ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

Scuola di Scienze
Dipartimento di Fisica e Astronomia
Corso di Laurea Magistrale in Fisica

Prototype of Machine Learning "as a Service" for CMS Physics in Signal vs Background discrimination

Relatore:
Prof. Daniele Bonacorsi

Correlatori:
Dott. Valentin Kuznetsov
Prof. Andrea Castro

Presentata da:
Luca Giommi

Anno Accademico 2016/2017

Start by doing what's necessary;
then do what's possible; and suddenly
you are doing the impossible.
[Saint Francis of Assisi]

Contents

Sommario

Gli esperimenti LHC al CERN raccolgono e analizzano una grande mole di dati. Il successo di queste sfide scientifiche è garantito da un enorme potenza di calcolo e capacità di storage, che opera su reti ad alta performance, attraverso complessi modelli di calcolo sull'infrastruttura LHC Computing Grid. Attualmente in presa dati (Run-2), LHC ha un chiaro e ambizioso programma sperimentale per i prossimi decenni. Si prevedono cospicui investimenti nell'hardware dei rivelatori, ma anche in ricerca e sviluppo in software e computing, che permettano di acquisire, gestire, processare e analizzare l'ammontare di dati che verrà prodotto nell'era di High-Luminosity LHC (HL-LHC).

L'ascesa dell'Intelligenza Artificiale - associata alla crescita dei Big Data, al progresso tecnologico e all'impatto della democratizzazione delle risorse mediante una efficiente allocazione a costi accessibili con soluzioni cloud - pone nuove sfide. Al tempo stesso, offre tecnologie estremamente promettenti, non solo per il mondo commerciale, ma anche per gruppi scientifici come ad esempio gli esperimenti di fisica ad alta energia. Machine Learning e Deep Learning stanno rapidamente evolvendo verso approcci per caratterizzare e descrivere i dati con il potenziale di cambiare radicalmente come i dati stessi vengono ridimensionati e analizzati, anche ad LHC.

Questa tesi ha lo scopo di contribuire alla costruzione di una soluzione Machine Learning "as a service" per le esigenze della fisica di CMS, ovvero un end-to-end data service per fornire modelli di Machine Learning allenati al software framework di CMS. Verso questo obiettivo, il lavoro di questa tesi contribuisce in primo luogo a un prototipo di questa infrastruttura, e in secondo luogo ad uno specifico caso d'uso: la discriminazione Segnale / Fondo nello studio dei decadimenti completamente adronici del quark top in CMS, effettuato con tecniche Machine Learning scalabili.

Il Capitolo 1 presenta una panoramica sul Modello Standard nella fisica ad alte energie, sull'acceleratore LHC e i suoi esperimenti.

Il Capitolo 2 presenta una beve panoramica del Computing Model di CMS.

Il Capitolo 3 introduce i concetti e le terminologie del Machine Learning, offrendo una panoramica sulle sue applicazioni (attuali e future) nella fisica ad alte energie e in particolare in CMS.

Il Capitolo 4 presenta il risultato originale di questo lavoro. Viene presentata l'architettura e le componenti della nuova infrastruttura Machine Learning "as a service". Viene quindi discusso come è stato costruito il modello - dalla preparazione e validazione dei dati, all'allenamento e test del modello - per questo caso d'uso. Questo modello è stato applicato alla sfida di discriminare Segnale dal Fondo nell'analisi del decadimento completamente adronico del quark top in CMS. Infine, viene presentato il prototipo funzionante di questa infrastruttura funzionante.

Abstract

Big volumes of data are collected and analysed by LHC experiments at CERN. The success of this scientific challenges is ensured by a great amount of computing power and storage capacity, operated over high performance networks, in very complex LHC computing models on the LHC Computing Grid infrastructure. Now in Run-2 data taking, LHC has an ambitious and broad experimental programme for the coming decades: it includes large investments in detector hardware, and similarly it requires commensurate investment in the R&D in software and computing to acquire, manage, process, and analyse the shear amounts of data to be recorded in the High-Luminosity LHC (HL-LHC) era.

The new rise of Artificial Intelligence - related to the current Big Data era, to the technological progress and to a bump in resources democratization and efficient allocation at affordable costs through cloud solutions - is posing new challenges but also offering extremely promising techniques, not only for the commercial world but also for scientific enterprises such as HEP experiments. Machine Learning and Deep Learning are rapidly evolving approaches to characterising and describing data with the potential to radically change how data is reduced and analysed, also at LHC.

This thesis aims at contributing to the construction of a Machine Learning "as a service" solution for CMS Physics needs, namely an end-to-end data-service to serve Machine Learning trained model to the CMS software framework. To this ambitious goal, this thesis work contributes firstly with a proof of concept of a first prototype of such infrastructure, and secondly with a specific physics use-case: the Signal versus Background discrimination in the study of CMS all-hadronic top quark decays, done with scalable Machine Learning techniques.

Chapter 1 presentes an overvew of Standard Model in HEP and the LHC accelerator and experiments.

Chapter 2 presents a summary of the CMS Computing Model.

Chapter 3 introduces Machine Learning concepts and terminology, offers an overview about its application (current and future) in HEP, and specifically in CMS.

Chapter 4 presents the original results of this work: it discusses the architecture and components of the new Machine Learning "as a service" infrastructure, how a Machine Learning model was built - from data preparation and validation

to model training and testing - for this use-case, how it was applied to the challenge of S/B discrimination in the all-hadronic top quark decay analysis, and finally presents a proof-of-concept of the current working prototype of such infrastructure.

Chapter 1

High Energy Physics at LHC

1.1 Overview of the LHC accelerator

The Large Hadron Collider (LHC) [1, 2] is a two-ring particle accelerator and collider (Figure 1.1) built by the European Organization for Nuclear Research (CERN). It is located beneath the Franco-Swiss border near Geneva in Switzerland where the Large Electron-Positron collider (LEP) previously existed [3]. The purpose of the LHC is to give scientists an experimental apparatus that would enable to advance the knowledge in the high energy physics such as experimentally looking for theoretically predicted particles, test the validity of supersymmetric theories, search for particles that would indicate the existence of dark matter, etc. On July 4th, 2012, the discovery of the Higgs boson, predicted by the Standard Model [4], was announced as a major milestone in the history of LHC.

LHC consists of a 27 km long circular ring, designed to accelerate protons and heavy ions at high energies. LHC is characterised by two accelerated beams that travel in opposite directions inside different parts of the same pipe at ultrahigh vacuum.

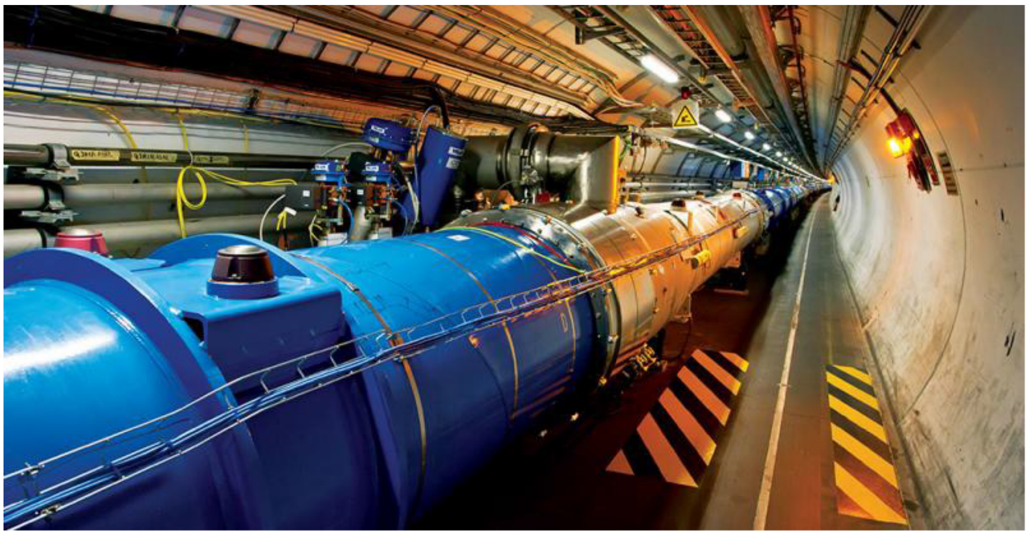


Figure 1.1: The LHC tunnel.

The acceleration process for protons is done in five steps (see Figure 1.2). Initially, hydrogen atoms are ionized in order to produce protons and then injected in a linear accelerator called LINAC 2. When protons reach an energy of 50 MeV they subsequently enter another component called Booster, where their energy goes up to 1.4 GeV. They then enter the Proton Synchrotron (PS) where 277 electromagnets push the protons to 99,9% the speed of light: at this point, each proton has an energy of 25 GeV. Proton bunches are then accelerated in the Super Proton Synchrotron (SPS), a circular particle accelerator with a circumference of 7 km. After protons have reached an energy of 450 GeV, they are injected into the LHC in two separate pipes in which they move in opposite directions; here the particles can be accelerated up to their maximum designed energy of 6.5 TeV. Currently this is the maximum energy of each proton beam reachable after the last upgrade, completed in March 2015: this was the start of Run-2, the second LHC data taking period. The two LHC channels intersect in four caverns, where the four detectors are deployed. Here protons can collide and the products of the collisions can be revealed.

The vacuum system is necessary so the particles do not lose energy in the acceleration process due to impacts with the molecules that constitute air. The LHC vacuum system is made up of three individual vacuum systems: the insulation vacuum for cryomagnets, the insulation vacuum for helium distribution, and the beam vacuum.

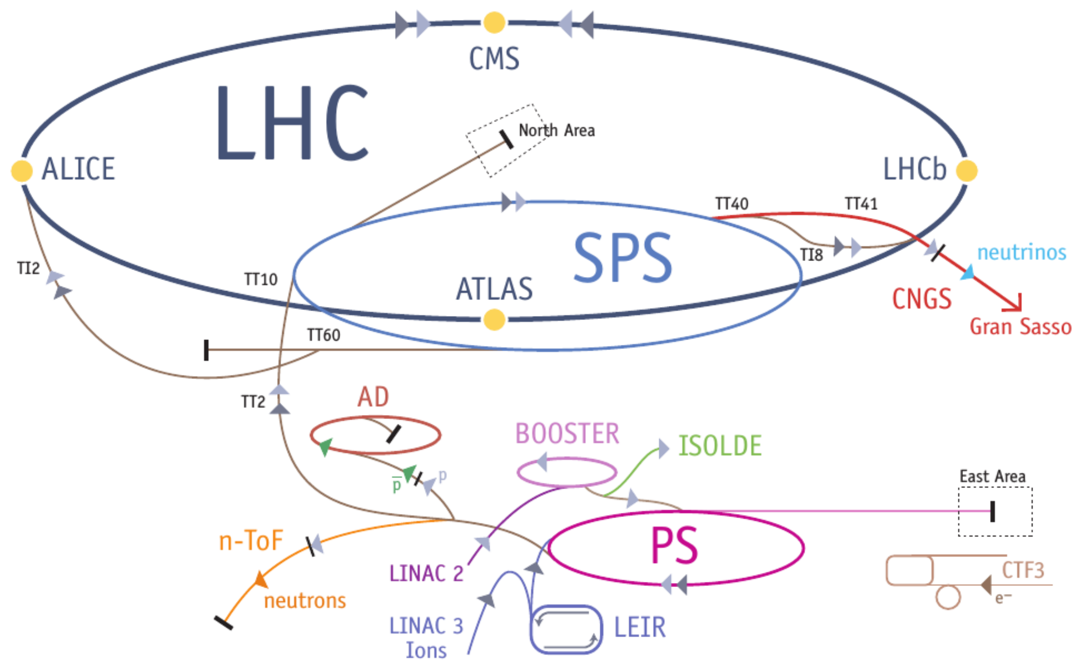


Figure 1.2: The LHC accelerator complex at CERN.

Electromagnets

There is a large set of magnets in the LHC (dipoles, quadrupoles, sextupoles, octupoles, decapoles), for a total of about 9600 magnets, 1232 being main dipoles and

392 lattice quadrupoles. Each type of magnet contributes to optimizing the particles' trajectory: dipoles have the function to maintain the beams in their circular orbit, while quadrupoles are used to focus the beam down to the smallest possible size at the collision points, thereby maximizing the chance of two protons smashing head-on into each other (Figure 1.3). The dipoles of the LHC represent one of the most relevant technological challenge in the LHC design. In a proton accelerator like LHC, the maximum energy that can be achieved is directly proportional to the strength of the dipole field, given a specific acceleration circumference. At the LHC the dipole magnets are superconducting electromagnets, which are able to provide a magnetic field of 8.3 T over their length. No solutions could have been designed using 'warm' magnets instead of superconducting ones. The LHC dipoles use niobium-titanium (NbTi) cables, which become superconducting below a temperature of 10 K (i.e. they conduct electricity without resistance). LHC operates at 1.9 K, even lower than the temperature of deep outer space, 2.7 K).

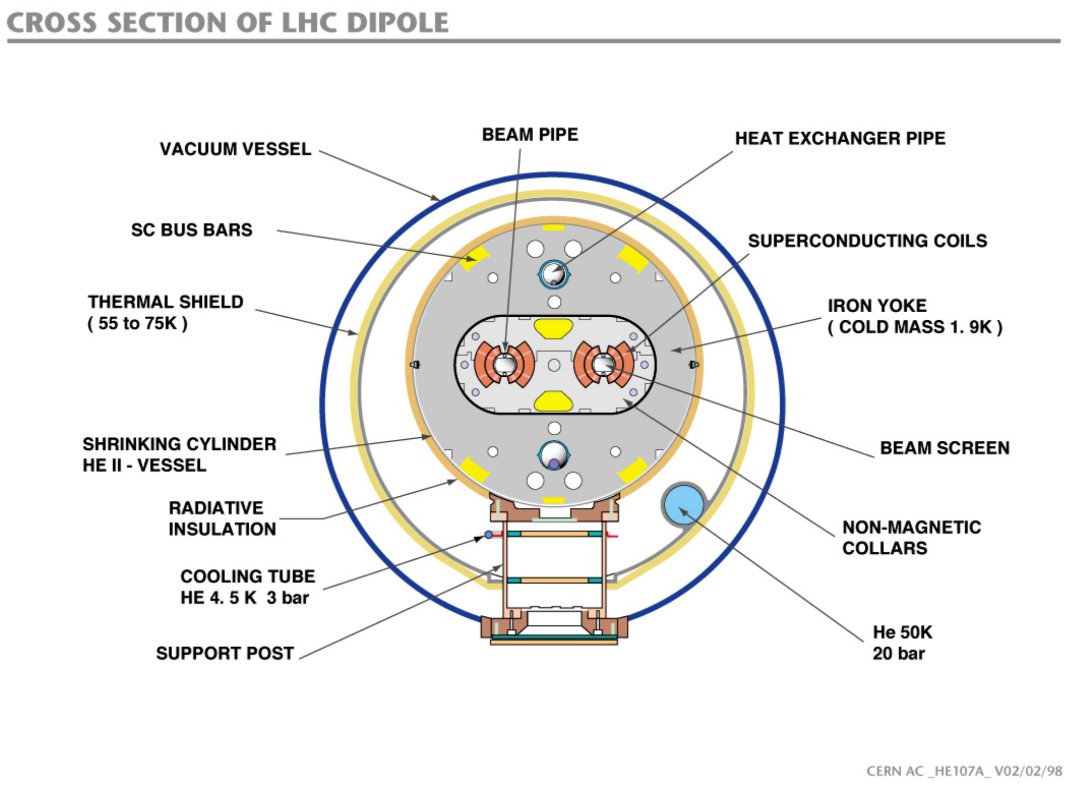

CROSS SECTION OF LHC DIPOLE
VACUUM VESSEL
BEAM PIPE
HEAT EXCHANGER PIPE
SC BUS BARS
SUPERCONDUCTING COILS
THERMAL SHIELD (55 to 75K)
IRON YOKE (COLD MASS 1. 9K)
SHRINKING CYLINDER HE II - VESSEL
BEAM SCREEN
RADIATIVE INSULATION
NON-MAGNETIC COLLARS
COOLING TUBE HE 4. 5 K 3 bar
He 50K 20 bar
SUPPORT POST
CERN AC _HE107A_ V02/02/98

Figure 1.3: Cross section of the LHC dipole.

Radiofrequency cavities

Along LHC length there are radiofrequency (RF) cavities, namely metallic chambers containing an electromagnetic field. Their primary purpose is to accelerate charged particles and to divide protons in packages while keeping them grouped. To enable an RF cavity to accelerate particles, an RF power generator supplies an electromagnetic field. The RF cavity is molded to a specific size and shape so that electromagnetic waves become resonant and build up inside the cavity (see Figure 1.4). Charged particles passing through the cavity experience the overall force and direction of the resulting electromagnetic field, which transfers energy to push them forward along the accelerator. The field in an RF cavity is made to oscillate (switch direction) at a given frequency, so timing the arrival of particles is important. At the LHC, each RF cavity is tuned to oscillate at 400 MHz. The ideally timed proton, with exactly the right energy, will see zero accelerating voltage when the LHC is at full energy. Protons with slightly different energies arriving earlier or later will be accelerated or decelerated so that they stay close to the energy of the ideal particle. In this way, the particle beam is sorted into discrete packets called "bunches". During the energy-boosting process the protons in the bunches shift collectively to experience an overall acceleration on each passage through the cavities, picking up the energy needed to keep up with the increasing field in the LHC powerful magnets. Top energy is reached in around 15 minutes, the bunches having passed the cavities around 1 million times. The 16 RF cavities on the LHC are housed in four cylindrical refrigerators called "cryomodules" – two per beam – which keep the RF cavities working in a superconducting state, without losing energy to electrical resistance.

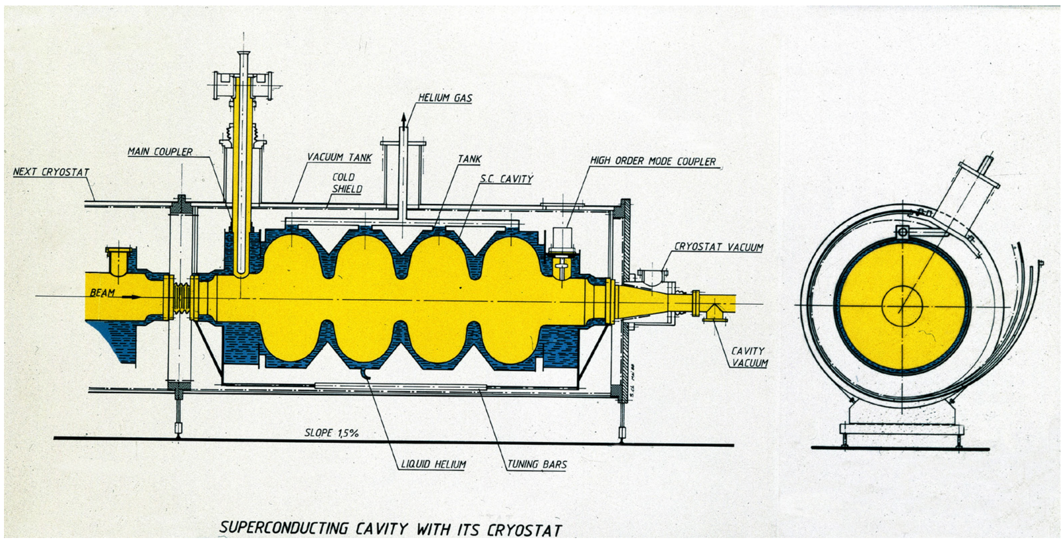


Figure 1.4: Schematic drawing of a superconducting cavity.

At the LHC, during operation conditions, each proton beam is divided into 2808 bunches, each of them composed by about 10^{11} protons. Their dimension is not constant along the circumference when they travel far from the collision point: their size is about few centimeters length and 1 millimeter width. In the collision points, the protons are collimated and the packages are compressed to about 16 nm. At full luminosity, the packages are separated by about 7m (in time this is 25 ns).

LHC technical parameters

An important parameter which characterizes a particle accelerator is the machine luminosity ($\mathcal{L}$) defined as:

$$\mathcal{L} = \frac{f_{rev} n_b N_b^2}{4\pi \epsilon_n \beta^*} H \qquad (1.1)$$

where f_{rev} is the revolution frequency, n_b is the number of bunches per beam, N_b is the number of particles in each colliding beam, ϵ_n is the normalized transverse beam emittance, β^* is the beta function at the collision point and H the hourglass factor. The number of events that occur each second is:

$$N_{event} = \mathcal{L}\sigma_{event} \tag{1.2}$$

The ATLAS and CMS experiments run at high machine luminosity $\mathcal{L} = 10^{34}\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$, whereas ALICE and LHCb operates, respectively, at a lower luminosity $\mathcal{L} = 10^{27}\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$ and $\mathcal{L} = 10^{32}\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$.

Following in Table 1.1 are listed some of the most relevant parameters of LHC.

Table 1.1: LHC main technical parameters.

Quantity (measurement unit)	**value**
Circumference (m)	26 659
Magnets working temperature (K)	1.9
Number of magnets	9593
Number of dipoles	1232
Number of quadrupoles	392
Number of radiofrequency cavities	16
Protons nominal energy (TeV)	6.5
Maximum intensity of the magnetic field (T)	8.33
Instantaneous luminosity record ($\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$)	2.06×10^{34}
Number of proton bunches per beam	2808
Number of protons per bunch	1.1×10^{11}
Minimum distance between bunches (m)	~7
Revolutions per second	11 245
Collisions per second (millions)	600

1.2 The experiments at LHC

Four major experiments at the LHC use detectors to analyse the myriad of particles produced by collisions in the accelerator iself. These experiments are run by large collaborations of scientists from Institutes all over the world. Each experiment is characterized by different detectors and subdetectors. The four main experiments are called ALICE, ATLAS, CMS and LHCb. Among other smaller experiments, TOTEM, LHCf and MoEDAL can be listed. The four main experiments are installed each in a cavern built around the four collision points.

ALICE

The Large Ion Collider Experiment (ALICE, Figure 1.5) [5, 6] is a general-purpose, heavy-ion detector devoted to study the strong interaction, in particular quark-gluon plasma at extreme values of energy density and temperature during the collision of heavy nuclei (Pb). Collisions in the LHC generate temperatures more than 100000 times hotter than the centre of the Sun. For part of each year, the LHC provides collisions between lead ions, recreating in the laboratory conditions similar to those just after the Big Bang. Under these extreme conditions protons and neutrons "melt", freeing the quarks from their bonds with the gluons. This is a state of matter called "quark-gluon plasma". ALICE studies it as it expands and cools, observing how it progressively gives rise to the particles that constitute the matter of our universe today. In order to study quark-gluon plasma, the ALICE collaboration uses a 10,000-tonne detector which is 26 m long, 16 m high, and 16 m wide. The detector sits in a vast cavern 56 m below ground close to the village of St Genis-Pouilly in France.

The ALICE detector is constituted of several subdetectors. Moving from the beam pipe outwards, there are the following subdetectors: the Inner Tracking System, the Time Projection Chamber, the Time Of Flight, the Ring Imaging Cherenkov, the Transition Radiation Detector, the Electromagnetic Calorimeters and the Muon Spectrometer. ALICE exploits several different subdetectors to perform particle identification: the Time Of Flight (TOF), which is able to separate π/K and K/p for $p_T < 1\,GeV/c$ with a 3σ precision; the High Momentum Particle Identification, which is exclusively dedicated to particle identification of hadrons with $p_T > 1\,GeV/c$; and the Transition Radiation Detctor, whose major purpose is to identify electrons of $p_T > 1\,GeV/c$. Furthermore, close to the beam pipe, the ALICE detector has the Inner Tracking System based on silicon pixels and microstrips, useful to reconstruct heavy flavours decays. The most complex subdetector in ALICE is the Time Projection Chamber (TPC) that provides informations both on the particles trajectories and the energy deposited. An additional information of the energy deposited is provided by two electromagnetic calorimeters.

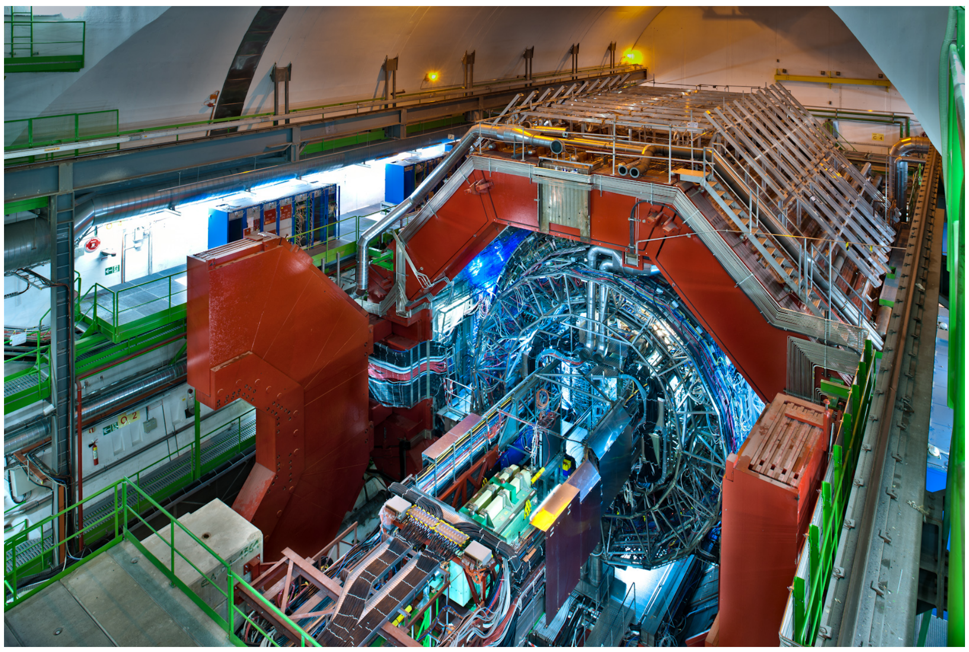


Figure 1.5: The ALICE detector.

ATLAS

A Toroidal LHC ApparatuS (ATLAS, Figure 1.6) [7, 8] is an experiment whose main purpose is to investigate new physics beyond the Standard Model up to the existence of dark matter and extra dimensions, exploiting the extremely high energy at the LHC.

The 7000-tonne ATLAS detector is the largest volume particle detector ever constructed: it is 46 m long, 25 m high and 25 m wide. It sits in a cavern 100 m below ground near the main CERN site, close to the village of Meyrin in Switzerland. The detector is made by four main layers: the Magnet system, that bends the trajectories of charged particles; the Inner Detector, which measures the path of charged particles; the Calorimeters, which identify photons, electrons, and hadrons; the Muon Spectrometer, which recognises the presence of muons.

Inside the solenoid (which provides a 2T axial field), the electromagnetic calorimeter, based on Liquid Argon is deployed. Outside the solenoid, the hadronic calorimeter is installed: it is a sampling calorimeter, in which steel is used as an absorber and the chosen active material is scintillating tiles.

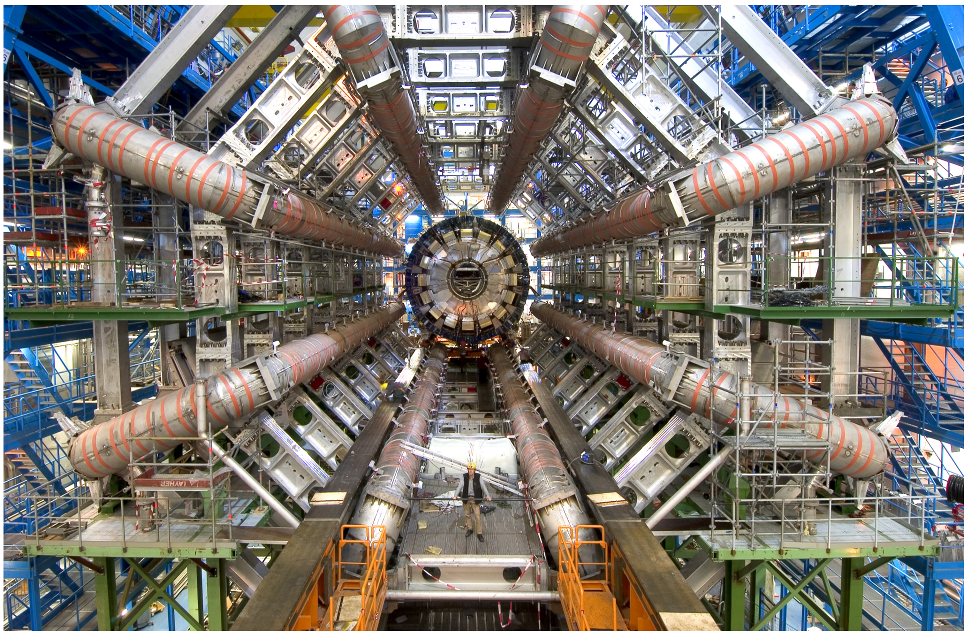


Figure 1.6: The ATLAS detector.

CMS

The Compact Muon Solenoid (CMS, Figure 1.7) [9, 10] is a general-purpose detector at the Large Hadron Collider (LHC). It has a broad physics programme ranging from studying the Standard Model, including the Higgs boson, to searching for extra dimensions and particles that could make up dark matter. Although it has the same scientific aims as the ATLAS experiment, it uses different technical solutions and a different magnet-system design. The CMS detector is built around a huge solenoid magnet. This takes the form of a cylindrical coil of superconducting cable that generates a field of 4 T, that is about 100,000 times the magnetic field of the Earth. The field is confined by a steel "yoke" that forms the bulk of the detector's 14,000-tonne weight. An unusual feature of the CMS detector is that instead of being built in-situ, like the other giant LHC detectors, it was constructed in 15 sections at ground level before being lowered into an underground cavern near Cessy in France and reassembled. The whole detector is 21 metres long, 15 metres wide and 15 metres high.

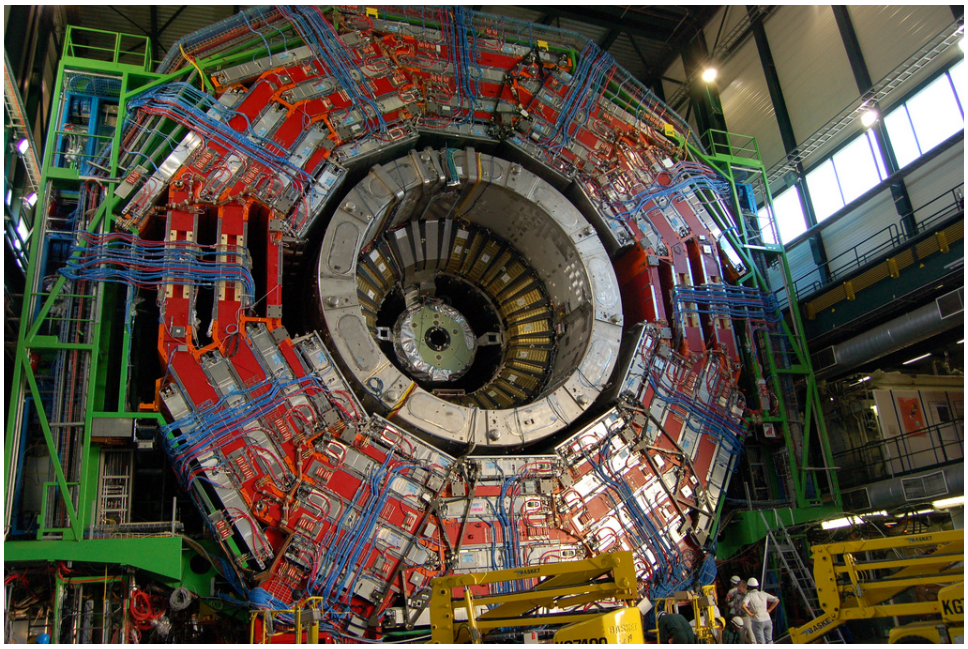


Figure 1.7: The CMS detector.

LHCb

The Large Hadron Collider beauty (LHCb, Figure 1.8) [11, 12] experiment is specialized in investigating the slight differences between matter and antimatter by studying the "beauty quark". In particular, the values of the Cabibbo Kobayashi Maskawa Matrix are measured in order to better understand the CP violation, which is strictly linked with the concentration of matter and antimatter in the universe. Instead of surrounding the entire collision point with an enclosed detector as ATLAS and CMS do, the LHCb experiment uses a series of subdetectors to detect mainly forward particles, those thrown forward by the collision in one direction. The first subdetector is placed close to the collision point, with the others following one behind the other over a length of 20 metres. Moving from the interaction point outwards, the LHCb detector is made of the following subdetectors: the vertex locator VELO, a first aerogel Ring Imaging Cherenkov, a silicon Trigger Tracker, the Magnet followed by three Tracking stations, a second aerogel Ring Imaging Cherenkov, a Scintillator Pad Detector and Preshower, an Electromagnetic Calorimeter, a Hadronic Calorimeter and lastly a series of Muon chambers. The ALICE detector is 5600-tonne LHCb detector is made up of a forward spectrometer and planar detectors. It is 21 metres long, 10 metres high and 13 metres wide, and sits 100 metres below ground near the village of Ferney-Voltaire, France.

Figure 1.8: The LHCb detector.

1.3 The CMS experiment

A more detailed description of the subdetectors of the CMS experiment - which is the experiment in which this thesis work was done - is given in this section.

CMS [10], as introduced in the previous section, is a general-purpose detector at the LHC. It is designed to investigate a wide range of Physics, including the search for the Higgs boson, extra dimensions and particles that could reveal the presence of dark matter.

CMS is built around a huge superconducting solenoid and it is made by different layers of detectors that identify the different particles and allow to build up a picture of the collision events. The detector acts like a giant filter where each layer is designed to stop, track or measure a different type of particle emerging from proton-proton and heavy-ion collisions. In order to work correctly (that is to detect and to identify particles), CMS needs:

- a high resolution vertex detector, to give an accurate reconstruction of the decay vertex of the heaviest particles;
- a high quality central tracking system, to give accurate momentum measurements of charged particles;
- a high resolution method to detect and measure electrons and photons (through an electromagnetic calorimeter);
- a hadron calorimeter, designed to detect particles from the hadronic shower;
- a high performance system to detect and measure muons.

Therefore the detector is made up of six concentric layers. If we start from the collision point and go outwards, we find (see Figure 1.9 and 1.10): the vertex detector, the Tracker, the Electromagnetic Calorimeter (ECAL), the Magnet, the Hadronic Calorimeter (HCAL) and the Muon detector.

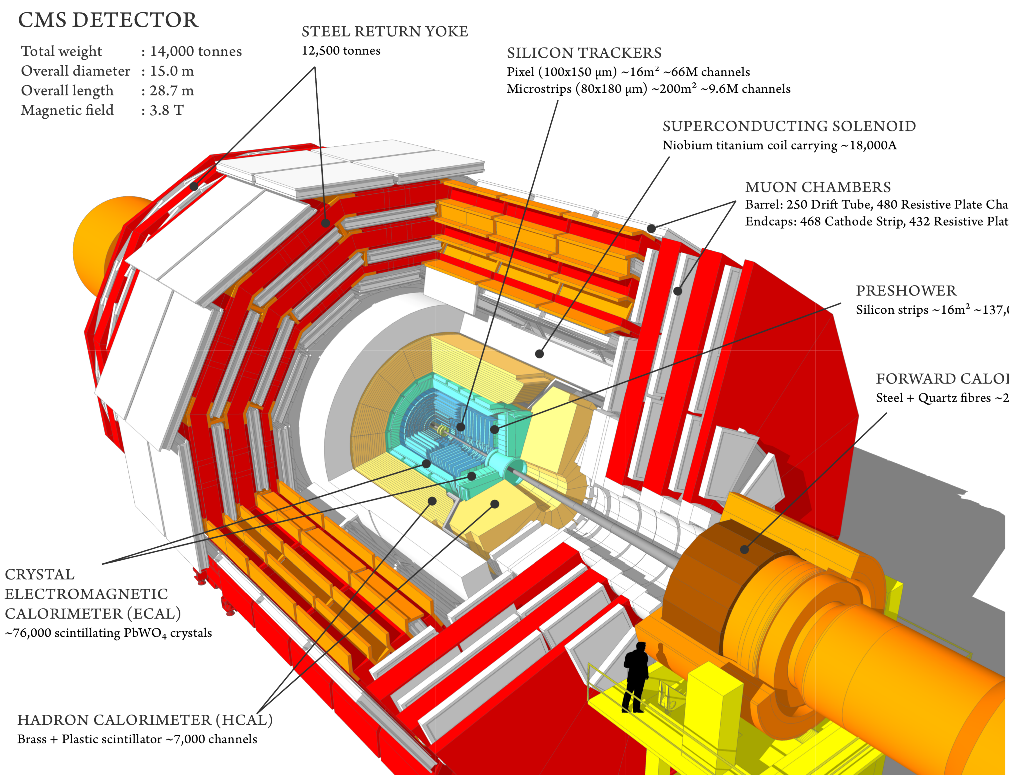

CMS DETECTOR
Total weight : 14,000 tonnes
Overall diameter : 15.0 m
Overall length : 28.7 m
Magnetic field : 3.8 T
STEEL RETURN YOKE
12,500 tonnes
SILICON TRACKERS
Pixel (100x150 μm) ~16m² ~66M channels
Microstrips (80x180 μm) ~200m² ~9.6M channels
SUPERCONDUCTING SOLENOID
Niobium titanium coil carrying ~18,000A
MUON CHAMBERS
Barrel: 250 Drift Tube, 480 Resistive Plate Chambers
Endcaps: 468 Cathode Strip, 432 Resistive Plate Chambers
PRESHOWER
Silicon strips ~16m² ~137,000 channels
FORWARD CALORIMETER
Steel + Quartz fibres ~2,000 Channels
CRYSTAL ELECTROMAGNETIC CALORIMETER (ECAL)
~76,000 scintillating PbWO4 crystals
HADRON CALORIMETER (HCAL)
Brass + Plastic scintillator ~7,000 channels

Figure 1.9: Pictorial view of a section of the CMS detector.

Particles from the collision point first meet a pixel detector and then a tracker, made entirely of silicon, that detect their positions as they move through the detector, thus allowing to measure their momentum. Outside the tracker, calorimeters measure the energy of passing particles. In the momentum measurement the particles should interact with the detector as little as possible, whereas the calorimeters are specifically designed to stop and absorb particles. Additionally muon tracks are measured by four layers of ad-hoc detectors, while the neutrinos escape from CMS undetected, although their presence can be indirectly inferred from the missing transverse energy of the event.

The cylindrical structure of the detector forms the basis of the event representations and is also reflected in the processed event data. For each event a location in cylindrical coordinates is recorded. The origin of the corresponding coordinate system is the interaction point. The direction of the beam defines the z-axis and the $x - y$ plane lies transverse to it. In particular, the x-axis forms a line from the interaction point to the center of the LHC ring with the y-axis perpendicular to it, pointing upwards. In practice, mostly cylindrical coordinates (r, ϕ, θ) are used as opposed to the cartesian ones to account for the structure of the detector. For this, the polar angle θ (angle between the direction of flight of the particle and the beam pipe) and the azimutal angle ϕ (angle around beam axis) are recorded. The previous definitions allow to introduce a series of quantities:

- particle transverse momentum: $p_T = \sqrt{p_x^2 + p_y^2}$

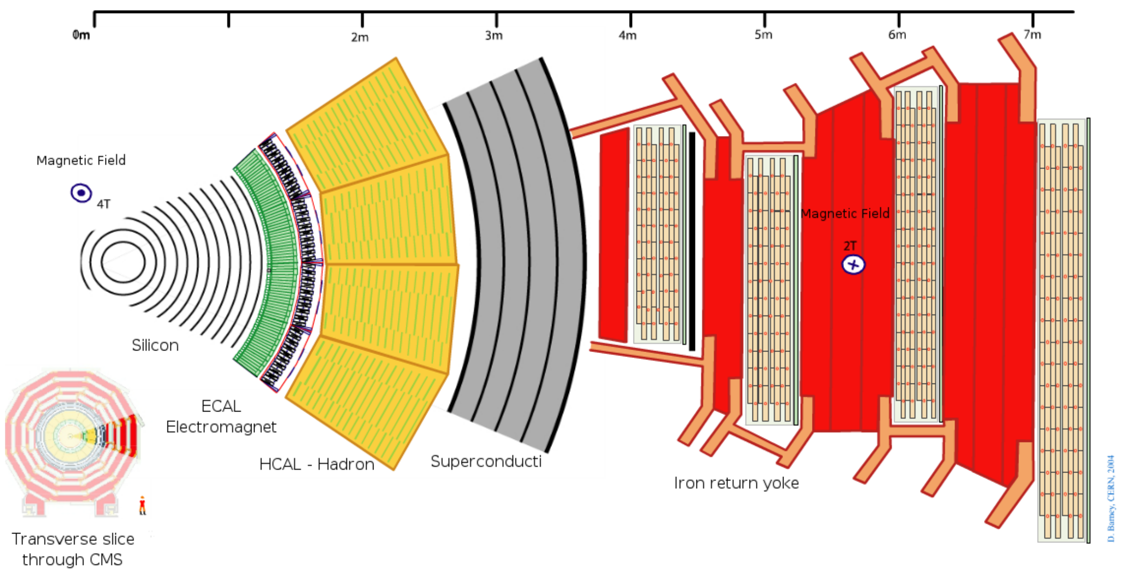


Figure 1.10: Schematic structure of the CMS subdetectors in section.

- transverse energy: $E_T = E\sin(\theta)$
- missing transverse energy (MET): $E_t^{missing} = \sum_i \vec{p}_T$ (summed over all the produced particles)
- rapidity (Lorentz invariant): $y = \frac{1}{2}\ln\frac{E+p_z}{E-p_z}$
- pseudo-rapidity: $\eta = -\ln\left(\frac{\theta}{2}\right)$

The tracking system allows to cover the pseudorapidity region $|\eta| < 2.5$, the calorimeter covers the range $|\eta| < 3.0$, forward calorimeters extend the coverage up to $|\eta| < 5.0$ and the muon detectors cover the range $|\eta| < 2.4$.

Magnet

The CMS magnet is a solenoid, that is a coil of superconducting wires, and creates a magnetic field when electricity flows through it. In CMS the solenoid has an overall length of 13 m and a diameter of 7 m and creates a magnetic field of about 4 T. It is the largest magnet of its type ever constructed and allows to place the tracker and the calorimeter detectors inside the coil, resulting in a detector that is overall very "compact", compared to detectors of similar weight.

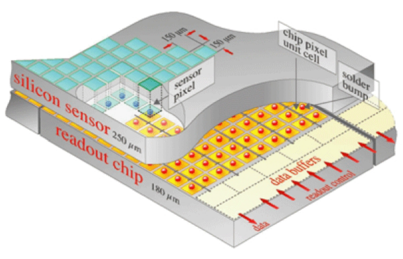
Tracker

One method to evaluate the momentum of a particle is using a magnetic field that bends charged particles. If we use the signal that comes from the tracker it is possible to reconstruct the bent path and then the momentum of the particle. The tracker can reconstruct the path of high-energy muons, electrons and hadrons as well as it can see tracks coming from the decay of very short-lived particles such as b-quarks. Moreover, it needs to detect particle paths accurately and to disturb particles as little as possible. Just a few measurement points are used to reconstruct the path of a particle and each measurement is accurate to 10 µm.

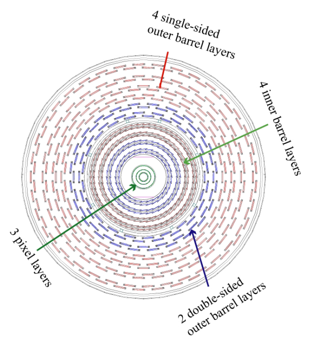
The tracker [13] is also the most inner layer of the detector and so it receives the highest flux of particles: the construction materials were therefore carefully chosen to be radiation hardy. The final design consists of a tracker entirely made of silicon: internally, at the core of the detector, there are three pixels levels that form the vertex detector and, after these, particles pass through ten microstrip detectors levels, up to reach a distance of 130 cm from the beam pipe (see Figure 1.11b). As particles travel through the tracker, the pixels and microstrips produce tiny electric signals that are amplified and detected.

The pixel detector (about the size of a shoebox) contains 65 million pixels, allowing to track the path of particles emerging from the collision with extreme accuracy. It is the detector closest to the beam pipe, with cylindrical layers at 4, 7 and 11 cm and disks at both ends. As previously said, it is crucial in reconstructing tracks of very short-lived particles; however, being so close to the collision means that the number of particles passing through is huge: indeed, the rate at 8 cm from the beam line amounts to about 10 million particles per square centimetre per second. Despite this huge number of particles passing through, the pixel detector is able to distinguish and to reconstruct the tracks. When a charged particle passes through, it gives enough energy to electrons to be ejected from the silicon atoms, creating electron-hole pairs. Each pixel uses electric current to collect these charges on the surface resulting in a small electric signal, which is then amplified.

After the pixels and on the way out of the tracker, particles pass through ten layers of silicon strip detectors, reaching out to a radius of 130 centimetres; this part of the tracker contains 15,200 highly sensitive modules with a total of 10 million detector strips that are read by 80,000 microelectronic chips. Each module consists of three elements: a set of sensors, its mechanical support structure and the readout electronics (see Figure 1.11a).



(a) CMS Silicon pixel detector


4 single-sided outer barrel layers
4 inner barrel layers
3 pixel layers
2 double-sided outer barrel layers

(b) Tracker layers seen by normal plane to beams

Figure 1.11: Schematic views of the CMS Tracker design.

The outer muon chamber system, provides a second measurement for the momentum of the muon which can be combined with the measurement obtained by the tracker in order to improve the overall resolution. This feature allows to reconstruct muons with an efficiency above 98%. Charged hadrons with a

$p_T > 10\,GeV$ are reconstructed with an efficiency close to 95% whereas hadrons with $1\,GeV < p_T < 10\,GeV$ are reconstructed with an efficency higher than 85%. The efficiency for high energy electrons is better than 90%.

ECAL

The Electromagnetic Calorimeter (ECAL) measures the energy of photons and electrons [14, 15]. It comprises lead tungstate crystals ($PbWO_4$), that are made primarily by metal and they are heavier than stainless steel but, with a touch of oxygen in this crystalline form, they are highly transparent and "scintillate" when electrons and photons pass through them. This means that crystals produce light in proportion to the particle's energy. The lead tungstate has several desirable properties compared to other crystal scintillators used in other experiments: short radiation length, relatively high critical energy, fast light emission. The main drawback of this material is the low light output emission compared to other inorganic scintillators; for this reason, the electromagnetic calorimeter needs very large photodiodes. Photodetectors are glued onto the back of each of the crystals to detect the scintillation light and convert it to an electrical signal that is amplified and processed.

The ECAL, made up by a barrel section and two "endcaps", forms a layer between the tracker and the HCAL. The cylindrical "barrel" consists of 61,200 crystals formed into 36 "supermodules", each of them weighs approximately 3 tonnes and containing 1700 crystals. The flat ECAL endcaps seal off the barrel and both are made up of almost 15,000 additional crystals. For extra spatial precision, the ECAL also contains preshower detectors that sit in front of the endcaps. These allow CMS to distinguish between single high-energy photons (often signature of interesting physics) and the less interesting close pairs of low-energy photons.

The energy resolution of the electromagnetic calorimeter can be expressed as:

$$\left(\frac{\sigma}{E}\right)^2 = \left(\frac{a}{\sqrt{E}}\right)^2 \oplus \left(\frac{\sigma_n}{E}\right)^2 \oplus c^2$$

where a accounts for the stochastic contribution, σ_n the noise and c^2 is a constant. The stochastic term describes the fluctuations in the shower development. The noise term includes contributions from electronic noise (dominant at low energy) and energy pile-up (dominant at high luminosity). For E expressed in GeV, the constants for the energy resolution for electrons in beam tests are: $a = 2.8\%$, $\sigma_n = 12\%$, $c^2 = 0.3\%$ [16].

HCAL

The Hadron Calorimeter (HCAL) [17] measures the energy of hadrons, particles made of quarks and gluons (like protons, neutrons, pions and kaons); additionally it provides an indirect measurement of the presence of non-interacting, uncharged particles such as neutrinos.

The detection and identification of these particles is important as they can tell us if new particles such as the Higgs boson or supersymmetric particles (much heavier versions of the standard particles we know) may have formed.

As these particles decay, they may produce new particles that do not leave trace of their presence in any part of the CMS detector: to spot these the HCAL must be hermetic, that is able to capture, as much as possible, every particle emerging from the collisions. In this way if we see particles shot out with an imbalance in the momentum and energy (measured in the sideways transverse direction relative to the beam line), we can deduce that "invisible" particles, such as neutrinos, are produced.

The CMS HCAL, as a sampling calorimeter, is made by alternating layers of "absorber" (brass) and fluorescent "scintillator" (plastic) materials that produce a rapid light pulse when the particle passes through. Special optic fibres collect up this light and feed it into readout boxes where photodetectors amplify the signal. When the amount of light in a given region is summed up over many layers of tiles in depth, called a "tower", this total amount of light yields a measure of a particle's energy.

The achievable energy resolutions vary across the various regions of HCAL. Nonetheless, these resolutions are described by this same formula:

$$\left(\frac{\sigma}{E}\right) = \frac{a}{\sqrt{E}} + c$$

where E is the deposited energy measured in GeV; c is equal to 5%; a is equal to 65% in the barrel, 85% in the endcap and 100% in the forward HCAL.

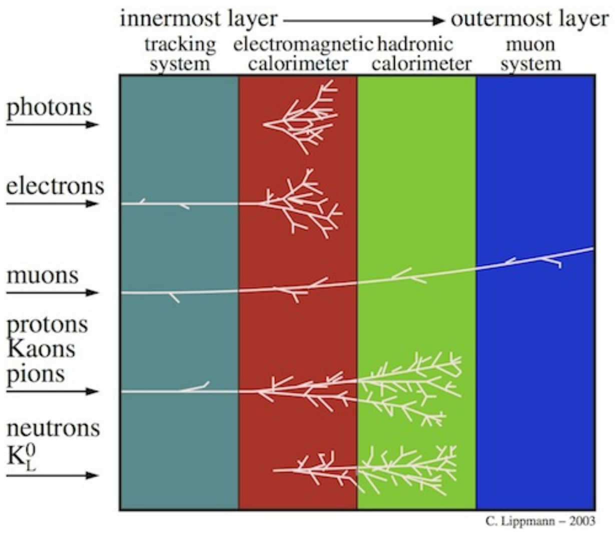
Muon detectors

As the acronyms expanded "Compact Muon Solenoid" [18] suggests, detecting muons is one of CMS most important tasks. Muons are charged particles like electrons and positrons but 200 times heavier in mass; we expect them to be produced in the decay of a number of potential new particles i.e. one of the clearest "signatures" of the Higgs boson is its decay into four muons.

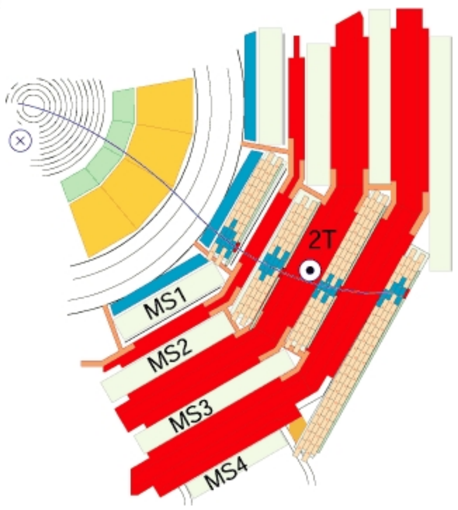
As muons can penetrate several metres of iron without interacting (mainly via multiple coulomb scattering), unlike most other particles they are not stopped by any of CMS calorimeters (Figure 1.12a). Therefore, chambers to detect muons are placed at the very edge of the detector where they are the only particles that might produce a signal.

The muon path is measured by fitting the hits from the four muon stations, which sit outside the magnet coil and they are interleaved with iron "return yoke" plates (shown in red in Figure 1.12b). These subdetectors trace the particle path by tracking particle positions through the multiple layers of each station, combined with tracker measurements. This yields a measurement of its momentum: we know that particles travelling with an higher momentum bend less in a magnetic field, and the CMS magnet is powerful enough to also bend the path of high-energy muons and better derive their momentum.

In total there are 1400 muon chambers: 250 Drift Tubes (DTs) chambers and 540 Cathode Strip Chambers (CSCs) track the particles position and contribute to the main muon trigger system, while 610 resistive plate chambers (RPCs) form a redundant trigger system, which quickly decides to keep the acquired muon data or



(a)



(b)

Figure 1.12: Paths of different particles traversing different CMS subdetectors (a) and pictorial representation of a muon traversing the CMS muon detectors (b)

not. Because of the different layers of detector and their heterogenity, the overall system is naturally robust and able to filter out background noise.

DTs and RPCs are arranged in concentric cylinders in the barrel region, while CSCs and RPCs make up the endcap disks that cover the ends of the barrel.

Data Acquisition and Trigger

When CMS is performing at its peak, about one billion proton-proton interactions will take place every second inside the detector. There is no way that data from all these events could be read out and recorded, and even if they could, most would be less likely to reveal interesting phenomena (e.g. they might be low-energy scattering collisions for instance, rather than energetic, head-on interactions.

Any LHC experiment needs a "trigger" system, able to select the potentially interesting events (such as those which may produce a Higgs particle or a supersymmetric particle) and reduce the rate to just a few hundred events per second, which can be read out and archived on storage for subsequent analysis.

However, because groups of protons collide 40 million times per second, there are only 25 ns before the next lot arrive. The solution is to store the data in pipelines that can retain and process information from many interactions at the same time. In order to avoid confusion between particles originating from two different events, the detectors must have very good time resolution and the signals from the millions of electronic channels must be synchronised so that they can all be identified as being from the same event.

The CMS trigger system [19] is organized in two levels. The first one is based on hardware (Level-1 (L1) Trigger) [20], and performs a data selection process in an extremely fast and completely automatic way: it selects data according to physics interest, for example an high energy value or a unusual combination of interacting particles in a event. This kind of trigger acts asynchronously in the phase of signals reception and performs the first event selection in CMS: it reduces the frequency of the acquired data to some hundreds of events per second. Events are then transferred and saved on storage for further processing, i.e. passed on to the next trigger step.

The second level of the CMS trigger is based on software (High-Level Trigger (HLT)) [21]. It is a software system implemented on a filter farm of thousands of commercial processors. The HLT has access to the complete read-out data and can therefore perform complex calculations similar to those made in the offline analysis software, if required for specially interesting events. The HLT contains many trigger paths, each corresponding to a dedicated trigger (such as a single-electron trigger or a 3-jets-with-MET trigger). A path consists of several steps (software modules), each module performing a well-defined task such as the reconstruction of physics objects (electrons, muons, jets, MET, etc).

The overall rate reduction capability is designed to be at least a factor of 10^6 for the combined L1 Trigger plus HLT. Ultimately, the triggered events come out of the CMS L1+HLT system at rate that is of the order of about 100 Hz.

1.4 LHC and CMS towards HL-LHC

Particle physics has a very broad experimental programme for the next few decades. Such programme aims at supporting the strategic goals designd by the high-energy physics community in fora such as the European Strategy for Particle Physics [22] for Europe and by the Particle Physics Project Prioritisation Panel [23] for the United States. Within this programme, the construction and operations of the High-Luminosity Large Hadron Collider [24, 25, 26] will be a major step forward.

The HL-LHC project comprised an upgrade of the current LHC, aiming at an in-depth investigation of the properties of the Higgs boson and its couplings to other particles. Within this program, the ATLAS and CMS collaborations will continue and extend their physics analyses, including measurements in the Higgs sector and searches for new physics Beyond the Standard Model (BSM). Any BSM discovery will lead to deeper explorations in the related physics, hoping to shed light e.g. on the nature of dark matter that makes up the majority of gravitational matter in the universe.

The HL-LHC is currently scheduled to start its data taking in 2026 (see Figure 1.13) and to operate through 2030s. In this time window, HL-LHC is designed to handle about 30 times more data than the LHC has currently collected (the total amount of LHC data collected so far by ATLAS and CMS is close to an exabyte). It is evident that approaches beyond simply scaling current solutions will be needed. This is discussed in more detail in 2.4.

1.5 The Standard Model and the Top quark

The SM [27, 28] explains how the basic building blocks of matter interact, governed by the fundamental forces. These basic building blocks of matter are

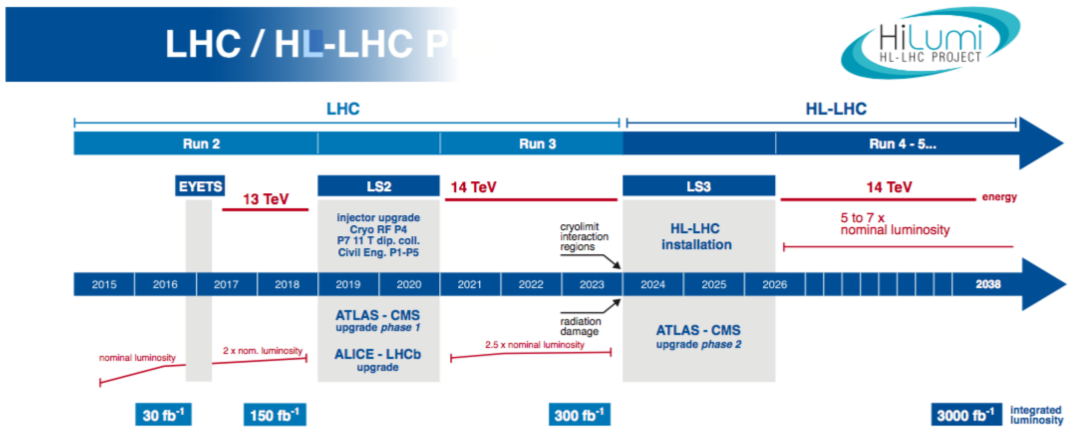


Figure 1.13: The current schedule for the LHC and HL-LHC upgrade and run. Currently, the start of the HL-LHC run is foreseen by mid 2026. The long shutdowns, LS2 and LS3, will be used to upgrade both the accelerator and the detector hardware.

the irreducibly smallest detectable particles, called elementary particles, while the fundamental forces the SM deals with are the strong, weak and electromagnetic ones.

The SM includes 12 elementary particles, divided into 6 quarks and 6 leptons, and the other 12 anti-particle counterparts. Quarks and leptons are related in pairs, or "generations". The lightest and most stable particles make up the first generation, whereas the heavier and less stable particles belong to the second and third generations. All stable matter in the universe is made from particles that belong to the first generation; any heavier particles quickly decay to the next most stable level. The six quarks are paired in the three generations – the "up quark" and the "down quark" form the first generation, followed by the "charm quark" and "strange quark", then the "top quark" and "bottom (or beauty) quark". Quarks also come in three different "colours" and only mix in such ways as to form colourless objects. The six leptons are similarly arranged in three generations – the "electron" and the "electron neutrino", the "muon" and the "muon neutrino", and the "tau" and the "tau neutrino". The electron, the muon and the tau all have an electric charge and a sizeable mass, whereas the neutrinos are electrically neutral and have very tiny mass.

Another important part of the SM is the fundamental forces. There are four fundamental in the universe: the strong force, the weak force, the electromagnetic force, and the gravitational force. They work over different ranges and have different strengths. Gravity is the weakest but it has an infinite range. The electromagnetic force also has infinite range but it is stronger than gravity. The weak and strong forces are effective only over a very short range and dominate only at the level of subatomic particles. Despite its name, the weak force is much stronger than gravity but it is indeed the weakest of the other three. The strong force, as the name suggests, is the strongest of all four fundamental interactions. Three of the fundamental forces result from the exchange of force-carrier particles, which belong to a broader group called “bosons”. Particles of matter transfer discrete amounts of energy by exchanging bosons with each other. Each fundamental force has its own corresponding boson – the strong force is carried by the “gluon”, the electromagnetic force is carried by the “photon”, and the “W and Z bosons” are carriers for the weak force. Although not yet found, the “graviton” should be the corresponding force-carrying particle of gravity. The Standard Model includes the electromagnetic, strong and weak forces and all their carrier particles, and explains extrimely well how these forces act on all of the matter particles. However, the most familiar force in our everyday lives, gravity, is not part of the SM, as fitting gravity comfortably into this framework has proved to be a difficult challenge. The quantum theory used to describe the micro world, and the general theory of relativity used to describe the macro world, are difficult to fit into a single framework. No one has managed to make the two mathematically compatible in the context of the Standard Model. But luckily for particle physics, when it comes to the minuscule scale of particles, the effect of gravity is so weak as to be negligible. Only when matter is in bulk, at the scale of the human body or of the planets for example, does the effect of gravity dominate. So the Standard Model still works well despite its reluctant exclusion of one of the fundamental forces.

The SM even includes the Higgs boson, the particle which explains why the other elementary particles, except photons and gluons, are massive.

Among all these particles, the top quark plays a very important role, being the most massive of all observed elementary particles (approximately 173 GeV). The large value of its mass, for instance, makes the top quark contribution dominant in loop corrections to many observables, like the W boson mass. Moreover, precise measurements of the W boson and the top quark masses are related to the mass of the Higgs boson, and are used to assess the self-consistency of the SM.

1.5.1 Top quark production and decay

According to the SM, the top quark [29] is an elementary particle belonging to the third generation of quarks. It is a fermion, with a spin 1/2 and an electric charge $+2/3\ e$, and it is the most massive of all observed elementary particles. It forms a weak isospin doublet together with the bottom quark, where the top quark is the up-type quark with the third component of the weak isospin $I_3 = +1/2$. As well as the other quarks, it is subject to the electromagnetic interaction (having electric charge), to the strong nuclear interaction (having colour charge), to the weak nuclear interaction (being part of a weak isospin doublet). Due to its huge mass, top quark mean lifetime is so short (about 5×10^{-25} s) that it decays semi-weakly mostly into a real W boson and a b quark before it can hadronize, without forming mesons or baryons with other quarks. In fact, unlike the other quarks, it has never been observed a bounded state involving this quark.

Its existence was postulated in 1973 by Makoto Kobayashi and Toshihide Maskawa to explain the observed CP violations in kaon decay, and was discovered in 1995 by the Collider Detector at Fermilab (CDF) and D0 experiments at Fermilab. Kobayashi and Maskawa won the 2008 Nobel Prize in Physics for the prediction of the top and bottom quark.

At the LHC, top quarks are mostly produced in pairs via strong interaction with a production cross section $\sigma_{t\bar{t}} \approx 830$ pb for $\sqrt{s} = 13$ TeV; however, there is a significant number of top quarks which are produced singly, via the weak interaction ($\sigma_t \approx 300$ pb for $\sqrt{s} = 13$ TeV). At leading order (LO), there are only a few processes which describe the production of $t\bar{t}$ production: the dominant production mechanism is from gluon-gluon fusion ($\approx 90\%$), while $q\bar{q}$ annihilation accounts for about 10%. The Feynman diagrams for the processes are shown in Figure 1.14.

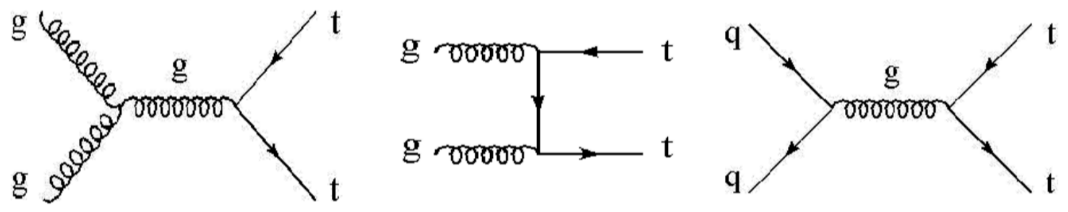


Figure 1.14: Feynman diagrams for $t\bar{t}$ production at LHC

The top quark decays through the weak interaction almost exclusively to a W boson and a bottom quark (with a branching ratio of approximately 96%), hence $t\bar{t}$ final states are classified by the decay products of the W boson, that can decay to either leptons or quarks. Therefore, three final states are distinguished as following:

- The dilepton channel, in which both W bosons decay to a lepton-neutrino doublet (branching ratio BR $\approx 5\%$):

$$t\bar{t} \rightarrow W^+bW^-\bar{b} \rightarrow l^+\nu_l b \;\; l'^-\bar{\nu}_{l'}\bar{b} \qquad with \;\; l = e, \mu$$

- The single-lepton channel, in which only one W decays to a lepton-neutrino doublet (BR $\approx 30\%$):

$$t\bar{t} \rightarrow W^+bW^-\bar{b} \rightarrow l^+\nu_l b \;\; q\bar{q}'\bar{b}$$

  or:

$$t\bar{t} \rightarrow W^+bW^-\bar{b} \rightarrow q\bar{q}'\bar{b} \;\; l^+\nu_l b$$

- The all-jets (or fully hadronic) channel, in which both W bosons decay hadronically, as represented in Figure 1.15 (BR $\approx 46\%$):

$$t\bar{t} \rightarrow W^+bW^-\bar{b} \rightarrow q'\bar{q}b \;\; q'\bar{q}\bar{b} \rightarrow j_1j_2j_3 \;\; j_4j_5j_6,$$

  producing 6 jets ("j_i" above) in the final state.

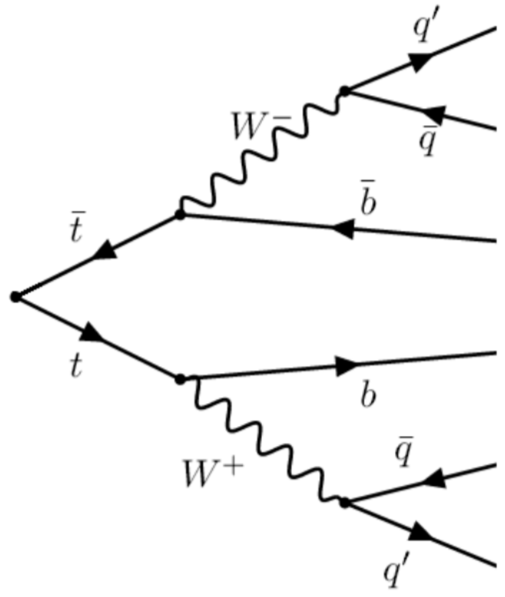


Figure 1.15: Feynman diagram for $t\bar{t}$ all-jets decay.

Chapter 2

The CMS Computing Model

During LHC operation, the accelerator produces a huge amount of data that must be stored and later analysed by scientists. The frequency of p-p or heavy ions collisions from LHC to each detector is $25 \cdot 10^9$ Hz, that produces 1 PB per second of data in each detector. As described in Section 1.3, the flux of data is selected by a trigger system that reduces the data stack produced to some hundreds of MegaBytes (MB) per second (for CMS, this is about 600 MB/s). To deal with all this data, a complex computing infrastructure has been designed and deployed, characterized by computing centers distributed worldwide, known as the Worldwide LHC Computing Grid (WLCG) [30, 31, 32, 33].

Including data produced by physics simulations and by detectors, the WLCG has to manage roughly 15 PB of data each year of normal conditions. On 29 June 2017, the CERN Data Center passed the milestone of 200 PetaBytes (PB) of data permanently archived on its tape libraries. In addition to this, each LHC computing model has to guarantee the data access by applications (jobs) on some computering resource somewhere on the globe, wherever the user might be located. The basic requirements, necessary to build such a computing system, are:

- ability to store and manage original and derived data;
- performant reconstruction code (to allow frequent re-reconstruction);
- streamed Primary Datasets (to allow priority-driven data distribution and processing);
- distribution of Raw and Reconstructed data altogether (to allow easy access to raw detector information);
- compact data formats (to allow hosting of multiple copies at multiple sites);
- consistent operation of production, reprocessing and bookkeeping.

Additionally the computing environment has to manage also the analysis processes, in what is referred to as "impredictable user analysis" activity.

The software that allows users to access computers that are distributed across the globe over high-performance networks is called "middleware", because it sits between the operating systems of the computers and the physics application software.

It includes the description of all the functionalities - and the needed software, middleware, hardware - that must be available to allow the collection, distribution and access of the data - produced by the accelerator and selected by each experiment's trigger system - in order to streamline the analysis of this data and the production of the desired physics output. All the interactions among each specific component of the overall system, and their management through a certain number of tools and services in real time, are also part of the computing model of each LHC experiment.

On top of a set of components that constitute the so-called "middleware-layer" - which might well (and should) be common across experiments - each experiment may also decide to add application layer software solutions which are experiment specific and perform very specific functionalities that cannot be offered by the previously introduced "common" layer. While common components are designed and developed by international projects to smoothly and coherently operate on WLCG resources, the application layer components are developed by software architects and developers of each experiment collaboration, and it is the experiment responsibility to make sure that these component can be operated in computing centers worldwide in an efficient way, and not disruptive of the activities of other experiments. All these very complex set-up and operations are steered by the WLCG management in close collaboration with the Software/Computing management of each LHC experiment.

2.1 CMS Computing Tiers in WLCG

As introduced in the previous section, the WLCG project is a global collaboration for the building and maintaining of a data storage and of an analysis infrastructure required by the experiments at the LHC. The main purpose of this infrastructure is to provide computing resources to store, distribute and analyse the data produced by LHC to all the users of the collaboration regardless of where they might be. This idea of a shared computing infrastructure is at the basis of the concept of the Grid. The WLCG cooperates with several Grid projects such as the European Grid Infrastructure (EGI) [34] and Open Science Grid (OSG) [35]. As mentioned in the previous section, the middleware projects provide the software layers on top of which the experiments add their own (different) application layer. At the middleware layer, the main building blocks that make up the infrastructure are the logical elements of a Grid site, namely:

- the Computing Element (CE) service, that manages the user's requests for computational power at a Grid site;
- the Worker Node (WN), where the computation actually happens on a site farm;
- the Storage Element (SE), that gives access to storage and data at a site. Data are stored on tapes and disks: tapes are used as long-term secure storage media, whereas disks are used for quick data access for analysis;
- the User Interface (UI), the machine on which users interact with the Grid;

- central services, that help users to access computing resources. Some examples are data catalogues, information systems, workload management systems, and data transfer systems.

Users make processing requests from one of the many entry points to the system. A processing request (a so-called "job") can include requests as: storage, processing capacity, availability of analysis software, etc. The computing Grid establishes the identity of the user, checks his credentials and searches for available sites that can provide the generic resources the user requested. Users do not have to worry about where the computing resources are located, as long as their needs are served: they can just tap into the Grid computing power and access storage on demand.

The WLCG is composed of centres belonging to few levels, called "Tiers", labelled 0, 1, 2 and 3. Each Tier is made up of computing resources centres and provides a specific set of services, different between one level and onother. Each Tier is hence a computing center that provides storage capacity, CPU power and network connectivity. The structure made by Tiers was formalized by the MONARC model, that foresaw a rigid hierarchy. According to the MONARC model, the number label associated to the Tier is intended to indicate the level and quantity of services and functionalities offered by the Tier to the community (and also, ultimately, its "size"): the lower the number the higher the level and quantity of services - for instance, a Tier-1 implements and offers more storage, CPU, network, additional services and related availability and support levels than a Tier-2. A computer center that take part to WLCG for one or more LHC experiments could theoretically cover different roles and perform different functions for each experiments: indeed for instance a Tier could have the function of Tier-1 for Alice and of Tier-2 for CMS. In Figure 2.1 and Figure 2.2 the hierarchy of computing centres in the CMS Computing model and the data flow among them is shown.

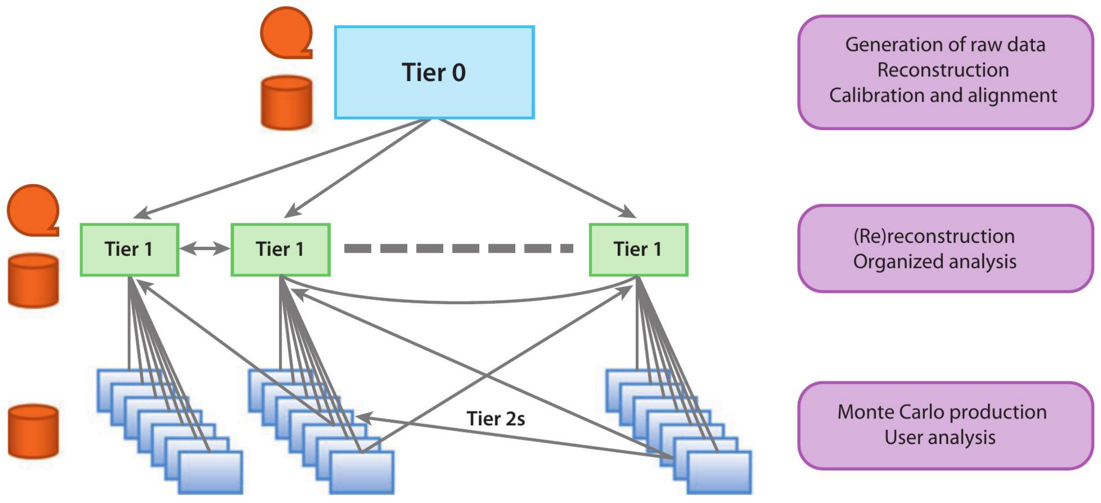


Figure 2.1: Original MONARC-driven tiered hierarchy of computing centres in the CMS Computing model (see text for explanations).

In Figure 2.3 the monthly transfer throughput across all Tiers for all LHC experiments in just one month (December 2017) is shown.

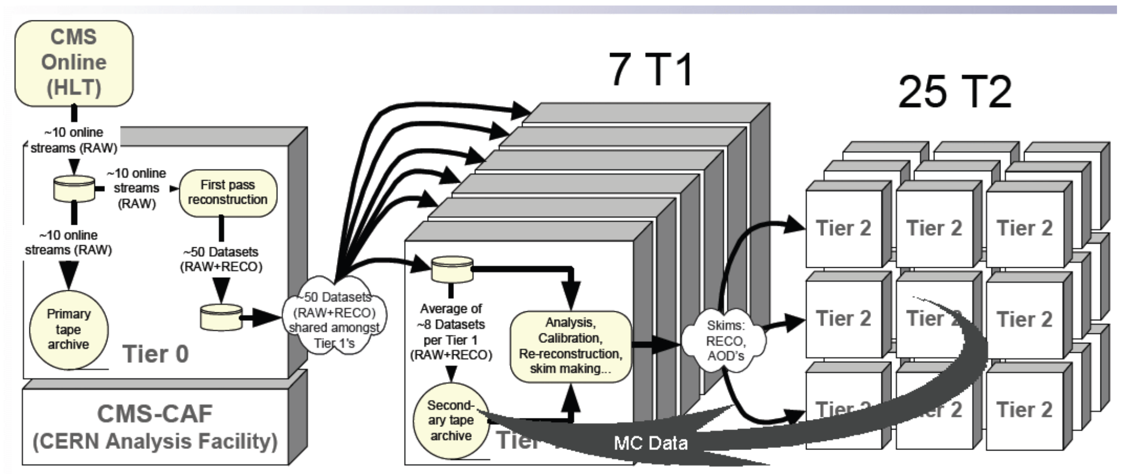


Figure 2.2: CMS WLCG Tiers and the data flows among them.

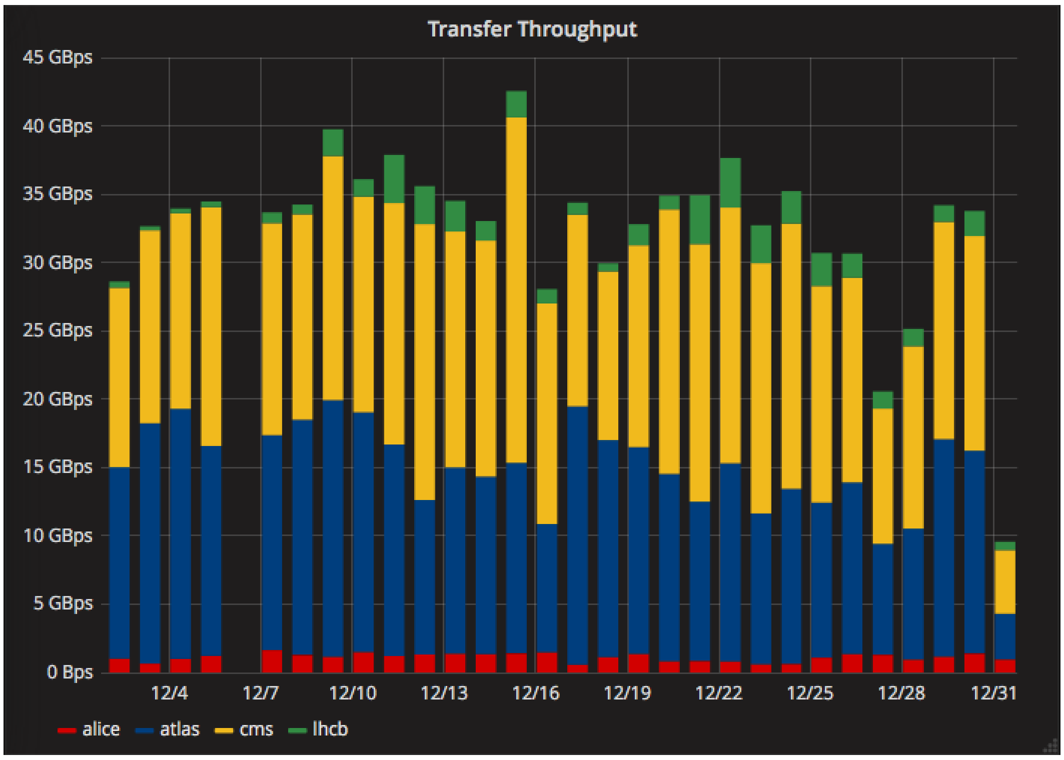


Figure 2.3: Example of a monthly transfer throughput across all Tiers for all LHC experiments in December 2017 [36].

2.1.1 Tier-0 and CMS-CAF

The Tier-0 (Figure 2.4) and CMS CERN Analysis Facility (CAF) are located at CERN. From 2012, the Tier-0 was extended via a link with the Wigner Research Centre for Physics in Budapest. This centre ensures more availability, thus allowing the Tier-0 to be operational also in case of CERN issues. The role of the Tier-0 is to receive RAW data from detectors and store them onto tapes, as well as permorm prompt reconstruction. Despite all the data from LHC passes through this central hub, it provides less than 20% of the Grid total computing capacity. The Tier-0 also distributes the RAW data and the reconstructed output to Tier-1s. The CMS Tier-0 classifies the promptly reconstructed data (in the RECO format, more later) into 50 "primary datasets", and it transfers them to Tier-1 centers. The mechanism through which integrity and availability are offered is built up as follows: data are stored in two copies, one at CERN (the so-called cold copy) and one distributed to Tier-1s (the so-called hot copy). In order to offer secure and performant transfers from the Tier-0 to the Tier-1s, a dedicate high-performance network infrastructure based on fiber optic has been designed and deployed, the so-called LHC Optical Private Network (LHCOPN [37]). While this was initially conceved for Tier-0 to Tier-1 connections only, it later expaned to also cover all the Tier-1 to Tier-1 connection, adding a crucial redundancy to the overall system.

CMS-CAF has the aim to provide support to all activities that require a very short time latency and a very quick asynchronous access to RAW data from Tier-0, such as calibration and alignment, detector/trigger commissioning or high priority physics analyses.

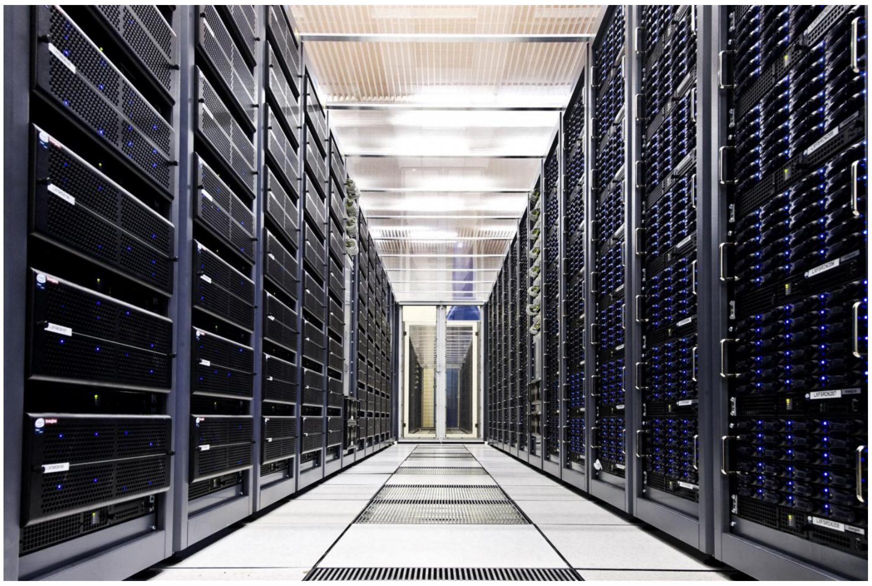


Figure 2.4: Servers at the CERN Data Centre, acting as Tier-0 for LHC experiments in the Worldwide LHC Computing Grid.

2.1.2 Tier-1s

The CMS Tier-1 pool consists of 13 computer centres in different part of the world (Italy, Germany, France, Spain, USA, Russia, UK and others), of which 7 are available to CMS during Run-2 (FZK in Germany, PIC in Spain, CCIN2P3 in France, CNAF in Italy, ASGC in TAiwan, RAL in UK and FNAL in USA) They are responsible for storing data, their reprocessing and further data sending to Tier-2 centres. At the Tier-1 sites the "hot copies" of real and simulated data are stored on both disk caches (for performant access) and tapes (for persistent archive). Together with consistent CPU power, the Tier-1 storage allows data reprocessing at each recalibration. Another goal of Tier-1s is to store simulated data produced e.g. at the Tier-2 level. Optical-fibre links working at 10 gigabits per second, as part of the extended LHCOPN, connect CERN to each of the 13 Tier-1 centres around the world.

2.1.3 Tier-2s and Tier-3s

Tier-2s are typically located at universities, laboratories and other scientific institutes that can store sufficient data and provide adequate computing power for specific analysis tasks. They handle a proportional share of the production and reconstruction of simulated events. There are around 155 Tier-2 sites around the world, among which about 50 used by CMS. Individual scientists can access the Grid through local (e.g. Tier-3) computing resources, which can consist of local clusters in a university department or even, in principle, an individual PC. Differently with respect to Tier-2 centres, a Tier-3 site does not sign any Memorandum of Understanding with WLCG, which mean there is no formal engagement between WLCG and Tier-3 resources. This implies no check on availability and thus a large flexibility in managing the site, which is why an average Tier-3 resource is rarely relied upon for Monte Carlo production, but it is ultimately a good resource for the local community of physics end-users and analysts.

2.2 The CMS data and simulation model

To extract a physics information useful for a high energy physics analysis, a physicist has to combine a variety of blocks, such as: reconstructed information from the recorded detector data, specified by a combination of trigger paths and possibly further selected by cuts on reconstructed quantities; Monte Carlo samples simulate the physics signal under study; background samples (specified by the simulated physics process). These informations are stored into *datasets* and *event collections*.

An *event collection* may correspond to the event data from a particular trigger selection from one given “run” (which corresponds to a single bunch crossing). These could very easily be any type of user-defined “ntuple”. A *dataset* is defined as any set of “event collections” that would naturally be grouped and analysed together as determined by physics attributes, like their trigger path or Monte Carlo physics generator, or by the fact that they represent a particular object model representation of those events (such as the RAW, RECO and AOD data formats). Datasets are split off at the Tier-0 and distributed to the Tier-1s.

An event collection is the smallest unit within a dataset that a user can select. Typically, the reconstructed information needed for the analysis would all be contained in one or a few event collection(s). Data are stored as ROOT files [38]. The smallest unit in computing space is called “file block”, which corresponds to a group of ROOT files likely to be accessed together (their dimension may vary in the range 1-10 TB). This requires a mapping from the physics abstraction of the event (event collection or dataset) to the file location.

2.2.1 CMS data types

Starting from RAW data produced from the online system, successive degrees of processing refine this data, apply calibrations and create higher level physics objects. Event information from each step in the simulation and reconstruction chain is logically grouped into what we call a data tier [39], where each one contains different levels of information about the same event. Indeed, each bit of data in an event must be written in a supported data format (a data tier may contain multiple data formats). A data format is essentially a C++ class, where a class defines a data structure (a data type with data members). The term data format can be used to refer to the format of the data written using the class (e.g., data format as a sort of template), or to the instantiated class object itself.

Therefore CMS uses a number of event data formats with varying degrees of detail, size, and refinement. The table in Figure 2.5 describes the various CMS event formats.

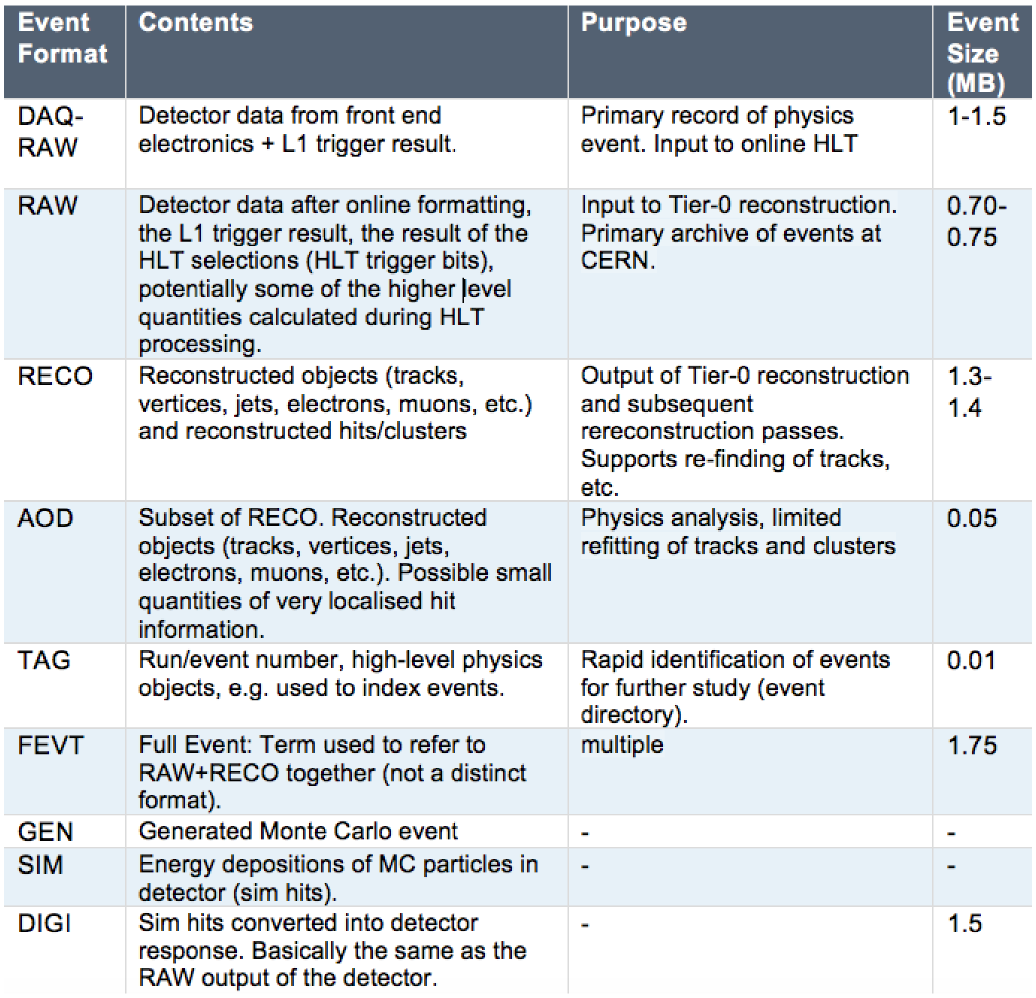
The three main data tiers written in CMS are:

- **RAW**: full event informations from the Tier-0, containing ’raw’ detector informations (detector particles hits, etc). In the vast majority of cases, RAW data are not directly used for analysis.

- **RECO** (“RECOnstructed data”): the output from first-pass processing at the Tier-0. This layer contains reconstructed physics objects, but istill contains plent of details about the event. RECO data can be used for analysis, but in general their size is too big for frequent use.

- **AOD** (“Analysis Object Data”): this is a “distilled” version of the RECO event information, and it is expected to be used for most analyses. AOD provides a trade-off between event size and complexity of the available information in order to optimize flexibility and speed for analyses.

In Figure 2.6 the physics elements in AOD, RECO and RAW formats are shown. RECO data contains objects from all stages of reconstruction. AOD are derived from the RECO information to provide data for physics analyses in a convenient, compact format. Typically, physics analyses don’t require users to rerun the reconstruction process on the data. Most physics analyses can run on the AOD data tier.



Event Format	Contents	Purpose	Event Size (MB)
DAQ-RAW	Detector data from front end electronics + L1 trigger result.	Primary record of physics event. Input to online HLT	1-1.5
RAW	Detector data after online formatting, the L1 trigger result, the result of the HLT selections (HLT trigger bits), potentially some of the higher level quantities calculated during HLT processing.	Input to Tier-0 reconstruction. Primary archive of events at CERN.	0.70-0.75
RECO	Reconstructed objects (tracks, vertices, jets, electrons, muons, etc.) and reconstructed hits/clusters	Output of Tier-0 reconstruction and subsequent rereconstruction passes. Supports re-finding of tracks, etc.	1.3-1.4
AOD	Subset of RECO. Reconstructed objects (tracks, vertices, jets, electrons, muons, etc.). Possible small quantities of very localised hit information.	Physics analysis, limited refitting of tracks and clusters	0.05
TAG	Run/event number, high-level physics objects, e.g. used to index events.	Rapid identification of events for further study (event directory).	0.01
FEVT	Full Event: Term used to refer to RAW+RECO together (not a distinct format).	multiple	1.75
GEN	Generated Monte Carlo event	-	-
SIM	Energy depositions of MC particles in detector (sim hits).	-	-
DIGI	Sim hits converted into detector response. Basically the same as the RAW output of the detector.	-	1.5

Figure 2.5: CMS data tiers and their characteristics.

Another important data tier that is not in the table is the MiniAOD format [40]. It was originally created in Spring 2014, its format evolved thanks to user feedback and in order to account new CMS reconstruction features, between the time of the original proposal and the starting of Run-2. The MiniAOD size is approximately 10% of the size of the Run-1 AOD format. The motivation for the MiniAOD format is to have a small and quickly derived data format on which the majority of CMS analyses can run. This format is targeted at having sufficient information to serve about 80% of CMS analysis, while dramatically simplifying the disk and I/O resources needed for analysis.

The writing of the different data tiers described above is due to two different workflows. The workflow named "data reconstruction" in CMS consists in the passage from informations contained in RAW data, through subsequent reprocessing stages, up to formats containing objects of interest for physics analysis.

The workflow named "simulation reconstruction", foresees that a first step is executed ("kinematics") based on several Monte Carlo generator events, following by a second step ("simulation") that simulates the detector answer when generated interactions occur, and finally a third step ("reconstruction") occurs, in which to simulate a real bunch crossing, the single interaction is combined with pile-up events

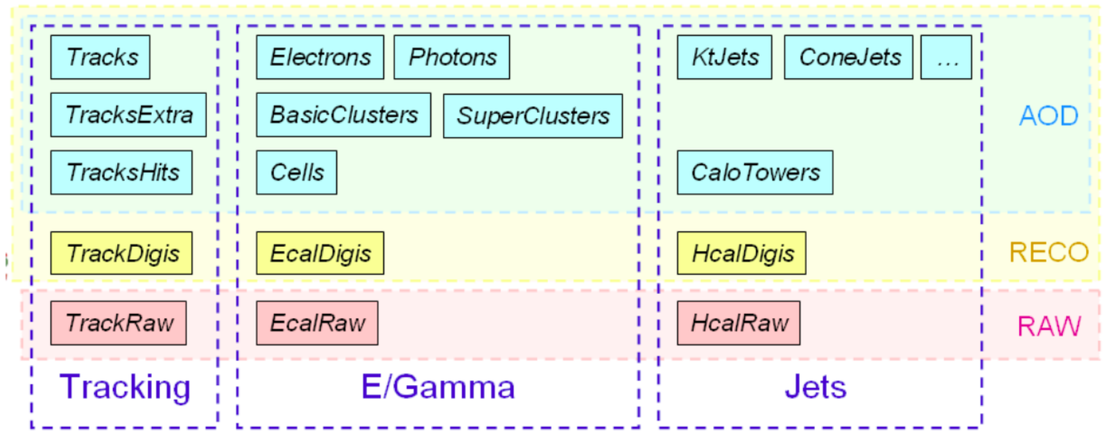


Figure 2.6: Illustration of RECO and AOD content.

and then rebuilt. In the end of this phase, the CMS data model expected that reference formats are called AOD and AODSIM for data and simulation respectively.

In addition to what mentioned above, the CMS collaboration also needs other types of data that are not based on the physics event as a unit, and they are hence called "Non-Event Data". There are four categories of such non-event data: construction data (information produced during the assembling of the detector), equipment management data (calibraton information of the CMS subdetectors); configuration data (calibration and alignment parameters); conditions data (description of the current operative condition of the detector). Non-Event Data are generated both during online data taking and offline analysis, and are stored in several CMS central databases. They are not discussed further in this thesis.

2.3 CMS services and operations

As stated in the beginning of this chapter the integration of resources at CMS Computing Centres into a single coherent system relies upon Grid middleware which presents a standardised interface to storage and CPU facilities at each WLCG site. At the same time, though, a number of CMS-specific distributed computing services operate above the generic Grid layer, allowing higher-level data and workload management functions. In some cases, these services require CMS-specific software agents to run at the sites, in addition to generic Grid services. An overview of the CMS Computing Services components is shown in Figure 2.7.

In the next two paragraphs, some of the most important services will be briefly presented, with the caveat that this is only a general high-level view, and some additionally existing services - which may rapidly evolve over the years - are not mentioned.

2.3.1 Data management

CMS requires tools to catalogue the data which exist, to track the location of the corresponding physical data files on site storage systems, and to manage and monitor the flow of data between sites. To provide the connection between abstract

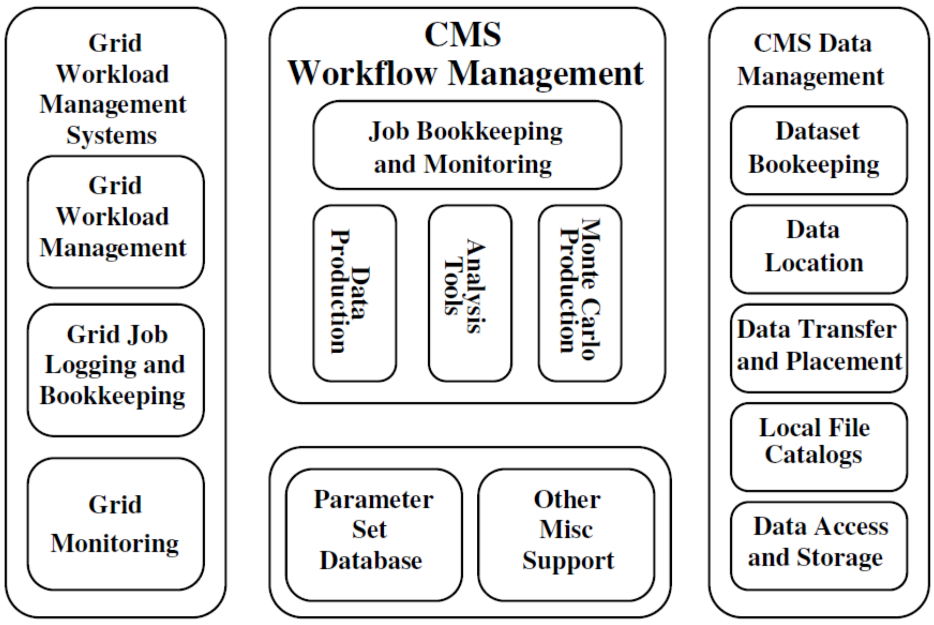


Figure 2.7: Overview of the CMS Computing Services.

datasets or event collections and physical files, a multi-tiered catalogue system is used: the Dataset Bookkeeping System (DBS) [41]. It provides a standardised and queryable means of cataloguing and describing event data. It is the principle means of data discovery for the user, answering the question "which data of this type exists in the system?" In particular it must express the relationships between "datasets" and "event collections" as well as their mapping to the packaging units of "file blocks" and "files". A second catalogue system functionality is needed to provide the mapping between file blocks to the particular sites at which they are located, including the possibility of replicas at multiple sites. Local File Catalogues at each site map logical files onto physical files in local storage. The Data Aggregation Service (DAS) is a service that provides a data search through a query language. The Physics Experiment Data Export (PhEDEx) [42, 43] is a reliable and scalable dataset replication system in production for CMS since 2004, and stands as one of the first systems developed on the Grid for a LHC experiments that still survives in LHC Run-2.

2.3.2 Worfklow management

The management of grid jobs is handled by a complex system, which nowadays has completely migrated towards the use of "pilots" on a HTCondor infrastructure [44]. The goal is to schedule jobs onto resources according to the policy and priorities of CMS, to assist in monitoring the status of those jobs, and to guarantee that site-local services can be accurately discovered by the application once it starts executing in a batch at the site. All these issues should be invisible to the user. The centrally-steered production activities are very different, in concepts and tools used, from the distributed analysis. In the former case, the actual management of the whole task is performed by a WMCore/WMAgent infrastructure [45], while on the latter case the job preparation, submission and monitoring is done via the CMS CRAB [46] distributed analysis toolkit. The distributed analysis comprises plenty of jobs running on the Grid which are intrinsically independent from each other. This fact allows the distribution of the the workload according to the computing center capabilities and the location of the input data. When the analysis has been assigned to a specific computing center, a CRAB job wrapper performs the set up of the environment, then it runs the user analysis on the local dataset and delivers the requested data to user. CRAB takes care of interfacing with the user environment, provides data-discovery and data-location services, manages the status reporting, monitoring, and user job output which can be put on a user-selected storage element. Via a simple configuration file, a physicist can thus access data available on remote sites as easily as he can access local data: all infrastructure complexities are hidden to users as much as possible. There is also a client-server architecture available, so the job is not directly submitted to the Grid but to a dedicated CRAB server, which, in turn, handles the job on behalf of the user, interacting with the Grid services.

2.4 Computing towards HL-LHC

As introduced in Section 1.4, starting in 2026 the HL-LHC will handle 30 times more data than the LHC has currently collected (with the total amount of LHC data collected so far by ATLAS and CMS summing up to close to an exabyte). While it is evident that approaches beyond simply scaling current solutions will be needed, it is less clear whether to evolve current solutions towards those scales, or implement new solutions, or a mix of both approaches. The trivial scaling assuming Moore's Law will not be sufficient, in a funding scenario that consists of more or less constant operational budgets year after year. Adiabatic evolutions - plus some revolutions - are expected in all area of computing at LHC.

First of all, the nature of computing hardware (CPU processors, storage, networks) is evolving with radically new paradigms, and the experience is developing and eventually deploying these solution is ahead of the LHC community. As the quantity and complexity of data to be processed will dramatically increase, more sophisticated analysis techniques will be required not only to maximize the physics throughput, but also to simply allow physicist to proficiently access the data. The software sector will need to go under solid evolutions: developing and deploying sustainable code for future and upgraded experiments, within these constraints, will be a technical challenge as well as a social challenge in terms of training and securing the skilled manpower for this complex task. In general, while usually "hardware upgrade" is clear to most funding agencies worldwide, a higher awareness of the need of a simultaneous “software and computing upgrade” is needed: a coherent approach to all upgrade plans needes towards HL-LHC will be the only way for the community to build an overall end-to-end system that will be able to cope with the data that will be collected, taking full advantage of all hardware upgrades and to ultimately complete the HL-LHC physics programme.

More details about the use of Machine Learning in HEP and in CMS are part of these evolutions towards HL-LHC, and will be discussed in Section 3.5.

Chapter 3

Machine Learning: concepts and methodology

3.1 The rise of Artificial Intelligence

Artificial Intelligence, Machine Learning and Deep Learning are very hot buzzwords nowadays. They often seem to be used interchangeably, and they are especially used very frequently when the topic is Big Data, Analytics, and - in a broad sense - the waves of technological change which are sweeping through our world. Despite their use, the word refer to quite different scopes.

- **Artificial Intelligence** (AI) is the original, broader concept that applies to machines being able to carry out tasks in a way that humans would consider "smart": this usually just means that some tasks are performed by machines in very accurate ways, close to how the same tasks would be performed by humans.
- **Machine Learning** (ML) is instead a current application of AI itself, based around the idea that humans should just be able to give machines access to enough data of sufficient quality, and let machines learn for themselves: machines would be able to perform a specific task after learning how to do so from the data itself, and without being explicitly, algorithmically programmed.
- **Deep Learning** (DL) is a technique for implementing ML, i.e. building artificial neural networks that are inspired by the understanding of the biology of human brains: while neurons are very well interconnected, though, an artificial neural network have discrete layers, connections, and directions of data propagation. The focus in this thesis will only be on Machine Learning approaches.

Machine Learning [47] is very widely adopted into daily technologies: most people probably use a learning algorithm dozens of times a day without even knowing. Every time people use a web search engine like Google or Bing, one of the reasons whay it works so efficiently is because a learning algorithm - regardless of its implementaion by Google or Microsoft - has learned how to rank web pages. Every time people read emails and the spam filter saves us from having to wade through tons of spam emails, a learning algorithm is acting behind the scenes. Every time people use Facebook or Apple photo typing application and it recognizes our friends photos, that also is based on machine (or deep) learning. The road through deep learning is showing that one of the best way towards machines operating better than humans in so many areas might go through learning algorithms that try to mimic how the human brain learns. Autonomous robotics, self-driving vehicles, computational biology: tons of disciplines are popping up in which machine learning is having a huge impact on. Few examples of machine learning applications are elaborated in the next paragraphs.

Database mining is one of the reasons why machine learning has so pervaded is the growth of the web and the growth of automation. All this means that we have much larger data sets than ever before. So, for example tons of Silicon Valley companies are today collecting web click data, also called *clickstream data*, and are trying to use machine learning algorithms to mine this data to understand and serve the users better. With the advent of automation, we now have electronic medical records, so if we can turn medical records into medical knowledge, then we can start to better understand diseases. With automation again, biologists are collecting lots of data about gene sequences, DNA sequences, and so on, and machines running algorithms are giving us a much better understanding of the human genome, and what it means to be human.

Self-customizing programs also use learning algorithms. Every time one goes to Amazon or Netflix or iTunes Genius, and the service recommends movies or products to you, a learning-based recommendation systems is behind it. Each of these services have millions of users; there is no way to write a specific program for each person, so the only way for some system to provide customized recommendations is for some software to learn by itself and customize itself to users' preferences.

Before going into details of Machine Learning as it is used in this thesis, it is worthwhile to underline that the underlying theory and ideas are definitely not new, and actually date back a few decades. The reasons why there is a new rise of AI in so many sectors nowadays might be related to three factors:

- (Big) Data. Learning algorithms are extremely data hungry, and finally the newest techologies seems to have reached a point where they can feed the algorithms with enough data;

- Technological progress. The storage capacity and density, the computational power, a better and greater bandwidth, together with lowered technology costs allowed to actually make a system to ingest/digest theinformation they needed actually possible;

- A bump in resources democratization and efficient allocation at affordable costs, as reflected by the rise of Cloud services (i.e. Amazon Web Services, Google Cloud Platform) and parallel computing operated by GPUs.

3.2 Machine Learning: a definition

An interesting (and operational) definition of ML is as follows:

A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E. [48]

Following this sentence, the key that connects past behavior to future behavior is the experience "E". Focusing on a task "T" (e.g. the car traffic pattern at an intersection), the experience "E" contains the data about all the traffic patterns that happened in the past: this is fed to a ML algorithm, which indeed learns about "T" from "E" and it will improve – as measured by "P", the metric chosen as performance measure - in predicting future traffic patterns. In this sense, ML can hence solve problems that cannot be solved by numerical means alone. Another way of stating the same is that ML is the discipline that gives computers the ability to learn without being explicitly programmed.

3.3 Supervised versus Unsupervised Learning

There are several different types of learning algorithms. The main two types are what we call *supervised learning* and *unsupervised learning.* In a simplified way, in supervised learning we are going to teach the computer how to do something, whereas in unsupervised learning we are going to let it learn how to do it by itself.

Supervised learning

In order to better understand what supervision means in a learning process, a couple of examples [49] are presented and briefly discussed in the following.

As a first example, let's suppose that a student collects data sets about US housing prices and plots the data, e.g. see Figure 3.1, where the size of different houses in square feet is shown on the x-axis, and the price of different houses in thousands of dollars is shown on the y-axis. Given this data, another person who owns a house of e.g. 750 square feet might want to know how much value he can get for the house. How can the learning algorithm help him? One thing a learning algorithm might be able to do is to put a straight line through the data or (better) to fit a straight line to the data (pink straight line in Figure 3.1): based on that, it might look that the house could be sold for about $150,000. Perhaps, this is not the only learning algorithm we could use: there might be a better one, for example a fit of a quadratic function or a second-order polynomial to this data could be tried out (blue curved line in Figure 3.1). And if we do that and make a prediction based on it, we could state that the house could be sold for about $200,000. Each of these two different approaches would be an example of a supervised learning algorithm: the term "supervised" here refers to the fact that we gave the algorithm a data set in which the "right" answers were given (i.e. a data set of house characteristics and prices in which for each entry in this data set we knew the right price, so that the toss of the algorithm was to just produce more of these right answers for new

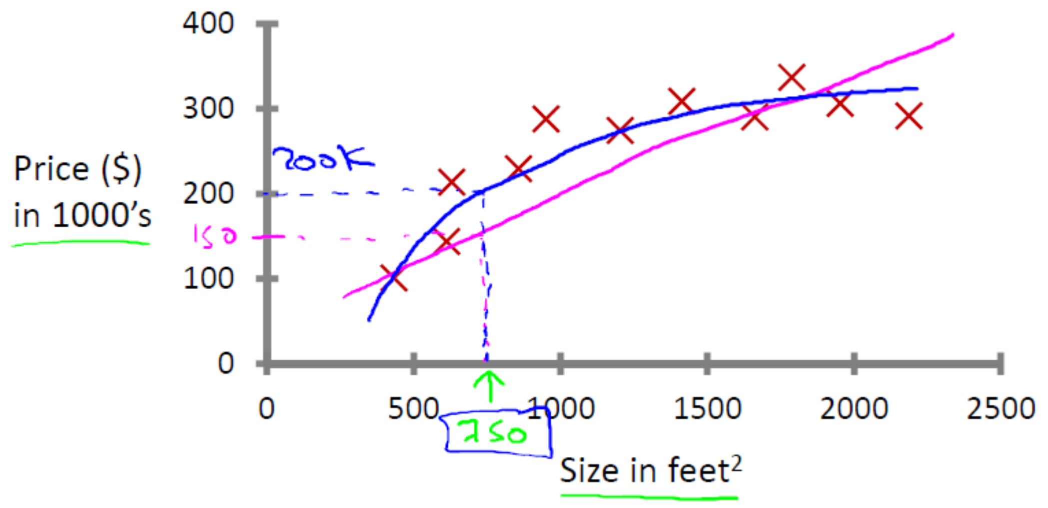


Figure 3.1: A first example of supervised learning. Details and reference in the text.

houses). This is also called a "regression" problem, namely we are trying to predict a continuous value output (in this case, the price: prices are actually discrete values, but one can usually think of the price of a house as a scalar number in the domain of the real numbers).

As another example of supervised learning, let's suppose that we want to look at medical records and try to predict the nature of a breast cancer as benign or malignant. One can take the medical dataset and plot the size of the tumor on the x-axis and a binary value (1 or 0) on the y-axis depending on the malignant/benign nature of the tumor.

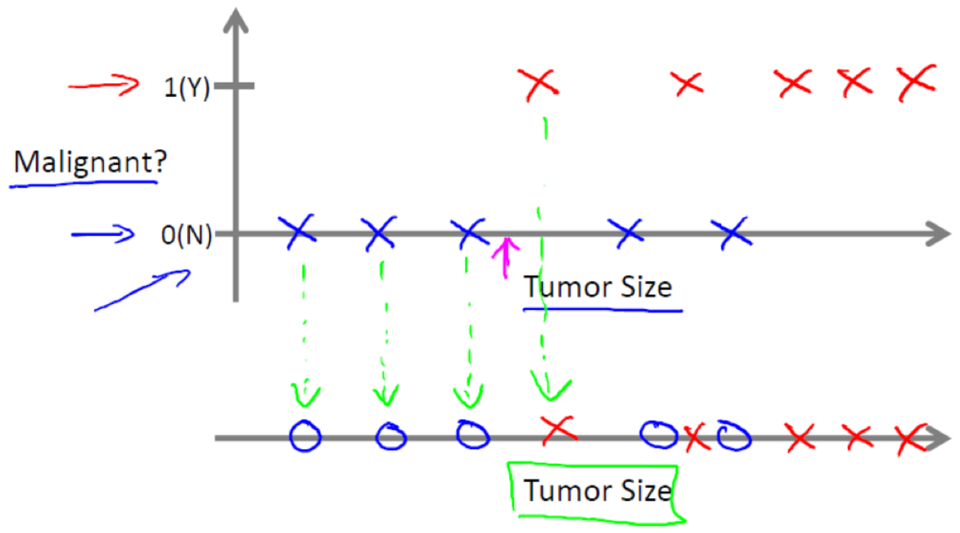


Figure 3.2: A second example of supervised learning. Details and reference in the text.

The data are displayed in Figure 3.2, in which five examples of benign tumors and five examples of malignant tumors are shown. Let's suppose that an additional data point might be related to a patient that has a breast tumor, and its size is known. One wonders which is the probability that this breast tumor is malignant versus benign. This is an example of a “classification” problem, instead: the term classification refers to the fact that here one is trying to predict a discrete value output, i.e. 0 or 1 (malignant or benign, in this example). In typical classification problems, it often happens to have more than two values for the possible output values. As a concrete example, suppose there are 3 types of breast cancers, and one might want to predict the discrete values of 0 (benign), 1, 2, 3 (all malignant). This would still be a classification problem: these other discrete values in the output set correspond to no cancer, or cancer of type 1, or 2, or 3. For clarity, we might use a slightly different set of symbols to plot this data, in particular to denote benign versus malignant, or negative versus positive examples. So, in Figure 3.2, instead of drawing crosses we draw circles for the benign tumors and crosses for the malignant ones (see the dropped down x-axis at the bottom of Figure 3.2) and we use only one attribute - the tumor size - in order to predict whether an additional tumor in the dataset would be malignant or benign. Typically, in ML problems one deals with many attributes, or features. In this example, one could e.g. know the age of the patient and add it as information to the size of the tumor. Our dataset will look like in Figure 3.3.

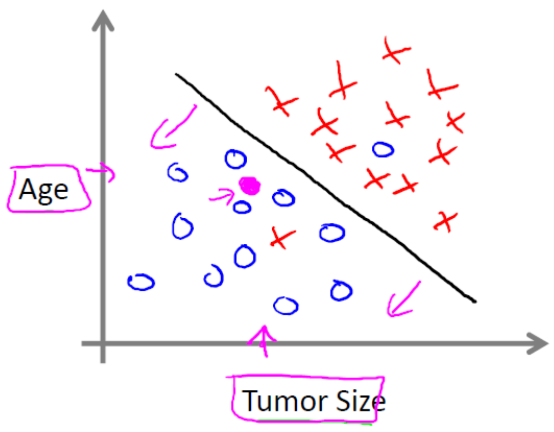


Figure 3.3: Further elaboration of the second example of supervised learning quoted in the text.

In this example, a learning algorithm should be able to throw a straight line through the data displayed on a bidimensional plot, and separate out the malignant tumors from the benign ones at its best. In real life cases, one can have many more attributes (clump thickness of the breast tumor, uniformity of cell size of the tumor, uniformity of cell shape of the tumor, etc), up to a very large number. Dealing with a very large number of features is an issue. Also storing a very large set of data without risking to go out of memory to process them is a potential problem. This is the kind of issues to face in basic supervised learning ML methods, and a large literature is available to map each problem to the best approach (e.g. in this specific example, perhaps an algorithm called “Support Vector Machine" offers a neat mathematical trick that would allow a computer to deal even with an infinite number of features).

Unsupervised learning

With respect to the supervised case, in unsupervised learning one is only given input data but no corresponding output variables, and the goal of unsupervised learning is to model the underlying structure or distribution in the data in order to learn more about the data itself.

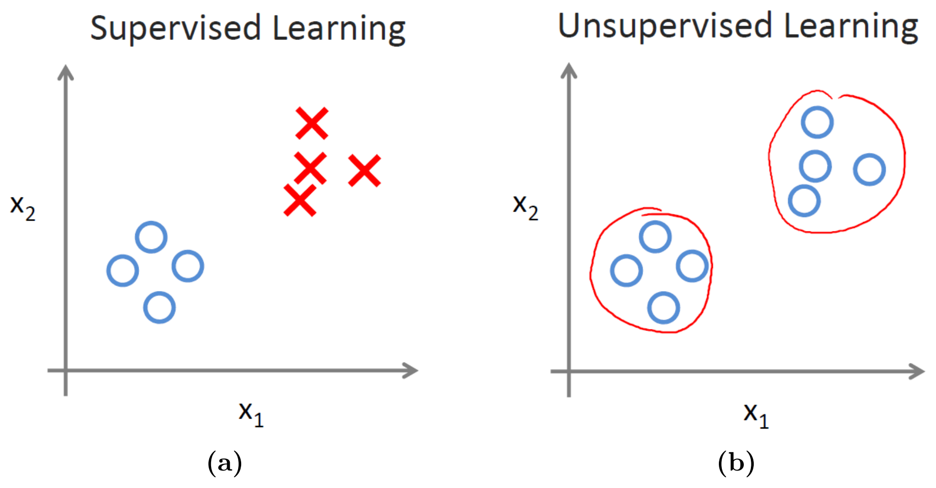


Figure 3.4: More on supervised vs unsupervised learning (see text for explanation).

Suppose one is given data as points in plots of Figure 3.4, and one needs to understand if there is some structure in such data. Given this data set, an unsupervised learning algorithm might decide that the data lives in two different clusters (also supervised algorithms may be able to break these data into two separate clusters). A clustering algorithm like this one turns out to be used in many applications, e.g. in Google News. What Google News does everyday is to go and look at thousands and thousands of new stories on the web, and group them into cohesive news stories, i.e. automatically cluster them together, in a way that the news that are all about the same topic get displayed together. Clustering algorithms and in general unsupervised learning algorithms are used in many other areas, e.g. in studying DNA microarray data in genomics, as outlined in the following example.

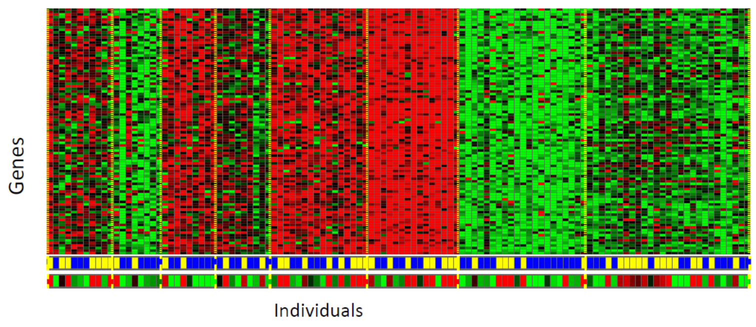


Figure 3.5: Example of DNA microarray data.

The basic idea is to examine a group of different individuals and for each to check if they do or do not have a certain gene. Technically you measure how much certain genes are expressed, and different colours (red, green, gray and so on) show the degree at which different individuals do have a specific gene or not (Figure 3.5). Running a clustering algorithm can group individuals into different categories according to these genes. This is unsupervised learning because we are not coding in the algorithm in advance which persons belong to each sub-group, but the system is able to automatically find structure in the data, i.e. automatically form clusters of individuals that belongs to the different types. The different in comparison to supervised approaches lies in the fact that we are not giving the algorithm the "right answers" in any previously available data set. Another area in which unsupervised learning is used is in the social network data analyses. Clustering here applies in finding groups of people that may related to each other, based on which persons we email the most, which Facebook friends two people have in common, etc. Another application, on the scientific side, is about the astronomical data analysis: clustering algorithms offer extremely powerful tools to dive into large quantity of data with no a-priori knowledge and extract meaningful insight to help e.g. the study of the origin of galaxies.

The ones mentioned above are examples of clustering, which is just one type of Unsupervised Learning. Another example can be explained through the so-called "cocktail party problem". In this example, a party is happening in a room full of people, everyone talking at the same time, hence with overlapping voices, making it hard for everyone to understand what everyone else is saying. Suppose there are microphones in the room: as they are at different distances from each speaker, every microphone records a different combination of any speakers voice. Take a simpler example with just 2 people in the room, and 2 microphones: every microphone will record the voice of each, but at different volumes depending on the distance. In this condition, a specific Unsupervised Learning algorithm could take the microphone recorders and try to find structure in the data, namely it might be able to identify the source voice patterns, despite at different volume levels in different microphone recordings, and ultimately disentangle them.

3.4 Machine Learning terminology

In the previous sections, highlights and examples on major types of ML challenges were described. In this section, a short list of general Machine Learning terms - widely used in literature - is offered in this section. It is intended as a guide to the reader throughout this thesis.

- **Big data** is a term that describes the large volume of data – both structured and unstructured. But it is not the amount of data that is important. It is how organizations use this large amount of data to generate insights. Companies use various tools, techniques and resources to make sense of this data to derive effective business strategies.

- **Classification** is supervised learning method where the output variable is a category, such as "Male" or "Female" or "Yes" and "No". For example classification Algorithms are Logistic Regression, Decision Tree, K-NN, SVM etc.

- **Clustering** is an unsupervised learning method used to discover the inherent groupings in the data. For example: grouping customers on the basis of their purchasing behaviour which is further used to segment the customers. And then the companies can use the appropriate marketing tactics to generate more profits. Example of clustering algorithms: K-Means, hierarchical clustering, etc.

- **Cross-validation** is the name given to a set of techniques that divide up data into training sets and test sets. The training set is given to the algorithm, along with the correct answers and becomes the set used to make predictions. The algorithm is then asked to make predictions for each item in the test set. The answers it gives are compared to the correct answers, and an overall score for how well the algorithm did is calculated.

- **Data mining** is a study of extracting useful information from structured/unstructured data taken from various sources. Data Mining is done for purposes like Market Analysis, determining customer purchase pattern, financial planning, fraud detection, etc.

- **Data science** is a combination of data analysis, algorithmic development and technology in order to solve analytical problems. The main goal is a use of data to generate business value.

- **Decision tree** is a type of supervised learning algorithm (having a pre-defined target variable) that is mostly used in classification problems. It works for both categorical and continuous input & output variables. In this technique, the population (or sample) are split into two or more homogeneous sets (or sub-populations) based on most significant splitter/differentiator in input variables.

- **Deep Learning** is associated with a machine learning algorithm (Artificial Neural Network, ANN) which uses the concept of human brain to facilitate the modeling of arbitrary functions. ANN requires a vast amount of data and this class of algorithms is highly flexible when it comes to model multiple outputs simultaneously.

- An **evaluation metric** is used to measure the quality of the statistical/machine learning model. For example few evaluation metrics are: AUC, Accuracy, Log-Loss, etc.

- In machine learning, **inference** often refers to the process of making predictions by applying the trained model to unlabeled examples. In statistics, inference refers to the process of fitting the parameters of a distribution conditioned on some observed data.

- **Machine Learning** refers to the techniques involved in dealing with vast data in the most intelligent fashion (by developing algorithms) to derive actionable insights. In these techniques, we expect the algorithms to learn by itself without being explicitly programmed.

- **Model** is "a specification of a mathematical (or probabilistic) relationship that exists between different variables". Because "modeling" can mean so many things, the term "statistical modeling" is often used to more accurately describe the kind of modeling that data scientists do.
- **Multivariate analysis** is a process of comparing and analyzing the dependency of multiple variables over each other.
- Neural net or **artificial neural network** is a robust function that takes an arbitrary set of inputs and fits it to an arbitrary set of outputs that are binary. In practice, Neural Networks are used in deep learning research to match images to features and much more. What makes Neural Networks special is their use of a hidden layer of weighted functions called neurons, with which you can effectively build a network that maps a lot of other functions. Without a hidden layer of functions, Neural Networks would be just a set of simple weighted functions.
- **Outlier** is an observation that appears far away and diverges from an overall pattern in a sample.
- **Overfitting** occours when a created model matches the training data so closely that the model fails to make correct predictions on new data.
- **Predictive analysis** is the analysis of data to predict future events, typically to aid in business planning. This incorporates predictive modeling and other techniques. Machine learning might be considered a set of algorithms to help implement predictive analytics. The more business-oriented spin of "predictive analytics" makes it a popular buzz phrase in marketing literature.
- **Regression** is supervised learning method where the output variable is a real value, such as "amount" or "weight". Example of Regression: Linear Regression, Ridge Regression, Lasso Regression.
- **Supervised Learning** algorithm consists of a target/outcome variable (or dependent variable) which is to be predicted from a given set of predictors (independent variables). Using these set of predictors, a function is generated that maps inputs to desired outputs, like: $y = f(x)$. Here the goal is to approximate the mapping function so well that when you have new input data (x) you can predict the output variables (Y) for that data. Examples of Supervised Learning algorithms: Regression, Decision Tree, Random Forest, KNN, Logistic Regression, etc.
- **Test set** is the subset of the data set used to test your model after the model has gone through initial vetting by the validation set.
- **Training set** is the subset of the data set used to train a model.

- In **Unsupervised Learning** algorithm, we do not have any target or outcome variable to predict/estimate. The goal of unsupervised learning is to model the underlying structure or distribution in the data in order to learn more about the data or segment into different groups based on their attributes. Examples of Unsupervised Learning algorithm: Apriori algorithm, K-means.
- **Validation set** is a subset of the data set, disjunct from the training set, used to adjust hyperparameters.

3.5 Machine Learning in HEP

As discussed primarily in [50], the HEP community if becoming aware of the potential that Artificial Intelligence has in many areas of HEP experiments, like Physics Analysis, Trigger decisions, Detector quality monitoring (and beyond), Software and Computing. A brief overview of Machine Learning in HEP in general, and in CMS in particular, is given in the next two sections.

3.5.1 Current practices and roadmap towards the 2020s

Machine Learning is a rapidly evolving approach in many areas, with the power of characterising and describing data in a way that may radically change how data is collected, eventually reduced and ultimately analysed. ML spans a wide range of possible applications in HEP, also. For example, ML may qualitatively and directly improve the physics reach of datasets, but may also indirectly allow more efficient use of processing and storage resources, thus effectively extending the physics reach of experiments at the same computing costs. It is fair to say that today the HEP community needs to build domain-specific applications on top of existing toolkits and ML algorithms developed by computer scientists and scientific software developers, but also data scientists - with all these professionals almost entirely found outside the HEP world. There will be (and already is) a visible trend in trying to change some ways HEP traditionally work to bridge the HEP world to the outside world. Using existing paradigms and techniques that are widely adopted by the non-HEP community will bring an advantage to the field in this sector, so it is expected that some more work will need to be done especially where typical HEP problems do *not* map well onto existing paradigms, as well as in trying to understand how these problems can be recast into some abstract formulations that might be of more general interest.

The portfolio of existing ML techniques and tools is large and in constant evolution, with a recent raise due to various factors (as previously exposed in this chapter). A variety of powerful ML approaches for classification (using pre-defined categories e.g. in supervised approaches), clustering (where categories are discovered inside the data), regression (where predictions are made for continuous - and not discrete - outputs), etc has emerged over the years. A large benefit comes from the fact that many of this have well-documented open source software implementations: e.g. parsing github [51] repositories or Kaggle [52] ML challenges is a huge source of documentation and code for anyone to start up a ML project. Some of the aforementioned tools have indeed been used productively in HEP already, and for more than 20 years, while others have been introduced relatively recently (for a review, see [53]). Nowadays, ML is starting to become ubiquitous in some HEP applications, most notably in final offline analyses, but it is also increasingly used in both online and offline reconstruction and particle identification algorithms, in the classification of reconstruction-level objects, such as jets, as well as in sectors that did not exploit ML up to 2-3 years ago, like the analysis of Computing metadata for resource usage optimization. It is difficult to predict how this will evolve in HEP. For sure, more and more powerful hardware and more performant ML algorithms will be available in the next years, and the toolset may become so powerfule that may candidade itself to replace existing, historical approaches in HEP. ML tools could e.g. replace the most computationally expensive parts of pattern recognition algorithms, as well as the parameter extraction algorithms for characterising reconstructed objects: the charged track and vertex reconstruction are one of the most CPU intensive elements of the current HEP software stack, and ML could improve its physics performance or execution speed. ML could increase its role at the analysis stage, and extend the physics reach of experiments, e.g. by just interpolating between mass points, or even by doing training in a systematics-aware manner. Many more examples might be found.

In the attempt (admittedly very difficult) to categorize the areas in which joint reserch with ML experts might offer the largest impact, the following ones could be highlighted.

- *Particle identification and particle properties.* In various detectors, e.g. in calorimeters or time projection chambers, the data can be represented as a 2D or 3D image (even 4D, if one includes timing information), so the problems can be cast as a computer vision task. In this case, Deep Learning is a promising approach: neural networks are good candidates to identify particles and extract many parameters (e.g. they can be used to reconstruct images from pixel intensities). In terms of Deep Learning architectures, convolutional (CNN), recurrent, (RNN) and adversarial neural networks may fit these tasks. Additionally, particle identification can also be explored to tag the flavour of jets (e.g. the so-called b-tagging). The investigation of these applications has started already at the LHC.

- *ML middleware and data formats for offline usage.* HEP relies on the ROOT data format, wheras the ML community worldwide has developed several other formats: some are general, some are often associated with specific ML tools. Most probably, the convergence among HEP and non-HEP data scientists may come more easily if non-ROOT data format would be used. In general, a desirable data format for ML applications should offer: high read-write speed for efficient training, sparse readability without needing to load the entire dataset into RAM, compressibility and in general a widespread adoption by the ML community. The thorough evaluation of different data formats and their impact on ML performance in the HEP context is a need, and should be pursued in the next few years.

- *Computing resource optimisations.* Managing large volume data transfers is one of the challenges facing current computing facilities. Networks play a crucial role in data exchange and so a network-aware application layer may significantly improve the operations of LHC experiments. ML is a promising technology to e.g. identify anomalies in network traffic, to predict and prevent network congestion, to detect bugs via analysis of self-learning networks, and for WAN path optimisation based on user access patterns.

- *ML "as a Service" (MLaaS).* While current cloud providers heavily rely on a MLaaS paradigm to exploit ML tools in order to make efficient use of resources, this is not (yet?) widely used in HEP. More on this topic in Chapter 4. In addition, a contribution towards this direction consists also in part of this thesis work.

- *Detector anomaly detection.* Data taking is continuously monitored by physicists taking shifts in person, to monitor and assess the quality of the incoming data fro the collisions. This is largely done using reference histograms produced by experts. A whole class of ML algorithms called anomaly detection might be useful for automating this critical task, thus freeing up plenty of human resources. In a nutshell, unsupervised algorithms would be able to learn from data and produce an alert when deviations are observed. After proper training, by monitoring many variables at the same time such algorithms could become sensitive to very subtle signs forewarning of imminent failure, and pre-emptive maintenance actions might be hence scheduled. These techniques are not now to industry, but are not (yet) applied in producton in HEP experiments.

- *Triggering and real-time analysis.* The increasing event complexity at HL-LHC will need to be dealt with: the use of sophisticated ML algorithms should be explored at all trigger levels. In this case there is some pioneering work among LHC collaborations, and this should be the guide for other HEP experiments. One of the challenges here is the trade-off in algorithm complexity and performance under very strict inference time constraints. Another challenge is related to technology tracking, in order to be able to invest on the ML techniques which will eventually allow HEP to maximally exploit future computing architectures.

- *Tracking.* As stated before, pattern recognition is a computationally challenging step. It will become a huge challenge in the HL-LHC environment. Adequate ML techniques for track pattern recognition on many-core processors may provide a solution that could be able to scale linearly with LHC intensity. Several efforts in the HEP community have started already on this task.

- *Simulation.* There has been plenty of recent progress in high fidelity fast generative models, such as Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs). These models are able to sample high dimensional feature distributions by learning from existing data samples, and they might offer a promising alternative for Fast Simulation. Several efforts in the HEP community have started already on this task. This application is promising to offer orders of magnitude increase in speed over existing Fast Simulation techniques.

- *Sustainable Matrix Element Method (MEM).* MEM is a powerful technique for making measurements of physical model parameters and direct searches for new phenomena. It is very computationally intensive, hence it is usage in HEP is quite limited. Neural networks can be used for numerical integration is not new, but it is a technical challenge to design a network sufficiently rich to encode the complexity of the ME calculation for a given process over the phase space relevant to the HEP signal process. For this task, Deep Neural Networks (DNNs) are good candidates to explore.

3.5.2 Machine Learning in the CMS experiment

ML techniques are an essential part of the CMS experiment since long time, with the power of ML being brought to the next level by Deep Learning in recent years. Only some highlights are offered below.

Examples in the physics analysis sectors are many, e.g. in the analyses $H \to \gamma\gamma$, $HH \to bbll\nu\nu$, and more. In reconstruction, ML is used for example in jet tagging with the task to find the particle id of a jet, e.g. b-quark with key features like the long lifetime of heavy flavour quarks, displaced tracks, etc (jet flavour tagging is also intrinsically a multi-class classification problem): a new ML-based flavour tagger is officially recommended since 2017 in all CMS analyses. Quark-gluon separation are being empowered by the use of convolutional neural networks (CNN), which are also used in ECAL in gamma versus electron showers separation. CNN are also used in CMS in the tracking sector. ML is also used in the Level-1 muon trigger to to try to improve the p_T resolution of the passing muons.

A sector in which ML/DL techniques are starting to be used since few years is also the area of CMS Software/Computing, with the main goals of computing resource Optimization and control of networks and production workflows. CMS data operations is one of the significant challenges for the upcoming HL-LHC. In the current infrastructure, CMS relies on in house solutions for managing the data. While these approaches work reasonably well today, ML can help automate and improve the overall system throughput and reduce operational costs. ML techniques can be applied in many areas of computing infrastructure, workflow and data management. For example, dataset placement optimization [54, 55] and reduction of transfer latency [56] can lead to a better usage of site resources and an increased overall throughput of analysis jobs. One of the current examples is predicting the "popularity" of a dataset from dataset usage, which helps reduce disk resource utilization and improve physics analysis time turn-over [57]. Data volume in data transfers is one of the challenges facing the current computing systems as thousand of users need to access thousands of datasets across the Grid. There is an enormous amount of metadata collected by application components, such as information about failures, file accesses etc. Resource utilization optimization based on this data, including Grid components and software stack layers, can improve overall operations. Understanding the data transfer latencies and network congestion may improve operational costs of hardware resources. Networks are going to play a crucial role in data exchange and data delivery to scientific applications in HL-LHC era. The network-aware application layer and configurations may significantly affect experiments' daily operations. ML applications can be in network security in identifying anomalies in network traffic; predicting network congestion; bug detection via analysis of self-learning networks, and WAN path optimization based on user access patterns.

Chapter 4

S/B discrimination in $t\bar{t}$ selection: a use-case for ML "as a service"

In the previous chapter, an overview on ML techniques was presented. In the examples discussed, it emerged that a variety of ML approaches and algorithms may be adequate for different scientific (or not) problems, and special care is needed in choosing the right approach for each use-case. In the scope of this thesis, we work towards a ML-as-a-service infrastructure, which can be applied to various HEP problems, and we specifically focus on a use case which is the Signal versus Background discrimination in all-hadronic top decays with CMS at the LHC.

Since this work is part of a major computing challenge (a full ML "as a service" infrastructure), a quick description of its design is presented in the early part of this chapter. The chapter follows with a description of how a ML set-up is designed and implemented to target the chosen physics use-case. The actual results of the application of such ML model to the top physics use-case are instead presented. The final section of this chapter quicky overviews and summarizes the work in progress in inserting this model into the ML "as a service" infrastructure.

4.1 Machine Learning "as a service" for CMS

As discussed in Section 3.5, the CMS experiment at CERN exploits various ML techniques in various physics and computing related projects. The construction and deployment of a ML project and its deployment for production use requires specific skills and it is a highly time-consuming task. There are no data science teams stably collaborating with CMS physicists and helping them to achieve their ML objectives. At the same time, the CMS physicists themselves rarely have specific data science skills to face such challenges alone. What is needed to design and run successfully a ML project is often not found in a basic CMS physicist expertize (or a HEP physicist, for what matters) whose primary competences are focussed on high energy physics, data analysis (including statistics), and whose ultimate goal is work towards a physics publication. Facing the need to improve a physics data analysis and understanding that ML might be an interesting exploration is hence just a first step towards actually embracing ML in an analysis. Such awareness of the existence of a gap across "two different worlds" (HEP data analysists and non-HEP data scientists) is admittedly growing in some contexts (see e.g. [50]), and work is being done to attack this issue. But such awareness largely still needs to be acknowledged in the overall worldwide HEP community. For example, in the non-scientific world, a developer who wants to build predictive analytics into their application, a customer success team that needs to know which accounts are likely to churn, or a data scientist looking to run models on faster and cheaper infrastructure, etc had to work alone in the past, while today they can start to shop for the ML product of their choice, a farfetched idea just several years ago. This step has not yet been done in the HEP community, which then risks to fall behind in being able to profit of open source solutions and tools that are available for everyone and may well benefit HEP physics too.

The work presented this thesis, done in the context of the CMS Computing group and in close connections with CMS Computing tools developers, contributes to what is considered to be a very ambitious goal in the medium-long term: build a ML "as a service" solution for CMS Physics needs, namely build an end-to-end data-service to serve ML trained model to the CMSSW framework [58]. The basic idea is as simple as this: instead of asking each physicists who wants to exploit ML in their own task to just learn how to do it and do it themselves independently, each user would ultimately build a modified data analysis code where "calls" to an external service - as simple and calls to functions - would be added to return a trained ML model output that could be directly used in the analysis code (e.g. in loops over events) in a streamlined manner, thus hiding all the complexity related to the ML machinery via outsourcing this to an external service. As it is easily understood, the scope of this thesis cannot cover such a large-scope project: the work done in this thesis is just one step in that direction, focusing mostly on how to prepare a ML task on a specific physics use-case and how to implement it in a way that it can be dealt with by such ML "as a service" infrastructure in the medium-long term. Actually, during this thesis work, it was possible also to reach an additional goal: the set-up a of a fully working TFaaS prototype to serve this physics use-case. This is discussed at the end of the chapter.

In this section, the architecture and components of such ML "as a service" infrastructure are presented and discussed.

4.1.1 Architecture and components

Choice of the ML framework

The popularity of the TensorFlow ML/DL framework makes this an excellent choice to apply ML algorithms for use in the CMS workflow pipeline. TensorFlow (TF) [59] is an open source software library for numerical computation that uses data flow graphs. TensorFlow was originally developed by researchers and engineers working on the Google Brain Team [60] within Google's Machine Intelligence [61] research organization for the purposes of conducting ML and DNN research. The system is general enough to be applicable in a wide variety of other domains as well, though. As stated above, data flow graphs are used. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (called "tensors") communicated between them. The architecture is flexible so to allow users to deploy computation to one or more CPUs (or GPUs) in a desktop, server, or even mobile device with a single API. In a not exhaustive list of features, one could mention:

- TF runs on Windows, Linux, and macOS, and also on mobile devices, including both iOS and Android;
- TF provides a very simple Python API, which is called TF.Learn (tensorflow.contrib.learn), compatible with Scikit-Learn;
- TF provides another simple API called TF-slim (tensorflow.contrib.slim) to simplify building, training, and evaluating NNs;
- TF's main Python API offers much more flexibility (at the cost of higher complexity) to create all sorts of computations, including any NN architecture you can think of;
- TF includes highly efficient C++ implementations of many ML operations, particularly those needed to build NNs. There is also a C++ API to define your own high-performance operations;
- TF provides several advanced optimization nodes to search for the parameters that minimize a cost function: TF automatically takes care of computing the gradients of the functions one defines, i.e. implements automatic differentiating (or autodiff);
- TF also comes with a great visualization tool called TensorBoard that allows you to browse through the computation graph, view learning curves, and more;
- once a model is done with TF, computations can be deployed to one or more CPUs or GPUs, as needed.

Google Tensorflow is of course not the only framework that was built to achieve such objectives. Naming a few possible alternatives, despite not in an exhaustive list:

- *Theano* [62] is a Python library that lets you to define, optimize, and evaluate mathematical expressions, especially ones with multi-dimensional arrays (i.e. ndarray in *NumPy* [63]). For problems involving large amounts of data, Theano allows to attain speeds that rival hand-crafted C implementations, and it was demonstrated to also surpass C on a CPU by orders of magnitude via taking advantage of recently released GPU architectures. Architecturally, Theano combines aspects of a computer algebra system with aspects of an optimizing compiler, a combination that is particularly useful for tasks in which complicated mathematical expressions need to be evaluated repeatedly and the overall evaluation speed is critical for a project. So, in a nutshell it can be seen as a Python library and optimizing compiler for manipulating and evaluating expressions, especially matrix-valued ones. Its advantage with respect to - e.g. pure Python like the *NumPy* library - is that it offers execution speed optimizations (it can use g++/nvcc to compile parts your expression graph into CPU/GPU instructions, and this would run much faster than pure Python), symbolic differentiation (Theano can build symbolic graphs for computing gradients) and stability optimizations (Theano is able to recognize some numerically unstable expressions and then compute them via more stable algorithms). It is often said that the closest Python package to Theano is SymPy [64], but Theano focuses more on tensor expressions than SymPy does, and for sure it has more machinery for compilation, while Sympy focuses more on sophisticated algebra rules, and supports a wide range of mathematical operations. In a sense, if *NumPy* is to be compared to MATLAB [65] and SymPy to Mathematica [66], Theano is a sort of hybrid of the two and tries to take the best of both worlds.

- *Caffe* [67] is a deep learning framework developed by Berkeley AI Research (BAIR) and by community contributors. Its expressive architecture encourages application, while models and optimization are defined by configuration and without hard-coding, so it is intended to ease the use of such a framework to a large community of users. The switch between CPU and GPU can be done by setting a single flag. The focus on performance and speed makes Caffe good for research experiments and industry deployment.

- *Torch* [68] is a scientific computing framework with wide support for ML algorithms that puts GPUs in the first place. It is based on a scripting language called LuaJIT and on an underlying C/CUDA implementation. It provides a powerful N-dimensional array management, lots of routines for indexing/slicing/transposing, linear algebra routines, support for neural networks, etc. The specific goal of Torch is to offer maximum flexibility and speed in building scientific algorithms while making the process as simple as possible: it comes with a a large ecosystem of community-driven packages that can be used for machine learning applied to computer vision, signal processing, parallel processing, image, video, audio, etc. With Torch a user can use all popular neural network and optimization librariesto implement complex neural network topologies, as well as build arbitrary graphs of neural networks, and parallelize them over CPUs/GPUs in an efficient manner.

- *MXNet* [69, 70] is an Apache incubator project for a multi-language ML library to ease the development of ML algorithms, especially focussed on deep neural networks. It blends declarative symbolic expressions with imperative tensor computation, all embedded in the host language. It is computation and memory efficient and can run on various heterogeneous systems, ranging from mobile devices to distributed GPU clusters.

- *CNKT* [71], now known as the Microsoft Cognitive Toolkit, allows to train and evaluate deep learning algorithms by scaling efficiently and working in a range of environments - from a CPU, to GPUs, to multiple machines. Its flexibily covers application of reinforcement learning, generative adversarial networks, supervised and unsupervised learning; it offers automatic hyperparameter tuning, and the ability to add new user-defined core-components on the GPU from Python.

Depending on the problem at hand, one ML framework might be better than others, but in general all of them are capable of dealing with most basic ML or DL applications. Also, they are all still being developed and improved (for example, the open source code of Tensorflow was released in 2015, Caffe in 2013, Theano in 2009), so it often happens that a framework offers a new functionality or gains a new speed factor with respect to competitors: it is part of the work of a data scientist to be up to date with all recent developments in all frameworks, and switch from one tool to another.

For the needs of the CMS experiment, Tensorflow was considered to be a sufficiently high standard: when its code became open source in 2015, some of its features already existed other frameworks, but its clean design, scalability, flexibility, documentation, production-readiness boosted it quickly to the top of the list of the ML frameworks worldwide. These aspects are also relevant for the work in this thesis. As a consequence, the TFaaS architecture has been based on Google Tensorflow.

An important additional aspect must be considered. To ease the use of all these ML frameworks, several other high-level APIs have been built independently on top of TensorFlow (and other frameworks), such as **Keras** [72] (which is now available also inside Tensorflow as *tensorflow.contrib.keras*). In particular, Keras is a high-level neural networks API written in Python. Keras is capable of running (for example) on top of TensorFlow, Theano, CNTK, and it was developed with a focus on enabling fast experimentation: being able to go from idea to result with the least possible delay is key to doing good research. Keras allows for easy and fast prototyping (through user friendliness, modularity, and extensibility), supports both convolutional networks and recurrent networks (as well as combinations of the two), and runs seamlessly on CPUs and GPUs. For these reasons, in this thesis work Keras has been used for a quick prototyping of a TF-based model for TFaaS (see Section 4.5).

The TFaaS architecture

The concept of "as a service" means that some tasks will be eventually performed on resources that are not on premises (namely, they are "on the cloud(s)”). The jargon terminology of “ML as a service” (MLaaS) is a sort of an umbrella definition for any idea that includes the exploitation of automated and semi-automated cloud platforms to cover most infrastructure issues in a ML project such as data preprocessing, model training and model evaluation, down to the final predictions. The prediction results should be bridged with the analyzer’s internal (local) IT infrastructure through REST APIs.

Amazon Machine Learning services [73] - as part of Amazon Web Services (AWS) [74], Azure Machine Learning [75] - as part of Microsoft Azure [76] and Google Cloud AI [77] - as part of Google Cloud Platform (GCP) [78], are the three leading cloud MLaaS services that allow for fast model training and deployment with little or even no data science expertise. These could offer a solution to those who are e.g. assembling a homegrown data science team out of available software engineers or programmers on a specific project, but lacks the actual facilities to deploy any developed solution and to run it at scale. The same needs may apply to HEP researchers who seeks for resources where to run their own ML projects, with an exception: such major cloud providers offer you the resources, but they do not provide you with the middleware needed to run the service you need on those resources. A full interface to the CMSSW framework, the ability to efficiently read the HEP input ROOT files (sometimes locally, sometimes remotely) and the overall orchestration of this cloud service from the inside of HEP analysis jobs are not functionalities that one should expect to be delivered by any cloud company: it is responsibility of each specific HEP experiments to design and develop these middleware components as needed.

For this purpose, CMS launched a project aiming to build a prototype of an end-to-end data-service to serve ML trained model to the CMSSW framework. This has been named **Tensorflow as a Service** (**TFaaS**), with the adoption of TF for this for the reasons discussed in the previous section. Its design and code (still under development) can be found in [79]. The overall architecture and its main components can be seen in Figure 4.1.

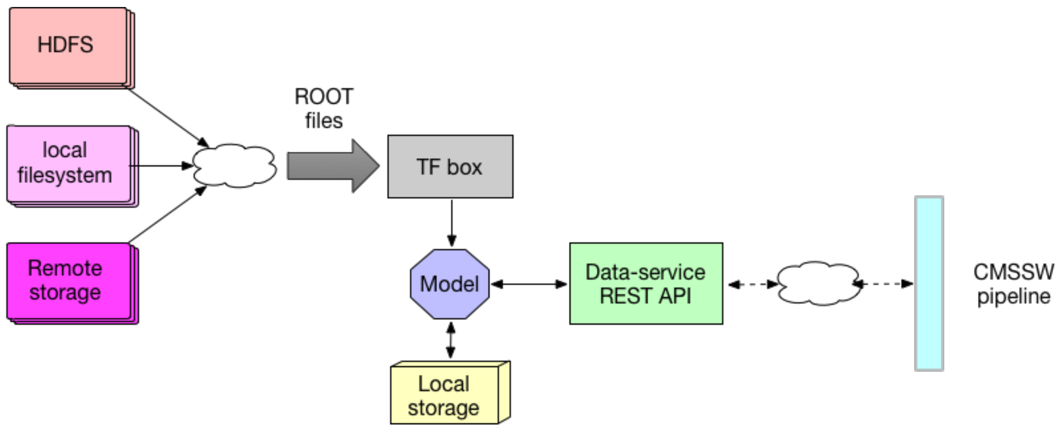


Figure 4.1: Schematic representation of TFaaS architecture (discussion in the text).

The TFaaS is designed to bridge the gap between what HEP ML users need and what cloud companies can provide. As can be seen in Figure 4.1, the CMSSW pipeline would use REST APIs to call a data-service as a front-end, which communicates with the back-end, consisting of local and cloud resources. The local, on premises resources will be primarily used as a storage cache. The model creation and training is actually outsorced to a "TF box" that can run on clouds, to which the needed ROOT files would be served from any storage system that allows XRootD connections, i.e. from virtually any storage system in the Worldwide LHC Computing Grid (WLCG) (see Chapter 2), or from HDFS systems. Such TF box, sitting somewhere on a remote machine, will be able to train the model as requested with the training dataset that can be read remotely, and then users can profit of this model as it will be served back to CMSSW applications in form of the actual predictions that were originally requested (e.g. in an event classification task, the trained model will be able to predict the type of event based on its characteristics).

As it can be extracted from the TFaaS workflow description of the previous paragraph, the main functionalities of TFaaS would be

1. read ROOT files and convert them into a suitable form as input to ML/DL systems;

2. train the model on a TF box and serve the trained ML/DL model via REST API with data exchange via a highly efficient transport layer (this would include to read data remotely and integrate service calls into CMSSW);

3. deploy the service to the cloud (e.g. rent GPUs to train the model), and use it for predictions.

For task 1 above, the DIANA-HEP [80] team - an initiative funded in US by the National Science Foundation and aiming at developing state-of-the-art software tools for HEP experiments which acquire, reduce, and analyze petabytes of data - launched the **uproot** project [81]. *Uproot* provides native access to ROOT IO without any particular knowledge of the ROOT file structure. *Uproot* is implemented as a pure Python ROOT reader that directly copies columnar ROOT data into (for example) *NumPy* arrays.

Despite still in development, its performance has been measured to be of the order of 50 kHz in reading from ROOT files of few GBs [82]. Even if this is beyond the scope of this thesis, a quick test to directly compare the event reading thoughput obtainable in reading ROOT files with *root2numpy* [83] versus *uproot* has been performed, and the results are shown in Figure 4.2. It must be noted that this

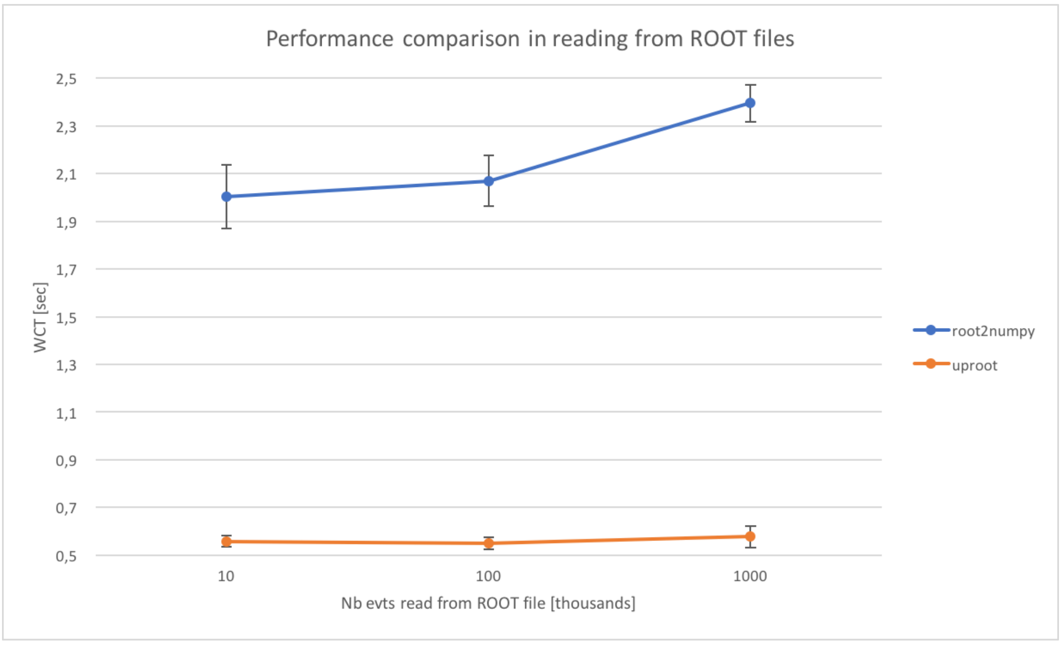


Figure 4.2: A quick performance comparison in reading from ROOT files using *root2numpy* and *uproot* (see text for discussion).

should not be taken as a strict performance comparison between the twp tools, because the testbed would need to be set-up much more carefully (e.g. file reading from local disks and not *afs* areas, additional care for possible causes of overhead in the timing in the two cases, more tests and hence more data points, etc). Still, a relative comparison is meaningful: as can be seen in the Figure 4.2, *i)* the overall events reading throughput with *uproot* is better than with *root2numpy* for any number of events read; *ii)* the throughput observed with *uproot* scales well with increasing number of events read, which is not the case for *root2numpy*, which might become a problem for large number (e.g. hundreds) of multi-GBs ROOT files on a production-scale infrastructure. Another aspect to mention is that *uproot* allows not only the reading of local ROOT files, but also the reading of remote files via XRootD [84]. It is worth noting, as will be further discussed in Section 4.3.1, that there is no special requirements on the input ROOT files, and *uproot* - also thanks to the work done in this thesis - is capable of dealing with any flat *TTree* structure in input.

For task 2 above, the transport layer exploited in the TFaaS architecture is the **Protocol buffers**. Developed by Google, protocol buffers (or "*protobuff*") are a language-neutral, platform-neutral extensible mechanism for serializing structured data. One could think of XML, but - naively summarizing - smaller, faster, and simpler. A user defines how data need to be structured once, and then one can use special generated source code to easily write and read the structured data to and from a variety of data streams and using a variety of languages. Protocol buffers currently supports generated code in Java, Python, Objective-C, and C++. The newest language version ("proto3") works also on Go, JavaNano, Ruby, and C#, with more languages to come.

In the scope of this thesis, a demonstration of a TFaaS working prototype was achieved, as it will be briefly discussed in Section 4.5.

4.2 S/B discrimination with ML

Most HEP analyses needs mechanisms to perform event classification. In particular, methods to discriminate signal events from background events, depending on the actual physics goal, are needed. This might be done at the event level (e.g. Higgs searches, or SUSY searches, or Top-mass measurement, ..), at the cone level (e.g. Tau versus quark-jet reconstruction, ..), at the track level (e.g. particle identification, ..), in the secondary vertex finding (e.g. b-tagging), in the flavour tagging, etc. This can only be done with input information from multiple variables coming from a variety of sources, like kinematic variables (masses, momenta, decay angles, ..), event properties (jet/lepton multiplicity, sum of charges, ..), event shape (sphericity, various type of high-order moments, ..), detector response (silicon hits, dE/dx, Cherenkov angle, shower profiles, muon hits, ..). Traditionally - despite different on a case by case basis - this is done by exploiting few powerful input variables and combine them. Nowadays, new ML-based methods allow to use up to 100 (and more) variables without loss of classification power.

Suppose one analyst has a data sample that consists of two types of events: in real life they will be mixed up together in the same sample, but we can use simulations to obtain samples of both, separate types and mix them together with class labels S for "Signal" and B for "Background" (in blue and red respectively) in Figure 4.3 (note we are restricting here to just two class cases, while many classifiers would be able to also deal with several classes). The question is how to set the

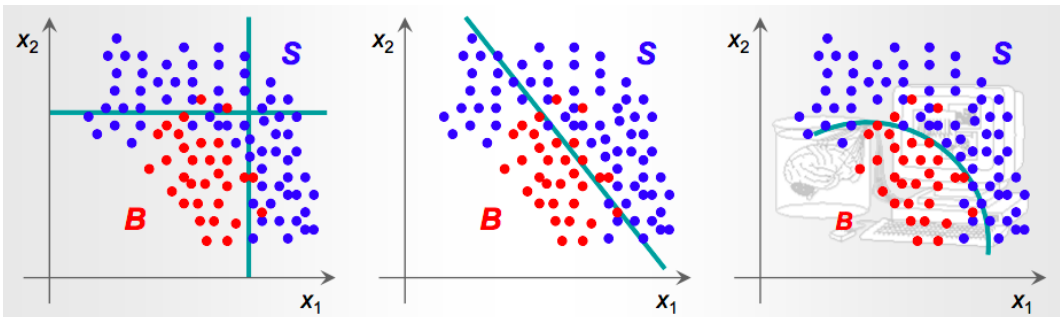


Figure 4.3: Example of Signal versus Background discrimination using decision boundaries. Discussion in the text.

decision boundaries, in terms of cut on specific variables, in order to select events of type S at best, given that we only have a limited set of discriminating variables - let's suppose only two, x_1 and x_2 in this example. In the three plots of Figure 4.3, from left to right, one is trying to achieve the goal by applying rectangular cuts (plot on the left), a linear boundary (plot in the center), a non linear boundary (plot on the right). Once decided on a set of possible boundaries, how to find the optimal one? The problem, as from this example, might seem very simple, but this is only because this example has 2 variables, and the human eye, and brain behind, have very good pattern recognition capabilities. These capabilities might become much weaker if we have more variables, e.g. dozens of them. Suppose that each event, if Signal or Background, has indeed "N" measured variables, which we can call "features". These many variables characterize the members of a given population: the same variables for signal and background are used, but the two groups have different PDFs. The problem becomes finding a mapping from a N-dimensional input/observable/"feature" space to one-dimensional output, i.e. a function that has N x_i arguments and gives a single variable as output. With a rule-based approach, PDF distributions of S and B are lotted and a cut can be found that maximizes the efficiency and the purity of the selected Signal sample, estimated on the simulated data. This is basically the idea (despite simplified here) behind a "Multivariate Analysis" (MVA) technique. And this is where ML techniques may naturally apply. Given a certain type of model class, a ML system might be able to automatically find the mapping discussed above, by using "known" or "previously solved" events, i.e. learning from known "patterns", such that the result output variable has good generalization properties when applied to really "unknown" events. This is precisely what a machine is supposed to be doing when applying supervised machine learning algorithms. Of course, there is no magic: one still needs to choose the discriminating variables, choose the class of models, tune the "learning parameters" (bias versus variance trade off), check generalization properties, consider trade off between statistical and systematic uncertainties, etc.

Operationally, what happen is that a program is "trained" on a predefined set of "training examples", which empowers its ability to reach an accurate conclusion when given new data. Note however that the goal of ML is never to make "perfect" guesses. The "best" values i.e. how "good" or "better" than others they are, depend essentially on which level of precision in the prediction one's problem needs, which quality of ML models we can afford to apply, e.g. also taking into account the non-infinite amount of computing resources we may have available to run a ML system. Ultimately, the goal of any successful ML effort is never to reach perfection, but only and always to make guesses that are good enough to be useful for the problem under study.

In the directions to follow to build a successful ML model, much care must be given to the choice of the training data. Regardless of how such data is operationally divided into in the model implementation (e.g. training vs validation vs test sub-samples) and considering instead "training data" the whole set of data used to build up the model, it is worth underlying that this must be a statistically significant random sample – as ML builds heavily on statistics. If the sample is not random, the price to pay is that the machine learns patterns in the data that are not actually there. If the sample is not large enough, the price to pay is that the machine will not learn enough, or (even worse) reach inaccurate conclusions.

4.3 Building a ML model for the $t\bar{t}$ selection use-case

As discussed in the previous section, supervised ML is a powerful tool for signal vs background discrimination in HEP. In this section we will present and discuss the steps followed to build up a supervised ML model for signal versus background discrimination in the fully-hadronic top analyses performed in CMS.

In the next subsections we will explain: *i)* how data has been prepared to be handled by ML algorithms (data preparation), *ii)* how imported data have been validated (data validation), *iii)* how the algorithm has been choosen and how the ML model has been built, and *iv)* how hyperparameters have been tuned.

A discussion of the physics use-case will be in Section 4.4.

4.3.1 Data preparation

The first challenge to face is the data preparation step. The core of the problem is that every input data for this study is in the form of ROOT data files. It is widely known why this format is used by the HEP community, but it has to be remarked that this format is not the most suitable to directly apply ML learning techniques. The first task to cope with has been to find a mean to "transition" from any ROOT input files to some format that is more adequate to interface with any existing ML framework that is used in other (non HEP) sciences, or in non-scientific sectors. The best design choice is enforced in the TFaaS architecture as of reading the ROOT file content into *NumPy* arrays to be kept in memory and further used with ML in a streamlined manner. On the other hand, for operational reasons of simplicity in debugging and data checking, the code was developed to also allow a dump of the data onto local disk as a tabular CSV (comma separated values) format, i.e. a format in which tabular data (numbers and text) are stored in plain text, with each line of the file acting as a data record, and each record consisting of one or more fields, separated by commas.

The task reduces itself then to coding the process of transformation from ROOT to CSV. This is definitely not a pioneering task. In the HEP community, a variety of interfaces to ROOT have been popping up over the time, ranging from more structured efforts to one-shot scripts written by individuals upon their specific needs. It must be noted, though, that a performant and scalable solution was seeked for this task, and compatibility with the TFaaS design was a requirement. TFaaS offers the *uproot* component for this task, as presented in 4.1.1. We actually evaluated also one alternative approaches, i.e. the use of *root2numpy*, a Python extension module that provides an efficient interface between ROOT and *NumPy*. From its documentation, it is reported that its internals are compiled in C++ and can therefore handle large amounts of data much faster than equivalent pure Python implementations. In our tests, though, we experienced *uproot* to be more performant 4.1.1: given it is natively supported in the TFaaS architecture, we opted for its adoption in the data preparation phase.

Another factor to consider is that the Monte Carlo samples and the data samples used for this study are officially produced by the CMS top PAG (Physics Analysis Group) and exploited for analysis by the all-hadronic top sub-group, and are used in this thesis as they are produced by the aforementioned teams, with no change or manipulation. They are organized with a flat structure, made by a *TTree* with its branches. It is crucial that any way to read such ROOT file to perform the transformation can handle such data organization. The *uproot* component has the native capability to deal with ROOT files with such flat structure, so this stood as an additional motivation for the adoption of *uproot*.

In order to operationally implement this choice, the author of this thesis collaborated to the coding and debugging of a scalable python script specifically for this purpose inside the overall TFaaS prototype design. This script acts as a high-performance wrapper to *uproot* in order to read ROOT files in a streamlined and performing way. The development of this component took a non-negligible part of the thesis work, in constant contact with the core *uproot* developers. Some of the main issues addressed and fixed are:

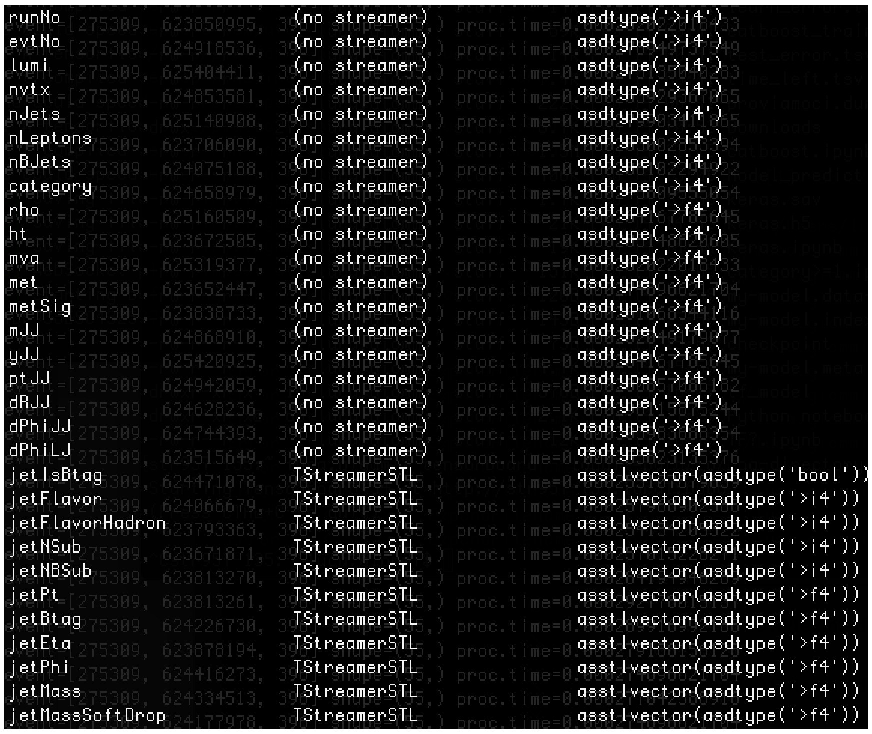
- ROOT file reading (that were not correctly and completely written) [85],
- the costruction of the *jagged arrays* [86] that allow to read arrays from ROOT files,
- the reading of vectors of booleans [87].

The fix to these issues resulted into new releases of the *uproot* component, which is gaining solidity over time also thanks to this thesis work. On the other hand, in this thesis all most recent beta versions functionalities could be proficiently exploited. The work in collaboration with the *uproot* development team will continue also after this thesis dissertation.

The aforementioned python script used to read ROOT files via *uproot* calls is complex and not fully reported verbatim in this thesis, but a small snapshot of how its main concept applies is below:

```
import uproot
t=uproot.open("input_file.root")["events"]
t.show()
for (array,) in t.iterate("triggerBit", entrysteps=1000,
          outputtype=tuple):
     print array
```

where it strikes for its simplicity to an end-user. In the first line the user imports the *uproot* python module. In the second line, *uproot* is used to open a ROOT file named *input_file.root* which has inside a *TTree* named *events*, that gets imported. In the third line, the content of the *TTree* is shown/printed, so that on the screen the user can see all the branches and their related type (see Figure 4.4). The last lines are the simplest a user can code in order to read the content of the file, using event bunches of a given size (1000), in particular from a specific branch (*triggerBit* in this example).



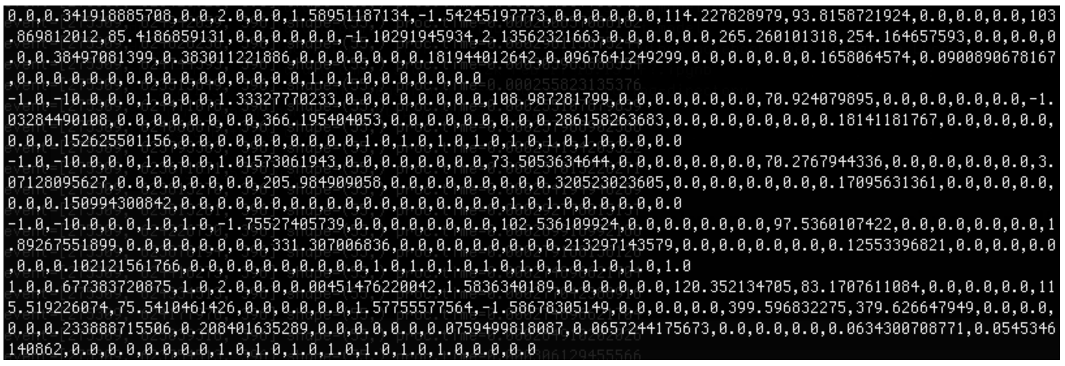
```
runNo            (no streamer)    asdtype('>i4')
evtNo            (no streamer)    asdtype('>i4')
lumi             (no streamer)    asdtype('>i4')
nvtx             (no streamer)    asdtype('>i4')
nJets            (no streamer)    asdtype('>i4')
nLeptons         (no streamer)    asdtype('>i4')
nBJets           (no streamer)    asdtype('>i4')
category         (no streamer)    asdtype('>i4')
rho              (no streamer)    asdtype('>f4')
ht               (no streamer)    asdtype('>f4')
mva              (no streamer)    asdtype('>f4')
met              (no streamer)    asdtype('>f4')
metSig           (no streamer)    asdtype('>f4')
mJJ              (no streamer)    asdtype('>f4')
yJJ              (no streamer)    asdtype('>f4')
ptJJ             (no streamer)    asdtype('>f4')
dRJJ             (no streamer)    asdtype('>f4')
dPhiJJ           (no streamer)    asdtype('>f4')
dPhiLJ           (no streamer)    asdtype('>f4')
jetIsBtag        TStreamerSTL     asstlvector(asdtype('bool'))
jetFlavor        TStreamerSTL     asstlvector(asdtype('>i4'))
jetFlavorHadron  TStreamerSTL     asstlvector(asdtype('>i4'))
jetNSub          TStreamerSTL     asstlvector(asdtype('>i4'))
jetNBSub         TStreamerSTL     asstlvector(asdtype('>i4'))
jetPt            TStreamerSTL     asstlvector(asdtype('>f4'))
jetBtag          TStreamerSTL     asstlvector(asdtype('>f4'))
jetEta           TStreamerSTL     asstlvector(asdtype('>f4'))
jetPhi           TStreamerSTL     asstlvector(asdtype('>f4'))
jetMass          TStreamerSTL     asstlvector(asdtype('>f4'))
jetMassSoftDrop  TStreamerSTL     asstlvector(asdtype('>f4'))
```

Figure 4.4: Output of the `show()` function. See text for details.

In summary, a python script which is part of the TFaaS infrastructure and which internally uses *uproot* has been co-developed by the author of this thesis work. It efficiently allows to convert files from the ROOT format into a structure (CSV) that is simpler and manageable by all commonly used ML frameworks and algorithms. In Figure 4.5, an example of the headers of a CSV file produced by this tool is shown. This concludes the data preparation part.

4.3.2 Data validation

A generic ML project may suffer a lot from missing attribute values for some data entries, or subsets of data which have been collected with less stringent quality than the rest of the data, etc. A "data validation" step is strongly suggested in any



```
0.0,0.341918885708,0.0,2.0,0.0,1.58951187134,-1.54245197773,0.0,0.0,0.0,114.227828979,93.8158721924,0.0,0.0,0.0,103
.869812012,85.4186859131,0.0,0.0,0.0,-1.10291945934,2.13562321663,0.0,0.0,0.0,265.260101318,254.164657593,0.0,0.0,0
.0,0.38497081399,0.383011221886,0.0,0.0,0.0,0.181944012642,0.0967641249299,0.0,0.0,0.0,0.165806457,0.0900890678167
,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,1.0,0.0,0.0,0.0
-1.0,-10.0,0.0,1.0,0.0,1.33327770233,0.0,0.0,0.0,0.0,108.987281799,0.0,0.0,0.0,0.0,70.924079895,0.0,0.0,0.0,0.0,-1.
03284490108,0.0,0.0,0.0,0.0,366.195404053,0.0,0.0,0.0,0.0,0.286158263683,0.0,0.0,0.0,0.0,0.18141181767,0.0,0.0,0.0,
0.0,0.152625501156,0.0,0.0,0.0,0.0,0.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,0.0,0.0
-1.0,-10.0,0.0,1.0,0.0,1.01573061943,0.0,0.0,0.0,0.0,73.5053634644,0.0,0.0,0.0,0.0,70.2767944336,0.0,0.0,0.0,0.0,3.
07128095627,0.0,0.0,0.0,0.0,205.984909058,0.0,0.0,0.0,0.0,0.320523023605,0.0,0.0,0.0,0.0,0.17095631361,0.0,0.0,0.0,
0.0,0.150994300842,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,1.0,0.0,0.0,0.0
-1.0,-10.0,0.0,1.0,1.0,-1.75527405739,0.0,0.0,0.0,0.0,102.536109924,0.0,0.0,0.0,0.0,97.5360107422,0.0,0.0,0.0,0.0,1
.89267551899,0.0,0.0,0.0,0.0,331.307006836,0.0,0.0,0.0,0.0,0.213297143579,0.0,0.0,0.0,0.0,0.12553396821,0.0,0.0,0.0
,0.0,0.102121561766,0.0,0.0,0.0,0.0,0.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0
1.0,0.677383720875,1.0,2.0,0.0,0.00451476220042,1.5836340189,0.0,0.0,0.0,120.352134705,83.1707611084,0.0,0.0,0.0,11
5.519226074,75.5410461426,0.0,0.0,0.0,1.57755577564,-1.58678305149,0.0,0.0,0.0,399.596832275,379.626647949,0.0,0.0,
0.0,0.233888715506,0.208401635289,0.0,0.0,0.0,0.0759499818087,0.0657244175673,0.0,0.0,0.0,0.0634300708771,0.0545346
140862,0.0,0.0,0.0,0.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,0.0,0.0
```

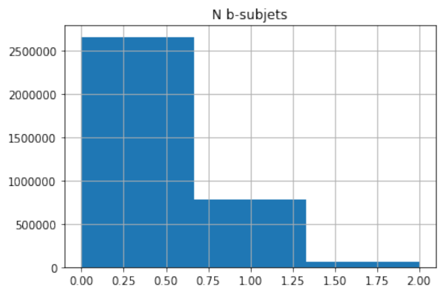
Figure 4.5: Heading lines of a CSV file produced by the tool discussed in the text. Records separated by commas are visible.

ML project, before jumping into the time-consuming phase of building and training a ML model.

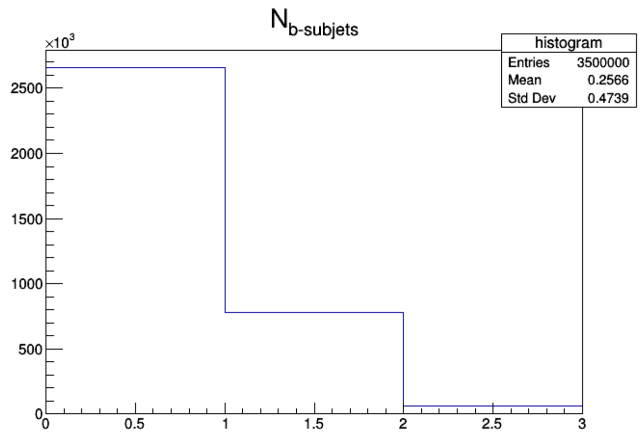
In this specific study, the data contained in the original ROOT files from the top physics group are validated by construction within the CMS frameworks that have been used the CMS procedures that have been followed to produce such data (mainly, coming from the CMS Physics Performance and Dataset coordination area and the CMS Software/Computing coordination area). We are not seeking any control over the correctness of the physics information here, but a "translation" step from a format to another was performed in the previous Data Preparation step, so it is wise to check for the consistence of the physics information contained in the newly created format with respect to the original ROOT format. Any inconsistencies may impact the physics meaningfulness of the overall exercise, and - if found - should be fixed immediately before proceeding.

As Data Validation step, a direct comparison of *all* variables in the ROOT file and in the CSV file has been performed. To do so, a simple python-based pipeline has been designed on a Jupyter notebook for efficient and interactive plot making and content checking. It must be noted that, in principle (as stated in the previous subsection) inside the TFaaS architecture there is no need to build and externally save any CSV file: once data are read from ROOT files, they can be inserted into *NumPy* arrays and kept in memory to be further used by ML algorithms. Nevertheless, one of the reasons why it was opted to dump CSV files out to local disk is indeed to facilitate also the Data Validation pipeline: with this choice, having CSV files on local disks, we can cleanly separate the two steps of the pipeline, i.e. the ROOT data reading in input, and the data validation/manipulation/correction (if needed).

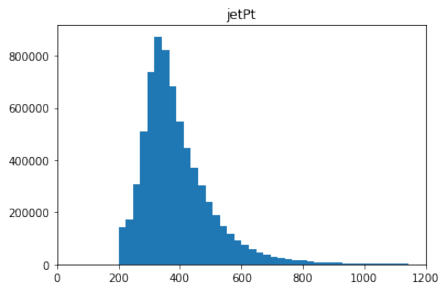
The Data Validation step proceeded as follows. For each ROOT file that contains data that are needed to perform ML, the TFaaS python script that read such data using *uproot* has been launched, and it converted the data into a CSV file. The number of branches/columns and the number of items inside each have been compared to the original information from ROOT. All the items inside each branch of each file were plotted on the Jupyter notebook (using the Matplotlib module) and all (both content-wise with code, and visually by e.g. comparing histograms) where checked for consistency against the original plots that could be traditionally done in a ROOT TBrowser. PyROOT [88] was used to perform these plots from ROOT files. This was done for all variables, but as examples: Figure 4.6a shows the $N_{b-subjets}$ variable, that contains integers, to be compared with Figure 4.6b from ROOT; Figure 4.6c shows the distribution of the "jetPt" variable, a vector of floating point values, against Figure 4.6d. The ROOT file and the produced CSV file have the exact same content for all variables. This concludes the Data Validation step.



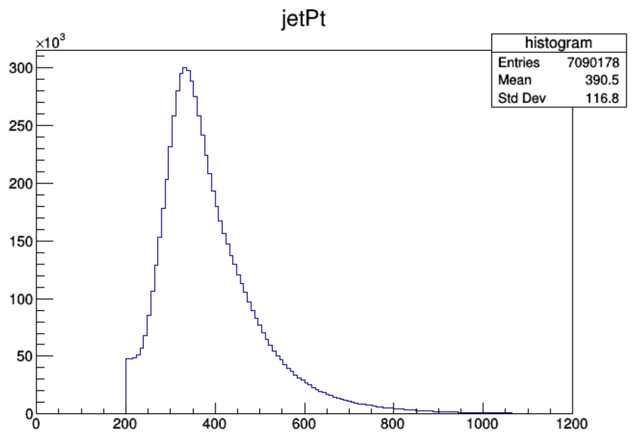
(a)



(b)



(c)



(d)

Figure 4.6: Two examples of the Data Validation step. An integer observable inside the branch $N_{b-subjets}$ and a vector of floating point values inside the branch "jetPt" after ROOT-to-CSV conversion are plotted via matplotlib from a Jupiter notebook ((a) and (c) respectively) and the same information with PyROOT from the ROOT files ((b) and (d) respectively).

4.3.3 Algorithm selection and model creation

In the previous sections, it was discussed how to prepare the data for ML - i.e. converting a ROOT file into *NumPy* arrays and/or a CSV (see Section 4.3.1) - and how to validate the data (see Section 4.3.2) before proceeding further. After these steps, a ML model for our task can start to be built.

First of all, a training phase is necessary to teach the model how to recognize and to distinguish signal from background. The CSV file used to train the model has a column called "prediction" that informs the model whether each event is signal (prediction=1) or background (prediction=0). In the literature, suggestions as of how to build a model that performs can be easily found (e.g. [89] or elsewhere):

A) **Definition of metrics to evaluate the model**: choose model evaluation metrics to quantify the model performance;

B) **Model training and testing**: estimate how well a model will generalize to out-of-sample data;

C) **Algorithm selection**: choose between different model classes, different algorithmic implementations.

In the scope of this thesis, all these steps have been followed, as discussed in the following sections of this chapter.

Another point worths a mention. In typical classification problems, two types of algorithms (dependent on the kind of output it creates) can be found:

- Some algorithms just create a *class output*, i.e. in a binary classification problem, the outputs will be either 0 or 1. Nowadays, also algorithms which can convert these class outputs to probability can be found (despite their use is somehow debated in the statistics community).
- Some algorithms instead give a *probability output*. Converting probability outputs to class output is just a matter of creating a threshold probability.

Examples of class output algorithms are e.g. SVM and kNN; examples of probability output algorithms are e.g. Logistic Regression, Random Forest, Gradient Boosting, Adaboost, etc (some more details can be found later in this same chapter).

In the scope of this thesis, a ML algorithm with probability output will end up to be selected as the "best" model, i.e. one from the second group discussed above. In this case, it was opted for the default value of 0.5 as threshold probability, i.e. an event is signal or background if that threshold is greater or lower than 0.5, respectively.

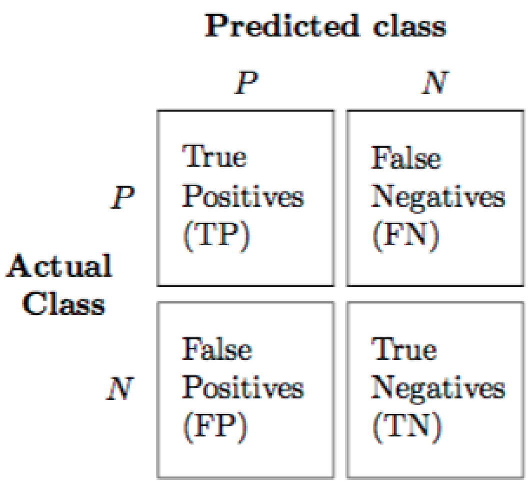
A) Definition of metrics to evaluate the model

The metric chosen to evaluate a set of ML algorithms is another crucial aspect. Its choice influences how the performance of ML algorithms is measured and compared among each other, and determine how the importance of different characteristics are weighted in the results and how the ultimate choice of which algorithm to choose is done (more on this topic can be found in literature e.g. in [90] or elsewhere). On the easy side, classification problems - like the one we are dealing with - are perhaps the most common type of ML problems: as such, there is a myriad of metrics that can be considered for adoption, in order to evaluate ML predictions for these problems.

- **Accuracy**. The classification accuracy is the number of correct predictions made as a ratio of all predictions made. This is the most commonly used evaluation metric for classification problems, with the drawback that it is also the most misused. It is really only suitable when there is an equal number of observations in each class (which is rarely the case) and when all predictions and prediction errors are equally important, which is often not the case.

- **Logarithmic Loss**. Logarithmic loss (or "logloss") is a performance metric for evaluating the predictions of probabilities of membership to a given class. The scalar probability between 0 and 1 can be seen as a measure of confidence for a prediction by an algorithm. Predictions that are correct or incorrect are rewarded or punished proportionally to the confidence of the prediction.

- **Confusion Matrix**. For supervised learning with two possible classes, all measures of performance are based on four variables obtained from applying the classifier to the test set: *TP* (true positive), *TN* (true negative), *FP* (false positive), *FN* (false negative). True Positive are those which we guess true (1) and really are true, True Negative are those which we guess false (0) and really are false, False Positive are those which we guess true and really are false, False Negative are those which we guess false and really are true. The confusion matrix is a specific table layout that allows visualization of all these values (see Figure 4.7): each column of the matrix represents the instances in a predicted class while each row represents the instances in an actual class.



		Predicted class	
		P	*N*
Actual Class	*P*	True Positives (TP)	False Negatives (FN)
	N	False Positives (FP)	True Negatives (TN)

Figure 4.7: Confusion matrix.

From the core elements of a confusion matrix, some more refined (and widely used) metrics, or "scorers", can be built:

- **Accuracy** = (TP+TN)/n
- **Precision** = TP/(TP+FP)
- **Recall** = TP/(TP+FN)
- **F1** = 2TP/(2TP+FP+FN)

The *Accuracy* was also discussed above; in the formula, n is the sum of TP+TN+FP+FN, i.e. it is the number of train examples. The *Precision*, also called *Positive predictive value*, is the proportion of predicted positives which are actual positive. The *Recall* is the proportion of actual positives which are predicted positive. The *F1* is the harmonic mean of Precision and Sensitivity (or TP rate)).

- **Area Under ROC Curve** (**AUC**). ROC stands for Receiver Operating Characteristic. The AUC represents the ability of the model to discriminate between positive and negative classes. An area of 1.0 represents a model that made all predictions perfectly. An area of 0.5 represents a model as good as random. A ROC (an example of which is shown in Figure 4.8) can be broken down into Sensitivity and Specificity:
  - *Sensitivity* is the TP rate, also called "recall". It is the number instances from the positive (1) class that were actually predicted correctly.
  - *Specificity* is the TN rate. It is the number of instances from the negative class (0) class that were actually predicted correctly.

  Indeed, a binary classification problem is really a trade-off between Sensitivity and Specificity.

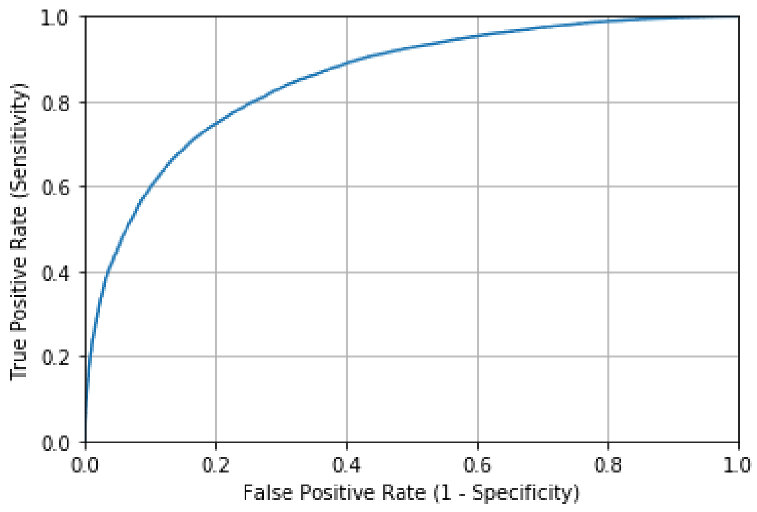


Figure 4.8: Example of a ROC curve. Discussion in the text.

Before proceeding, it is important to decide on the evaluation metrics to be used. The classification accuracy itself is the easiest classification metric to understand, but - if used alone - it has the drawback that it does not tell the underlying distribution of response values and what "types" of errors the classifier is making. The full confusion matrix gives a more complete picture of how a classifier is performing and allows to compute various other classification metrics, which can guide the analyzer in the model selection. In any case, the choice of a specific metric still may vary depending on the final objective. The AUC is useful as a single number that summarizes the classifier performance, with a higher value corresponding to a better classifier; it is anyway really useful even when there is high class imbalance (unlike the classification accuracy). Another thing to note is that AUC does not require to set a classification threshold.

In the scope of my thesis, a good balance in statistics between the signal and background samples occurs, so in principle the accuracy scorer is adequate; at the same time, as discussed above, the AUC is a more refined scorer, it does not require a classification threshold, and still offers a ranking of the classifier performance as a single number. In this work, given it is wise to use at least a couple of metrics, both accuracy and AUC will be used as model scorers.

B) Model training and testing

The performance evaluation for a ML algorithm applied to a specific problem goes through making predictions for new data about which one already knows the correct answer. One cannot anyway work on a ML algorithm on the training dataset and use predictions from this same dataset to evaluate the model performance, because of the overfitting issue.

In statistics, goodness of fit refers to how closely predicted values from a model match the observed (true) values. A model that - so to say - has learned the background instead of the signal is considered "overfit" because it fits the training dataset but has poor fit with new data sets [91]. It is anyway possible to better understand overfitting by looking at the opposite problem, i.e. underfitting. Underfitting occurs when a model is - in a way - just too simple, i.e. it is informed by too few features or regularized too much, so it is not learning flexibly enough from the dataset. Defining the "bias" as the difference between the model expected predictions and the true values, and the "variance" as the algorithm sensitivity to specific sets of training data, the literature in ML shows that simple learners tend to have less variance in their predictions but more bias towards wrong outcomes (see Figure 4.9). On the other hand, more complex learners tend to have more variance in their predictions. Both bias and variance are forms of prediction errors in ML, and typically the error from the bias can be reduced but might increase the error from the variance as a result, or vice-versa. This trade-off between too simple (high bias) versus too complex (high variance) is a key concept in statistics and ML, and one that affects all supervised learning algorithms.

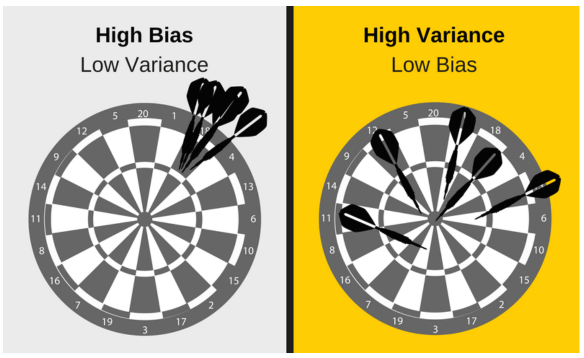


Figure 4.9: Pictorial representation of bias and variance trade-off [92].

A key challenge with overfitting, and with ML in general, is that one cannot know how well the model will perform on new data until one actually test it on this new data. In order to overcome overfitting, it is absolutely mandatory to evaluate any ML algorithm that is being considered on data that are *not* used to train the model itself. This process is documented in literature ([93] and elsewhere). The evaluation of a model is an estimate that we can use to understand how well an algorithm may actually do once used in practice. As such, it is not a guarantee of performance in itself. Once the model has been evaluated, then the algorithm will have to under-go re-training on the entire training dataset and get it ready for operational use.

This is the basic idea for a whole class of model evaluation methods called "cross validation". This is needed in the analysis presented in this thesis, so two approaches are briefly discussed below, and the motivations that lead to pick one of them will follow.

The **holdout method** is the simplest kind of cross validation [94]. The data set is separated into two sets, called the "training set" and the "testing set". In general, the relative size of the split can depend on the size and specifics of your dataset, although it is common to use 67% of the data for training and the remaining 33% for testing. The approximator function fits a function using the training set only. Then the approximator is asked to predict the output values for the data in the testing set (outout values that have never been seen before). The errors it makes are accumulated to give the mean absolute error on the test set, which is used as a model evaluation metric. The holdout method is ideal for large datasets (millions of records) where there is strong evidence that both data splits will still be statistically representative of the underlying problem. Because of its speed, it is also useful to use this approach when the algorithm is slow to train. Despite these positive factors, its evaluation can have a high variance, especially if the statistics is not so high. In such conditions, the evaluation may depend heavily on which data points end up in the training set and which end up in the test set, and thus the evaluation may be significantly different.

K-fold cross validation is one possible way to improve over the holdout method discussed above. The data set here is divided into k subsets (called "folds"), and the holdout method is repeated k times (e.g. where k is set to 5, 7, 10, or so). Each time, one of the k subsets is used as the test set and the other k-1 subsets are put together to form the training set (see Figure 4.10). Then, the average error across all k trials is computed, together with the standard deviation. The advantage of this method is to be less sensitive to how the data division happens to be made: every data point gets to be in a test set exactly once, and gets to be in a training set k-1 times. The variance of the resulting estimate is reduced as k is increased. The choice of k must allow the size of each test partition to be large enough to be a reasonable sample of the problem, whilst allowing enough repetitions of the train-test evaluation of the algorithm to provide a fair estimate of the algorithms performance on unseen data. The disadvantage of this method is that the training algorithm has to be rerun from scratch k times, which means it takes k times as much computation to make an evaluation, thus resulting evidently slower (note that there are several variants of this method, e.g. one is to randomly divide the data into a test and training set k different times, so one can freely choose how large each test set is and how many trials one averages over).

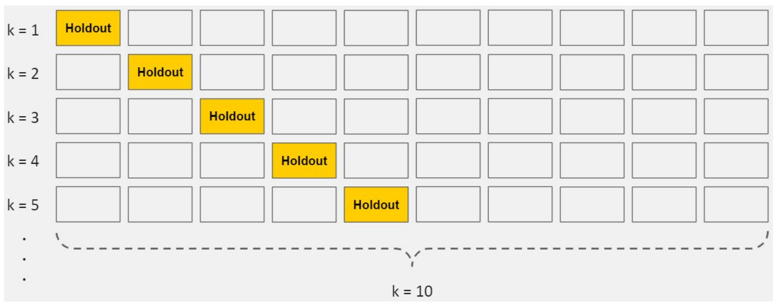


Figure 4.10: Pictorial representation of 10-Fold Cross Validation [91].

In the scope of this thesis, k-fold cross validation is used: it is indeed a powerful way to improve over the holdout method discussed above. It definitely increases the model validation, but this work is not time-constrained (at least not in this exploratory phase) and this drawback is hence felt not as a major limitation against the adoption of k-fold cross validation.

C) Algorithm selection

The next crucial phase is to select the best ML algorithm that fits our use-case. As it is not possible to know a-prori which algorithms are best suited to the specific problem, the only way to go is to try a number of algorithms, and focus with further attention only on those that prove themselves to be the most promising. In this step, spot-checking is a way of discovering which algorithms perform well on a specific ML problem. In the following, several algorithms will be listed by category: a full description of how all of these ML algorithms work in details and how they can be built from scratch goes beyond the scope of this thesis (more can be found e.g. on the *scikit-learn* web page [95]).

For the application of interest in this thesis work, an evaluation of two *linear* ML algorithms is a good starting point:

- **Logistic Regression**. Logistic regression assumes a Gaussian distribution for the numeric input variables and can model binary classification problems. Logistic regression is named for the function used at the core of the method, the logistic function.

- **Linear Discriminant Analysis** (LDA). It is a statistical technique for binary and multi-class classification. It is a classifier with a linear decision boundary, generated by fitting class conditional densities to the data and using Bayes' rule. The model fits a Gaussian density to each class, assuming that all classes share the same covariance matrix.

In addition to the algorithms above, also few *non-linear* ML algorithms have been considered for the use-case of this thesis work:

- **k-Nearest Neighbors** (kNN). It uses a distance metric to find the k most similar instances in the training data for a new instance and takes the mean outcome of the neighbors as the prediction.

- **Classification and Regression Trees**. Also named CART, or just decision trees, they construct a binary tree from the training data. Split points are chosen greedily by evaluating each attribute and each value of each attribute in the training data in order to minimize a cost function (like the Gini index).
- **Naive Bayes**. Naive Bayes calculates the probability of each class and the conditional probability of each class given each input value. These probabilities are estimated for new data and multiplied together, assuming that they are all independent (a simple or naive assumption). When working with real-valued data, a Gaussian distribution is assumed to easily estimate the probabilities for input variables using the Gaussian Probability Density Function.

Another possibility is to combine different models into so-called "*ensemble*" predictions. Among the most popular classes of such methods for combining the predictions from different models, it is worthwhile to quote:

- **Bagging**, that builds multiple models (typically of the same type) from different subsamples of the training dataset.
- **Boosting**, that builds multiple models (typically of the same type) each of which learns to fix the prediction errors of a prior model in the sequence of models.

Let's dig more into Bagging ensemble algorithms for a moment: more will be added later also on Boosting ensemble algorithms. The Bagging ensemble algorithms (also known as Bootstrap Aggregation), they involve taking multiple samples from the training dataset (with replacement) and training a model for each sample. The final output prediction is averaged across the predictions of all of the sub-models. The three bagging models that were considered to be promising for this thesis work are:

- **Bagged Decision Trees**. Bagging performs best with algorithms that have high variance. A popular example are decision trees, often constructed without pruning.
- **Random Forest**. Random Forests is an extension of bagged decision trees. Samples of the training dataset are taken with replacement, but the trees are constructed in a way that reduces the correlation between individual classifiers. Specifically, rather than greedily choosing the best split point in the construction of each tree, only a random subset of features are considered for each split.
- **Extra Trees**. Extra Trees are another modification of bagging where random trees are constructed from samples of the training dataset.

Back on digging more into Boosting ensemble algorithms, they create a sequence of models that attempt to correct the mistakes of the models before them in the sequence. Once created, the models make predictions which may be weighted by their demonstrated accuracy and the results are combined to create a final output prediction. The two most common Boosting ensemble algorithms are:

- **AdaBoost**. It was perhaps the first successful boosting ensemble algorithm. It generally works by weighting instances in the dataset by how easy or difficult they are to classify, allowing the algorithm to pay less attention to them in the construction of subsequent models.

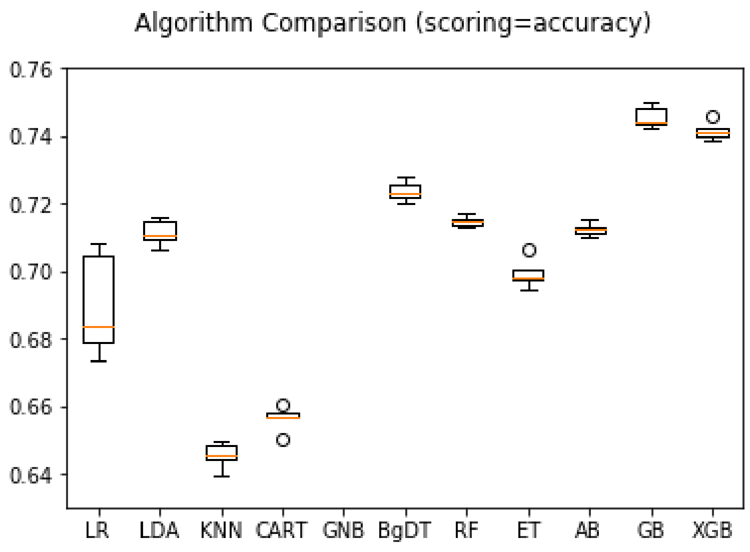
- **Stochastic Gradient Boosting**. Also called Gradient Boosting Machines, they are one of the most sophisticated ensemble techniques. It is also a technique that is proving to be perhaps one of the best techniques available for improving performance via ensembles.

In addition, another classifier called **XGBoost** is worth considering.

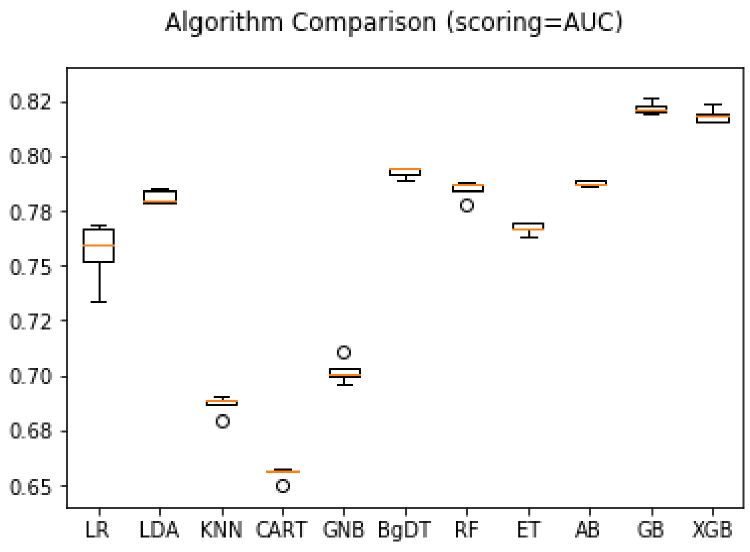
- **XGBoost (eXtreme Gradient Boosting)** [96]. is an advanced implementation of gradient boosting algorithm. It offers many improvements with respect to GBM [97], among which the most relevant are: *i)* it implements *regularization* [98] that helps to reduce overfitting;*ii)* it implements parallel processing, hence it delivers performances that are demonstratedly better than GBM; *iii)* it has a high flexibility because it allows users to define custom optimization objectives and evaluation criteria.

In the scope of this thesis, all the algorithms reviewed above have been tested as potentially good classifiers for the use-case under study. They have been tried on the training sample, and the results have been compared one with the other in order to choose the “best” algorithm for the physics goal. In the evaluation, as mentioned in a previous section, two different metrics have been considered: accuracy and AUC. The results of these comparisons are shown in Figure 4.11a and 4.11b, for accuracy and AUC as scorer, respectively.

Some details on the set-up to perform the comparison follow. All the mentioned classifiers have been used with default values for all their internal parameters. Naturally, all these algorithms would perform better than with the default configurations once all parameters would have been specifically tuned for the problem under study. For reasons of time, this would have been possible only for few algorithms, and only these could have been compared among themselves: in order to try out all classifiers, it was decided instead to use the default arguments for all of them, and identify a subset of them that seemed to perform better, even if with the default arguments. Figures 4.11a and 4.11b show the results of this comparison with default parameters. In this set-up, it can be seen that, in terms of accuracy, the best 5 models are (GB, XGB, BgDT, RF, AB), while in terms of AUC, the best 5 models are (GB, XGB, BgDT, AB, RF). Among these, both GBM and XGBoost seem to consistently rank higher than most other classifiers: GBM and XGBoost have comparable performances, with XGBoost standing as an improved algorithm primarily because of a higher processing speed and the implementation of regularization to reduce overfitting (as discussed earlier in this section). As a consequence, in the scope of this thesis, XGBoost was selected as the “best” algorithm and subsequent efforts will be spent to tune it further for the case under study.



(a)



(b)

Figure 4.11: Accuracy (a) and AUC (b) score for different ML algorithms on the training set. Standard matplotlib representation with boxes and outliers apply. In (a) the data point corresponding to GNB is not visible because lower than 0.6 accuracy.

4.3.4 Hyperparameters tuning

Now that a ML algorithm have been chosen, before proceeding in applying the model to make predictions, it is suggested to consider the concept of *iteration* and how it applies to eventually improve a model.

The first stage where the iteration plays a big role is at the model creation level. Any model, whether it be a regression model, a decision tree, or a neural network, is defined by many (sometimes even millions) of parameters. For example, a regression model is defined by its feature coefficients, a decision tree is defined by its branch locations, and a neural network is defined by the weights connecting its layers. There is a step in which the machine learns the right values for all of the model parameters, and this step is where iterative algorithms come into play. In this context, one of the shining successes in applied ML is the Gradient Descent (GD) [99] algorithm, and its modified counterpart, the Stochastic Gradient Descent (SGD) [100]. GD is an iterative method for finding the minimum of a function, which in ML typically is the loss (or cost) function (i.e. some metric that quantifies the cost of wrong predictions). GD calculates the loss obtained by a model with a given set of parameters, and then alters those parameters to reduce such loss, repeating via iterations this process until that loss cannot substantially be reduced further. The final set of parameters that minimize the loss ultimately define the best-fit model.

The next level where iteration plays a huge role is at what it is named in jargon the "micro" level, more commonly known as the "general model" or "model family". A model family can be thought as broad category of models with customizable structures. Logistic regressions, decision trees, SVMs, and neural networks are actually all different families of models. Each model family has a set of structural choices that must be made before actually fitting the model parameters. These structural choices are called *hyper-parameters*. Hyper-parameters are "higher-level" parameters that cannot be learned directly from the data using GD or other optimization algorithms. They describe structural information about a model that must be decided before fitting model parameters. In the seek for a better model, then, a good practice is iterating many times the chosen model in order to tune its hyper-parameters and find their best set.

In the scope of my thesis, XGBoost has lots of hyper-parameters and the tuning part is a very long process. The starting point is to consider default values for all the hyper-parameters and, for each set of iterations, a different hyper-parameter has to be changed in order to improve accuracy and AUC scores at each iteration round. Once the best value is found, the hyper-parameter will be set to that value for next iterations, where a different hyper-parameter will instead be put under tuning.

In Table 4.1 the accuracy and AUC scores for different set of hyper-parameters of the XGBoost-based model are shown:

- the first version is the base XGB model in which default values of the hyper-parameters are considered.
- the second version is obtained tuning only three hyper-parameters;

- the third is the final model obtained by the complete study of all the configurations for 10 selected hyper-parameters (ie. the main ones in XGB).

As it can be seen, any optimization leads to better scores, both in terms of accuracy and AUC: the intermediate tuning offers the largest improvement with respect to the default, but more can be achieved by applying also the final tuning. The very last version has been chosen as the "best" model for this thesis work, with a caveat. By "best" here it is not implied that nothing better could be achieved: it would not be unexpected that a deeper analysis of XGBoost or of another kind of model could produce best scores. Consider that other algorithm were compared to XGB based on the default parameters, and some hyper-parameters tuning for those models could have lead to a different choice of the "best" algorithm. This a common aspect of the model selection phase in applied ML problems: in principle, all possible optimizations of all existing algorithms should be tried out, and only at that stage a final choice of the "very best" model could be drawn. But, the goal of this thesis is not to produce an extremely optimized algorithm to be use in production for this use-case (for which a long time and much more work would be needed), while it is to define a "good enough" model that already has visible impact (improvements) over how the analysis was previously done, and which could represent a good working use-case for the further definition of a ML-as-a-service prototype. In this sense, the third version of XGBoost is already well optimized for our needs.

Algorithm	**Accuracy**	**AUC**
XGBoost_v1	(74.15 ± 0.25)%	(81.84 ± 0.30)%
XGBoost_v2	(76.96 ± 0.26)%	(85.04 ± 0.23)%
XGBoost_v3	(77.28 ± 0.30)%	(85.27 ± 0.21)%

Table 4.1: Accuracy and AUC scores for different set of hyper-parameters for the XGBoost-based model. An higher version number means a later stage in the tuning process.

As a summary, in the scope of this thesis, a majority of the hyperparameters of XGBoost have been tuned and ultimately the XGB-based model with "v3" optimization was chosen as the "best" model to proceed with.

Despite, for reasons of space, it is not optimal to show here the code developed in the entire ML pipeline to go from the training dataset to the final ML model with tuned hyperparameters, it might nevertheless be interesting for the reader to see a snippet of how a model that could be good for this case can be created, tested on training sample with k-fold cross validation (printing out for example AUC score), and saved for subsequent use.

This code - just explanatory and not representative of the entire code written for the full ML pipeline - can be found in the following.

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from xgboost.sklearn import XGBClassifier
from pickle import dump
```

```
filename = 'training_dataset.csv'
dataset = read_csv(filename)

(...)

kfold = KFold(n_splits=5, random_state=7)
model= XGBClassifier(
    reg_lambda=40,
    learning_rate=0.05,
    n_estimators=1215,
    max_depth=4,
    min_child_weight=9,
    gamma=0.7,
    subsample=0.7,
    colsample_bytree=0.9,
    objective= 'binary:logistic',
    nthread=4,
    scale_pos_weight=1,
    seed=27)
results = cross_val_score(model, X, Y, cv=kfold,
        scoring='roc_auc')
print("AUC", results.mean(), "dev_std", results.std())

dump(model, open('model.sav','wb'))
```

4.4 The $t\bar{t}$ selection use case

Standard Model and top Physics have been introduced in Section 1.5. In this section, more details on the selected physics use case are provided i.e. the Signal versus Background discrimination in the selection of $t\bar{t}$ events in the all-hadronic topology [101].

4.4.1 Statement of the physics goal

A top quark decays semi-weakly into a real W boson and a b quark in $\approx 93\%$ of the cases. Depending on how the W boson decays, we have three different channels which depend on the numbers of charged leptons. In this study we are interested in the "all-hadronic channel", in which both W bosons (coming from the decay of the two top quarks produced in the pp collisions) both decay hadronically (Figure 4.12), i.e.:

$$t\bar{t} \to W^+b \;\; W^-\bar{b} \to q\bar{q}'b \;\; \bar{q}q'\bar{b}.$$

The products of this decay would be six distinct (i.e. "resolved") jets, but if the top quarks have large transverse momentum p_T, the particles originating from the decay $t \to W^+b$ will receive a large boost and emerge quite collimated, yielding two distinct wide jets:

$$q\bar{q}'b \;\; \bar{q}q'\bar{b} \to jet_{boosted,1} \;\; jet_{boosted,2}.$$

Wide jets coming from particles with a relevant Lorentz boost are thus called "boosted jets". Jets are physical objects defined as sprays of collimated particles produced by the fragmentation and hadronization of quarks and gluons originated by the hard collision. At the recent 13 TeV run of LHC, jets with p_T up to few TeV

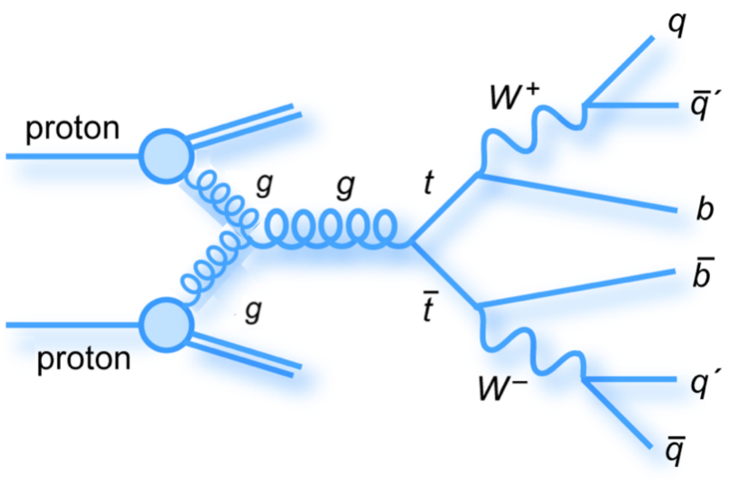


Figure 4.12: Full-hadronic decay of a $t\bar{t}$ pair.

can be observed. At such high p_T, the decay products of the top quarks can be so collimated that standard reconstruction methods start to be less efficient.

The study of such boosted jet in the SM framework may serve as a test of the model under extreme conditions, but may also - according to some of the physics Beyond the Standard Model (BSM) scenarios - provide us with the first observations of BSM signals. Hence, an efficient separation between highly boosted QCD jets and highly-boosted heavy objects can facilitate both tests of the SM and searches for first hints of physics beyond the SM. Such a separation can only be done by looking at the internal structure of these jets, namely, their substructure [102].

A great amount of pp collisions is produced at LHC, with only few of such events being relevant for analysis purposes, with the others constituting less interesting background events. The signal (S), in this case the $t\bar{t}$ pairs that decay in the all-hadronic channel, is overwhelmed by a huge background (B), making it impossible to discriminate S from B without an appropriate selection. The identification of jets that arise from the b quark hadronization, i.e. b-jets, is crucial in reducing the overwhelming background, which includes many other non-b jets stemming from gluons, light-flavour quarks (u, d, s) and the c quark fragmentation. The fraction of the $t\bar{t}$ pairs signal and the background events is quantitatively represented by the S/B (signal-to-background) ratio. Therefore, a selection of events that maximizes the purity of the data (computed as S/N, where N=S+B is the number of expected candidates) has been implemented using Machine Learning techniques. This study is complementary to the same one done through a multivariate analysis (MVA) based on the ROOT framework [103].

4.4.2 Data preparation and ML model creation

Our analysis is based on two samples: the data sample, which is collected by the CMS detector, and a Monte Carlo (MC) simulated sample, which contains reconstructed $t\bar{t}$ events. The data sample used for this analysis was collected during the 2016 LHC run of pp collisions at 13 TeV, corresponding to an integrated luminosity of 5.28 fb^{-1}. All is stored in ROOT files in the form of *TTrees* that contain all the relevant information on the major physics objects reconstructed in the detector (jets, leptons, photons, tracks). The ROOT files (both for data sample and simulated events) were converted into two more readable CSV (comma-separated values) produced using *uproot* (see Section 4.3.1), that became the starting point for ML applications.

In order to select different categories of events with different physics properties, we can apply specific selections on the events, such as:

- a trigger selection;
- a selection of the number of jets (N_{jets}), which pass some requirements on p_T;
- a selection of the number of identified charged leptons (e or μ) ($N_{leptons}$);
- a selection of the number of "b-tagged" jets (N_{b-jets}), i.e. jets associated with the hadronization of b quarks, and on the number of "b-tagged" subjets ($N_{b-subjets}$)

Now, what we want to do is to create a model that is able to distinguish signal events from background events. But before that, we have to train the model teaching it, event by event, if an events is signal or if it is background. The signal events come from the MC sample whereas for the background events we have decided to recur to the data themselves, but these data contain also $t\bar{t}$ candidates. The question is how may we make the signal small enough with respect to the background. In order to do so, we want to introduce some prerequisites that suppress the $t\bar{t}$ signal. If we ask that a b quark is not recognized, in the MC sample we do not change things too much, because we know that these events are always signal (the event yield will be almost halved and nothing will change kinematically because kinematic features are independent from the request of b-tagging); but now in the data sample, where above all there are generic jets, asking a b-tag has a very small efficiency (even if the trigger is prescaled) and so the events without b-tag are constituted by background with an higher purity. In summary, this request does not change too much the MC sample, whereas on the data sample this means to amplify the background events by a great factor. So, the first request that we apply to the events inside the two CSV files is not to require b-tag in the trigger. Such a trigger which requires only a jet with $p_T > 280\,GeV$ and another with $p_T > 200\,GeV$ is prescaled, that is only one out of several triggered events is really kept. In this way, we will use data from the MC sample to teach the model how a signal event is, while we will use data from data sample to teach the model how a background event behaves. We apply the same trigger to both files because we want to build the model with events that pass the same constraints, and so the model will recognize events independently by these constraints.

Other requests are then made on the events, that is $N_{jets} >= 2$ (so we want to take both jets coming from the two top quarks, if the event is a signal, and they will be the two leading jets, that are the most energetic ones). Then we want $N_{b-subjets} = 0$ (that is there are no subjets inside a jet recognized as produced by quark b) for the same reason of b-tagging explained before.

To properly analyze the all-hadronic boosted topology, other kinematic requests for the two wide-jets are necessary, in addition to those implemented on the p_T by the trigger. Boosted jets in fact have high p_T, so we require $p_T > 400$ GeV for both jets.

Furthermore, a lepton veto is required in order to select only the hadronic decays. This is carried out by the request that the number of identified charged leptons $N_{leptons}$ be equal to 0.

Many different models have been created using different techniques and classifiers (see Section 4.3.3 and 4.3.4). The model with the best accuracy is an XGBoost with a particular set of parameters, giving an accuracy of 77% on a test set, that was extracted from the two samples before training the model.

It is interesting to see the feature importances score for those variables used to train the model (see Figure 4.13). In the x-axis there is a part of the variables used, where the index is used to identify the components of those variables that are vectors, because they are related to different jets. Here we can recognize, for example, N_{jets} and $jetP_t\_i$ (that is the transverse momentum of the different jets).

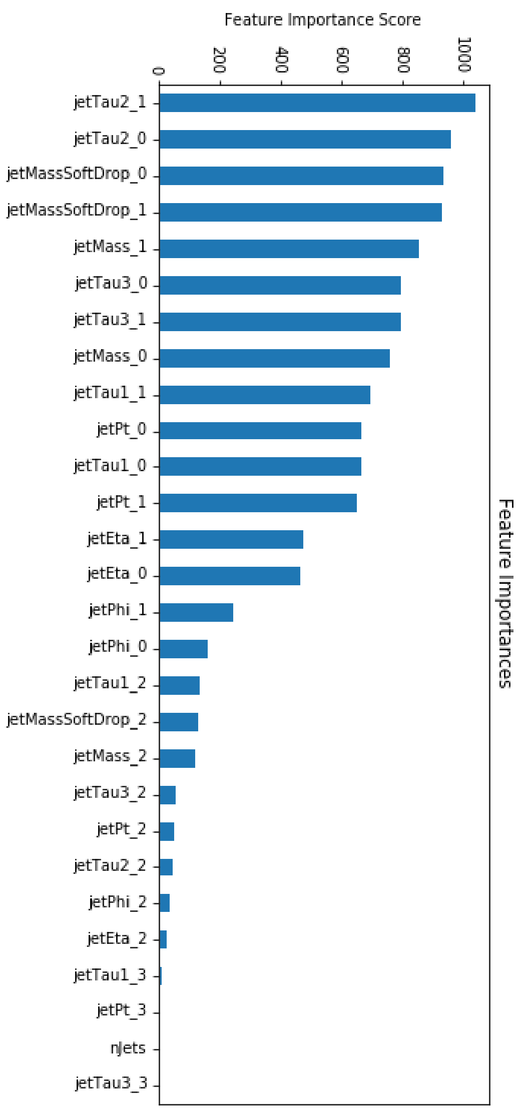


Figure 4.13: Feature importances score of the variables used to train the model.

After we have trained the model using the two samples (with the same constraints for both), now we have to apply the model. Because we are interested in selecting only the events in which the top quark decays into a b quark and a W boson (where the particles produced by the hadronizations are merged in a single wide jet because of the high-p_T of the top quark), b-tagging is aimed at the subjet structure of the boosted jets, so what we mean here is to ask a jet with a b-tagged subjet. So what we have to ask as constraints to the two CSV files are the same requests that we have done previously but now we have to change the trigger we are using, asking first of all a b-tag in the fit (hence removing the prescaling). Then we have decided to try different cases asking $N_{b-subjets} = 1$, $N_{b-subjets} = 2$ and $N_{b-subjets} >= 1$.

4.4.3 Results

From the two original data samples, at this point we have created two samples that we will use to test the model. The first one, coming from the MC sample, allow us to know what is the *efficiency* of the model, that represents how many signals are correctly identified with respect to the whole sample. The second one, coming from the data sample, allow us to know what is the *purity* of the candidates obtained by the model, that represents the expected signal divided by the number of signal candidates.

Our goal is to compare the results (in terms of efficiency and purity) obtained by these ML techniques (outside the ROOT framework) with those obtained by a more common MVA using ROOT. In particular, the MVA associates to each event inside the sample a variable (called "mva") in the range 0-1: closer it is to 0 (1), the more the event is classified as background (signal, respectively) by the MVA.

Efficency, purity and other related quantities are computed as follows. The number of signal events expected in the data sample is:

$$N_{t\bar{t}} = \sigma_{t\bar{t}} \cdot \mathcal{L} \cdot \epsilon \cdot SF, \tag{4.1}$$

where $\sigma_{t\bar{t}} = 832\ pb$ is the theoretical cross section, $\mathcal{L} = 5.28\ fb^{-1}$ is the integrated luminosity related to the sample of data we are using, $SF = 0.7$ is a scale factor (that is introduced because from the official analysis the cross section obtained is smaller than the nominal one) and ϵ is the efficiency computed as:

$$\epsilon = \frac{N_{sig}^{MC}}{N_{tot}^{MC}}, \tag{4.2}$$

where N_{sig}^{MC} is the number of events MC that are predicted as signal by the ML model or pass a *mva* cut, and $N_{tot}^{MC} = 7.72 \times 10^7$ is the number of MC generated events. Then, calling $S = N_{t\bar{t}}$ the expected signals in N candidates of the data sample, the background will be $B = N - S$, and $purity = S/N$.

As we have already said in the previous section, we apply the ML model to the data asking $N_{b-subjets} = 1$ or $N_{b-subjets} = 2$ or $N_{b-subjets} >= 1$ (added of course to the other constraints). In order to better compare the results of MVA and ML, we decided to apply different cuts on the *mva* variable (where each event that has a value of *mva* greater than the fixed threshold will be considered a signal candidate) and compute efficiency and purity for each case. In Figure 4.14 the

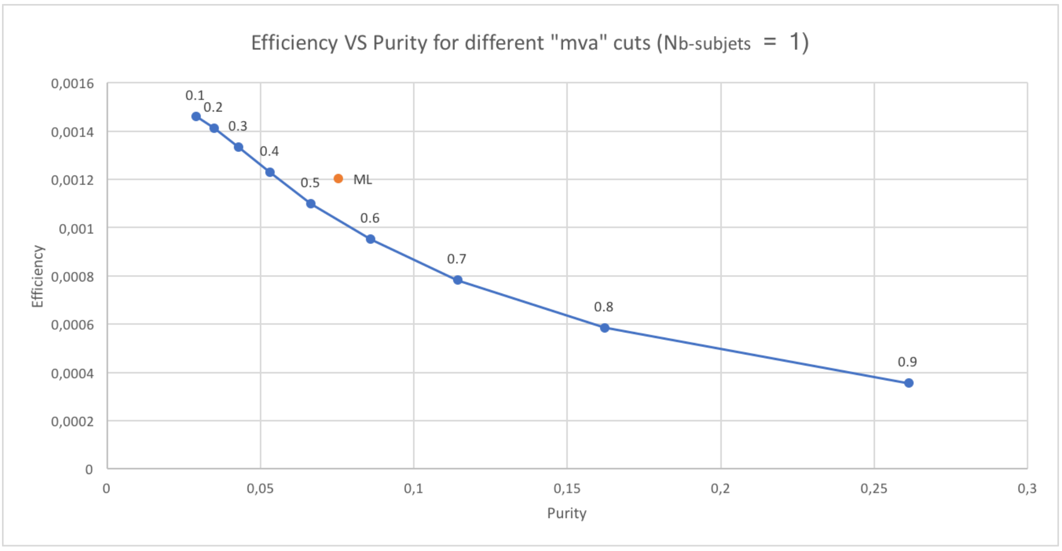


Figure 4.14: Efficiency and purity for different *mva* cuts (the values quoted above the dots), compared to the same obtained by ML, in the case of $N_{b-subjets} = 1$.

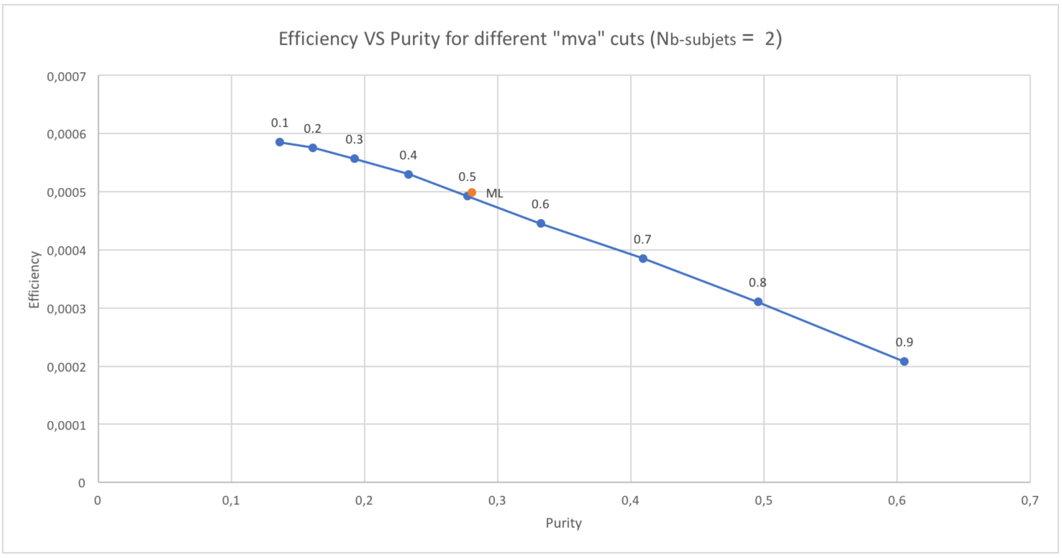


Figure 4.15: Efficiency and purity for different *mva* cuts (the values quoted above the dots), compared to the same obtained by ML, in the case of $N_{b-subjets} = 2$.

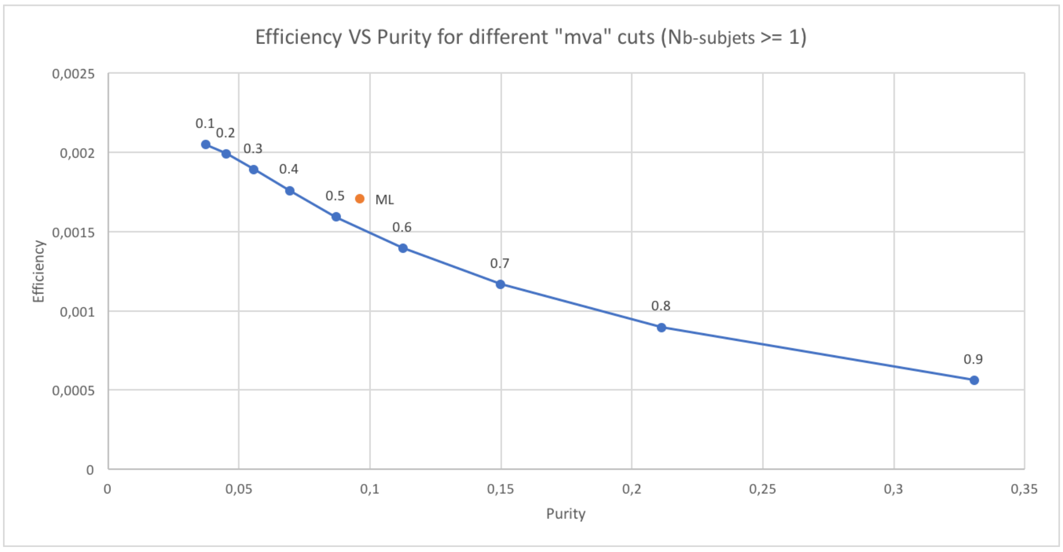


Figure 4.16: Efficiency and purity for different *mva* cuts (the values quoted above the dots), compared to the same obtained by ML, in the case of $N_{b-subjets} >= 1$.

case $N_{b-subjets} = 1$ is shown, in Figure 4.15 the case $N_{b-subjets} = 2$ is shown, and in Figure 4.16 the case $N_{b-subjets} >= 1$ is shown.

The best way to make a direct comparison between the ML method and the MVA method is to fix the efficiency of the MVA method to the one obtained by ML, and then compare the two purities. Another test one could make is to add the variable *mva* to build the model in order to see how much the job done by MVA can help ML to offer better predictions. The results for the three different selections of $N_{b-subjets}$ are shown in Table 4.2, Table 4.3 and Table 4.4.

Quantity	**Value**	**Value (ML with *mva*)**
N_{sig}^{MC}	92956	92612
N_{ML}	49234	48620
N_{MVA}	66696	65886
S	3704	3690
Efficiency	0.12%	0.12%
Purity (ML)	7.5%	7.6%
Purity (MVA)	5.55%	5.60%

Table 4.2: Results for $N_{b-subjets} = 1$.

It can be seen that ML model allowed to obtain comparable resuls in term of purity with respect to MVA method, once we have fixed the efficiency to the one that ML gives, in particular the results are slightly (higher) - i.e. better - for the ML approach, in all the three $N_{b-subjets}$ categories. Given the short time spent on preparing this ML model, and the margins of improvement coming to further refinement of the ML model and algorithm tuning, this outcome is in itself quite remarkable as it shows the effectiveness of ML techniques in this physics use-case.

A further study regards the distribution of *mva* variable before and after ML.

Quantity	**Value**	**Value (ML with *mva*)**
N_{sig}^{MC}	38524	38524
N_{ML}	5467	5437
N_{MVA}	5665	5665
S	1535	1535
Efficiency	0.05%	0.05%
Purity (ML)	28.1%	28.2%
Purity (MVA)	27.1%	27.1%

Table 4.3: Results for $N_{b-subjets} = 2$.

Quantity	**Value**	**Value (ML with *mva*)**
N_{sig}^{MC}	131833	131353
N_{ML}	54701	54074
N_{MVA}	70451	69620
S	5253	5233
Efficiency	0.17%	0.17%
Purity (ML)	9.6%	9.7%
Purity (MVA)	7.5%	7.5%

Table 4.4: Results for $N_{b-subjets} >= 1$.

This is shown for the three cases of $N_{b-subjets}$ in Figure 4.17, Figure 4.18 and Figure 4.19 - when considering MC samples - or otherwise Figure 4.20, Figure 4.21 and Figure 4.22 - when considering data samples. From the data sample ones, it is possible to see that after ML the *mva* distribution has a shape with a smooth peak shifted to larger values of *mva*, because the ML method is selecting signal candidates with large values of *mva*. On the contrary, on MC samples, after ML the yield decreases because those events that have lower values of *mva* are cut.

As a final study, it is also interesting to see what effect the ML has in the distribution of the *jet invariant mass*. For the computation of the jet invariant mass a particular algorithm (soft-drop [104]) is used, which removes wide-angle soft radiation from a jet in order to mitigate the effects of the contamination from initial state radiation (ISR), underlying event (UE) and multiple hadron scattering (pileup). So, if we take the output variable obtained by this algorithm, we have better definition of the invariant mass of a jet. Now, if we consider the case of $N_{b-subjets} = 2$ we are selecting events that have a high probability to come from the all-hadronic $t\bar{t}$ channel, where both b quark are recognized. In Figure 4.23 the distribution of the invariant mass of the leading jet after the "soft-drop declustering" algorithm for the MC sample is shown.

It can be noticed that the distribution - after applying ML - has a peak a slightly larger than the same obtained after *mva* cut (with the same efficiency obtained with ML, and so we have the same number of entries), and in addition it is shifted to higher mass values. Here we are considering only the leading jet and so what we suppose to see is the distribution of the decay products of a top quark. In fact, we can see a well defined peak related to the mass of the top quark (even if it is still too far from the 173 GeV), and a slightly smaller peak related to the W boson

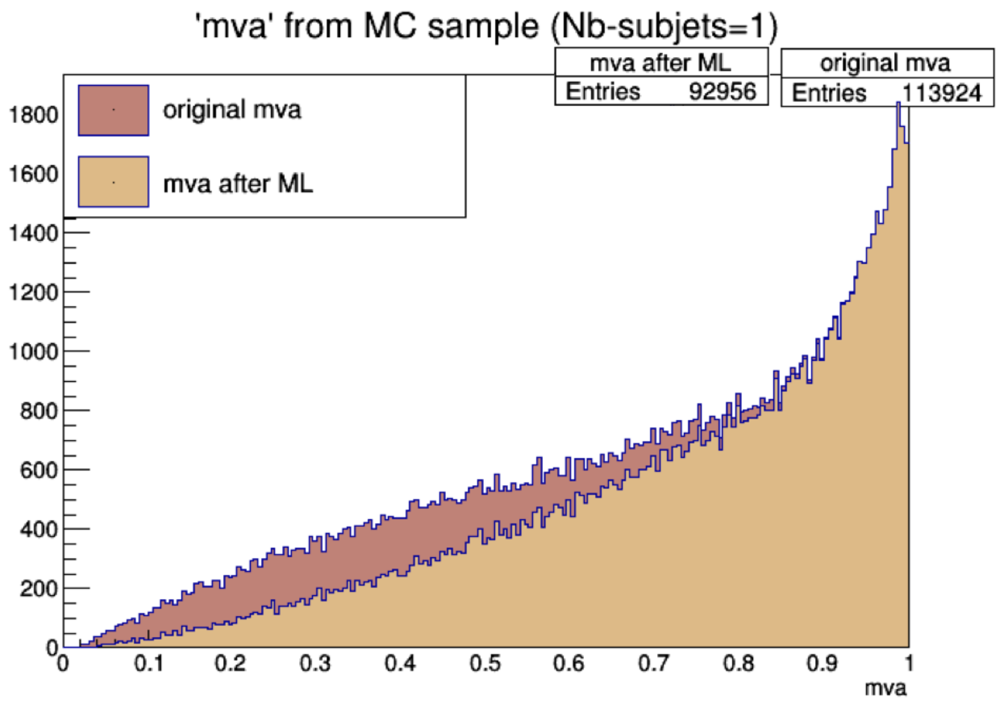


Figure 4.17: Distribution of *mva* of the MC sample with $N_{b-subjets} = 1$.

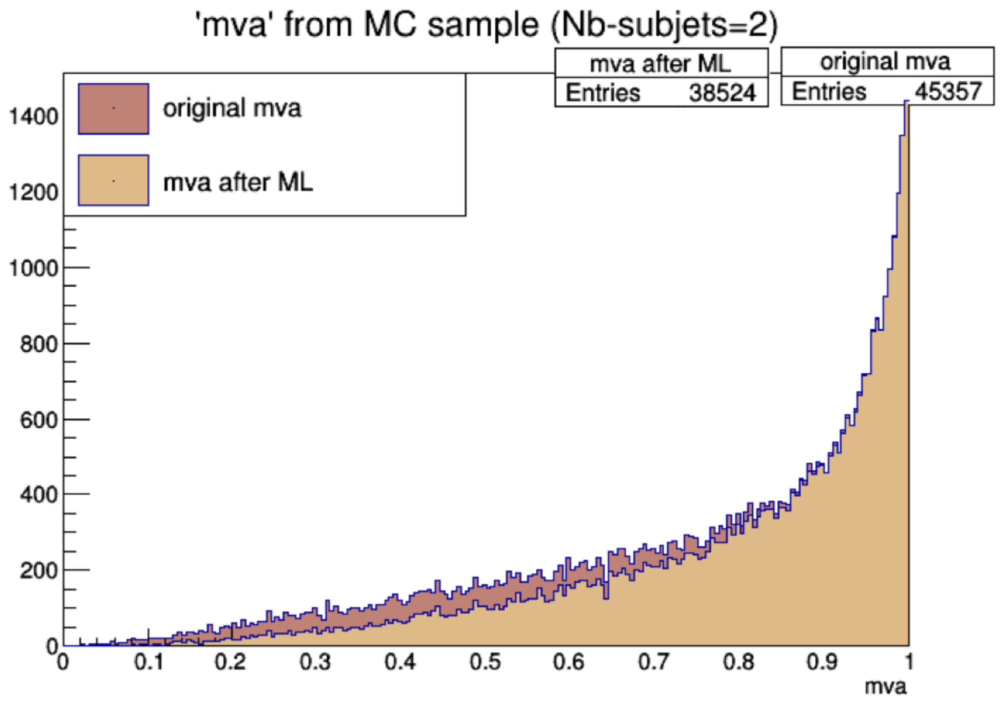


Figure 4.18: Distribution of *mva* of the MC sample with $N_{b-subjets} = 2$.

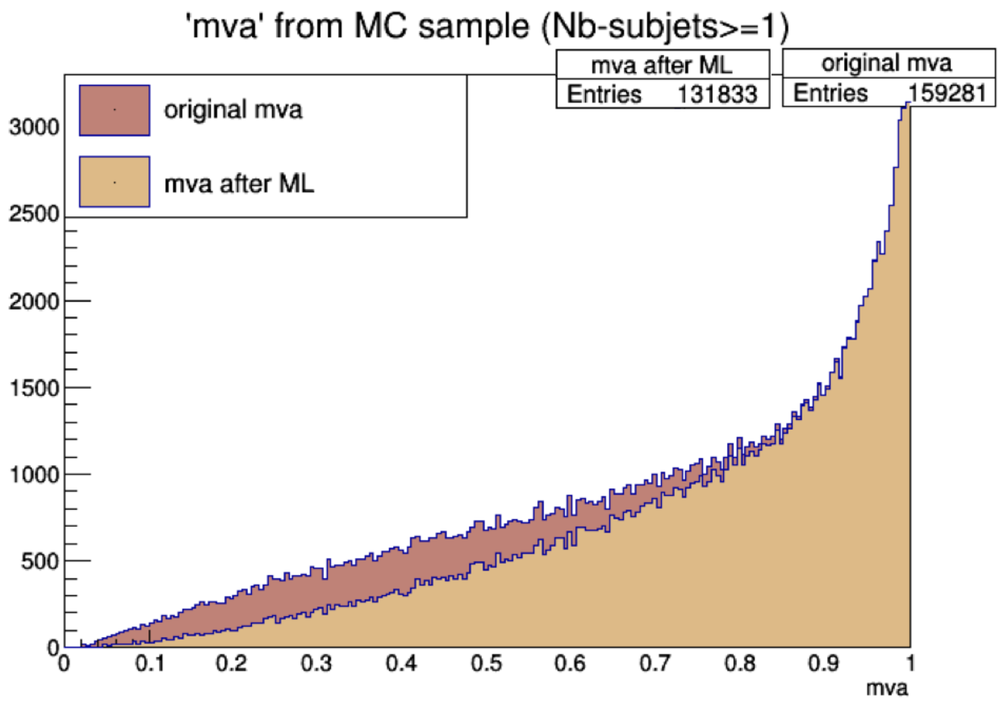


Figure 4.19: Distribution of *mva* of the MC sample with $N_{b-subjets} >= 1$.

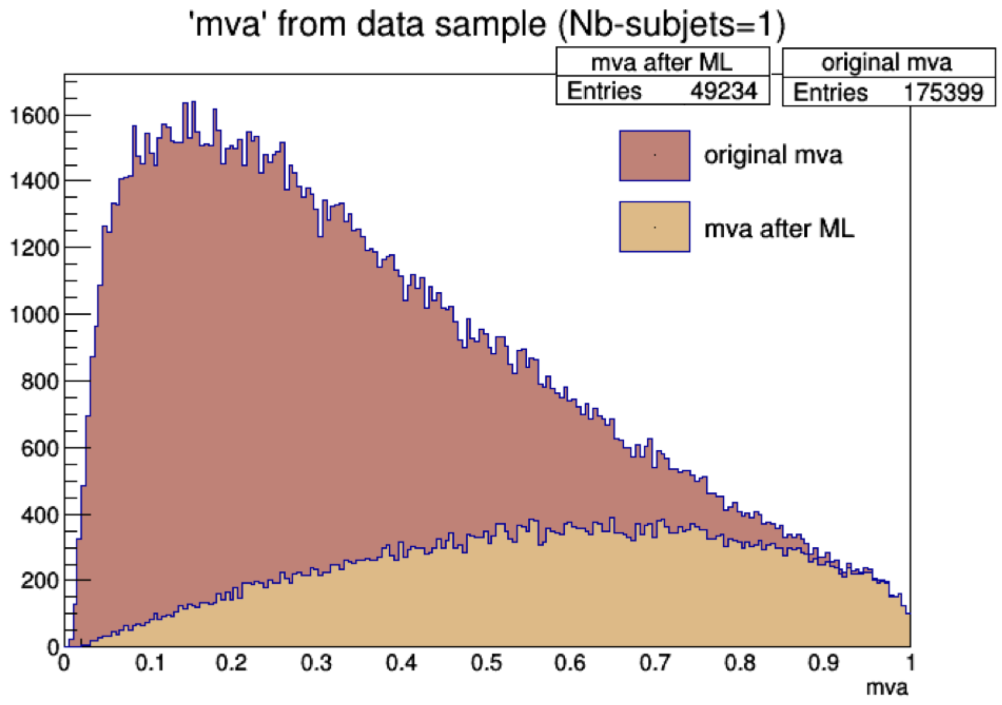


Figure 4.20: Distribution of *mva* of the data sample with $N_{b-subjets} = 1$.

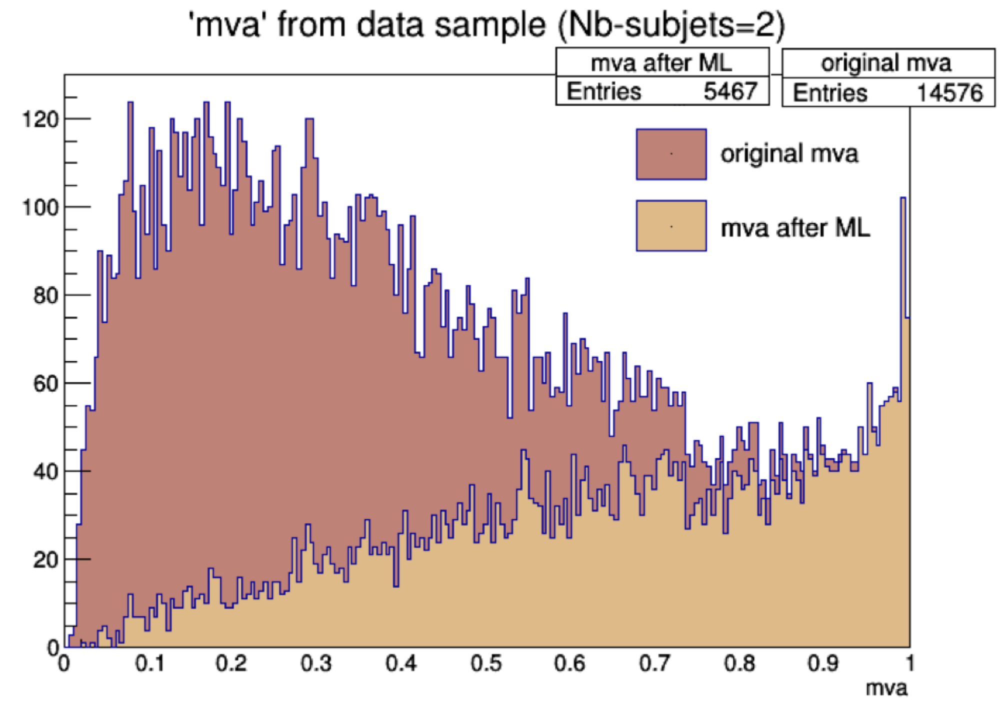


Figure 4.21: Distribution of *mva* of the data sample with $N_{b-subjets} = 2$.

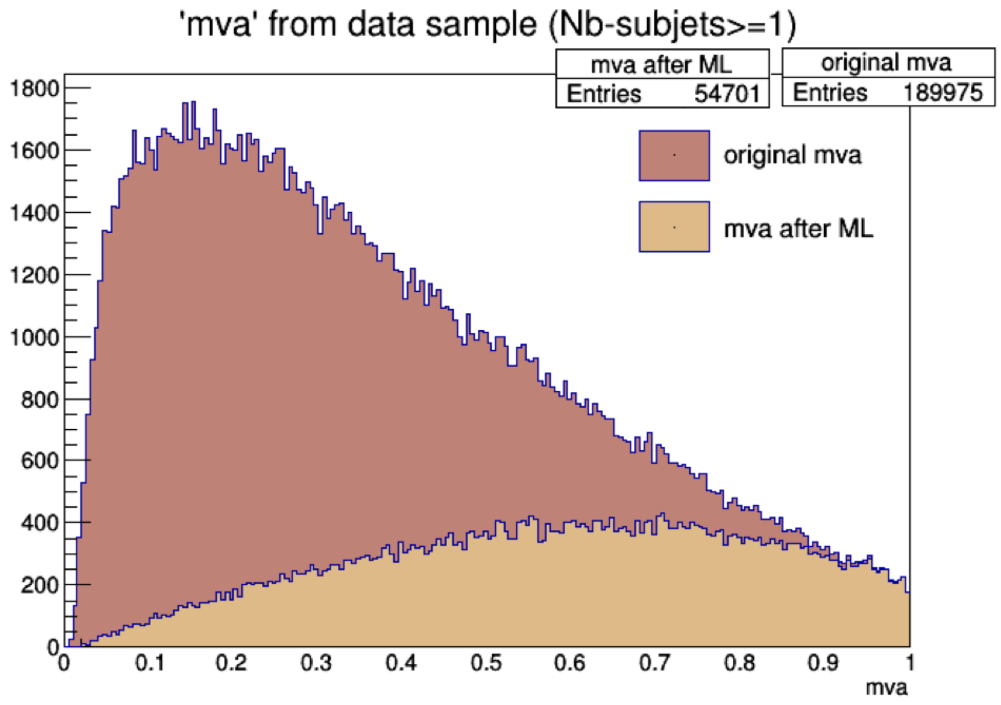


Figure 4.22: Distribution of *mva* of the data sample with $N_{b-subjets} >= 1$.

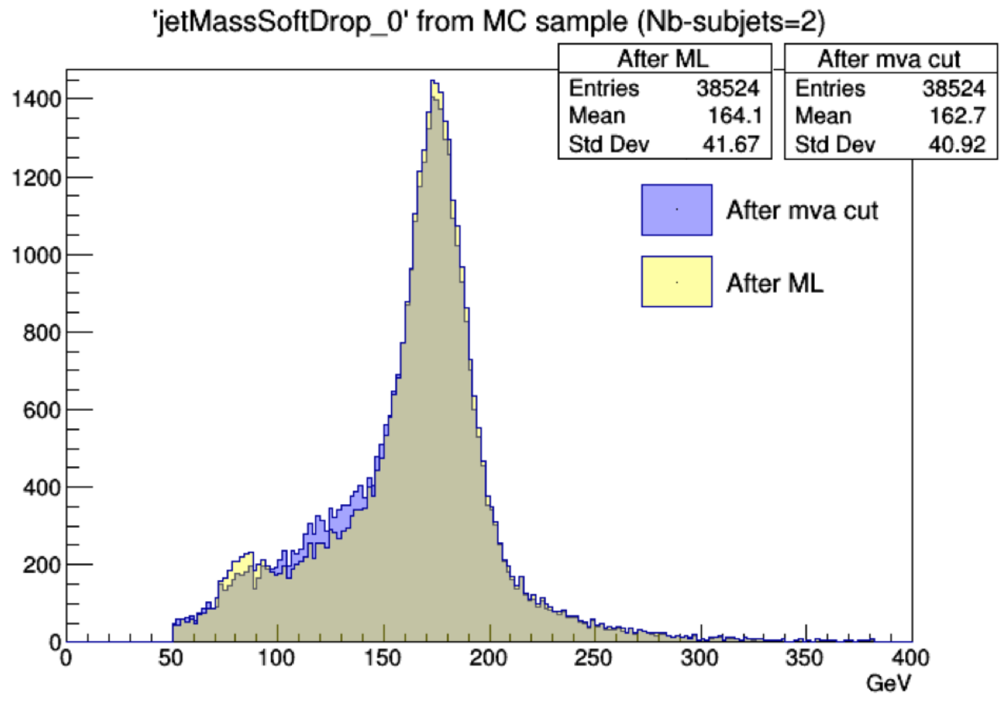


Figure 4.23: Invariant mass of the leading jet after the "soft-drop declustering" algorithm for the MC data with $N_{b-subjets} = 2$.

(where the signal coming from the b quark has been lost during the reconstruction). It is also interesting to see that in Figure 4.13, the variable related to the soft-drop mass of the leading jet is at the third position for the feature importance score, underlying the fact of how much it has been important to build the model.

4.5 Demonstration of the TFaaS prototype

In this section, a step-by-step description of how a model can be used in the TFaaS prototype is presented. A caveat applies, though: this section is only meant to illustrate some key points of the overall workflow, not to stand as a guide for anyone interested to fully reproduce the set-up. For the latter, any interested reader should consult [79] and the related documentation (constantly improving as the project proceeds).

While the "best" model in the selected physice use-case was built using scikit-learn, for this demonstration, two similar models have been created using Keras with a Tensorflow (v 1.0.0) backend. Details of such models, and the differences among the two, are not relevant for understanding this demonstration. The input CSV file is the same for both models, and its size is approximately 39 MB. The first model for this demonstration was done on keras with 0 efforts, and has a very low accuracy (48.2%). The second model for this demonstration was done on keras with some efforts (more could have been made, but this was not the point of this exercise), and has a higher accuracy (66.5%). The Keras models can be saved on local disk as *.h5* files:

```
model.save('file_name.h5')
```

Doing so, the first saved model is 46 KB in size, and the second 128 KB.

The models must be converted to Tensorflow using the *keras_to_tensorflow* package [105]. This package loads a trained keras model, freezes the nodes (converts all Tensorflow variables to Tensorflow constants), and saves the inference graph and weights into protobuf files (*.pb* and *.pbtxt*). Note that there are actually two different formats that a ProtoBuf can be saved in. TextFormat (*.pbtxt* extension) files are in a human-readable form, which makes it nice for debugging and editing, but can get large when there is numerical data like weights stored in it. Binary format (*.pb* extension) files are a lot smaller than their text equivalents, even though they are not as readable. In our case, reference commands that are needed are:

```
./keras_to_tensorflow.py -input_model_file [your_saved_model.h5]
                         -output_model_file [output.pb]
./keras_to_tensorflow.py -input_model_file [your_saved_model.h5]
                         -output_model_file [output.pbtxt]
```

This file (any format) can then be used to deploy the trained model for inference. During freezing, other nodes of the network, which do not contribute the tensor that would contain the output predictions, are prunned (this results in a smaller, optimized network).

TFaaS needs to be installed. The code can be found at [79].

Once this is done, you need to install the TFaaS GO server, and for this you need to have the GO (golang) language [106] on your system. To do so, the GO language package for your favorite distribution must be downloaded and installed, the GOROOT environment variable must be set to point to the installed GO distribution, and the GOPATH environment variable must be set to point to your local area where your GO packages will be stored.

Then, the GO tensorflow library needs to be installed on your system. A good reference instructions can be found in [107]. Download and extract the TensorFlow C library into e.g. */usr/local/lib* by invoking the following shell commands:

```
TF_TYPE="cpu" # Change to "gpu" for GPU support
 TARGET_DIRECTORY='/usr/local'
 curl -L
   "https://storage.googleapis.com/tensorflow/libtensorflow/libtensorflow-\
   ${TF_TYPE}-$(go env GOOS)-x86_64-1.6.0-rc1.tar.gz" |
 sudo tar -C $TARGET_DIRECTORY -xz
```

Once installed you'll need to get GO Tensorflow code. It will be compiled against library you just downloaded and installed: a proper set-up of *LIBRARY_PATH* variables should be checked.

Now that the TensorFlow C library is installed, invoke *go get* as follows to download the appropriate packages and their dependencies:

```
go get github.com/tensorflow/tensorflow/tensorflow/go
go get github.com/tensorflow/tensorflow/tensorflow/go/op
```

Download necessary dependencies for TFaaS:

```
go get github.com/vkuznet/x509proxy
go get github.com/golang/protobuf
go get github.com/sirupsen/logrus
```

Invoke *go test* as follows to validate the TensorFlow for GO installation:

```
go test github.com/tensorflow/tensorflow/tensorflow/go
```

Another way to check that GO is properly working (without which the TFaaS GO server will not work) is to try a simple Hello World in GO. A simple GO code like this:

```
package main

import (
    tf "github.com/tensorflow/tensorflow/tensorflow/go"
    "github.com/tensorflow/tensorflow/tensorflow/go/op"
    "fmt"
)

func main() {
    // Construct a graph with an operation that produces a string constant.
    s := op.NewScope()
    c := op.Const(s, "Hello from TensorFlow version " + tf.Version())
    graph, err := s.Finalize()
    if err != nil {
        panic(err)
    }

    // Execute the graph in a session.
    sess, err := tf.NewSession(graph, nil)
    if err != nil {
        panic(err)
    }
    output, err := sess.Run(nil, []tf.Output{c}, nil)
    if err != nil {
        panic(err)
    }
    fmt.Println(output[0].Value())
}
```

can be run via the command

```
go run hello_tf.go
```

and should print out an hello message with the installed Tensorflow version number. Once all this is done, you should make sure that produced executable is linked against tensorflow library, e.g. via *ldd* command.

Now you can build TFaaS GO server by running *make* and you will get the TFaaS executable which is ready to act as a server for your models. The *.pb* files (i.e. TF model files in protobuf data-format) should now be moved to the same node where TFaaS in installed, to be ready for the next step. Note that each TF model use input entry-point to feed the data (vector or matrix) into TF model. This entry point has a name either assigned by TF code or via explicit assignment. One may inspect the TF model in text format to find out which layer names where used and hence extract the input layer name and the output layer name.

In order to serve TF models with the TFaaS GO server, one needs to set-up server certificates to start HTTPs server. Best is to generate self-signed host certificates. An example how to generate self-signed certificates is:

```
openssl req -new -newkey rsa:2048 -nodes -keyout server.key -out server.csr
```

Varius arguments will needed to be asked, but they are self-explanatory.

Attention must be put also to prediction labels. For models where there are multiple labels we need to create prediction labels file. It is a simple text file which lists its label on every line.

Once all this is done, all pieces are in place to run a TFaaS server, which can be done via this command:

```
openssl req -new -newkey rsa:2048 -nodes -keyout server.key -out server.csr
```

At this stage, after proper set-up of the user credential, the server can be run:

```
./tfaas -dir [dir_with_models] -serverCert server.csr -serverKey server.key
-modelLabels [label_file] -inputNode [input_layer_name] -outputNode [input_layer_name]
-modelName [model.pb]
```

One can see the server running:

```
INFO[0000] Starting HTTP server    Addr=":8083"
```

Once the server runs, one can prepare in any language a JSON with the records than one wants to send to the server, and send it like this:

```
# host settings
headers="Content-type: application/json"
host=https://localhost:8083
curl -k --key ~/.globus/userkey.pem --cert ~/.globus/usercert.pem -H "$headers"
-d @/tmp/[my_json_file] $host/json
```

and the server will reply with predictions according to the trained model.

The steps above were tried for both the aforementioned models, and it is remarkable to observe and report that in both cases <u>the predictions from a local ML model</u> and <u>from the remote TFaaS</u> were <u>the same</u> in both models, the lower accuracy one and the higher accuracy one. This means that any inference of the ML model that can be done locally, now can altrnatively be done via simple calls from the analysis code (e.g. in C++, or python) to such external ML " as a service" solution. As a demonstration of this, this is what a single, one-shot call from the client to a running server looks like from the command line:

```
$ curl -s -L -k --key ~/.globus/userkey.pem --cert ~/.globus/usercert.pem
-H "Content-type: application/json" -d '{"keys": ["nJets", "nLeptons",
"jetEta_0", "jetEta_1", "jetEta_2", "jetEta_3", "jetEta_4", "jetMass_0",
"jetMass_1", "jetMass_2", "jetMass_3", "jetMass_4", "jetMassSoftDrop_0",
"jetMassSoftDrop_1", "jetMassSoftDrop_2", "jetMassSoftDrop_3",
"jetMassSoftDrop_4", "jetPhi_0", "jetPhi_1", "jetPhi_2", "jetPhi_3",
"jetPhi_4", "jetPt_0", "jetPt_1", "jetPt_2", "jetPt_3", "jetPt_4",
"jetTau1_0", "jetTau1_1", "jetTau1_2", "jetTau1_3", "jetTau1_4",
"jetTau2_0", "jetTau2_1", "jetTau2_2", "jetTau2_3", "jetTau2_4",
"jetTau3_0", "jetTau3_1", "jetTau3_2", "jetTau3_3", "jetTau3_4"],
"values": [2.0, 0.0, 0.9228423833849999, -1.1428750753399999,
0.0, 0.0, 0.0, 155.239425659, 142.709609985, 0.0, 0.0, 0.0,
83.5365982056, 120.549507141, 0.0, 0.0, 0.0, 1.9305502176299998,
-1.17742347717, 0.0, 0.0, 0.0, 481.419799805, 449.04394531199995,
0.0, 0.0, 0.0, 0.296700358391, 0.286615312099, 0.0, 0.0, 0.0,
0.164555206895, 0.19625715911400002, 0.0, 0.0, 0.0,
0.117722302675, 0.155229091644, 0.0, 0.0, 0.0]}'
http://localhost:8083/json
[0.65876985]
```

As a response, it can be seen that a floating point umber is returned ("[0.65876985]"): this means that this specific event is predicted by the trained model to be a Signal event (> 0.5) and not a Background event. Tests done with large set of events show that the results obtainable locally with a given model are the same ones that are returned by the server vis this kind of calls. Of course, for plenty of events this is not done command-line, but all event entries are passed via JSON format to the server in a unique, large client query, returning a vector of predictions. As an additional demonstration, Figure 4.24 shows the ROC curve obtained with the better model that was uploaded to the TFaaS infrastructure. As it can be seen, the obtained value of AUC is a 71%, consistent with the ML model loaded into TFaaS.

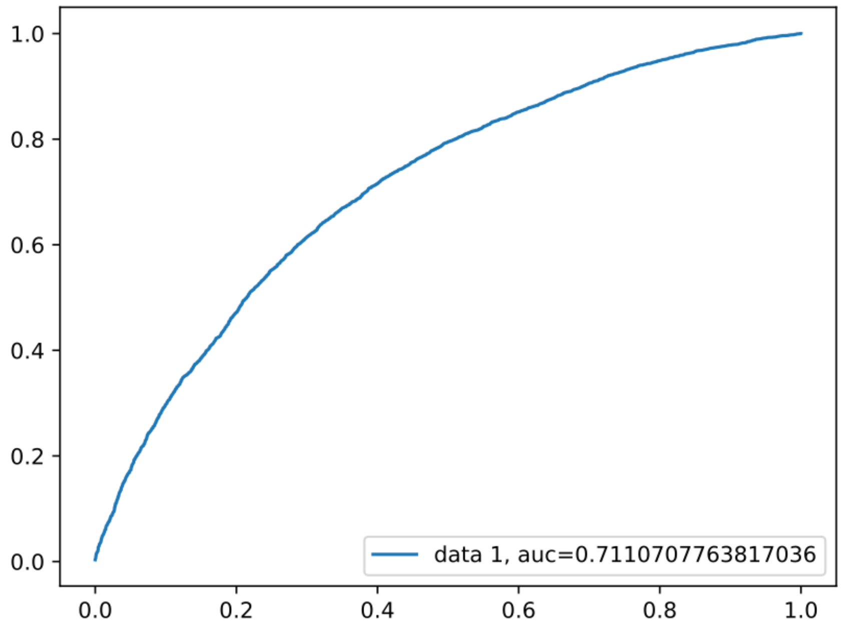


Figure 4.24: Plot of the ROC curve with AUC value for the Keras model (more in the text).

Conclusions

This thesis aimes at contributing to the construction of a Machine Learning "as a service" solution for CMS Physics needs, namely an end-to-end data-service to serve Machine Learning trained model to the CMS software framework.

To this ambitious goal, this thesis work contributes firstly with a proof of concept of a first prototype of such infrastructure, and secondly with a specific physics use-case: the Signal versus Background discrimination in the study of CMS all-hadronic top quark decays, done with scalable Machine Learning techniques.

After evaluation of most adequate Machine/Deep Learning frameworks, the Machine Learning "as a service" architecture has been based on Google Tensorflow, and called "Tensorflow as a service" (TFaaS). Keras has been selected and used for a prototyping of a Tensorflow-based model for such infrastructure. The selected transport layer exploited in the TFaaS architecture is the Protocol buffer by Google. The data preparation step was performed according to several approaches, and the *uproot* solution by the DIANA-HEP project was selected as the most performing and scalable. An effort in data validation was also pursued, to seek to the maximum quality in the data input to the model. A large set of metrics was considered to evaluate the model, and both Accuracy and AUC were selected as model scorers. The problem was formulated as a Supervised Classification Machine Learning problem. The model training has been performed with k-fold cross validation. A dozen of ML algorithms have been considered, and a model was built on each and then compared among each other, yielding the selection of a model based on eXtreme Gradient Boosting (XGBoost). After hyper-parameters tuning - which was tried in multiple configuration and one with 10 tuned parameters was chosen - the model reached a >77% Accuracy and >85% AUC scores. The model was applied to the S/B discrimination in the all-hadronic top quark decay analysis, and results show that this approach is better than the one used in currently published results, obtained with MVA from the ROOT package, improving the purity of the sample at fixed MVA efficiency. As a final result, a proof-of-concept of the current working prototype of the TFaaS infrastructure while exploiting the aforementioned model for the physics case under study has been discussed in details - in its implementation and operation - and has been demonstrated to work end-to-end as from the design goals. In fact the obtained value of AUC using the infrastructure is a 71%, consistent with the ML model loaded into TFaaS.

These results have been presented at the International Symposium on Grids

Clouds 2018 (ISGC 2018), in Taipei in March 2018. Next steps would be to exercise the infrastructure with other use-cases, and to strengthen its deployment flexibility with cloud services.

Bibliography

[1] O. S. Brüning et al. *LHC Design Report*. Ed. by CERN library copies. Vol. 1, 2, 3. 2012 (cit. on p. 1).

[2] L. Evans and P. Bryant. “LHC Machine”. In: *Journal of Instrumentation* 3.08 (2008). Ed. by IOPscience, S08001. URL: http://iopscience.iop.org/1748-0221/3/08/S08001 (cit. on p. 1).

[3] S. Myers R. ABmann M. Lamont. “A Brief History of the LEP Collider”. In: Suppl. 109, 17-31 (2002). Ed. by Nucl.Phys. B (cit. on p. 1).

[4] S. Chatrchyan et al. [CMS Collaboration]. “Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC”. In: *Phys. Lett. B* 716 (2012). Ed. by IOPscience, p. 30 (cit. on p. 1).

[5] The ALICE Collaboration et al. “The ALICE experiment at the CERN LHC”. In: *Journal of Instrumentation* 3.08 (2008). Ed. by IOPscience, S08002. URL: http://iopscience.iop.org/1748-0221/3/08/S08002 (cit. on p. 5).

[6] *The Alice Collaboration*. URL: http://aliceinfo.cern.ch (cit. on p. 5).

[7] The ATLAS Collaboration et al. “The ATLAS Experiment at the CERN Large Hadron Collider”. In: *Journal of Instrumentation* 3.08 (2008). Ed. by IOPscience, S08003. URL: http://iopscience.iop.org/1748-0221/3/08/S08003 (cit. on p. 7).

[8] *Atlas experiment*. URL: http://atlasexperiment.org/detector.html (cit. on p. 7).

[9] The CMS Collaboration et al. “The CMS experiment at the CERN LHC”. In: *Journal of Instrumentation* 3.08 (2008), S08004. URL: http://stacks.iop.org/1748-0221/3/i=08/a=S08004 (cit. on p. 7).

[10] *The CMS Detector*. URL: https://cms.cern/detector (cit. on pp. 7, 9).

[11] The LHCb Collaboration et al. “The LHCb Detector at the LHC”. In: *Journal of Instrumentation* 3.08 (2008). Ed. by IOPscience, S08005. URL: http://iopscience.iop.org/1748-0221/3/08/S08005 (cit. on p. 8).

[12] *The LHCb Collaboration*. URL: http://lhcb.web.cern.ch/lhcb (cit. on p. 8).

[13] V. Karimäki et al. *The CMS tracker system project: Technical Design Report*. Technical Design Report CMS. Geneva: CERN, 1997. URL: https://cds.cern.ch/record/368412 (cit. on p. 12).

[14] C. Grupen and B. Shwartz. *Particle Detectors*. 2nd ed. Cambridge Monographs on Particle Physics, Nuclear Physics and Cosmology. Cambridge University Press, 2008, pp. 241–242. DOI: 10.1017/CBO9780511534966 (cit. on p. 13).

[15] *The CMS electromagnetic calorimeter project: Technical Design Report*. Technical Design Report CMS. Geneva: CERN, 1997. URL: http://cds.cern.ch/record/349375 (cit. on p. 13).

[16] P. Adzic. "Energy resolution of the barrel of the CMS Electromagnetic Calorimeter". In: *Journal of Instrumentation* 2.04 (2007), P04004. URL: http://stacks.iop.org/1748-0221/2/i=04/a=P04004 (cit. on p. 13).

[17] *The CMS hadron calorimeter project: Technical Design Report*. Technical Design Report CMS. Geneva: CERN, 1997. URL: https://cds.cern.ch/record/357153" (cit. on p. 13).

[18] *The CMS muon project: Technical Design Report*. Technical Design Report CMS. Geneva: CERN, 1997. URL: http://cds.cern.ch/record/343814 (cit. on p. 14).

[19] V. Khachatryan et al. "The CMS trigger system". In: *Journal of Instrumentation* 12.01 (2017), P01020. URL: http://stacks.iop.org/1748-0221/12/i=01/a=P01020 (cit. on p. 15).

[20] C. Foundas. "The CMS Level-1 Trigger at LHC and Super-LHC". In: *Cornell University Library* (2008). URL: https://arxiv.org/abs/0810.4133 (cit. on p. 15).

[21] A. Perrotta. "Performance of the CMS High Level Trigger". In: *Journal of Physics: Conference Series* 664.8 (2015), p. 082044. URL: http://stacks.iop.org/1742-6596/664/i=8/a=082044 (cit. on p. 16).

[22] *The European Strategy for Particle Physics*. URL: https://council.web.cern.ch/en/content/european-strategy-particle-physics (cit. on p. 16).

[23] *High Energy Physics Advisory Panel (HEPAP)*. URL: https://science.energy.gov/hep/hepap/ (cit. on p. 16).

[24] *The High-Luminosity LHC*. URL: https://home.cern/topics/high-luminosity-lhc (cit. on p. 16).

[25] G. Apollinari et al. "High Luminosity Large Hadron Collider HL-LHC". In: *CERN Yellow Report* 5 (2015), pp. 1–19. DOI: 10.5170/CERN-2015-005.1. arXiv: 1705.08830 [physics.acc-ph] (cit. on p. 16).

[26] Apollinari G. et al. *High-Luminosity Large Hadron Collider (HL-LHC): Technical Design Report V. 0.1*. CERN Yellow Reports: Monographs. Geneva: CERN, 2017. URL: https://cds.cern.ch/record/2284929 (cit. on p. 16).

[27] D. J Griffiths. *Introduction to elementary particles; 2nd rev. version*. Physics textbook. New York, NY: Wiley, 2008. URL: https://cds.cern.ch/record/111880 (cit. on p. 16).

[28] C. Patrignani et al. “Review of Particle Physics”. In: *Chin. Phys.* C40.10 (2016), p. 100001. DOI: 10.1088/1674-1137/40/10/100001 (cit. on p. 16).

[29] K. Kröninger, Andreas B. Meyer, and P. Uwer. “Top-Quark Physics at the LHC”. In: *The Large Hadron Collider: Harvest of Run 1*. Ed. by Thomas Schörner-Sadenius. 2015, pp. 259–300. DOI: 10.1007/978-3-319-15001-7_7. eprint: 1506.02800. URL: https://inspirehep.net/record/1375310/files/arXiv:1506.02800.pdf (cit. on p. 18).

[30] C. Grandi et al. *CMS Computing Model: The "CMS Computing Model RTAG"*. Tech. rep. CMS-NOTE-2004-031. CERN-LHCC-2004-035. LHCC-G-083. Geneva: CERN, 2004. URL: http://cds.cern.ch/record/814248 (cit. on p. 21).

[31] G. L. Bayatyan et al. *CMS computing: Technical Design Report*. Technical Design Report CMS. Submitted on 31 May 2005. Geneva: CERN, 2005. URL: http://cds.cern.ch/record/838359 (cit. on p. 21).

[32] J. D. Shiers. “The Worldwide LHC Computing Grid (worldwide LCG)”. In: *ComputerPhysics Communications* 219-223 (2007) (cit. on p. 21).

[33] *WLCG Project*. URL: http://www.cern.ch/lcg (cit. on p. 21).

[34] *European Grid Infrastructure (EGI)*. URL: http://www.egi.eu/ (cit. on p. 22).

[35] *Open Science Grid (OSG)*. URL: http://www.opensciencegrid.org (cit. on p. 22).

[36] *WLCG Transfers Dashboard*. URL: http://monit-grafana-open.cern.ch/dashboard/db/wlcg-transfers-dashboard?orgId=16 (cit. on p. 24).

[37] *Large Hadron Collider Optical Private Network*. URL: https://twiki.cern.ch/twiki/bin/view/LHCOPN/WebHome (cit. on p. 25).

[38] *ROOT*. URL: https://root.cern.ch (cit. on p. 27).

[39] *WorkBookDataFormats*. URL: https://twiki.cern.ch/twiki/bin/view/CMSPublic/WorkBookDataFormats (cit. on p. 27).

[40] G. Petrucciani, A. Rizzi, and C. Vuosalo. “Mini-AOD: A New Analysis Data Format for CMS”. In: *Journal of Physics: Conference Series* 664.7 (2015), p. 072052. URL: http://stacks.iop.org/1742-6596/664/i=7/a=072052 (cit. on p. 28).

[41] M. Giffels, Y. Guo, and D. Riley. “Data Bookkeeping Service 3 – Providing event metadata in CMS”. In: *Journal of Physics: Conference Series*. Conference Series 513.4 (2014), p. 042022. URL: http://stacks.iop.org/1742-6596/513/i=4/a=042022 (cit. on p. 30).

[42] T. Wildish et al. “From toolkit to framework - the past and future evolution of PhEDEx”. In: *Journal of Physics*. Conference Series 396.3 (2012). Ed. by IOPscience, p. 032118. URL: http://iopscience.iop.org/1742-6596/396/3/032118 (cit. on p. 30).

[43] J. Rehn et al. “PhEDEx high-throughput data transfer management system”. In: *CHEP06* (2006). Ed. by GridPP. URL: http://www.gridpp.ac.uk/papers/chep06_tuura.pdf (cit. on p. 30).

[44] *HTCondor infrastructure*. URL: https://research.cs.wisc.edu/htcondor/mw/ (cit. on p. 30).

[45] *WMAgent monitoring*. URL: https://github.com/dmwm/WMCore/wiki/WMAgent-monitoring (cit. on p. 30).

[46] *CRAB*. URL: https://twiki.cern.ch/twiki/bin/view/CMSPublic/SWGuideCrab (cit. on p. 31).

[47] *Machine Learning*. URL: http://www.wikiwand.com/en/Machine_learning (cit. on p. 33).

[48] Thomas M. Mitchell. *Machine Learning*. 1st ed. New York, NY, USA: McGraw-Hill, Inc., 1997. ISBN: 0070428077, 9780070428072 (cit. on p. 35).

[49] *Machine Learning tutorial*. URL: https://class.coursera.org/ml-005/lecture (cit. on p. 35).

[50] The HEP Software Foundation (HSF). “A Roadmap for HEP Software and Computing R&D for the 2020s”. In: (2017). arXiv: 1712.06982 [physics.comp-ph] (cit. on pp. 42, 48).

[51] *GitHub*. URL: https://github.com/ (cit. on p. 42).

[52] *Kaggle*. URL: https://www.kaggle.com/ (cit. on p. 42).

[53] D. Bonacorsi et al. “Machine Learning in HEP”. In: *Nature Review in print* (2018) (cit. on p. 43).

[54] V. Kuznetsov et al. “Predicting dataset popularity for the CMS experiment”. In: *Journal of Physics: Conference Series* 762.1 (2016), p. 012048. URL: http://stacks.iop.org/1742-6596/762/i=1/a=012048 (cit. on p. 45).

[55] D. Bonacorsi et al. “Exploring Patterns and Correlations in CMS Computing Operations Data with Big Data Analytics Techniques”. In: *PoS* ISGC2015 (2015), p. 008 (cit. on p. 45).

[56] D. Bonacorsi et al. “Monitoring data transfer latency in CMS computing operations”. In: *Journal of Physics: Conference Series* 664.3 (2015), p. 032033. URL: http://stacks.iop.org/1742-6596/664/i=3/a=032033 (cit. on p. 45).

[57] L. Giommi. “Predicting CMS datasets popularity with machine learning”. Bachelor thesis. 2015. URL: http://amslaurea.unibo.it/9136/ (cit. on p. 45).

[58] *CMSSW framework*. URL: https://twiki.cern.ch/twiki/bin/view/CMSPublic/WorkBookCMSSWFramework (cit. on p. 48).

[59] *Tensorflow*. URL: https://www.tensorflow.org (cit. on p. 48).

[60] *Google Brain Team*. URL: https://research.google.com/teams/brain (cit. on p. 48).

[61] *Google's Machine Intelligence*. URL: https://research.google.com/pubs/MachineIntelligence.html (cit. on p. 48).

[62] *Theano*. URL: http://deeplearning.net/software/theano (cit. on p. 49).

[63] *NumPy*. URL: http://www.numpy.org (cit. on p. 49).

[64] *SimPy*. URL: http://www.sympy.org (cit. on p. 50).

[65] *MATLAB*. URL: https://www.mathworks.com/products/matlab.html (cit. on p. 50).

[66] *Mathematica*. URL: http://www.wolfram.com/mathematica/?source=nav (cit. on p. 50).

[67] *Caffe*. URL: http://caffe.berkeleyvision.org (cit. on p. 50).

[68] *Torch*. URL: http://torch.ch (cit. on p. 50).

[69] *MXNet*. URL: https://mxnet.apache.org (cit. on p. 50).

[70] T. Chen et al. "MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems". In: *CoRR* abs/1512.01274 (2015). arXiv: 1512.01274. URL: http://arxiv.org/abs/1512.01274 (cit. on p. 50).

[71] *Microsoft Cognitive Toolkit*. URL: https://www.microsoft.com/en-us/cognitive-toolkit (cit. on p. 50).

[72] *Keras*. URL: https://keras.io (cit. on p. 51).

[73] *Amazon Machine Learning*. URL: https://aws.amazon.com/aml (cit. on p. 51).

[74] *Amazon Web services*. URL: https://aws.amazon.com (cit. on p. 51).

[75] *Azure Machine Learning*. URL: https://docs.microsoft.com/en-us/azure/machine-learning/preview/overview-what-is-azure-ml (cit. on p. 51).

[76] *Microsoft Azure*. URL: https://azure.microsoft.com (cit. on p. 51).

[77] *Google Cloud AI*. URL: https://cloud.google.com/products/machine-learning (cit. on p. 51).

[78] *Google Cloud Platform*. URL: https://cloud.google.com (cit. on p. 51).

[79] *TFaaS*. URL: https://github.com/vkuznet/TFaaS (cit. on pp. 52, 83, 84).

[80] *DIANA-HEP*. URL: http://diana-hep.org (cit. on p. 53).

[81] *uproot*. URL: https://github.com/scikit-hep/uproot (cit. on p. 53).

[82] *Personal communication with V. Kuznetsov, CMS ML forum meeting, February 14th, 2018* (cit. on p. 53).

[83] *Root2numpy*. URL: http://scikit-hep.org/root_numpy (cit. on p. 53).

[84] *XRootD*. URL: http://xrootd.org (cit. on p. 54).

[85] *Problem with iterate function*. URL: https://github.com/scikit-hep/uproot/issues/44 (cit. on p. 57).

[86] *Fail to construct JaggedArray*. URL: https://github.com/scikit-hep/uproot/issues/43 (cit. on p. 57).

[87] *vector<bool> not supported*. URL: https://github.com/scikit-hep/uproot/issues/45 (cit. on p. 57).

[88] *PyROOT*. URL: https://root.cern.ch/pyroot (cit. on p. 60).

[89] *Evaluating a classification model*. URL: http://www.ritchieng.com/machine-learning-evaluate-classification-model (cit. on p. 61).

[90] *Metrics To Evaluate Machine Learning Algorithms in Python*. URL: https://machinelearningmastery.com/metrics-evaluate-machine-learning-algorithms-python (cit. on p. 61).

[91] *Overfitting in Machine Learning: What It Is and How to Prevent It*. URL: https://elitedatascience.com/overfitting-in-machine-learning (cit. on pp. 64, 66).

[92] *WTF is the Bias-Variance Tradeoff?* URL: https://elitedatascience.com/bias-variance-tradeoff (cit. on p. 64).

[93] *Evaluate the Performance of Machine Learning Algorithms in Python using Resampling*. URL: https://machinelearningmastery.com/evaluate-performance-machine-learning-algorithms-python-using-resampling (cit. on p. 65).

[94] *Cross Validation*. URL: https://www.cs.cmu.edu/~schneide/tut5/node42.html (cit. on p. 65).

[95] *scikit-learn, Machine Learning in Python*. URL: http://scikit-learn.org/stable/ (cit. on p. 66).

[96] *Scalable and Flexible Gradient Boosting*. URL: http://xgboost.readthedocs.io/en/latest/ (cit. on p. 68).

[97] *Complete Guide to Parameter Tuning in XGBoost (with codes in Python)*. URL: https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/ (cit. on p. 68).

[98] *How to avoid Over-fitting using Regularization?* URL: https://www.analyticsvidhya.com/blog/2015/02/avoid-over-fitting-regularization/ (cit. on p. 68).

[99] *Introduction to Gradient Descent Algorithm (along with variants) in Machine Learning*. URL: https://www.analyticsvidhya.com/blog/2017/03/introduction-to-gradient-descent-algorithm-along-its-variants/ (cit. on p. 70).

[100] *Optimization: Stochastic Gradient Descent*. URL: http://ufldl.stanford.edu/tutorial/supervised/OptimizationStochasticGradientDescent/ (cit. on p. 70).

[101] E. Ballabene. "Measurement of the $t\bar{t}$ production cross section at 13 TeV in the all-jets boosted regime with CMS at LHC". Bachelor thesis. Alma Mater Studiorum Università di Bologna, 2016/2017 (cit. on p. 72).

[102] P. Nath et al. "The Hunt for New Physics at the Large Hadron Collider". In: *Nucl. Phys. Proc. Suppl.* 200-202 (2010), pp. 185–417. DOI: 10.1016/j.nuclphysbps.2010.03.001. arXiv: 1001.2693 [hep-ph] (cit. on p. 73).

[103] *TMVA, Toolkit for Multivariate Data Analysis with ROOT*. URL: http://tmva.sourceforge.net (cit. on p. 73).

[104] A. J. Larkoski et al. "Soft Drop". In: *JHEP* 05 (2014), p. 146. DOI: 10.1007/JHEP05(2014)146. arXiv: 1402.2657 [hep-ph] (cit. on p. 79).

[105] *Keras to Tensorflow package*. URL: https://github.com/vkuznet/keras_to_tensorflow.git (cit. on p. 84).

[106] *Golang*. URL: https://golang.org/ (cit. on p. 84).

[107] *Go Tensorflow Library*. URL: https://www.tensorflow.org/versions/master/install/install_go (cit. on p. 84).

Acknowledgments

With this master thesis I am closing a big chapter of my studying life and starting a new journey. My passion for physics began when I was in highschool and from then on it kept giving me satisfaction till this very day. I would like to start by thanking my family for their support and for believing in me.

I would also like to thank my supervisor Daniele for the constant guidance throughout this thesis and during these years in university. I thank Valentin Kuznetsov for the technical support and for giving me the opportunity to work with him. I would also like to thank Andrea Castro for his support of top quark physics and for being always present and helpful.

Last but not least I would like to thank all of my friends and all my volleyball teams who were standing by my side during my university life.

Bologna, 23 Marzo 2018 Luca Giommi