHIA 8 + POEHEG-BOX V2 or Madgraph aMC@NLO + PYTHIA 8 with 343981 channel that gives 4lep final state. The simulated dataset contains:

- Monte-carlo Weight (mc_Weight)
- Scale factor for pileup ($scaleFactor\_PILEUP$)
- Scale factor electron efficiency ($scaleFactor\_ELE$)
- Scale factor muon efficiency ($scalFactor\_MUON$)
- Scale factor lepton Trigger efficiency ($scaleFactor\_LepTRIGGER$)

When visualizing the MC dataset by jupyter notebook, we obtain the results shown in the figure [3.2]:



```
data["Signal ($m_H$ = 125 GeV)"].head() #Simulated Data list
```

mcWeight	scaleFactor_PILEUP	scaleFactor_ELE	scaleFactor_MUON	scaleFactor_LepTRIGGER	totalWeight
28.485731	0.351687	1.000000	0.984327	0.832276	0.000018
28.485731	0.553627	1.000000	0.985408	0.999803	0.000034
28.485731	1.554649	0.813747	1.000000	0.998660	0.000078
28.485731	1.241677	1.000000	0.978418	0.988703	0.000074

Figure 3.2: Simulated ATLAS data features visualized with jupyter notebook.

3.2 Enhancing Higgs signal using ATLAS opendata

The data source provided by ATLAS collaboration is accompanied by a set of jupyter notebook documents, they can be analysed directly from the browser, either online or locally by downloading the dataset from the CERN website. In our thesis, we work locally with python notebooks codes applying Machine Learning Models in the search to enhance the *Higgs* boson.

We are dealing with two kinds of machine learning models, “Artificial Neural Network” (ANN) and “Boosted Decision Tree” (BDT). We are interested in the machine learning models because they have the ability to do many analyses with high quality, saving countless human-hours of design and analysis work, and they are of a great help with our problem : “the *Higgs* signal and background noise classification”.

In this work, we add a lot of improvements to the original jupyter notebooks provided by CERN and introduce several cuts to the dataset. We then study the impact of the parameters on our machine learning models. By doing so, we aim to get a good separation between the *Higgs* signal (S) and background (B), therefore “enhance the *Higgs* boson signal”.

The python libraries and tools we use in this work are :

• Uproot : a reader and a writer of the ROOT file format, typically used in particle physics.

• pandas : a software library for storing data as dataframes, a format widely used in Machine Learning.

• NumPy : a library that provides numerical calculations such as histogramming.

• Matplotlib : a plotting library, common for making plots, figures, images, visualisations.

• Scikit-learn : a software machine learning library, easy and efficient tools for predictive data analysis with machine learning model.

3.2.1 Optimization with machine learning methods

Throughout our work, we are using machine learning methods (ANN and BDT) as efficient tools that allow us to *enhance the Higgs signal* from the monte-carlo simulated dataset. In order to achieve that, we apply various changes on the codes to find the region in the multi-dimensional phase space where the *Higgs* signal is more dominant than the background noise, and get the maximum signal over background ratio. We also add the experimental dataset to the ML models to see their results. We are also going to illustrate why those methods are convenient and powerful than the traditional method that is based on data cuts.

3.2.2 Traditional cuts and optimization

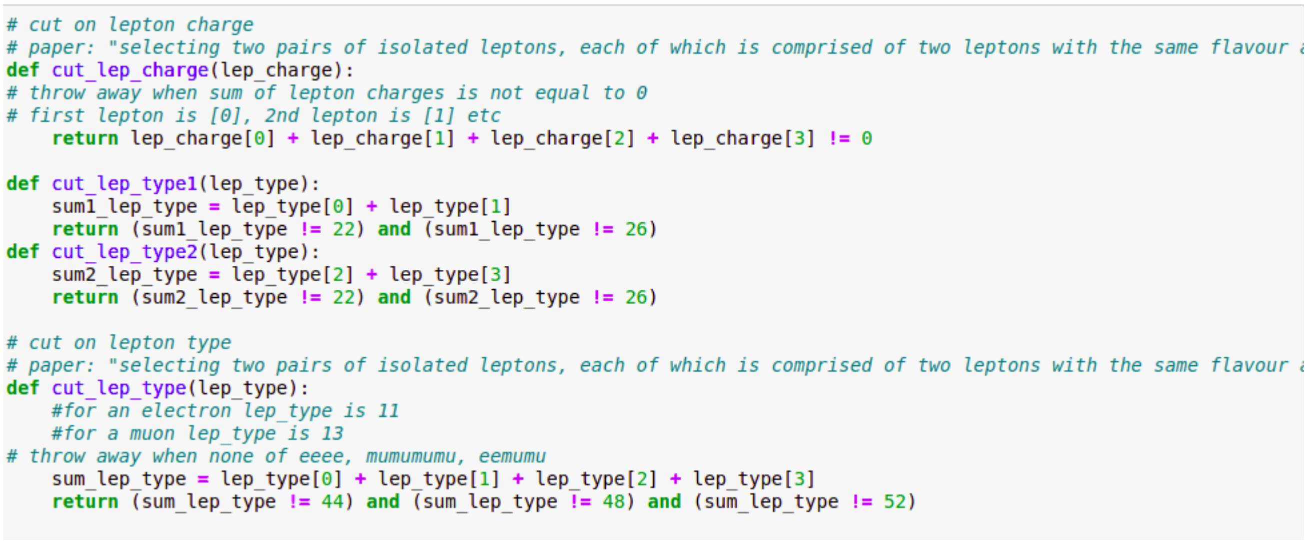
Beside the original code, we introduce a number of numerical cuts just on *lep_pt_1* and *lep_pt_2* (cuts on leptons transverse momentum, pseudo-rapidity, and charge) in the data that contains the *Higgs* signal and the background events, in order to reduce the contribution of the background. Those cuts are as following:

• Selecting two pairs of isolated leptons, each of which is composed of two leptons with the same flavour and opposite charge.

• Each Electron must satisfy $p_T > 7000$ MeV and be measured in the pseudorapidity range $|\eta| < 2.47$.

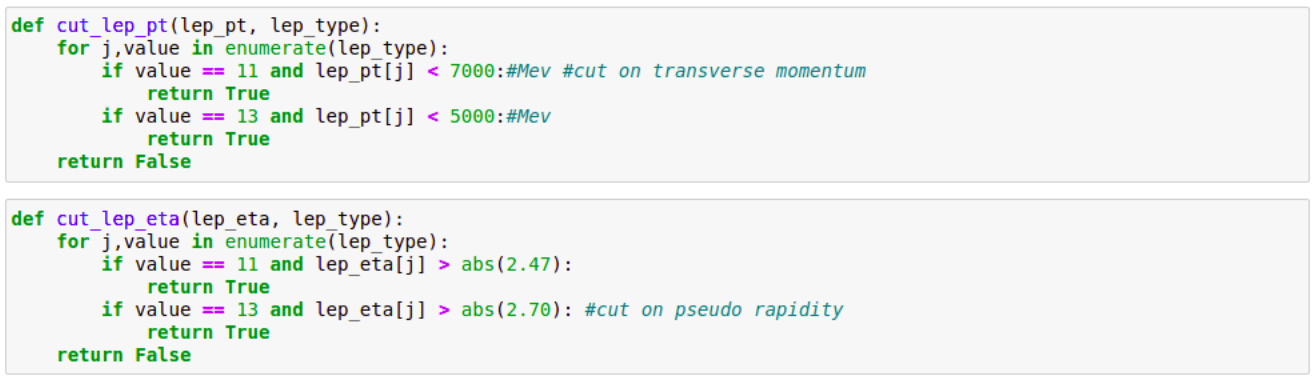
• Each Muon must satisfy $p_T > 6000$ MeV and be measured in the pseudorapidity range $|\eta| < 2.7$.

The additional cuts are made by adding a new cells in jupyter notebook as is illustrated in the figures [3.3] and [3.4]:



```
# cut on lepton charge
# paper: "selecting two pairs of isolated leptons, each of which is comprised of two leptons with the same flavour
def cut_lep_charge(lep_charge):
# throw away when sum of lepton charges is not equal to 0
# first lepton is [0], 2nd lepton is [1] etc
    return lep_charge[0] + lep_charge[1] + lep_charge[2] + lep_charge[3] != 0

def cut_lep_type1(lep_type):
    sum1_lep_type = lep_type[0] + lep_type[1]
    return (sum1_lep_type != 22) and (sum1_lep_type != 26)
def cut_lep_type2(lep_type):
    sum2_lep_type = lep_type[2] + lep_type[3]
    return (sum2_lep_type != 22) and (sum2_lep_type != 26)

# cut on lepton type
# paper: "selecting two pairs of isolated leptons, each of which is comprised of two leptons with the same flavour
def cut_lep_type(lep_type):
    #for an electron lep_type is 11
    #for a muon lep_type is 13
# throw away when none of eeee, mumumumu, eemumu
    sum_lep_type = lep_type[0] + lep_type[1] + lep_type[2] + lep_type[3]
    return (sum_lep_type != 44) and (sum_lep_type != 48) and (sum_lep_type != 52)
```

Figure 3.3: Cells from Jupyter Notebook showing the applied cuts on two isolated pairs of leptons with the same flavour and opposite charge (11 for electron and 13 for muon).



```
def cut_lep_pt(lep_pt, lep_type):
    for j,value in enumerate(lep_type):
        if value == 11 and lep_pt[j] < 7000:#Mev #cut on transverse momentum
            return True
        if value == 13 and lep_pt[j] < 5000:#Mev
            return True
    return False
```

```
def cut_lep_eta(lep_eta, lep_type):
    for j,value in enumerate(lep_type):
        if value == 11 and lep_eta[j] > abs(2.47):
            return True
        if value == 13 and lep_eta[j] > abs(2.70): #cut on pseudo rapidity
            return True
    return False
```

Figure 3.4: Cells from Jupyter Notebook showing the applied cuts on leptons lep_pt_1 and leptons lep_eta (11 for electron and 13 for muon).

In the next step which is the cuts optimization, we noticed that there is a difference between the results of the original code and our modified code in the S/B ratio. After adding a number of cuts on the data the contribution of the background is reduced which gives a better S/B ratio as is illustrated in the figures [3.7]. Although adding cuts can boost the $Higgs$ signal a bit but it is insufficient.

Let's set a cut at 8 GeV in the distributions of signal and background (1st plot) [3.5], this means keeping all events above 8 GeV in the signal and background histograms, we then take the ratio of signal and background events that pass this cut. This gives us a starting value for S/B ratio (2nd plot) [3.6].

We then increase the value of cuts from 8 GeV to 13 GeV, whereat these values the cuts are throwing away more background than signal, therefore the S/B ratio increases, but around 14 GeV a point comes where we start throwing away too much signal, thus, S/B ratio starts to decrease.

Imagine having to separately optimize about 4 variables, since applying a cut on one variable could change the distribution of another, which means being obligated to re-optimise, this is where Machine Learning algorithms (ANN and BDT detailed [2]) come in handy. Machine learning models can optimise all variables at the same time what gives a better signal over background classification than the individual cuts ever could.

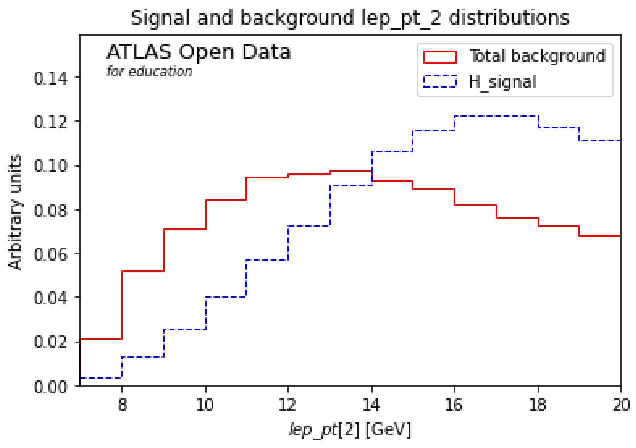


Figure 3.5: S and B *lep_pt_2* distribution (1st plot) after introducing cuts on *lep_2*.

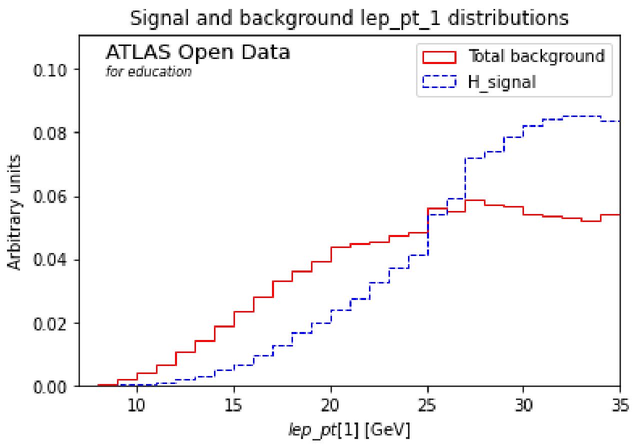
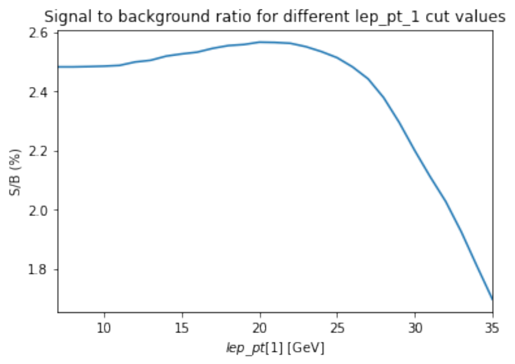
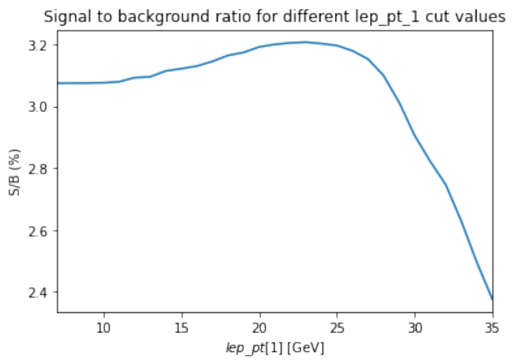


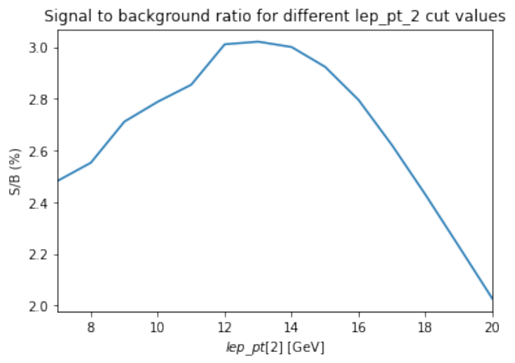
Figure 3.6: S and B *lep_pt_1* distribution (2st plot) after introducing cuts on *lep_1*.



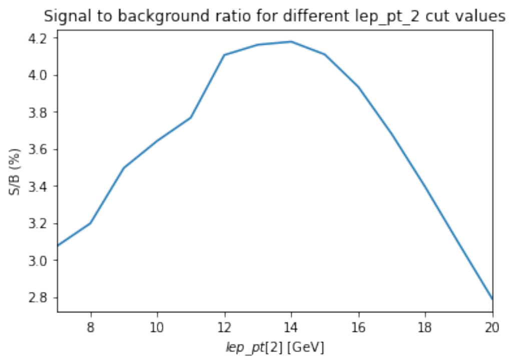
(a) S/B ratio before introducing cuts on $lep\_P_T\_1$ $(P_T > 7GeV)$.



(b) S/B ratio after introducing cuts on $lep\_P_T\_1$ $(P_T > 7GeV)$.



(c) S/B ratio before introducing cuts on $lep\_P_T\_2$ $(P_T > 7GeV)$.



(d) S/B ratio after introducing cuts on $lep\_P_T\_2$ $(P_T > 7GeV)$.

Figure 3.7: S/B ratio results after applying cuts on 3.3 and 3.4 variables.

3.2.3 Covariance and Correlation

Correlation is a dimensionless measure used to represent how strong two random variables are related to each other. It refers to the scaled form of *covariance* and measures both the strength and direction of the linear relationship between two variables. *Correlation* ranges between -1 and $+1$ and it is not influenced by the change in scale.

Since the data collected for the process $pp \longrightarrow H \longrightarrow ZZ^* \longrightarrow 4lep$ contains more than two observables $(x_1, x_2, x_3, x_4......, x_n)$, we can form the *covariance* between those variables as follow:

$$V_{ij} = \langle x_i x_j \rangle - \langle x_i \rangle \langle x_j \rangle \tag{3.1}$$

If there is a tendency for positive fluctuations in x_i to be associated with positive fluctuations in x_j (and therefore negative with negative) then the product $(x_i - \bar{x})(x_j - \bar{x})$ tends to be positive and the *covariance* is greater than 0.

If the variables are independent, then a positive variation in x_i is likely to be associated with a positive or a negative variation in x_j, in this case the *covariance* is zero.

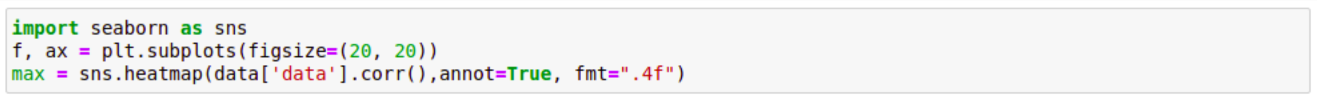
Covariance is useful, but it has dimensions. Often one uses the *correlation*, which is just :

$$\rho_{ij} = \frac{V_{ij}}{\sigma_i \sigma_j} \tag{3.2}$$

$$\rho_{ii} = 1 \ for \text{ i=j} \tag{3.3}$$

It is easy to show that ρ lies between 1 (complete *correlation*) and -1 (complete *anticorrelation*), $\rho = 0$ corresponds to x_i and x_j independent.

We can show the heatmap by applying the following commands on jupyter notebook :

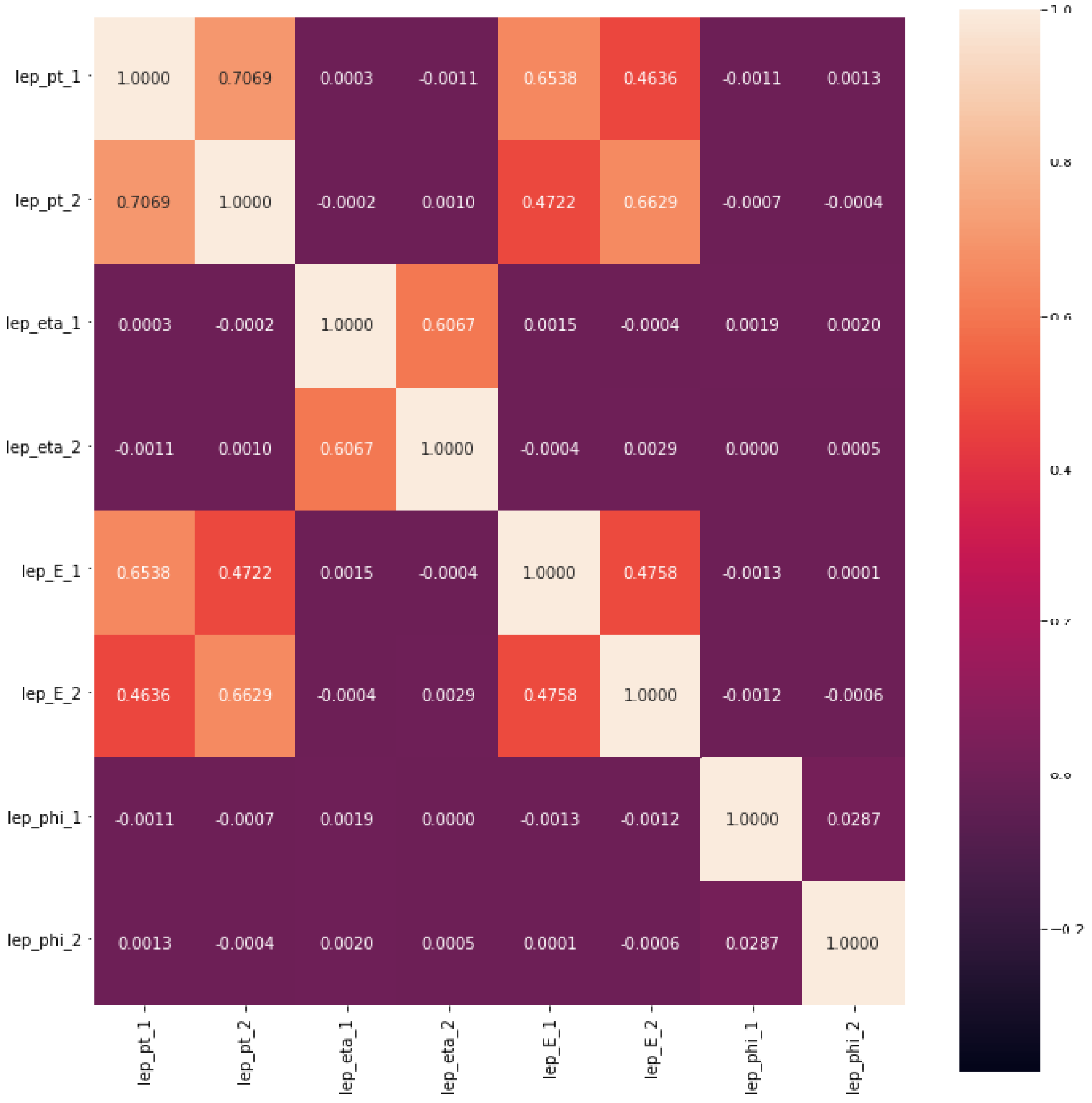


```
import seaborn as sns
f, ax = plt.subplots(figsize=(20, 20))
max = sns.heatmap(data['data'].corr(),annot=True, fmt=".4f")
```

Figure 3.8: Part from the code with commands to get heatmap that shows the correlation between experimental data.

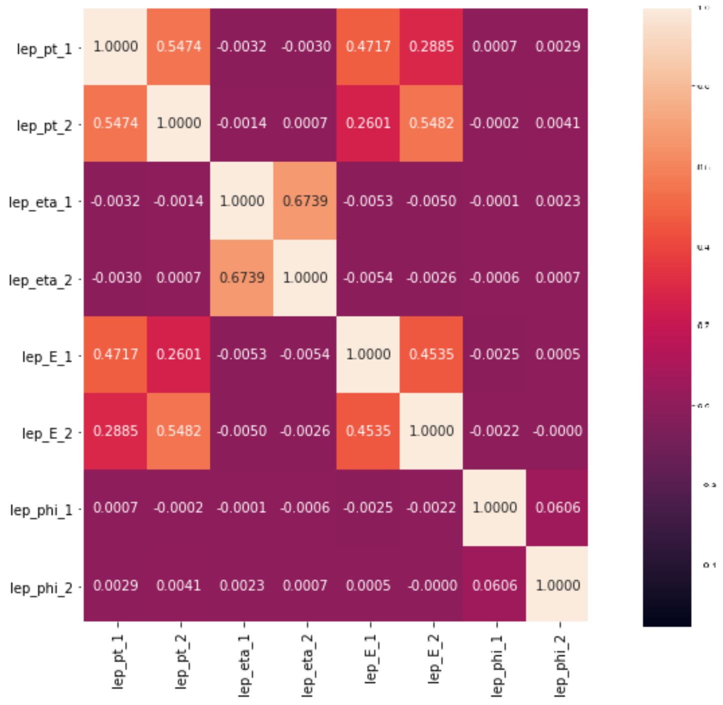
The diagonal of V is σ_i^2. The diagonal of ρ is 1 as it is shown in figure [3.10] for our collected data.

We can see that there is a strong *correlation* in the diagonal of the heatmap, what means each variable is correlated to itself. Also there are some variables that don't correlate very well, on the othere hand, there are data variables that don't correlate at all, those information on correlation between data variables can be used later in our work in order to choose the best input variables for best optimization that lead us to *enhance* the *Higgs* signal to background ratio.

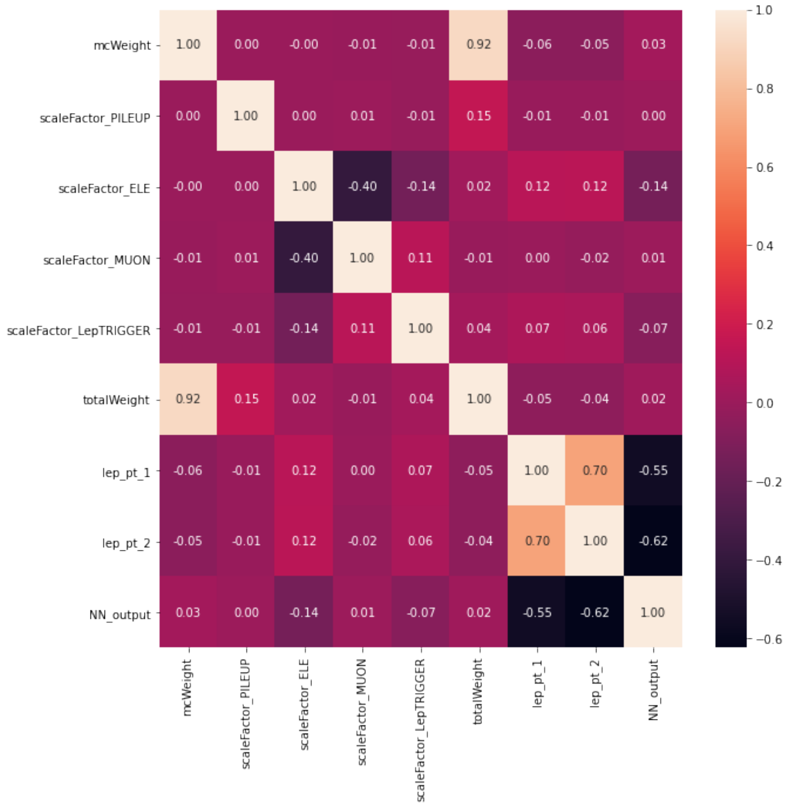


	lep_pt_1	lep_pt_2	lep_eta_1	lep_eta_2	lep_E_1	lep_E_2	lep_phi_1	lep_phi_2
lep_pt_1	1.0000	0.7069	0.0003	-0.0011	0.6538	0.4636	-0.0011	0.0013
lep_pt_2	0.7069	1.0000	-0.0002	0.0010	0.4722	0.6629	-0.0007	-0.0004
lep_eta_1	0.0003	-0.0002	1.0000	0.6067	0.0015	-0.0004	0.0019	0.0020
lep_eta_2	-0.0011	0.0010	0.6067	1.0000	-0.0004	0.0029	0.0000	0.0005
lep_E_1	0.6538	0.4722	0.0015	-0.0004	1.0000	0.4758	-0.0013	0.0001
lep_E_2	0.4636	0.6629	-0.0004	0.0029	0.4758	1.0000	-0.0012	-0.0006
lep_phi_1	-0.0011	-0.0007	0.0019	0.0000	-0.0013	-0.0012	1.0000	0.0287
lep_phi_2	0.0013	-0.0004	0.0020	0.0005	0.0001	-0.0006	0.0287	1.0000

Figure 3.9: *Correlation* between *experimental data* variables.



(a) *Correlation* between *Higgs signal* variables for simulated data.



(b) *Correlation* between *background* variables for simulated data.

Figure 3.10: Heatmaps show correlation factor for higgs events and background events.

Both correlation and covariance are closely related to each other and they still differ a lot:

Tableau 3.1: *Correlation* vs *covariance* characteristics

Covariance	Correlation
Indicates the change scale of random variables.	Indicates how strongly random variables are related.
Covariance is a measure of correlation.	Correlation refers to the scaled form of covariance.
Indicates the direction of linear relationship between variables.	Measures strength and direction of linear relationship between variables.
Varies between $-\infty$ and $+\infty$	Varies between -1 and +1
Affected by the change in scale.	Not affected by the change in scale.
Gets its units from the unit product of two variables.	Dimensionless.

3.2.4 Optimization with Artificial Neural Network

Since the traditional cuts don't give us maximum signal to background ratio, we choose artificial neural network as a modern alternative way that can take correlations between features for more information in multidimensional space data, and optimize all variables at the same time. Therefore, gives best signal and background classification which is more powerful than individual cuts (traditional method).

The Training phase We feed our ANN the parameters or "the kinematic variables" $lep\_P_T\_1$ and $lep\_P_T\_2$ both at the same time as inputs from the labeled simulated data (0 for background events and 1 for signal events) as training set. We use $MLPClassifier$ ($the\ multilayer\ perceptron\ classifier$) as our neural network model, since we have a classification between *Higgs* signal and background noise events, we will train our ANN model with 66% from the data, while the other 33% is preserved for the testing phase.

Inputs Data Preprocessing for Neural Network The multilayer perceptron is very sensitive to features scaling. The neural network in python may have difficulties to converge before the allowed maximum number of iterations if the data is not normalized. It is the case for our features for the ANN $lep\_P_T\_1$ and $lep\_P_T\_2$ that contain a high scale value. So we normalize our inputs with a tool from sklearn library called "$StandardScaler$" for scaling inputs to small values.

Effect of changing ANN parameters in optimization

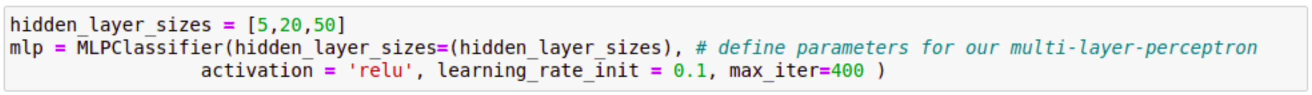
In this part we change a number of parameters in the code exactly in our neural network model $MLPClassifier$ respectively:

• First we change number of hidden layers from 1 with 2 neurons to 2 hidden layers (the first hidden layer with 2 neurons and the second hidden layer with 10 neurons), and we fix 200 *iteration* with a *learning rate* =0.001, we put "*sigmoid*" or what is called "*logistic*" as an activation function.

• In the second modification, we change the neural network parameter again but this time we create an ANN with 3 hidden layers (the first hidden layer contains 5 neurons, the second hidden layer contains 20 neurons, the third hidden layer contains 50 neurons), with 200 *iteration*, *learning rate* = 0.001 and "*Relu*" as activation function.

• For the third modification, we keep our ANN with three hidden layers but we change the *learning rate* to 0.1 and for the number of iteration we use this time 400 *iteration*.

The modifications we tried while experiencing the ANN model, helped us pick the best combinations of parameters to show the evolution of ANN model performance from the lowest to a higher efficiency.



```
hidden_layer_sizes = [5,20,50]
mlp = MLPClassifier(hidden_layer_sizes=(hidden_layer_sizes), # define parameters for our multi-layer-perceptron
                    activation = 'relu', learning_rate_init = 0.1, max_iter=400 )
```

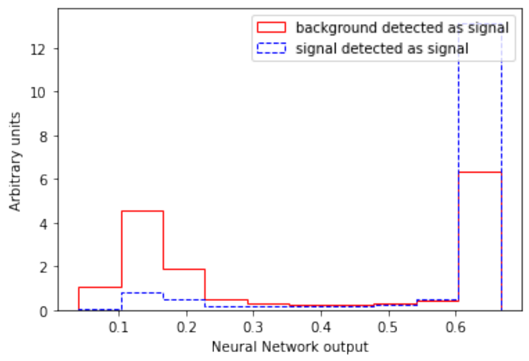
Figure 3.11: Cell from the code showing the change of parameters for the ANN.

After processing we will pay attention to three following illustrations representing the results

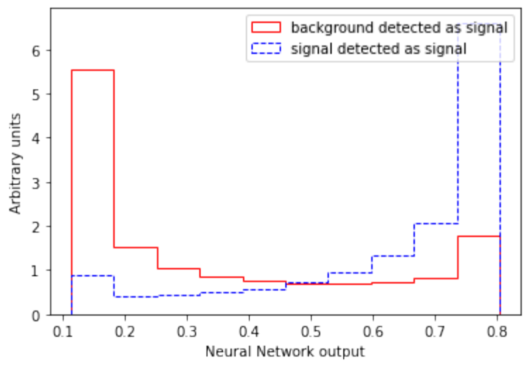
:

- The ANN outputs (separation of signal and background events).
- The ROC curve.
- The Higgs signal to background ratio.

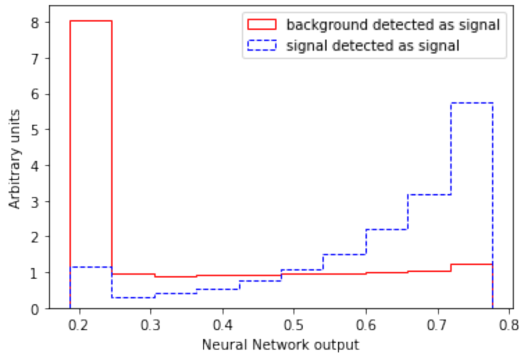
The results after training the neural networks is illustrated in the figures below:



(a) 2 hidden layers and *logistic* as activation function, *learning rate* =0.001, *max iteration* =200.



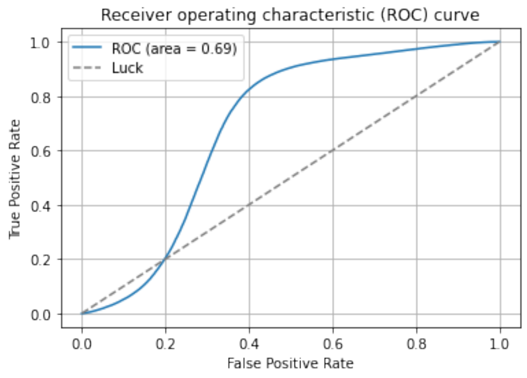
(b) 3 hidden layers and *relu* as activation function, *learning rate* =0.001, *max iteration* =200.



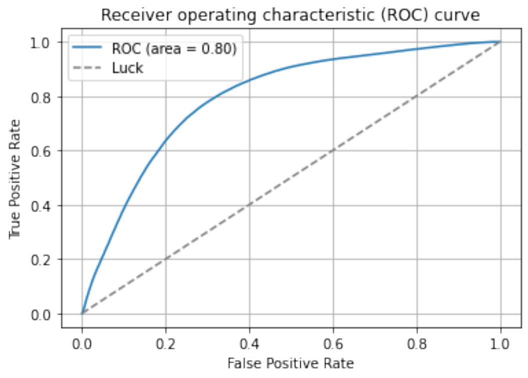
(c) 3 hidden layers and *relu* as activation function, *learning rate* 0.1, *max iteration* = 400.

Figure 3.12: Higgs signal and Background ANN output distribution with different parameters.

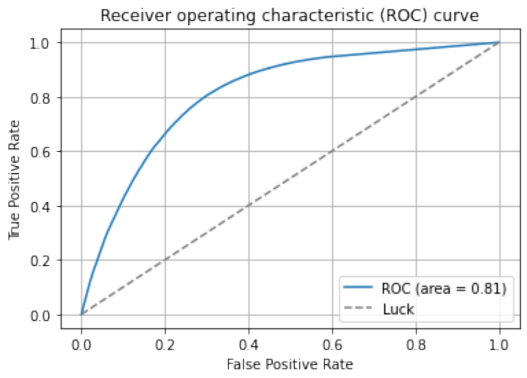
After analyzing the results of the ANN experiments with different hidden layers, we can clearly see that the ANN with 2 hidden layers (with *logistic* activation function, *learning rate* =0.001, *max iteration* =200.) has a very bad classification between the values 0.6 and 0.7 [3.12]. There is a huge number of background events (represented by arbitrary units) incorrectly classified as *Higgs signal* with a probability of 60% and 70%. Which is very bad as performance for our ANN model. When adding another hidden layers with more neurons to our ANN and changing the activation function from *logisctic* to *ReLU*, the number of background events classified as background decrease a little bit, and the separation between the *Higgs* signal and background events become apparent. Then, when keeping the number of hidden layers and the activation function while changing the learning rate and *max iteration*, we notice how the background detected as signal is reduced comparing to the previous tries, and the separation become even more apparent.



(a) 2 hidden layers and *logistic* as activation function, *max iteration* =200, *learning rate* =0.001.



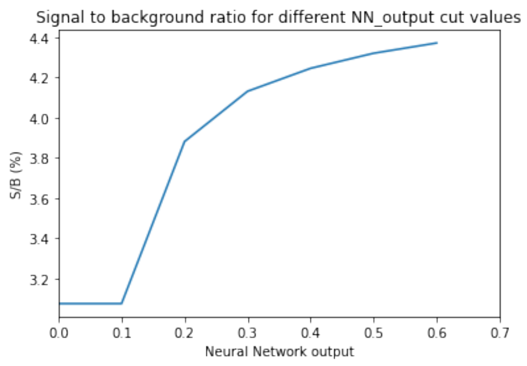
(b) 3 hidden layers and *relu* as activation function, *max iteration* =200, *learning rate* =0.001.



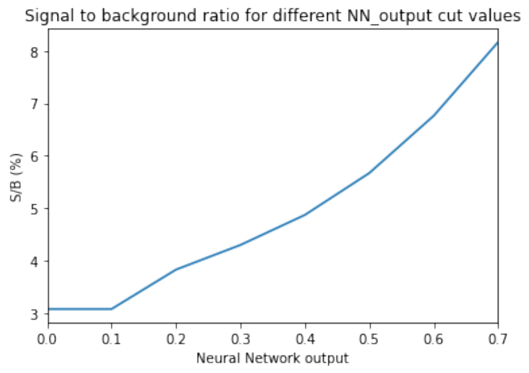
(c) 3 hidden layers and *relu* as activation function, *max iteration* = 400, *learning rate* 0.1.

Figure 3.13: Roc curve with different parameters.

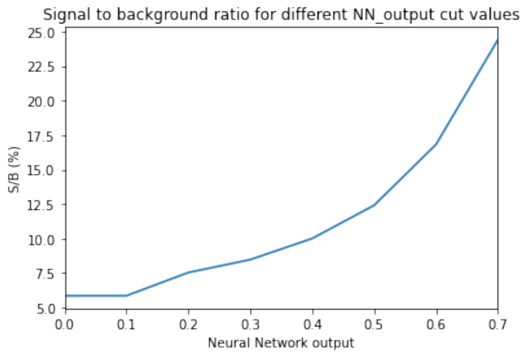
We can clearly see that the area under the curve [3.13] for the ANN with 3 hidden layers is bigger than the ANN with 2 hidden layers, which means that the model with 3 hidden layers is more capable of signal and background separation task than the 2 hidden layers model. The true positive rate is higher than the false positive rate which leads to high sensitivity to *Higgs* *signal* classification. Also, the same change in the results from the signal to the background ratio in curve [3.14] is noticeable. Adding more hidden layers and changing the activation function to *ReLU* gives high *S*/*B* ratio.



(a) 2 hidden layers and logistic as activation function, *learning rate* =0.001, *max iteration* =200.



(b) 3 hidden layers and relu as activation function, *max iteration* =200, *learning rate* =0.001.



(c) 3 hidden layers and *relu* as activation function, *max iteration* = 400, *learning rate* 0.1.

Figure 3.14: Higgs signal over background ratio for different parameters.

Checking Overtraining After finding the optimal architecture, we have to assure that our model does not face overtraining problems (discussed in [2.4.4]). Fortunately, the code has a prepared part for this, it uses the 33% of the data preserved for testing. What permit the illustration of figure [3.15], thus confirming that our ANN did not overtrain.

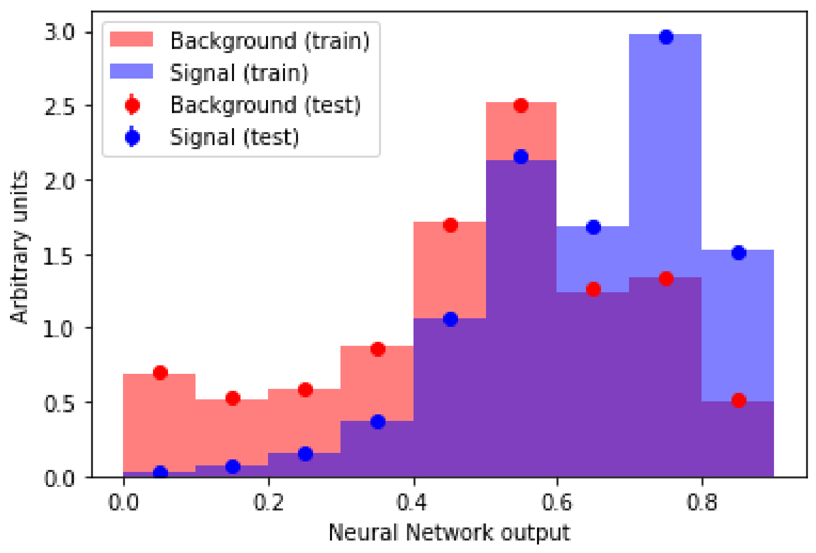
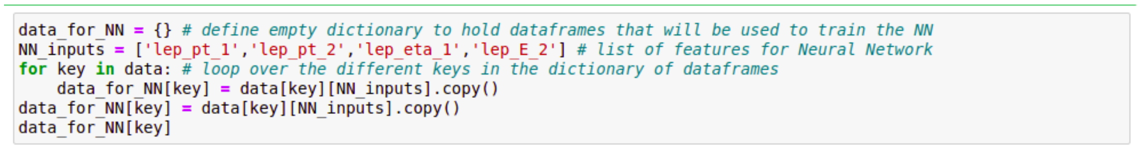


Figure 3.15: Artificial neural network overtraining check.

As it is clear in the Figure [3.15], the testing results represented by the dots match perfectly the training results represented by the histograms, thus, the ANN can preserve a good efficiency of classification when confronted with unknown data (testing data).

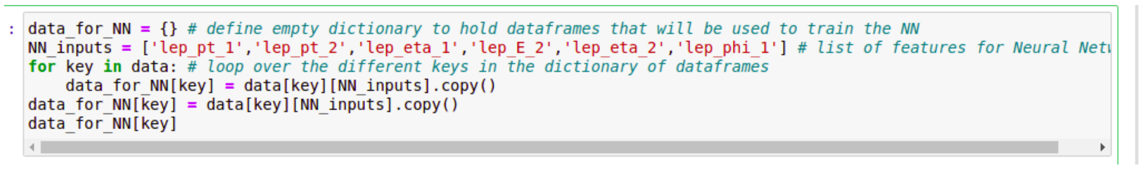
Effect of adding more variables into the neural network

In this part, we add a number of new dynamical variables (kinematic) in the neural network beside the *lep_pt_1* and *lep_pt_2* that have a correlation factor ρ=0.54 as inputs. First, we change the input variables to "*lep_pt_1*" and "*lep_eta_2*" that have a correlation factor of ρ=-0.003, then we change again the inputs to "*lep_eta_1*" and "*lep_E_2*" where ρ =-0.005 according to the heatmap [3.10], in order to see the effect of correlation and anticorrelation on our Artificial neural networks performance. As a final modification, we add respectively one by one the variables from the set: *lep_pt_1*, *lep_pt_2*, *lep_eta_1*, *lep_E_2*, *lep_eta_2*, and *lep_phi_1*, some of them are decorrelated and others are correlated. The modifications are illustrated in the next figures:



```
data_for_NN = {} # define empty dictionary to hold dataframes that will be used to train the NN
NN_inputs = ['lep_pt_1','lep_pt_2','lep_eta_1','lep_E_2'] # list of features for Neural Network
for key in data: # loop over the different keys in the dictionary of dataframes
    data_for_NN[key] = data[key][NN_inputs].copy()
data_for_NN[key] = data[key][NN_inputs].copy()
data_for_NN[key]
```

(a)

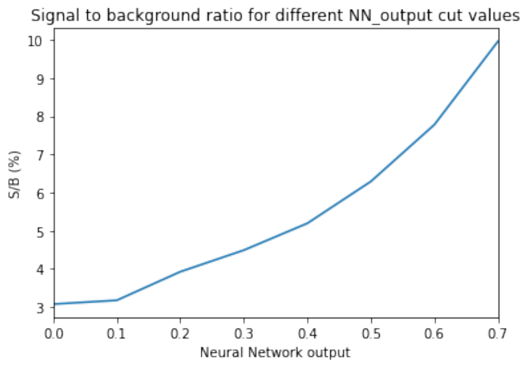


```
: data_for_NN = {} # define empty dictionary to hold dataframes that will be used to train the NN
NN_inputs = ['lep_pt_1','lep_pt_2','lep_eta_1','lep_E_2','lep_eta_2','lep_phi_1'] # list of features for Neural Netw
for key in data: # loop over the different keys in the dictionary of dataframes
    data_for_NN[key] = data[key][NN_inputs].copy()
data_for_NN[key] = data[key][NN_inputs].copy()
data_for_NN[key]
```

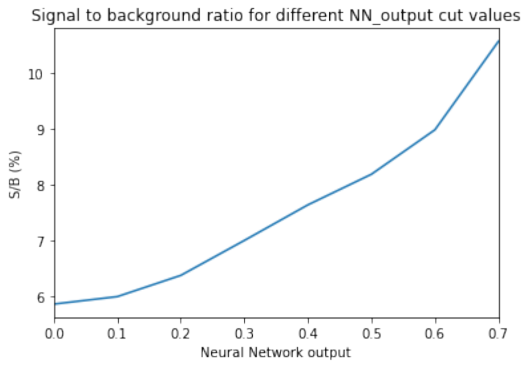
(b)

Figure 3.16: Cells from the code showing the modifications of adding different variables.

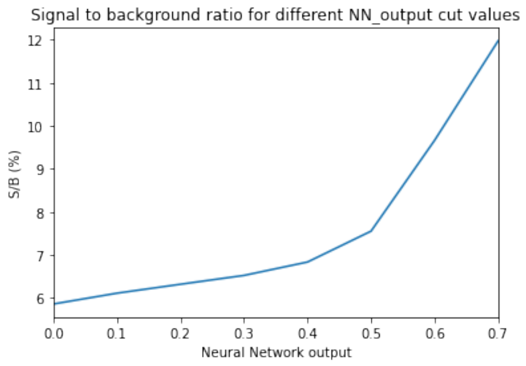
The results are represented as following:



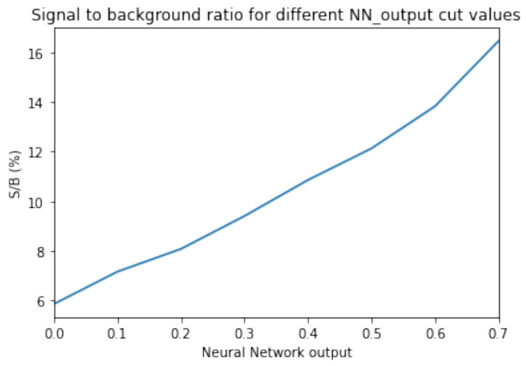
(a) *lep_pt_1,lep_pt_2* ρ=0.54.



(b) *lep_pt_1,lep_eta_2* ρ=-0.003.



(c) *lep_eta_1,lep_E_2* ρ=-0.005.



(d) *lep_pt_1,lep_pt_2,lep_eta_1,lep_E_2,lep_eta_2,lep_phi_1*.

Figure 3.17: Higgs signal over the background ratio after testing the correlated and decorrelated variables on the ANN model.

We can see that the optimization with decorrelated inputs variables gives better results for the signal to background ratio than the correlated variables. This case of decorrelation is more discussed in [37]. For this reason the heatmap is very important in machine learning models to get the maximum efficiency.

3.2.5 Optimization with Boosted Decision Tree

Another alternative optimization method is the "Boosted decision tree". Here, the model we used is *DecisionTreeClassifier* from *sklearn.tree* for decision tree, and *AdaBoostClassifier* from *sklearn.ensemble* for boosting. In our first tries, we set the variables as $lep\_P_T\_1$ and $lep\_P_T\_2$, both correlated extracted from the labeled simulated data, as we attempt to get the best optimization by updating model's parameters.

Note Just like the ANN, we are going to check the results from :

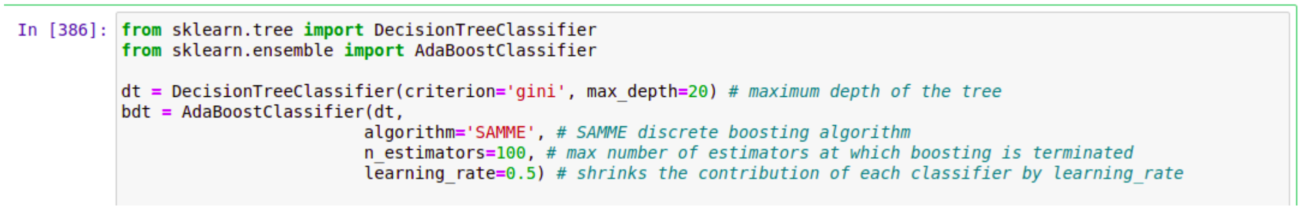
• The BDT score (signal and background) distribution,

• ROC curve,

• and signal to background ratio.

Updating BDT parameters

In the BDT optimization method we are modifying the following parameters: *maximum depth of the tree*, *max number of estimators*, and *learning rate*.



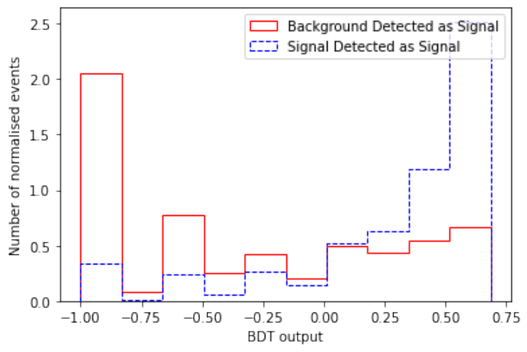
```
In [386]: from sklearn.tree import DecisionTreeClassifier
          from sklearn.ensemble import AdaBoostClassifier

          dt = DecisionTreeClassifier(criterion='gini', max_depth=20) # maximum depth of the tree
          bdt = AdaBoostClassifier(dt,
                                   algorithm='SAMME', # SAMME discrete boosting algorithm
                                   n_estimators=100, # max number of estimators at which boosting is terminated
                                   learning_rate=0.5) # shrinks the contribution of each classifier by learning_rate
```

Figure 3.18: Cell from jupyter notebook showing the parameters of BDT.

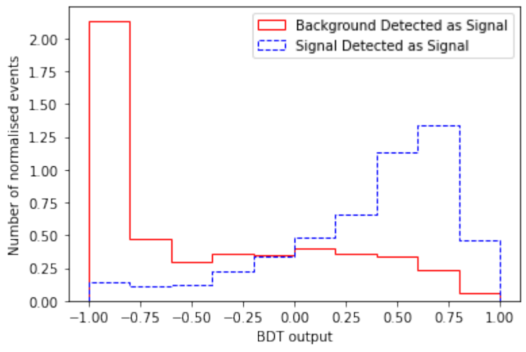
Since every parameter has an impact on our results, we try to find out about their effect, by updating them one by one, to fit with the best possible results. We started by changing the Maximum Depth of the tree which would allow the BDT model to seperate more data, starting by 2, 10, 15, 20, 25, to finally decide to work with 20. Technically we could use bigger numbers but it wouldn't give us any different and/or better results. We then started modifying the number of estimators to get acceptable results. We began with 12 until we got to 100, which gave us the best results. We couldn't change it to a bigger number since our devices started taking a really long time (up to 30 min). The last thing we had to check was the learning rate. We set it to various values, from 0.01 to 1 but the only usable results we got were when we set the learning rate at 0.5, which makes sense since it was the value set by default in the original code file. Eventually, the best parameters we found are:

- Maximum Depth = 20.
- Number of Estimators = 100.
- Learning rate = 0.5.

In the figures [3.19] extracted from the code, we can clearly see the evolution in the results, from the first try to the best.

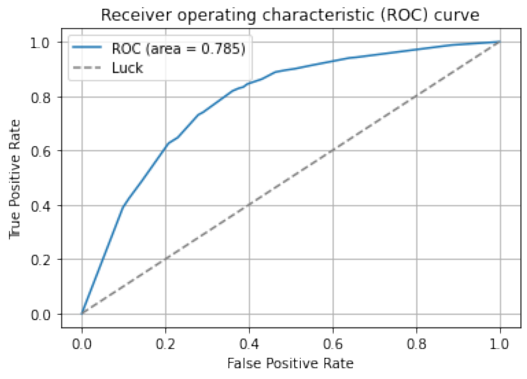


(a) *Maximum Depth* = 2, *Number of Estimators = 10, Learning rate = 0.5*

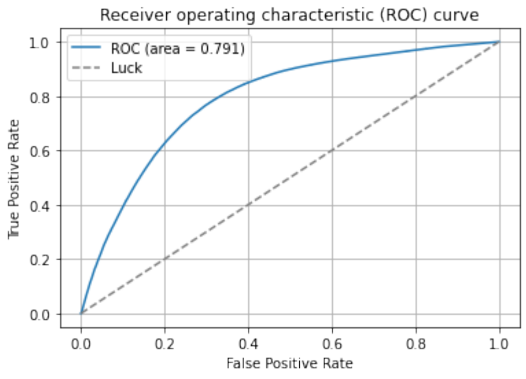


(b) *Learning rate* as 0.05, *Maximum Depth = 20, Number of Estimators = 100*

Figure 3.19: Optimized BDT parameters for an efficient Signal-Background Separation.

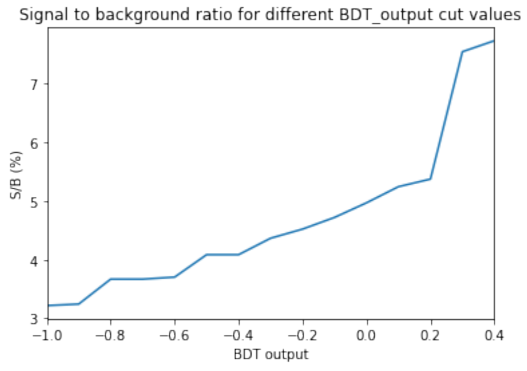


(a) *Maximum Depth* = 2, *Number of Estimators = 10, Learning rate = 0.5*

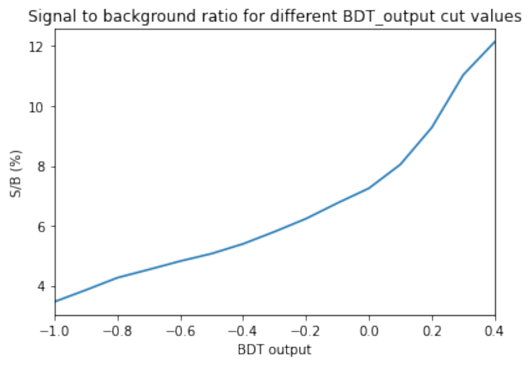


(b) *Learning rate* as 0.05, *Maximum Depth = 20, Number of Estimators = 100*

Figure 3.20: Optimized BDT parameters for an efficient ROC Curves.



(a) *Maximum Depth* = 2, *Number of Estimators = 10*, *Learning rate = 0.5*



(b) *Learning rate* as 0.05, *Maximum Depth = 20*, *Number of Estimators = 100*

Figure 3.21: Optimized BDT parameters for an efficient Signal/Background Ratio.

Now that we have done our experiments, while applying a change in various parameters. It becomes apparent how each of them affect the results. We have seen how the more we increase the depth of the tree and add the number of estimators (base learners), the algorithm would explore more from -1 to +1, while the background detected as signal decreases. Also in terms of S/B ratio we got up to 14 % with tree depth equals 20 or more. The same applies for the ROC curve. The only exception we found was with the learning rate where only 0.5 gave good results, which means whenever we increase or decrease the learning rate it would give worst results.

Checking Overtraining Now that we have the combination of the best parameters, we have to confirm that the BDT model is not overtraining (explained in [2.5.3]). Just like in the ANN, by using the 33% of the data preserved for testing we can illustrate the figure [3.22] and deduce that the model indeed doesn't overtrain. As it is clear, the testing results represented by the dots match perfectly the training results represented by the histograms, hence, when the BDT model is faced with unknown data (testing data), it can preserve a good efficiency of prediction.

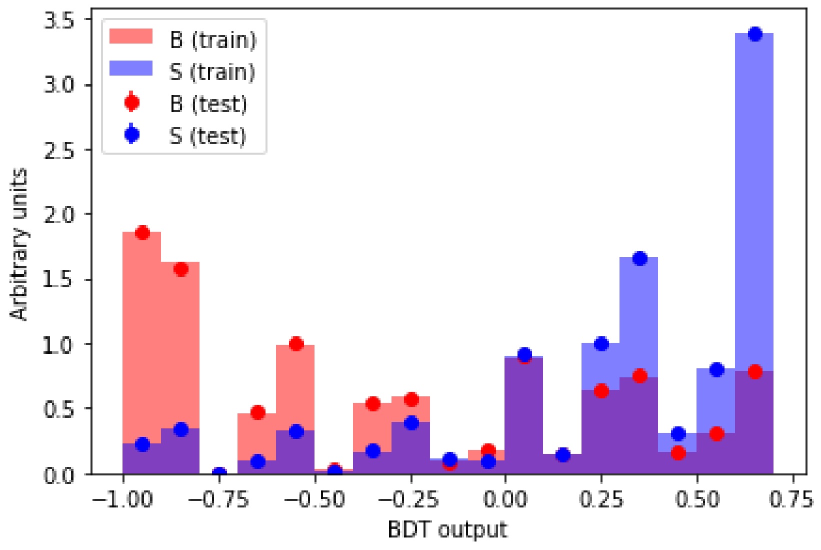
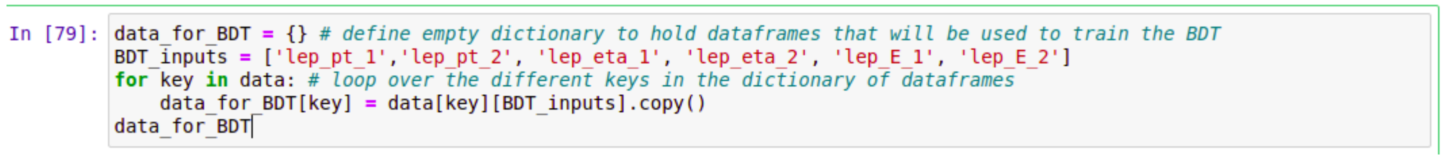


Figure 3.22: Boosted decision tree overtraining check.

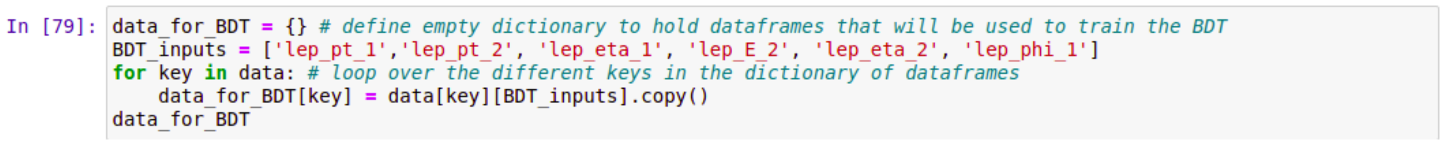
Updating optimization variables on BDT

Here, we start by adding some variables with positive correlation (meaning positive between variables of the same type of variables corresponding different particles) represented in the heatmap: "*lep_pt_1*", "*lep_pt_2*", "*lep_eta_1*", "*lep_eta_2*", "*lep_E_1*", "*lep_E_2*". Then, we continue by adding variables with negative correlation or as called decorrelation: "*lep_pt_1*", "*lep_pt_2*", "*lep_eta_1*", "*lep_E_2*", "*lep_eta_2*", "*lep_phi_1*". The modifications we made in the code are as following:



```
In [79]: data_for_BDT = {} # define empty dictionary to hold dataframes that will be used to train the BDT
         BDT_inputs = ['lep_pt_1','lep_pt_2', 'lep_eta_1', 'lep_eta_2', 'lep_E_1', 'lep_E_2']
         for key in data: # loop over the different keys in the dictionary of dataframes
             data_for_BDT[key] = data[key][BDT_inputs].copy()
         data_for_BDT
```

(a) Correlation.

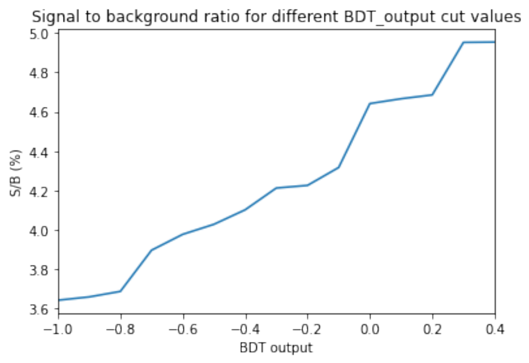


```
In [79]: data_for_BDT = {} # define empty dictionary to hold dataframes that will be used to train the BDT
         BDT_inputs = ['lep_pt_1','lep_pt_2', 'lep_eta_1', 'lep_E_2', 'lep_eta_2', 'lep_phi_1']
         for key in data: # loop over the different keys in the dictionary of dataframes
             data_for_BDT[key] = data[key][BDT_inputs].copy()
         data_for_BDT
```

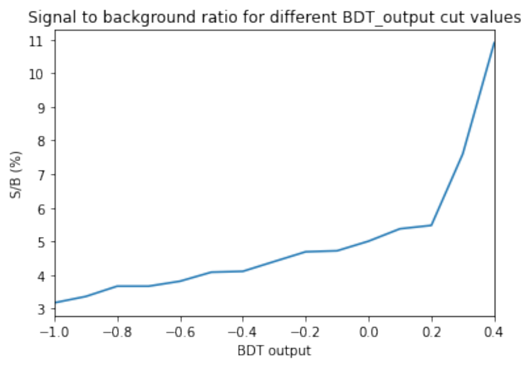
(b) Decorrelartion.

Figure 3.23: Cells from the code showing the modifications in the variables.

The outputs show us the apparent change represented in the signal to background ratio:



(a) Correlation.



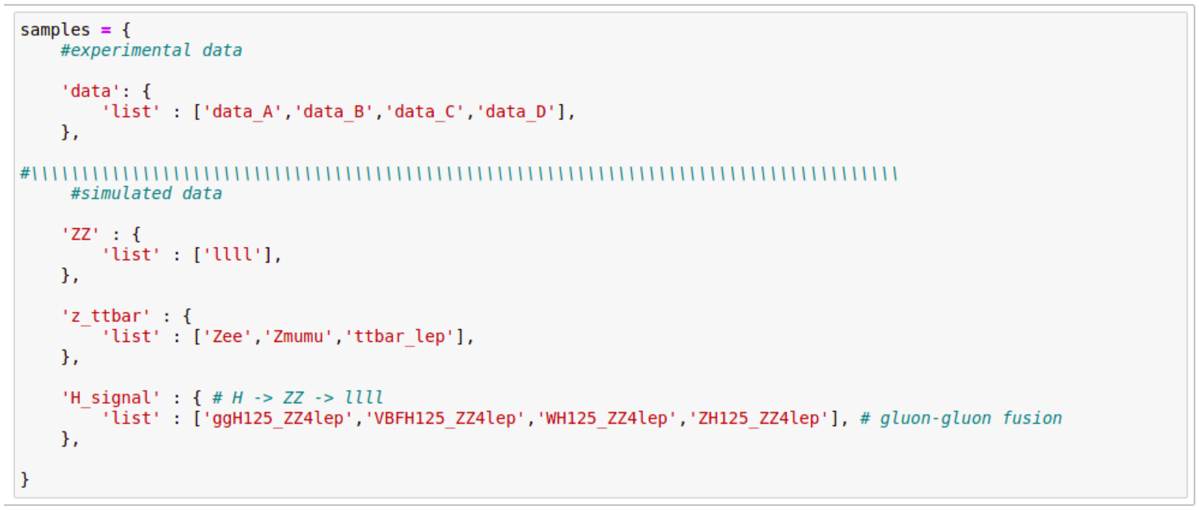
(b) Decorrelartion.

Figure 3.24: Higgs signal over the background ratio after testing the correlated variables and decorrelated variables on the BDT model.

The results showed obvious differences between the optimization with correlated and decorrelated variables. Here decorrelated variables gave much better results in terms of the signal to background ratio.

3.3 Dealing with experimental data

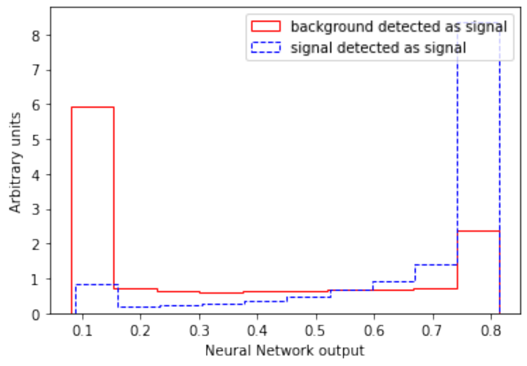
Using the labeled simulated data, we found a good architecture and trained models that we trust. The architecture is represented by the best combination of parameters to give the best possible results. Now, we can take our experiments to the next level, and challenge the ML models with the real data "*experimental data*" which is harder to classify since its events do not carry labels.



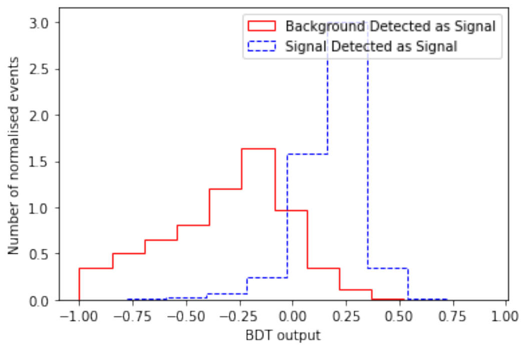
```
samples = {
    #experimental data

    'data': {
        'list' : ['data_A','data_B','data_C','data_D'],
    },

#\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
     #simulated data

    'ZZ' : {
        'list' : ['llll'],
    },

    'z_ttbar' : {
        'list' : ['Zee','Zmumu','ttbar_lep'],
    },

    'H_signal' : { # H -> ZZ -> llll
        'list' : ['ggH125_ZZ4lep','VBFH125_ZZ4lep','WH125_ZZ4lep','ZH125_ZZ4lep'], # gluon-gluon fusion
    },

}
```

Figure 3.25: Cell from the code to show where we added the experimental data samples.

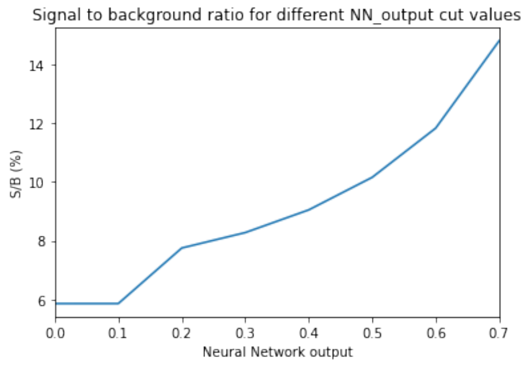
The results from dealing with the experimental data are shown in the figures [3.26].



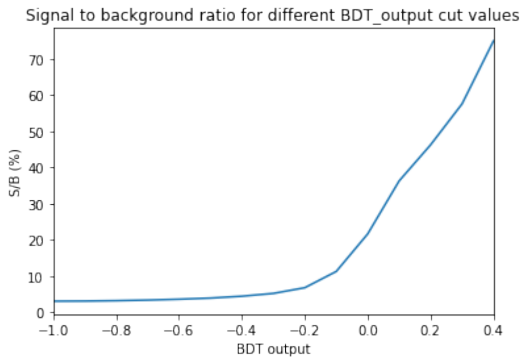
(a) ANN output with adding experimental data.



(b) BDT score with adding experimental data.



(c) ANN signal to background ratio with adding experimental data.



(d) BDT signal to background ratio with adding experimental data.

Figure 3.26: Results of testing machine learning models on experimental data.

We can see that our models make a good classification (ANN and BDT output), even though we added the non-labeled experimental data. The same applies with the signal to background ratio.

3.4 Discussion

Since the traditional method does not exactly give a good optimization comparing to the other machine learning methods we used, hence, we won't be discussing it. Over our encounter with ANN and BDT, we observed how the slightest change on the inputs have a great impact on the outputs. This impact is not only in term of the performance (Signal over background ratio, ROC curve, and signal background separation) but also on the needed time to obtain the results. Of course, it is worth mentioning that all of this would much differ between the methods, whether we used the simulated or the experimental data, and also the capacity of our used devices.

3.4.1 Discussing Artificial Neural Network results

After testing our Artificial neural network model ($MLPClassifier$) with different parameters and experiencing it with new samples and variables, we noticed the following results :

• Adding more hidden layers for our neural network means increasing the number of neurons which means a deeper processing for our inputs dataset, therefore leads to a good classification whether the event is *Higgs* signal or background event.

• Putting *ReLU* as activation function is faster to compute than the *sigmoid* function, and its derivative is faster to compute. Meaning *ReLU* converge faster to the global minima where the weights get updated throughout the optimization process [2.9]. This makes a significant difference to training and inference time for neural networks.

• Increasing the iterations number means increasing the number of back and forth in the back-propagation process in order to update the weights, given by that a very good result in classification task.

• Since the learning rate may be the most important hyperparameter when configuring the

ANN, adapting our ANN model with the right learning rate decreases the processing time, and updates the weights get very well.

• Adding decorrelated variables into our artificial neural network increases its efficiency because using non interconnected variables means giving more information to the ML models, which means more features that help to recognize each data event (*Higgs* signal or background noise).

3.4.2 Discussing Boosted Decision Tree results

In the BDT optimization method we are modifying the following parameters : *maximum depth of the tree*, *max number of estimators*, and *learning rate*.

• Maximum Depth gives better splits for signal and background therefore reduces error.

• Number of estimators (base learners, also called weak classifiers) decreases the number of incorrectly classified events.

• Using gini index (GI) criterion purity gives the best first split in the root node, also good decision tree classification, and S/B ratio, better than entropy criterion.

• Fixing the Learning rate reduces the misclassified events and also those incorrectly classified.

3.4.3 Discussing dealing with experimental data

As we saw in the previous section, artificial neural network and boosted decision tree machine learning models were trained using the **labeled** simulated data, before we applied on them the experimental dataset. As we can see in the figures [3.26], the results are not similar between the two models. The difference between the models is not caused by the lack of efficiency but could be the result of many problems, such as a conflict between the best parameters and the capacity of the used computer, or the imbalance in the efficiency of the used parameters. Regardless of that, it is very important to mention that we challenged the machine learning models to classify unknown experimental data, and it was indeed successful. Therefore, our goal was fulfilled, to enhance *Higgs* signal.

Conclusion

The *Higgs* boson signal existence is naturally insignificant in the ATLAS data, since it is produced from rare events. In this Master thesis, we used Machine Learning optimization methods, in the form of Artificial Neural Network and Boosted Decision Tree, as alternative modern ways that allow us to explore the multidimensional phase space and extract the region where the $Higgs$ signal is more dominant.

We began by exploring some background knowledge about the Higgs boson, starting by the process of how it is generated experimentally at the LHC, passing by the detection and identification processes at the ATLAS detector, and the difficulties of detecting the $Higgs$ which is buried in a huge fraction of background events. We then explored the trigger system that allows us to reduce the huge amount of data to a manageable size, to be easily handled using the capacity of our computers. We then introduced about the simulated dataset and how it is generated. After assembling all the needed theoretical notions, it was time to apply them on a live code. Fortunately, we had access to ATLAS opendata source, the one that not only contained the prepared machine learning codes that we based our models on, but also the datasets (simulated and experimental) we used in our work.

We had started our experiments with the simulated dataset that allowed us to construct an optimal architecture that found the best region which is enriched with the Higgs signal. We achieved that by adjusting different parameters simultaneously. This optimal adjustment is represented by extending the depth of the neural network and boosting the decision tree with the Adaboost algorithm. We can simplify the way optimization methods work with the Higgs signal to be pretty similar to the way microscopes work to amplify the tiny objects and make them more visible. Using decorrelated variables for our machine learning models increases efficiency because adding the decorrelated variables means adding more information to the model. Then we repeated the procedure with the experimental dataset but working the optimal architecture. To finally conclude that both artificial neural networks and boosted decision trees are efficient at their job with a reasonable value, and as a small criticism, both models cannot have efficient optimization and quick processing all at the same time.

Beside the methods that we used there are plenty others which are used in parallel in the research area, such as: CNN, RNN, QBDT, and more, some of them can be more efficient. The procedure we took using machine learning methods to enhance the higgs signal is similar to the one used today in high energy physics research, which may be the gate to unravel a new physics that goes beyond the *Standard Model.* It is promising that the Higgs particle discovery is the first of many to come, among which signals are even more rare, with higher energy and bigger masses.

Appendix A

The Standard Model of Particle Physics

Theories and discoveries of many physicists over the past century have provided understanding of the fundamental structure of matter, to conclude that our universe is made up of a number of fundamental particles and governed by four fundamental forces, where the SM of particles help us to understand the basics of these particles and which three of the four forces of nature are interconnected. It was made to explain with high accuracy the results of a large number of experiments and to predict a wide variety of phenomena. The SM was tested by numerous experiments since it first appeared and has been shown to successfully describe high energy particle interactions.

The Standard Model is the name given in the 1970s to a theory of fundamental particles and how they interact. It incorporated all that was known about subatomic particles at the time and predicted the existence of additional particles as well.

There are seventeen named particles in the Standard Model, organized into the chart shown below. The last particles discovered were the W and Z bosons in 1983, the top quark in 1995, the tau neutrino in 2000, and the Higgs boson in 2012.

The mechanism that breaks electroweak symmetry in the standard model which gives mass to elementary particles, implies the existence of a scalar particle was called the **Higgs boson**, it is produced by the quantum excitation of the Higgs field with zero spin, no electric charge, no colour charge. For many years, the big problem was that no experiment had observed the higgs boson, which would have confirmed the theory that explains the origin of mass, it wasn't until july 4th ,2012, that the ATLAS and CMS experiments at the LHC announced that they had both observed a new particle which exhibits the same characteristics with those of the Higgs boson predicted by the theory, with a mass in the 126 Gev region.

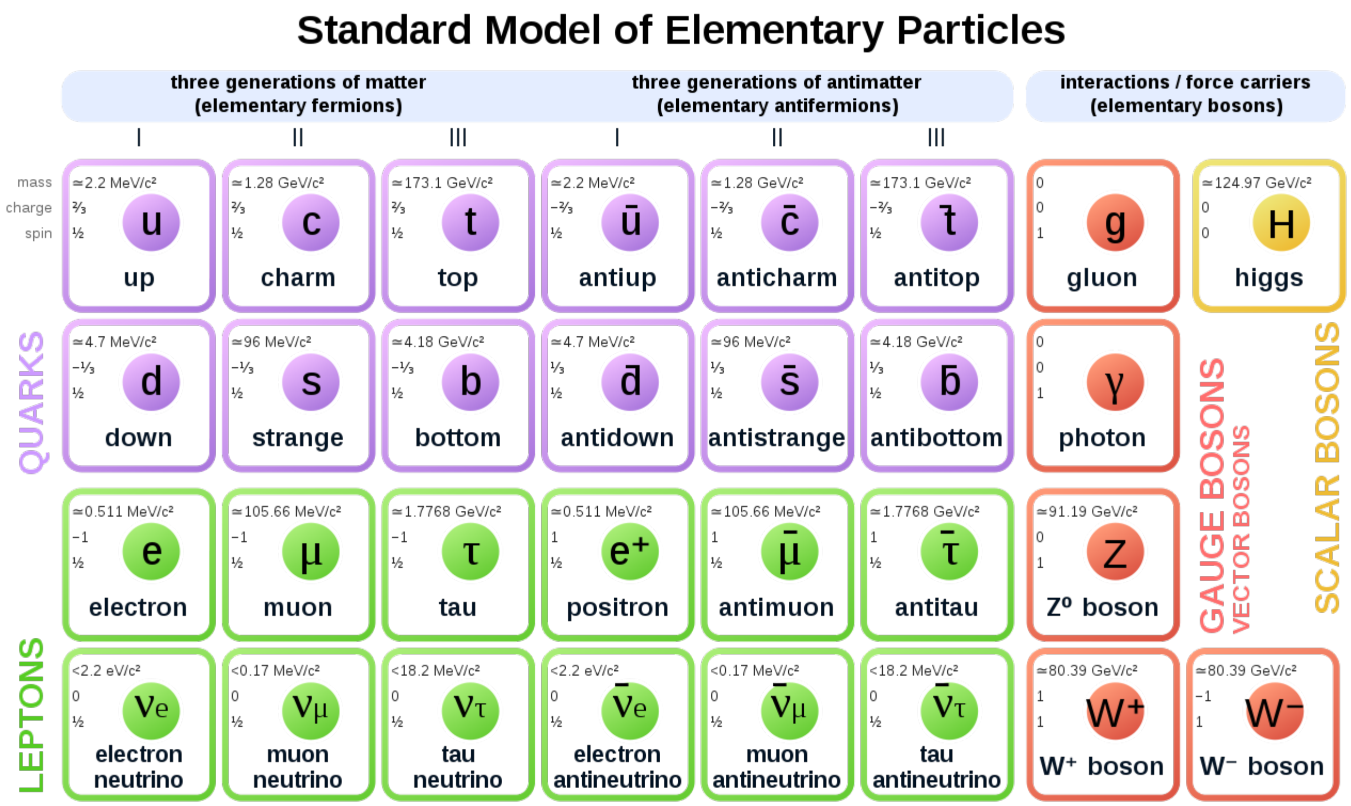

Standard Model of Elementary Particles
three generations of matter (elementary fermions)
three generations of antimatter (elementary antifermions)
interactions / force carriers (elementary bosons)
I
II
III
I
II
III
mass
charge
spin
≃2.2 MeV/c² ⅔ ½ u up
≃1.28 GeV/c² ⅔ ½ c charm
≃173.1 GeV/c² ⅔ ½ t top
≃2.2 MeV/c² −⅔ ½ ū antiup
≃1.28 GeV/c² −⅔ ½ c̄ anticharm
≃173.1 GeV/c² −⅔ ½ t̄ antitop
0 0 1 g gluon
≃124.97 GeV/c² 0 0 H higgs
≃4.7 MeV/c² −⅓ ½ d down
≃96 MeV/c² −⅓ ½ s strange
≃4.18 GeV/c² −⅓ ½ b bottom
≃4.7 MeV/c² ⅓ ½ d̄ antidown
≃96 MeV/c² ⅓ ½ s̄ antistrange
≃4.18 GeV/c² ⅓ ½ b̄ antibottom
0 0 1 γ photon
≃0.511 MeV/c² −1 ½ e electron
≃105.66 MeV/c² −1 ½ μ muon
≃1.7768 GeV/c² −1 ½ τ tau
≃0.511 MeV/c² 1 ½ e⁺ positron
≃105.66 MeV/c² 1 ½ μ̄ antimuon
≃1.7768 GeV/c² 1 ½ τ̄ antitau
≃91.19 GeV/c² 0 1 Z Z⁰ boson
<2.2 eV/c² 0 ½ νe electron neutrino
<0.17 MeV/c² 0 ½ νμ muon neutrino
<18.2 MeV/c² 0 ½ ντ tau neutrino
<2.2 eV/c² 0 ½ ν̄e electron antineutrino
<0.17 MeV/c² 0 ½ ν̄μ muon antineutrino
<18.2 MeV/c² 0 ½ ν̄τ tau antineutrino
≃80.39 GeV/c² 1 1 W⁺ W⁺ boson
≃80.39 GeV/c² −1 1 W⁻ W⁻ boson
QUARKS
LEPTONS
GAUGE BOSONS
VECTOR BOSONS
SCALAR BOSONS

Figure A.1: Standard model of elementary particles (*).

Appendix B

Machine Learning overview

Machine Learning techniques are making a huge leap in the scientific field, and becoming much easier allowing anyone to put it in use to achieve their purposes, not like until recent years where it was only available to large companies and institutions.

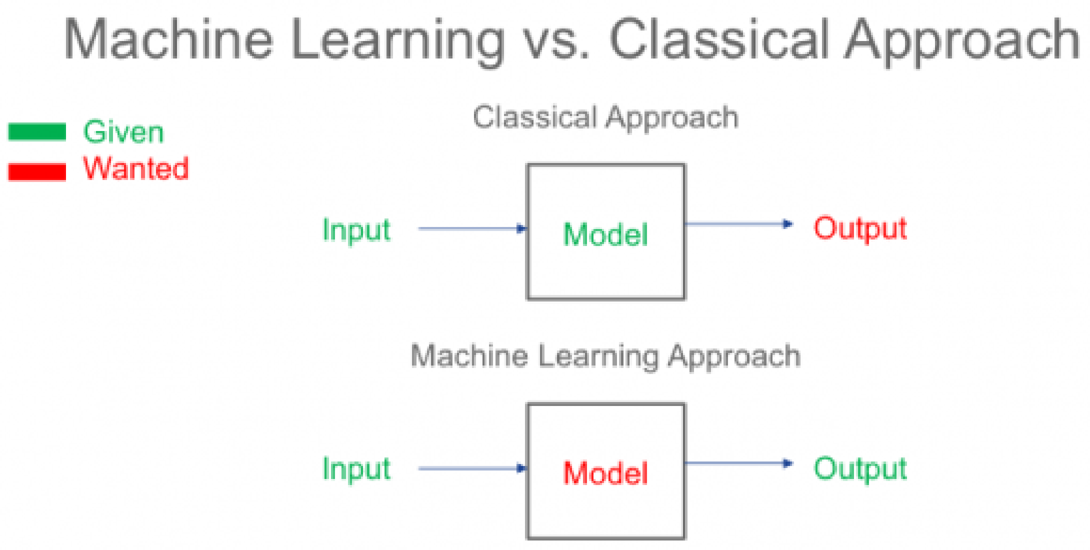

Machine Learning vs. Classical Approach
Given
Wanted
Classical Approach
Input
Model
Output
Machine Learning Approach
Input
Model
Output

Figure B.1: Machine Learning and Classical Models (*).

Machine Learning is divided into two main areas: supervised learning and unsupervised learning. Although it may seem that the first refers to prediction with human intervention and the second does not, these two concepts are more related with what we want to do with the data.

• One of the most common uses of supervised learning is to make future predictions based on behaviors or characteristics that have been already seen in the data (historical data). Supervised learning makes it possible to search for patterns in historical data. For example, e-mails are labeled as "spam" or "legitimate" by users. The prediction process begins with an analysis of which characteristics or patterns the marked emails have. It can be determined, for example, that spam email is the one that comes from certain IP addresses, has a certain text and/or images, and more. Once all patterns have been determined, that means the learning phase is done, new mails that have never been marked as spam or legitimate are compared with patterns to be classified based on their characteristics.
• On the other hand, unsupervised learning uses historical data that has no target field. The aim is to explore the data and find some structure or to organize it. For example, it is often used to group customers with characteristics or behaviors similar to those of highly segmented marketing campaigns.

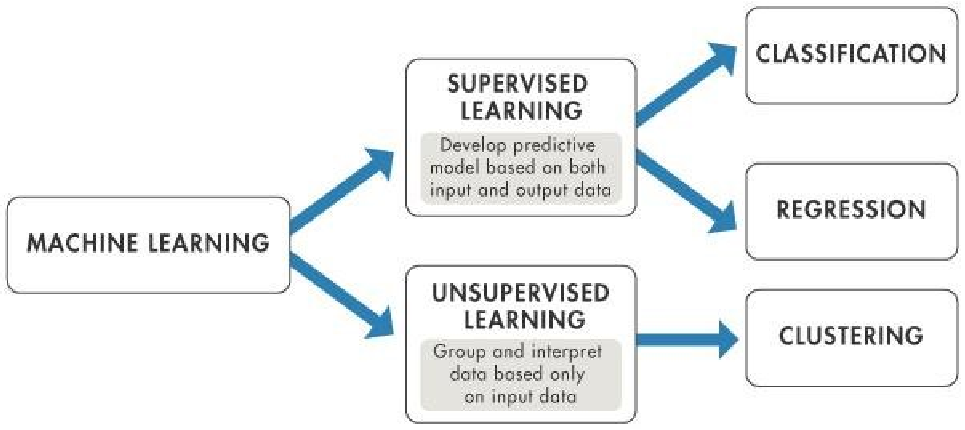

MACHINE LEARNING
SUPERVISED LEARNING
Develop predictive model based on both input and output data
UNSUPERVISED LEARNING
Group and interpret data based only on input data
CLASSIFICATION
REGRESSION
CLUSTERING

Figure B.2: Supervised vs Unsupervised Learning (*).

B.1 Learning and training

It is the process in which the patterns of a data set are detected, that is the heart of machine learning. Once patterns are identified, predictions can be made with new data entered into the system. For example, historical data from book purchases on an online website can be used to analyze customer behavior in their purchasing processes (titles visited, categories, purchase history…), group them into behavioral patterns, and make purchase recommendations to new customers who follow known or learned patterns.

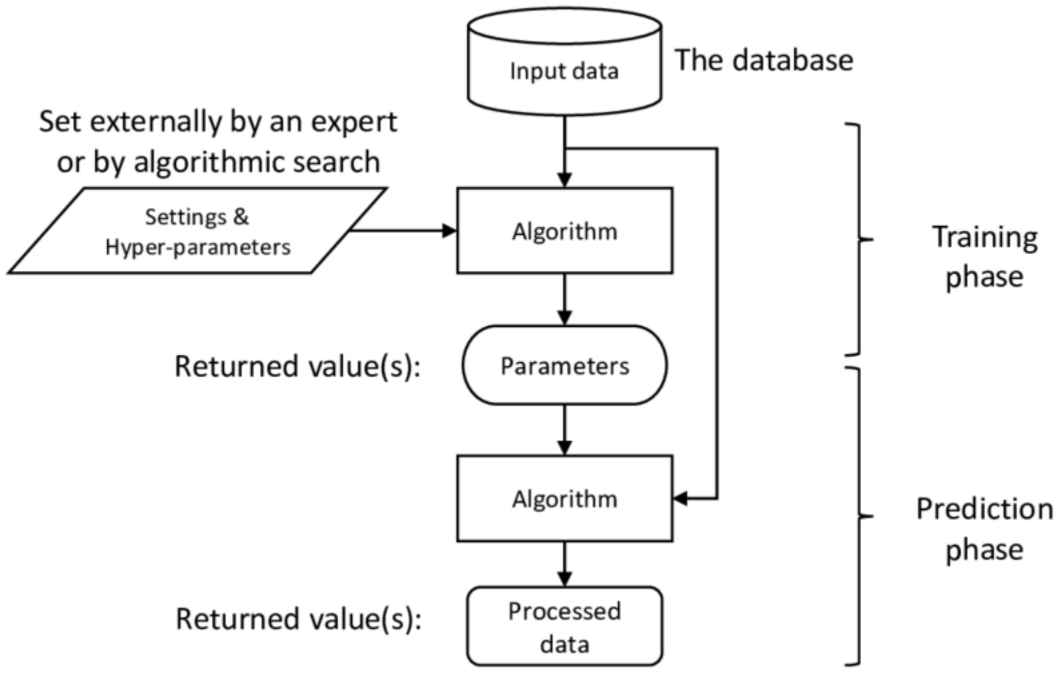


Figure B.3: Learning and training (*).

Appendix C

Illustratif example of Neural network technique

The term *Artificial Neural Network* is derived from Biological neural networks that form the structure of a human brain. Similar to the human brain, artificial neural networks have neurons that are interconnected to one another in various layers of the network. These neurons are known as nodes. Typically Artificial Neural Network looks something like in the figure below.

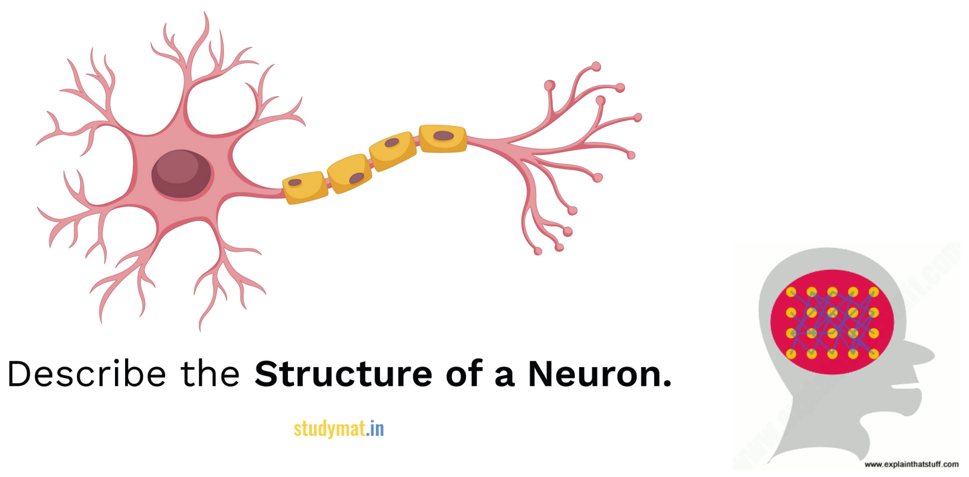


Figure C.1: Structure of simple neuron (*).

C.1 What does a neuron do ?

The operations done by each neurons are pretty simple :

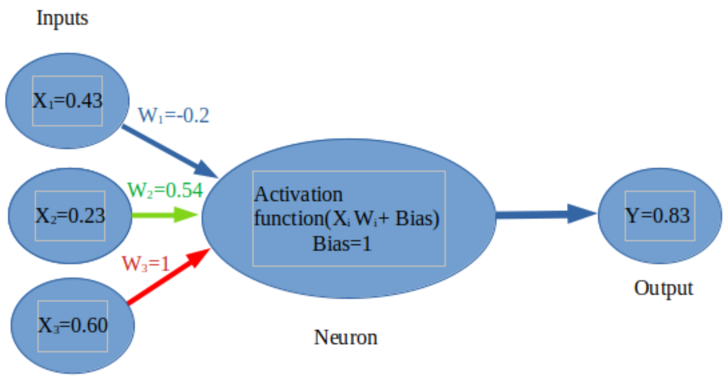


Figure C.2: Simple neuron.

First, it adds up the value of every neuron from the previous column it is connected to. On the Figure C.2, there are 3 inputs (X_1 =0.43, X_2 =0.23, X_3 =0.60) coming to the neuron, so 3 neurons of the previous column are connected to our neuron.
before being added, this value is multiplied, by another variable called *weight* ($w_1 = -0.2$, $w_2 = 0.54$, $w_3 = 1$) which decides the connection between the 3 neurons, where each connection has its own weight. The weights are the only values that will be modified during the learning process, also a bias value may be added to the total value calculated. The bias is chosen before the learning phase and is not a value coming from a specific neuron, but can be useful to fire the performance of the network. After all those summations, the neuron finally applies a function called *activation function* to the obtained value transforming by the inputs to a binary distribution. The so-called activation function usually serves to turn the total value calculated before to a number between 0 and 1. There is other functions exist and may change the limits of our function, but keep the same aim of limiting the value as :

• sigmoid : $\frac{1}{1+\exp(-x)}$ The function takes any real value as input and outputs values in the range 0 to 1

• The hyperbolic tangent activation function ($Tanh$): $\frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$ The function takes any real value as input and outputs values in the range -1 to 1

• Linear rectfill function ($ReLU$) :$max(x, 0)$ The function takes any real value as input and outputs values in the range 0 if $x \leq 0$ and x if x >0

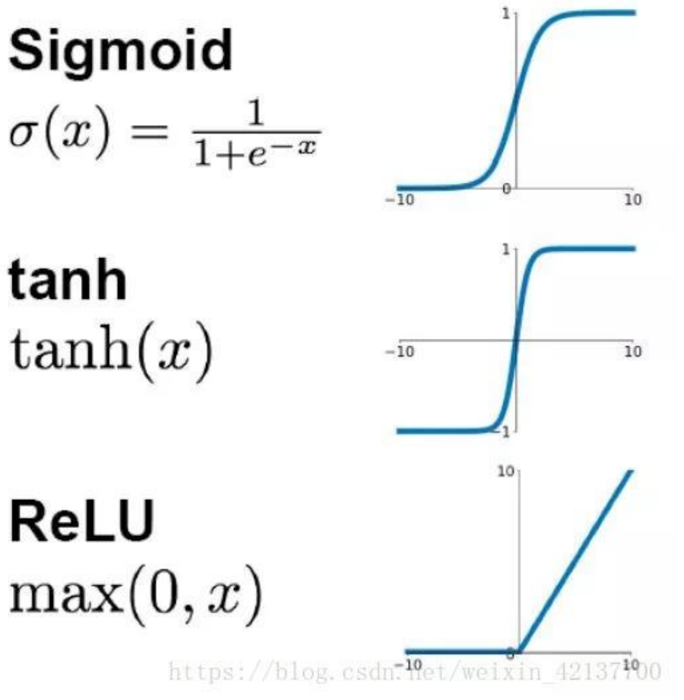


Figure C.3: Schematic representation of multiple activation functions and their limits.

That's all what a neuron does. Take all values from connected neurons multiplied by their respective weight, add them, and apply an activation function. Then, the neuron is ready to send its new value to other neurons.

How does a neural network learn ? Neural networks generally perform supervised learning tasks, building knowledge from data sets where the right answer is provided in advance. The networks then learn by tuning themselves to find the right answer on their own, increasing the accuracy of their predictions.
The most common supervised learning method is based on the gradient descent learning rule. The method optimises the network weights such that a certain objective error function E is minimised by calculating the gradient of E in the weight space and moving the weight vector along the *negative gradient*. For a single $Artificial\ Neuron\ AN$ the objective function is

usually an error function which measures the AN's error in approximating the target vector y:

$E = \sum_{i=1}^{n}(y_{predected} - y_{actual})^2$

The weights of the AN are adjusted by exposing it more times to the training data set. For a single training pattern the weights of an AN are adjusted with the formula : $W_{new} = W_{old} - \eta \frac{dE}{dW_{old}}$ where η is the learning rate which represents the size of the steps taken in the negative direction of the gradient.

C.2 Simple Artificial neural networks

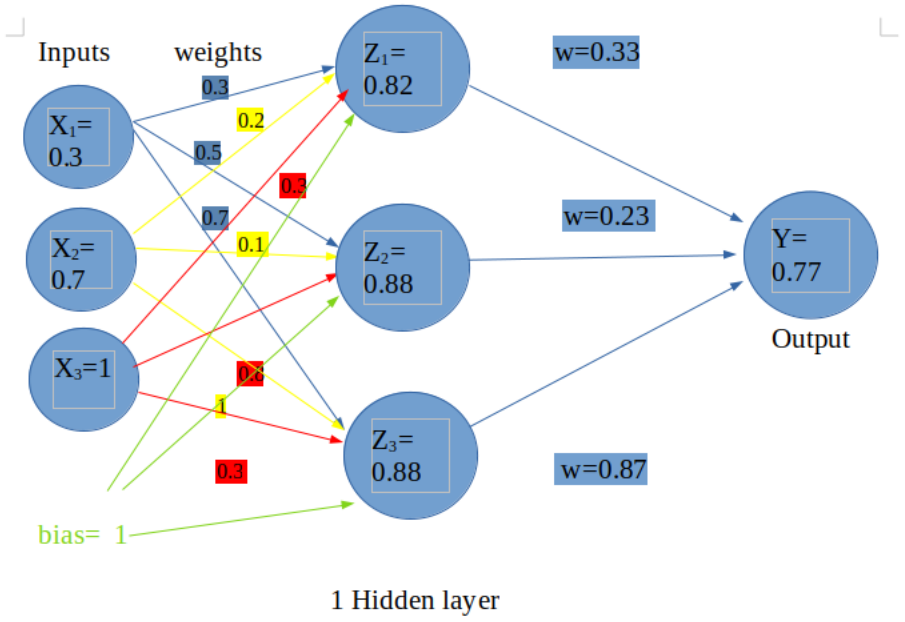


Figure C.4: Example of simple artificial network with 1 hidden layer.

Input Layer: As the name suggests, it accepts inputs in several different formats in ou simple networks we provide 3 inputs $x_1 = 0.34$, $x_2 = 0.70$, $x_3 = 1$, with a *bias*=1

Hidden Layer: The hidden layer presents $in - between$ input and output layers. It performs all the calculations to find hidden features Z_1, Z_2, Z_3, by the following steps :

• The first step is to calculate $x = \sum_{i=1}^{3} x_i w_i$, where W_i are the weights represented by colored rectangles with different values.

• The second step is to apply them on a non linear transformation by using sigmoid function as activation function : $sigmoid = \frac{1}{1+\exp(-x)}$

$sigmoid = \frac{1}{1+\exp(-\sum_{i=1}^{3} x_i w_i}$

$Z_i = \frac{1}{1+\exp(-\sum_{i=1}^{3} x_i w_i + bias)}$

$Z_1 = \frac{1}{1+\exp(-(0.3*0.3+0.7*0.2+1*0.3)+1)} = 0.82$

$Z_2 = \frac{1}{1+\exp(-(0.3*0.5+0.7*0.1+1*0.8)+1)} = 0.88$

$Z_3 = \frac{1}{1+\exp(-(0.3*0.7+0.7*0.8+1*0.3)+1)} = 0.88$

Output Layer: The input goes through a series of transformations using the hidden layer, which finally results in output that is conveyed using this layer. The artificial neural network takes input and computes the weighted sum of the inputs and includes a bias. This computation is represented in the form of a transfer function. It determines the weighted total is passed as an input to an activation function to produce the output. Activation functions choose whether a node should fire or not. Only those who are fired make it to the output layer. There are distinctive activation functions available that can be applied upon the sort of task we are performing.

$y = \frac{1}{1+\exp(-\sum_{j=1}^{3} x_j w_j)} + bias$

$y = \frac{1}{1+\exp(-(0.82*0.33+0.88*0.23+0.88*0.87))}$ =0.77

Neuron representation in python

In this example we try to predict target y. all the steps from learning to updating the weights are implemented in the code.

```
import numpy as np
def sigmoid(x):
    return 1/(1+np.exp(-x))
def sigmoid_derivative(x):
    return x*(1-x)
inputs = np.array([[0,0,1],
                   [1,1,1],
                   [1,0,1],
                   [0,1,1]])
outputs=np.array([[0,1,1,0]]).T
np.random.seed(1)
weights=np.random.random((3 ,1))-1
bias=0.3
for i in range(100000):
    inputs_layer = inputs
    first_outputs = sigmoid(np.dot(inputs_layer, weights)+bias)
    error = outputs-first_outputs
    adjustment=error*sigmoid_derivative(first_outputs)
    weights+=np.dot(inputs_layer.T, adjustment)
print(first_outputs)
[[0.00301772]
 [0.99753714]
 [0.99799165]
 [0.00246107]]
```

Listing C.1: Python simple neuron example to predict the maping of the vector target y

Appendix D

Illustratif example of boosted decision tree

To solve classification problems with mixed data, it is only possible to cut over and over again and apply more conditions in order to get a good classification.

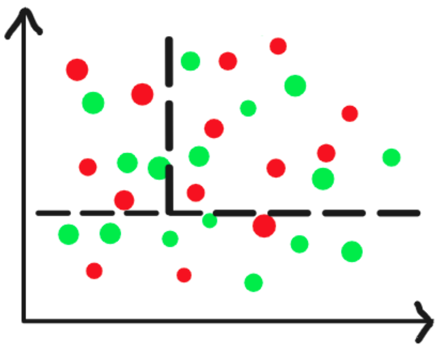


Figure D.1: Exemple of mixed data.

A decision tree is a type of probability algorithm tree that makes a decision about some process that contains conditional control statements, and it is represented by a diagram or chart that determines the course of the action or the statistical probability. Boosting a decision tree reduces error mainly by reducing bias. The algorithm of Boosting decision trees learns by fitting the residual of the trees that preceded it. Thus, boosting in a decision tree ensemble tends to improve accuracy with some small risk of less coverage.

As an example, we can create a decision tree that predicts whether a particle is a Fermion or a Boson, using the following data;

Tableau D.1: Table of data for classification

Name	Type	Antiparticle	Spin	Label
Antitau	Lepton	yes	$\frac{1}{2}$	0
Muon	Lepton	no	$\frac{1}{2}$	0
Gluon	Gauge Boson	no	1	1
Z	Gauge Boson	no	1	1
Muon neutrino	Lepton	no	$\frac{1}{2}$	0
Photon	Gauge Boson	no	1	1
W	Gauge Boson	no	1	1
Positron	Lepton	yes	$\frac{1}{2}$	0

Where 1 is assigned to bosons and 0 to fermions. The decision tree for this classification would look something like that:

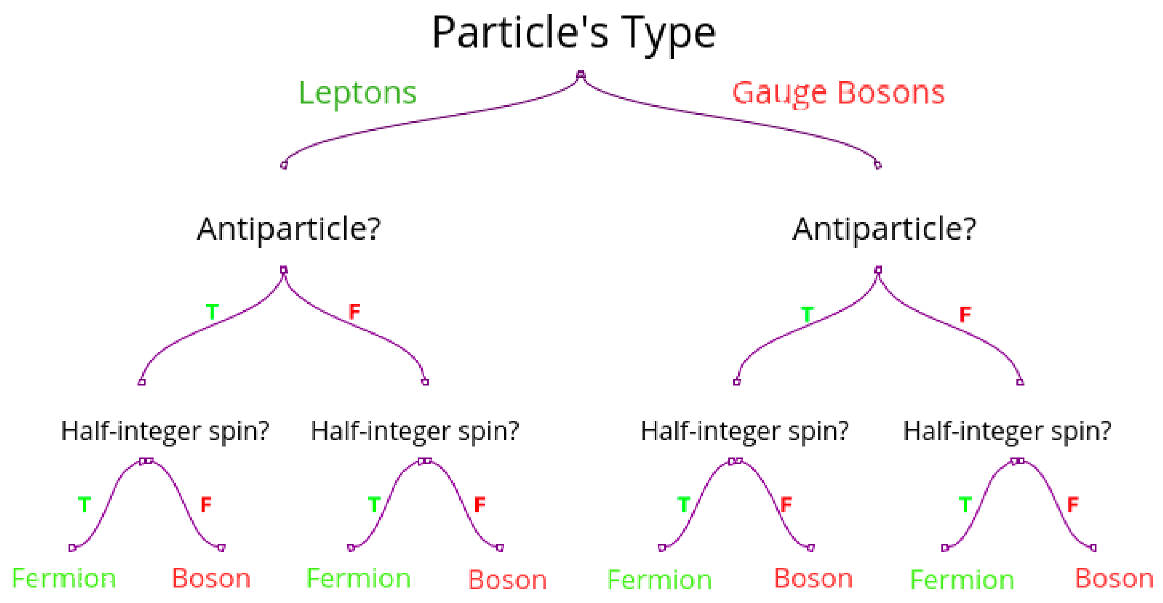


Figure D.2: Simple decision tree that classify particles.

Starting from a node or called interior nodes if it is preceded by a branch : where the data is splitted based on an input feature. The branch of the decision tree represents a possible decision or outcome, here, the features are: particle's type, spin, and whether it is an antiparticle or not. Leaves are the terminal nodes that represent a class label or probability, containing the final classified outputs. A decision tree learns by taking a set of input features and splits the data based on those features. Each split is meant to maximize information gain and minimize entropy, where the gain is the difference in entropy before and after the split.

Boosted decision tree representation in python

We can create the previous decision tree using python programming language,

```python
import pandas as pd

df = pd.read_csv("bosons.csv")
df.head()

inputs = df.drop('Label',axis='columns')

target = df['Label']

from sklearn.preprocessing import LabelEncoder
le_Name = LabelEncoder()
le_Type = LabelEncoder()
le_Antiparticle = LabelEncoder()
le_Spin = LabelEncoder()

inputs['Name_n'] = le_Name.fit_transform(inputs['Name'])
inputs['Type_n'] = le_Type.fit_transform(inputs['Type'])
inputs['Antiparticle_n'] = le_Antiparticle.fit_transform(inputs['
    Antiparticle'])
inputs['Spin_n'] = le_Spin.fit_transform(inputs['Spin'])

inputs
```

```

inputs_n = inputs.drop(['Name', 'Type', 'Antiparticle', 'Spin'],axis='
    columns')

inputs_n

target

from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier

dt = DecisionTreeClassifier(criterion='gini', max_depth=10)
model = AdaBoostClassifier(dt,
                              algorithm='SAMME',
                              n_estimators=10,
                              learning_rate=0.5)

model.fit(inputs_n, target)

model.score(inputs_n,target)
```

Listing D.1: Python example of particles classification

This code assigns for each of the input a number using the command *LabelEncoder* to classify them, then by using *sklearn* and *AdaBoostClassifier* we boost the tree and minimize the errors. Also the code can predict whether another particle is a boson or fermion using the following code line:

```
model.predict([[2,0,1,1]])
```

Listing D.2: Python example of particles classification

and the output give $array([0])$ meaning it assigned the number 0 to it, and classified it as a Fermion

Appendix E

Scikit-learn toolkit

Scikit-learn is a Python module integrating a wide range of machine learning algorithms for medium-scale supervised and unsupervised problems. It was initially developed by David Cournapeau as a Google summer of code project in 2007. This package focuses on bringing machine learning to non-specialists using a general purpose high-level language. Emphasis is put on ease of use, performance, documentation, and API consistency. It has minimal dependencies and is distributed under the simplified BSD license,encouraging its use in both academic and commercial settings. Source code, binaries, and documentation can be downloaded from http://scikit-learn.sourceforge.net.

The library is built upon the SciPy (Scientific Python) that must be installed before you can use scikit-learn. This stack that includes:

- **NumPy**: Base n-dimensional array package;
- **SciPy**: Fundamental library for scientific computing;
- **Matplotlib**: Comprehensive 2D/3D plotting;
- **IPython**: Enhanced interactive console;
- **Sympy**: Symbolic mathematics;
- **Pandas**: Data structures and analysis.

Appendix F

The ROC curves

A ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters:

• True Positive Rate

• False Positive Rate

True Positive Rate (TPR) is a synonym for recall and is therefore defined as follows:

$$TPR = \frac{TP}{TP + FN} \tag{F.1}$$

False Positive Rate (FPR) is defined as follows:

$$FPR = \frac{FP}{FP + TN} \tag{F.2}$$

• TP : True Positive is the number of higgs signal events classified as higgs signal events after a certain threshold value

• TN : True negative is the number of background events classified as background events after a certain threshold value

• FP : False positive is the number of background events classified as higgs signal events after a certain threshold value

• FN : False negative is the number of higgs signal events classified as background events after a certain threshold value

A ROC curve plots TPR vs. FPR at different classification thresholds. Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives. The following figure shows a typical ROC curve. To compute the points in an

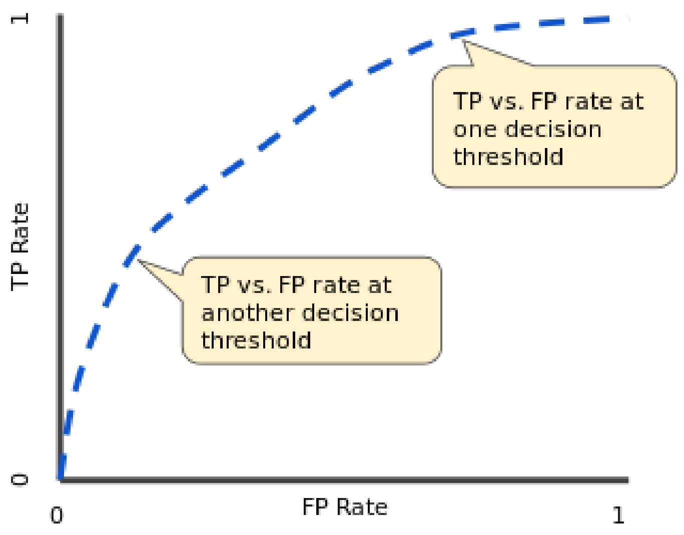


Figure F.1: TP vs. FP rate at different classification thresholds (*).

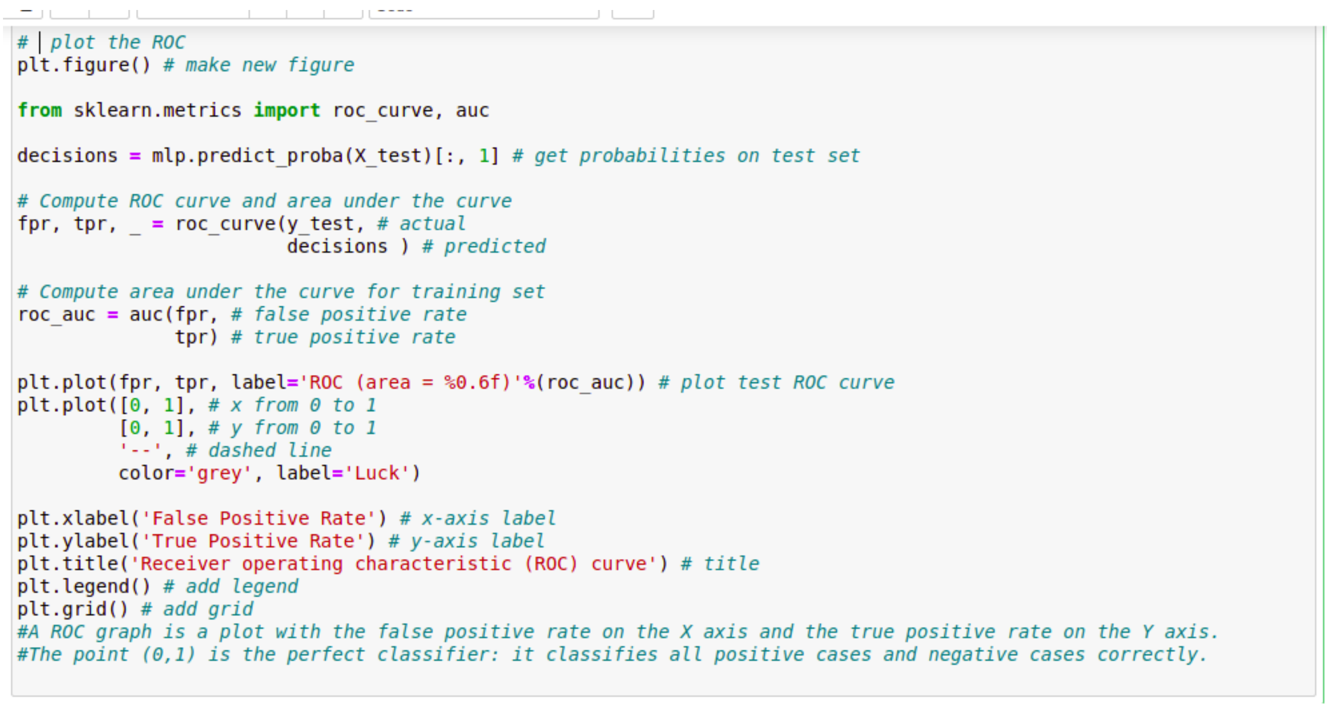
ROC curve, we could evaluate a logistic regression model many times with different classification thresholds, but this would be inefficient. Fortunately, there's an efficient, sorting-based algorithm that can provide this information for us, called AUC.

F.1 AUC

AUC stands for "Area under the Curve".It measures the entire two-dimensional area underneath the ROC curve (think integral calculus) from (0,0) to (1,1) and provides an aggregate measure of performance across all possible classification thresholds. One way of interpreting AUC is as the probability that the model ranks a random positive example more highly than a random negative example. The bigger the AUC is the better our ML model is.

F.1.1 How does *sklearn.metrics* draw ROC curves in the case of Higgs signal and background classification

In the case of Higgs signal and background classification in high energy physics, the number of events is divided into two types (higgs boson events and background events), then we have in this case a binary classification task. Let's take any machine learning model (ML) for example (ANN or BDT) in order to solve the classification task between higgs signal and the background events with two different values for y as target value (1 when we have higgs signal and 0 for background) that our machine learning model try to predict it. We import the ROC curve model from sklearn with the following command :



```
# plot the ROC
plt.figure() # make new figure

from sklearn.metrics import roc_curve, auc

decisions = mlp.predict_proba(X_test)[:, 1] # get probabilities on test set

# Compute ROC curve and area under the curve
fpr, tpr, _ = roc_curve(y_test, # actual
                        decisions ) # predicted

# Compute area under the curve for training set
roc_auc = auc(fpr, # false positive rate
              tpr) # true positive rate

plt.plot(fpr, tpr, label='ROC (area = %0.6f)'%(roc_auc)) # plot test ROC curve
plt.plot([0, 1], # x from 0 to 1
         [0, 1], # y from 0 to 1
         '--', # dashed line
         color='grey', label='Luck')

plt.xlabel('False Positive Rate') # x-axis label
plt.ylabel('True Positive Rate') # y-axis label
plt.title('Receiver operating characteristic (ROC) curve') # title
plt.legend() # add legend
plt.grid() # add grid
#A ROC graph is a plot with the false positive rate on the X axis and the true positive rate on the Y axis.
#The point (0,1) is the perfect classifier: it classifies all positive cases and negative cases correctly.
```

Figure F.2: Commands to plot the ROC curve to show performance of the used machine learning model in the classification (higgs signal or background task).

For example, suppose our machine learning model had predicted a number of target values shown in the table.

y target list			
actual target y	y predicted by ML model	y predicted for threshold (0)	y predicted for threshold (0.2)
1	0.8	1	1
0	0.96	1	1
1	0.4	1	1
1	0.3	1	1
0	0.2	1	0
1	0.7	1	1

The model *sklearn.metrics* takes the first threshold value as 0 (y predicted for threshold (0)). As we can see the first y=0.8 predicted by our Machine Learning model y=0.8 is greater than 0 so definitely is going to become 1 as is illustrated in the table, and others values become 1 also. Starting from this operation we calculate the TPR= $\frac{4}{4+0}$ = 1 since we have 4 true positive and 0 false negative. For the FPR= $\frac{2}{2+0}$ = 1 since we have 2 false positive and 0 true negative. Also the same operation to calculate TPR and FPR for the threshold value 0.2 and others 0.3,0.4....1.

sklearn.metrics model takes those TPR and FPR as coordinates (TPR,FPR) to draw the ROC curve starting from (1,1) that we found for the threshold 0 to the (0,0). After connecting all the dots together we will get the curve shown in the figure below :

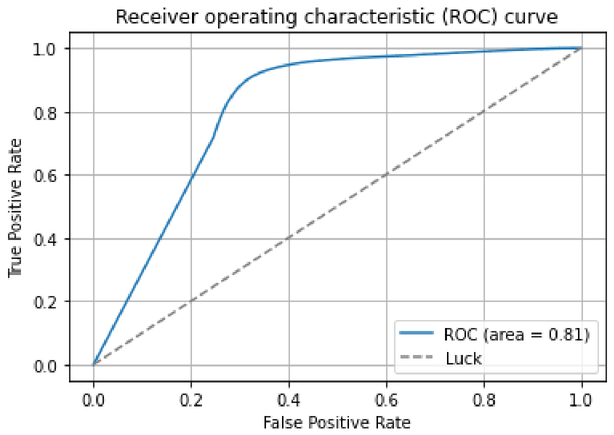


Figure F.3: ROC is the blue curve, AUC (Area Under Curve) is the area under the choppy curve .

The ROC curve should always be greater than the discrete line, it should never be less than (0.5,0.5), or else the model is weak with random predictions.

Bibliography

[1] M. D. Schwartz. TASI Lectures on Collider Physics. arXiv:1709.04533, (2017).

[2] L. Lista. Practical Statistics for Particle Physicists. arXiv:1609.04150v2, (2017).

[3] CERN. The Large Hadron Collider.

[4] CERN. ATLAS `https://home.cern/science/experiments/atlas`.

[5] Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC. Physics Letters B, (2012).

[6] F. H. Sawy M. T. Ghoneim and M. T. Hussein. Particle production in proton-proton collisions. arXiv:1505.06287, (2015).

[7] J. Prochazka. Elastic proton-proton collisions at high energies. PhD thesis, Institute of Particle and Nuclear Physics, (2018).

[8] Stanford ATLAS. Particle Collision and Detection, (2016).

[9] S. Bock. Higgs Physics at the LHC. PhD thesis, Heidelberg University, (2010).

[10] M. Y. Hussein. Higgs boson production at the lhc. arXiv:1703.03952, (2017).

[11] A. DJOUADI. Higgs Physics: Theory. arXiv:1203.4199, (2011).

[12] CMS Collaboration. Particle-flow reconstruction and global event description with the CMS detector. `doi:10.1088/1748-0221/12/10/P10003.`, (2009).

[13] ATLAS Collaboration. Performance of the ATLAS Detector using First Collision Data. arXiv:1005.5254, (2018).

[14] C. Germain I. Guyon B. Kégl V. V. Gligorov, G. Cowan and D. Rousseau. Real time data analysis at the LHC:present and future. arXiv:1509.06173, (2015).

[15] W. Lukas. Measurement of Charged-Particle Distributions in Proton-Proton Interactionsat $\sqrt{s} = 8$ TeV with the ATLAS Detector at the LHC. Master's thesis, University of Innsbruck, (2016).

[16] ATLAS Collaboration. Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC. Published by Elsevier B.V. Open access under CC BY-NC-ND license., (2012).

[17] C. Anastopoulos. Background Studies for the $Higgs \longrightarrow ZZ^* \longrightarrow 4l$ channel for the discovery of the Higgs Boson at the LHC. (2009).

[18] ATLAS Experiment. Trigger and Data Acquisition.

[19] M. Machet. Higgs boson production in the diphoton decay channel with CMS at the LHC : first measurement of the inclusive cross section in 13 TeV pp collisions, and study of the Higgs coupling to electroweak vector bosons. PhD thesis, Université Paris-Saclay, (2016).

[20] P. Nadolsky F. Olness. Research in Theoretical High-Energy Physics at Southern Methodist University. Technical report, Southern Methodist University.

[21] R.S. Thornec A.D. Martina, W.J. Stirlingb and G. Wattc. Parton distributions for the LHC. arXiv:0901.0002v3, (2009).

[22] F. Cerutti L. Del Debbio S. Forte A. Guffanti J. I. Latorre J. Rojo R. D. Ball, V. Bertone and M. Ubiali. Impact of Heavy Quark Masses on Parton Distributions and LHC Phenomenology. arXiv:1101.1300v3, (2011).

[23] G. Luisoni. An introduction to POWHEG. arXiv:0709.2092, (2017).

[24] A. Greljo. Tutorial on HiggsPO tool. (2016).

[25] T.Isolabella. A machine learning approach to particle physics data analysis: the process $J/\psi \longrightarrow \gamma p\bar{p}$. Master's thesis, University of Groningen, (2019).

[26] M. Paganini. Machine Learning Solutions for High Energy Physics: Applications to Electromagnetic Shower Generation, Flavor Tagging, and the Search for di-Higgs Production. PhD thesis, Yale University, (2019).

[27] R. J. Barlow. Practical statistics for particle physics. arXiv:1905.12362v2, (2021).

[28] K. Cranmer D. Guest and D. Whiteson. Deep Learning and Its Application to LHC Physics. arXiv:1806.11484v1, (2018).

[29] D. Bourilkov. Machine and Deep Learning Applications in Particle Physics. International Journal of Modern Physics A, arXiv:1912.08245v1, (2019).

[30] J. Therhaag. Introduction to neural networks in high energy physics. EPJ Web of Conferences 55,02003 (2013), (2013).

[31] Dr. S. Goswami. https://towardsdatascience.com/early-stopping-a-cool-strategy-to-regularize-neural-networks-bfdeca6d722e, 2020.

[32] M. Guth. Signal Region Optimisation Studies Based on BDT and Multi-Bin Approaches in the Context of Supersymmetry Searches in Hadronic Final States with the ATLAS Detect. Master's thesis, UNIVERSITY OF FREIBURG, Department of Mathematics and Physics.

[33] J. Zhu Y. Liu I. Stancu B.P. Roe, H. Yang and G. McGregor. Boosted Decision Trees as an Alternative to Artificial Neural Networks for Particle Identification. Nucl.Instrum.Meth. A543 (2005) 577-584, arXiv:0408124, (2018).

[34] G. Cowan. Topics in Statistical Data Analysis for High-Energy Physics. (2010).

[35] Y. Freund and R. E. Schapire. ADecision-Theoretic Generalization of On-Line Learning and an Application to Boosting. journal of computer and system sciences 55, 119-139 (1997), (1997).

[36] L. Serkin. The release of the 13 TeV ATLAS Open Data: using open education resources effectively. (2020).

[37] Constantin Weisser Ouail Kitouni, Benjamin Nachman and Mike Williams. Enhancing searches for resonances with machine learning and moment decomposition. J.High Energ.Phys.2021,70(2021), arXiv:2010.09745, (2019).