Università degli Studi di Milano

FACOLTÀ DI SCIENZE E TECNOLOGIE
Corso di Laurea in Fisica

Elaborato Finale

Improving Suppression of Jets from Pileup at High Luminosity LHC Using Timing Information from the High Granularity Timing Detector for ATLAS

Relatore:
Prof. Leonardo C. Carminati

Correlatori:
Dott. Silvia Resconi
Dott. Marianna Testa
Dott. Ruggero Turra

Candidato:
Martino M. L. Pulici
Matricola 918820

Anno Accademico 2019–2020

A mio nonno

Sommario

In un acceleratore insieme alle collisioni di interesse ("collisioni dure"), ne avvengono di secondarie ("collisioni soffici" o "di *pileup*"): distinguere i getti provenienti dalle collisioni dure da quelli provenienti dalle collisioni soffici sarà una delle sfide nella prossima fase ad Alta Luminosità di LHC. Per far fronte all'aumento degli eventi secondari sarà introdotto l'*High Granularity Timing Detector*: esso assegnerà una misura temporale alle tracce di particelle cariche ed escluderà tutte le tracce non compatibili con il tempo globale di evento. La distinzione di getti primari e secondari può essere effettuata utilizzando un *Boosted Decision Tree*, un algoritmo di apprendimento supervisionato che ottimizza una Funzione Obiettivo per classificare efficientemente e correttamente alcuni dati. In questa tesi il primo approccio al problema di classificazione di getti primari e secondari è stato l'utilizzo di sole variabili di alto livello per l'addestramento, ottenendo un miglioramento considerevole utilizzando variabili di tempo. È stato quindi esplorato il potenziale delle variabili di basso livello, con e senza l'utilizzo dell'informazione temporale. È stato trovato che la configurazione base di basso livello ottiene prestazioni superiori rispetto al miglior caso con variabili di alto livello, con un ulteriore miglioramento garantito dall'utilizzo di informazioni temporali di basso livello.

Abstract

In an accelerator, together with the collisions of interest ("hard collisions"), secondary ones ("soft" or "pileup collisions") take place: discriminating jets coming from hard collisions from the ones coming from soft collisions will be one of the main challenges in the forthcoming will be introduced: it will assign a time measurement to charged particle tracks and exclude all the tracks which are not compatible with the global event time. The discrimination of signal and background jets can be done by using a Boosted Decision Tree, a supervised learning algorithm that optimizes an Objective Function in order to efficiently and correctly classify some data. In this thesis, the first approach to the signal and background jet classification problem was to use only high-level jet variables in the training, obtaining a considerable improvement when timing variables were used. Then, the potential of low-level track variables was explored, with and without the usage of timing information. It was found that the basic track configuration performs notably better than the best jet case, with a further improvement granted by the usage of low-level timing information.

Contents

List of figures

List of tables

Foreword

In an accelerator, particles are propelled at speeds close to the light's speed and are made to collide. Events of this kind are particularly crucial for physicists since at such high speeds the collision energy can be converted into matter (according to the well-known $E = mc^2$ equation), enabling the investigation of sub-atomic particle creation and other phenomena. For example, the collision between protons can lead to the production of Higgs bosons or top quarks, whose study is crucial for understanding the fundamental laws governing the Universe. These interesting particles have very short lifetimes, quickly decaying into lighter ones. In some cases, what is measured is a collection of hadrons (jets) emerging from the collision vertices.

In addition to the primary collisions, where physics events of interest occur, additional ones take place. In these secondary collisions, typically low transverse momentum jets are produced. These unwanted jets are known as pileup jets, and discriminating them from signal ones is one of the main challenges at hadron colliders. Doing so is particularly challenging when the instantaneous luminosity is high, such as at LHC, even more at HL-HLC. In this thesis, an algorithm based on machine learning techniques will be used to discriminate between hard-scatter jets and pileup jets. The information deriving from the new High Granularity Timing Detector will be used to increase the algorithm's discriminating power.

Until now, timing information was included in high (jet) level variables, potentially missing some low-level information contained in the single tracks which compose jets. This thesis has two aims: the first one is to check whether the usage of track information may increase the algorithm's performance; the second one is to evaluate the importance of track-level timing information provided by the High Granularity Timing Detector.

This thesis is organized as follows: in Chapter 1, the Large Hadron Collider and the ATLAS detector will be presented, together with a description of the HGTD detector; in Chapter 2, the jet reconstruction procedure will be briefly illustrated; in Chapter 3, the concept of Boosted Decision Trees will be introduced, looking at the LightGBM framework and its characteristics with particular attention; in

Chapter 4, the high-level variables characterizing a jet will be defined and analyzed, before evaluating their usage in the training of the BDT; in Chapter 5, the low-level variables will be studied, and their impact on the BDT performance will be assessed; in Chapter 6, the results found in Chapters 4 and 5 will be compared, in order to come to a conclusion regarding the two aims of the thesis.

Part I

THEORETICAL BACKGROUND

Chapter 1

The Large Hadron Collider and the ATLAS detector

1.1 Luminosity

Luminosity is one of the most important quantities to describe collisions and can be intuitively understood as the measure of how many collisions are happening in an accelerator [1]. Since a higher number of collisions makes it more likely to observe a certain process, luminosity L can be more precisely defined as the proportionality factor between the event rate $\frac{\mathrm{d}N}{\mathrm{d}t}$ and the cross section σ:

$$\frac{\mathrm{d}N}{\mathrm{d}t} = L\sigma \tag{1.1}$$

where the cross section σ is a way to measure the probability of something happening and is measured in barns ($1\,\mathrm{b} = 10^{-28}\mathrm{m}^2$).

It is also intuitive that the event rate must be proportional to the number of bunches n_b, to the square of the number of protons per bunch N^2_{bunch}, to the collision frequency f, and to the reciprocal of the beams' area S, giving:

$$\frac{\mathrm{d}N}{\mathrm{d}t} = \frac{n_b N^2_{\mathrm{bunch}} f}{S}\sigma \tag{1.2}$$

Combining Eqs. (1.1) and (1.2) it is then possible to find an explicit expression for luminosity:

$$L = \frac{n_b N^2_{\mathrm{bunch}} f}{S} \tag{1.3}$$

Another essential quantity to describe the accelerator is given by the integrated luminosity, $\int L\mathrm{d}t$, which is a measurement of the data size.

1.2 Hard and soft collisions

In a particle accelerator, the majority of events originate from large-distance collisions between protons. Since the transferred momentum is small, events of this kind take the name of "soft collisions" [2]: the jets produced tend to have large longitudinal momentum and small transverse momentum. An example of an event with a very high number of pileup vertices is shown in Fig. 1.1. Even though these collisions represent the majority of interactions, they are not very interesting.

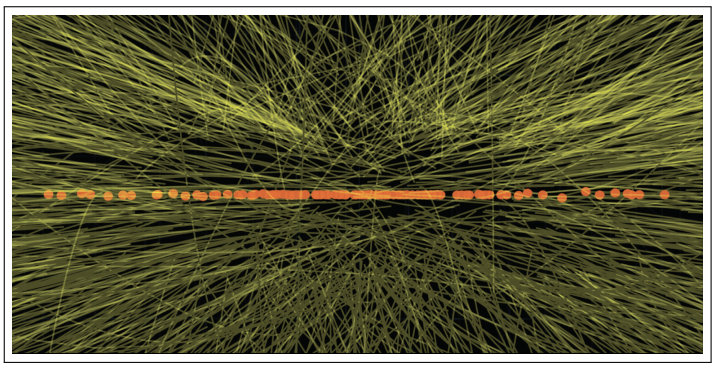

Figure 1.1: An event with very high pileup

Sometimes, though, collisions occur between two partons of the incoming protons, resulting in large momentum transfer ("hard collisions"). For such events, the resulting jets can be produced at large angles, creating massive particles. These interesting events are sporadic, though: for example, the W boson production through quark-antiquark annihilation has a cross section 10^5 times smaller than the total inelastic p-p cross section.

1.3 The Large Hadron Collider

The Large Hadron Collider (LHC) [3] is the world's largest and most powerful particle accelerator, built between 1998 and 2008 by the European Organization for Nuclear Research (CERN, from the French *Conseil Européen pour la Recherche Nucléaire*). As shown in Fig. 1.2, LHC is not the first accelerator built at CERN, but rather the last step in a series of projects starting in 1959 with the Proton Synchrotron, which accelerated protons to an energy around 500 times smaller than LHC. LHC is designed to accelerate two proton (or heavy ion) beams at a center of mass energy of 7 TeV. Its luminosity values range around $10^{34}\text{cm}^{-2}\text{s}^{-1}$, as shown in Fig. 1.3, with a record set in 2017 of $2.06 \times 10^{34}\,\text{cm}^{-2}\text{s}^{-1}$. Integrated luminosity for various years is also shown in Fig. 1.4.

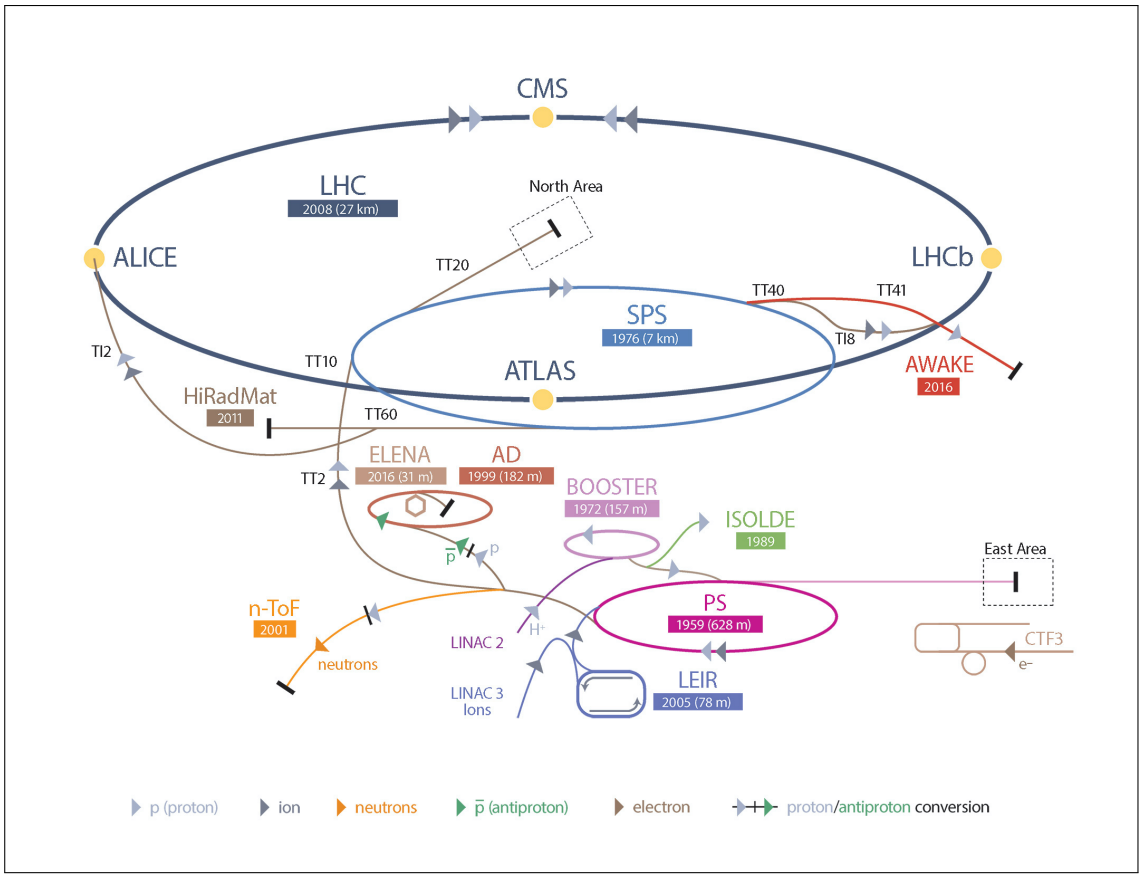

Figure 1.2: The various stages of CERN accelerators

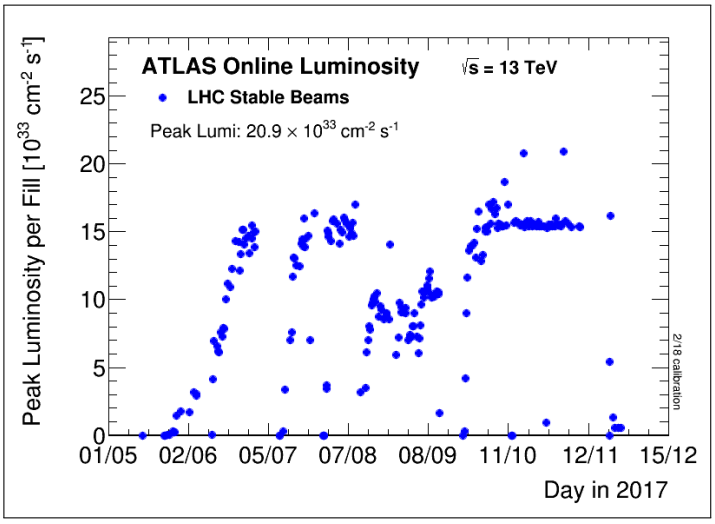

Figure 1.3: Peak luminosity at LHC in 2017 for p-p collisions

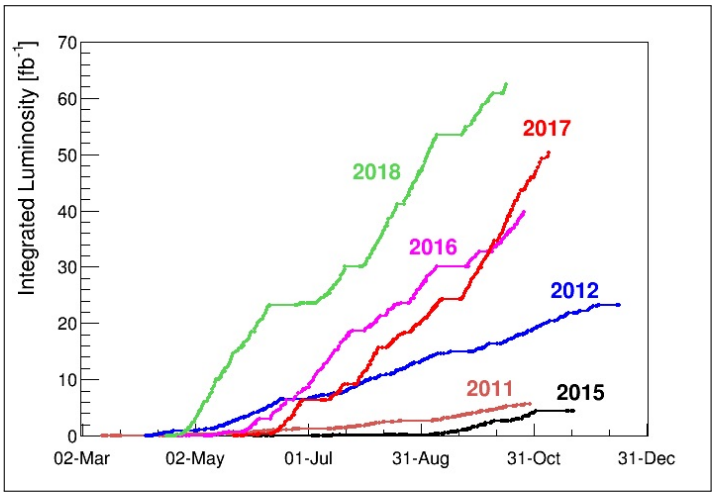

Figure 1.4: Integrated luminosity at LHC

Momentum at collision	7 TeV
Momentum at injection	450 GeV
Machine circumference	26 658.883 m
Revolution frequency	11.2455 kHz
Luminosity	$10^{34}\text{cm}^{-2}\text{s}^{-1}$
Particles per bunch	11×10^{11}
Bunch separation	24.95 ns
Bunch spacing	7.48 mm

Table 1.1: Various LHC machine parameters

LHC consists of a 27-kilometer ring of superconducting magnets and accelerating cavities, making two high-energy particle beams travel at energies close to 7×10^{12} eV, with a speed corresponding to 99.999 999 % that of light, in ultrahigh vacuum[1]. The strong magnetic field needed, which reaches 8.3 T, is maintained by 1624 superconducting electromagnets (Fig. 1.5) chilled to a temperature of −271.3 °C by a distribution system of liquid helium. Some other machine parameters are reported in Table 1.1. When the beams reach the required energies they are made to collide at four locations: A Toroidal LHC ApparatuS (ATLAS), Compact Muon Solenoid (CMS), A Large Ion Collider Experiment (ALICE) and LHC-beauty (LHCb).

[1]Equivalent to some 10^{-6} mbar.

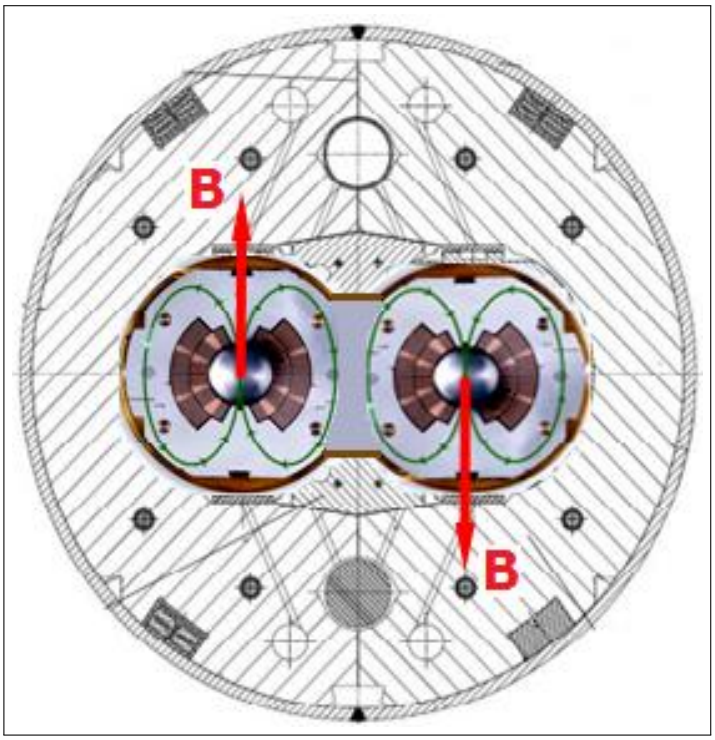

Figure 1.5: A cross section of the LHC magnet system

1.4 ATLAS

ATLAS [4], shown in Fig. 1.6, is a general-purpose particle physics experiment, comprising about 3000 scientific authors from 181 institutions around the world, designed to exploit the opportunities provided by LHC to their fullest. The basic questions ATLAS physicists are trying to answer are the ones regarding the predictions of the Standard Model, such as the existence of the Higgs boson finally discovered in 2012. Other investigation areas are the search for extra dimensions and for particles that could make up dark matter.

ATLAS [5] is the largest detector ever constructed, 46-kilometer long, 25 m in diameter, sitting in a cavern 100 m below ground and weighing around 7000 t. It consists of three different concentrically wrapped detecting subsystems which individually record trajectory, momentum, and energy of particles. Every second in the ATLAS detector over a billion particle interactions take place. Out of them, around a thousand are flagged as potentially useful and recorded, with a ratio of one in a million interactions.

The detector comprises four main components: the Inner Detector, the Calorimeter, the Muon Spectrometer, and the Magnet System.

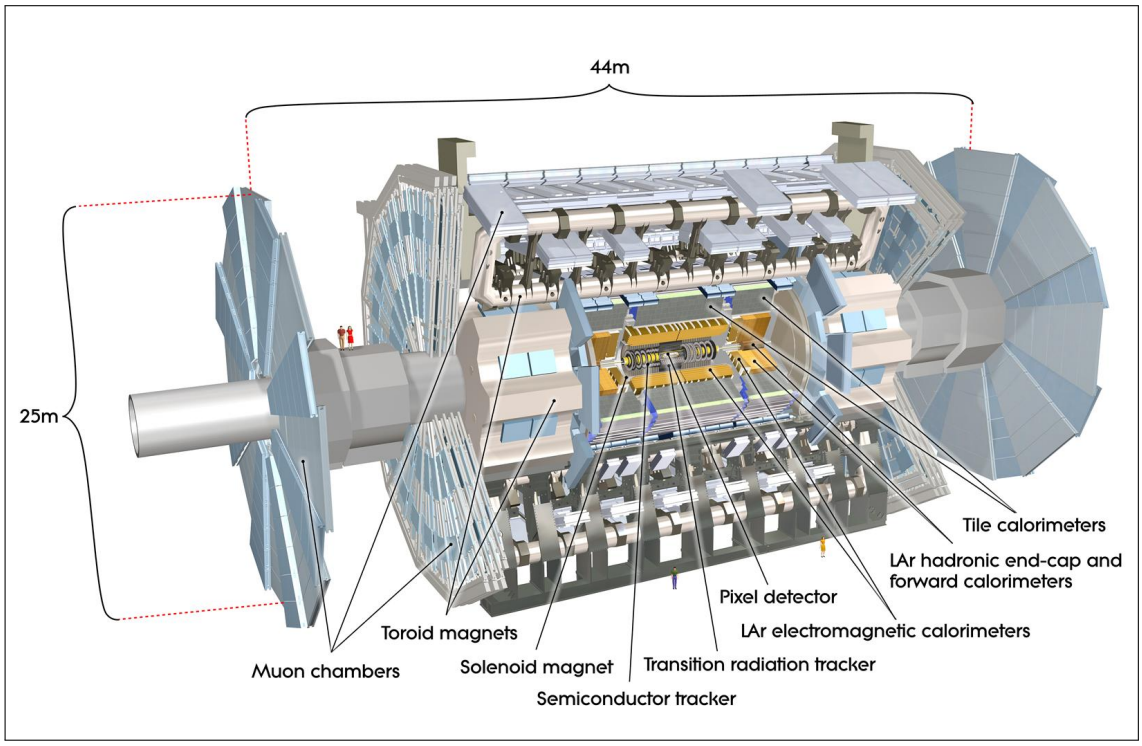

Figure 1.6: The ATLAS detector

1.4.1 The ATLAS frame of reference

When dealing with ATLAS data, some coordinates are given using the frame of reference shown in Fig. 1.7. The z coordinate represents the longitudinal position along the beam axis, x represents the coordinate pointing toward the center of LHC, while y is directed upwards. Two angular coordinates are also defined: an azimuthal angle θ and a polar angle φ.

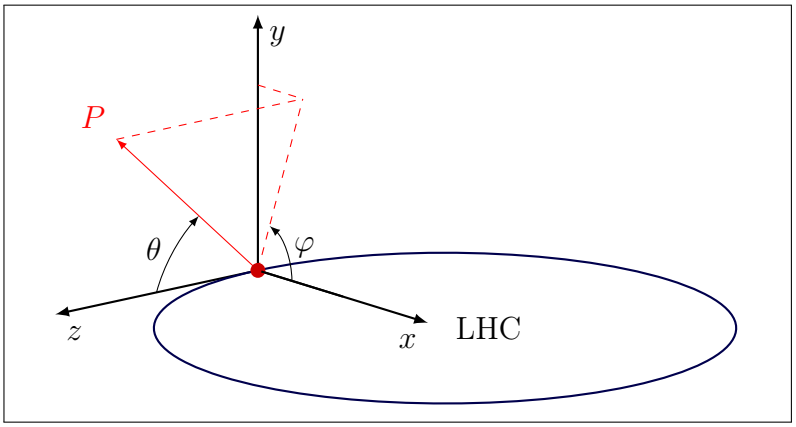

Figure 1.7: The ATLAS frame of reference

In addition, another variable, known as pseudorapidity, is often used. Pseudorapidity η is a spatial coordinate defined as:

$$\eta := -\log\left[\tan\left(\frac{\theta}{2}\right)\right] \tag{1.4}$$

Pseudorapidity can therefore range between $\eta = -\infty$ and $\eta = +\infty$.

1.4.2 The Inner Detector

The Inner Detector (Fig. 1.8) is immersed in a magnetic field parallel to the beam axis and is the component closest to the beam pipe. It covers the pseudorapidity region $|\eta| < 2.5$ and consists of one barrel and two end-caps. It measures direction, momentum, and charge of electrically-charged particles produced in the collisions: in particular, its typical momentum resolution is:

$$\frac{\sigma_{p_T}}{p_T} \approx 0.05\,\% p_T \oplus 1\,\% \tag{1.5}$$

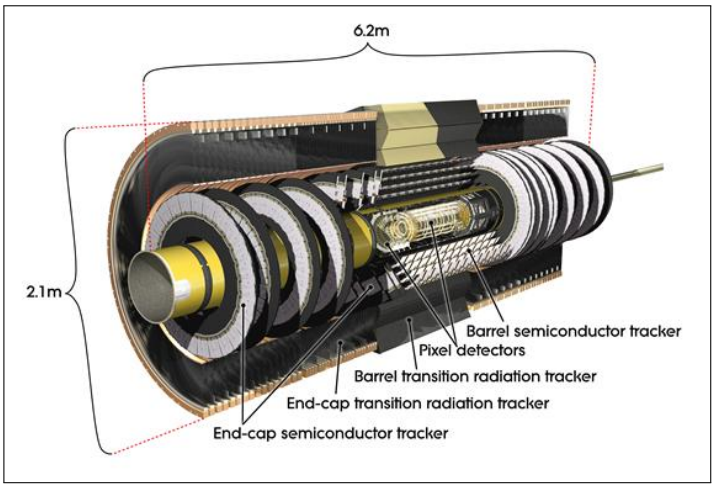

Figure 1.8: The Inner Detector

The Inner Detector is composed of three different systems of sensors: the Pixel Detector, the Semiconductor Tracker, and the Transition Radiation Tracker.

The Pixel Detector

The Pixel Detector is made up of silicon pixel sensors arranged in four cylindrical layers in the barrel and three discs in the end-caps. It has an area of $1.7\,\mathrm{m}^2$ and is composed of 80 million pixels, for a power consumption of 15 kW. Each of the 1744 modules has 46 080 pixels, exhibiting a typical single-hit resolution of $14 \times 115\,\mu\mathrm{m}^2$ in $R\varphi$.

The Semiconductor Tracker

The Semiconductor Tracker (SCT) is composed of silicon micro-strips with a 80 µm pitch, and organized in 4088 two-sided modules, for a total of over six million channels. The 60 m^2 of silicon are distributed over nine cylindrical barrel layers and nine planar end-cap discs glued together in couples with a stereo angle of 10 µm to provide four space points. The SCT allows the position of charged particles to be recorded with an accuracy of 17 µm per layer in $R\varphi$.

The Transition Radiation Tracker

The Transition Radiation Tracker (TRT) has a volume of 12 m^3, for a total of 350 000 readout channels. The basic detector element of the TRT is a 4-millimeter-diameter straw tube with a 0.03-millimeter-diameter gold-plated tungsten wire in the middle. Each straw tube works as a drift chamber. There are 50 000, 144-centimeter-long straws in the barrel, and 250 000, 39-centimeter-long straws in both end-caps. The TRT has a precision of around 0.17 mm and provides additional information on the particle type, exploiting the measurement of the transition radiation.

1.4.3 The Calorimeter

The function of the Calorimeter (Fig. 1.9) is to measure the particle energy lost through the detector [6]. Calorimeters are designed to stop entirely or absorb most of the particles coming from a collision and are typically composed of alternating passive and active layers. Calorimeters can be electromagnetic, measuring the energy of electrons and photons, or hadronic, measuring the energy of hadrons.

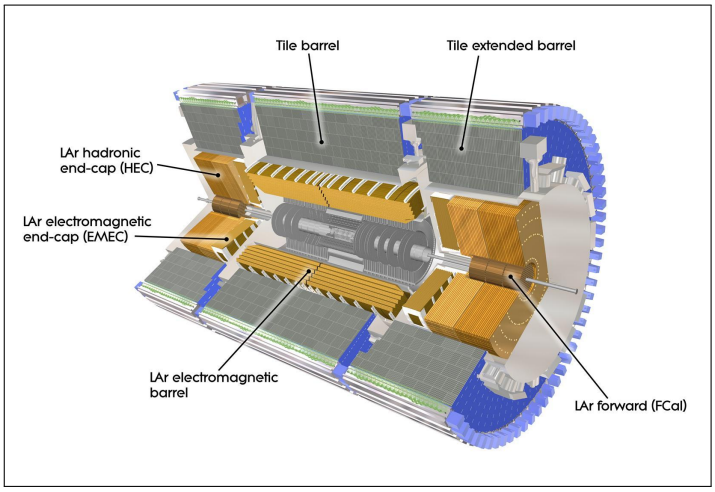

Figure 1.9: The ATLAS Calorimeter

The Electromagnetic Calorimeter

The Electromagnetic Calorimeter has a length of 6.65 m and an external radius of 2.25 m. The central barrel region covers the pseudorapidity region of $|\eta| < 1.4$, while the end-caps extend the coverage up to $|\eta| = 3.2$. It is made up of 2-millimeter layers of liquid argon as the active medium and copper as the passive one. It has a thickness of 22 times the radiation length in the barrel and more than 24 times the radiation length in the end-caps. It is segmented in 190 000 cells along η and φ and has four longitudinal layers. Its nominal resolution is:

$$\frac{\sigma_E}{E} \approx \frac{10\,\%}{\sqrt{E}} \oplus 0.7\,\% \tag{1.6}$$

The Hadronic Calorimeter

The Hadronic Calorimeter is 6.10-meter-long, with an outer radius of 4.5 m, covering the pseudorapidity range $|\eta| < 3.9$. The central barrel region ($|\eta| < 1.7$) uses scintillators and iron as passive medium. In the end-cap Hadronic Calorimeter (HEC), liquid argon is the active medium, with copper and tungsten as absorbers. For $\eta = 0$ the thickness is 11 times the nuclear interaction length. Its nominal resolution is:

$$\frac{\sigma_E}{E} \approx \frac{50\,\%}{\sqrt{E}} \oplus 3\,\% \tag{1.7}$$

The Forward Calorimeter

The Forward Calorimeter (FCal) is placed at a distance of 4.7 m from the interaction point and measures both electromagnetic and hadronic showers. The electromagnetic part utilizes liquid argon as the active medium and copper as the passive; the hadronic parts use tungsten as the passive material. Its nominal resolution is:

$$\frac{\sigma_E}{E} \approx \frac{100\,\%}{\sqrt{E}} \oplus 10\,\% \tag{1.8}$$

1.4.4 The Muon Spectrometer

As already discussed, muons pass through the Inner Detector and the Calorimeters; therefore, the need for a specific Muon Spectrometer (Fig. 1.10) arises. It is made up of 4000 individual muon chambers and, using four different technologies, identifies and measures the momenta of muons, exploiting the curvature in the toroidal magnetic field. The Muon Spectrometer has four subsections: the Monitored Drift Tubes, the Cathode Strip Chambers, the Thin Gap Chambers, and the Resistive Plate Chambers.

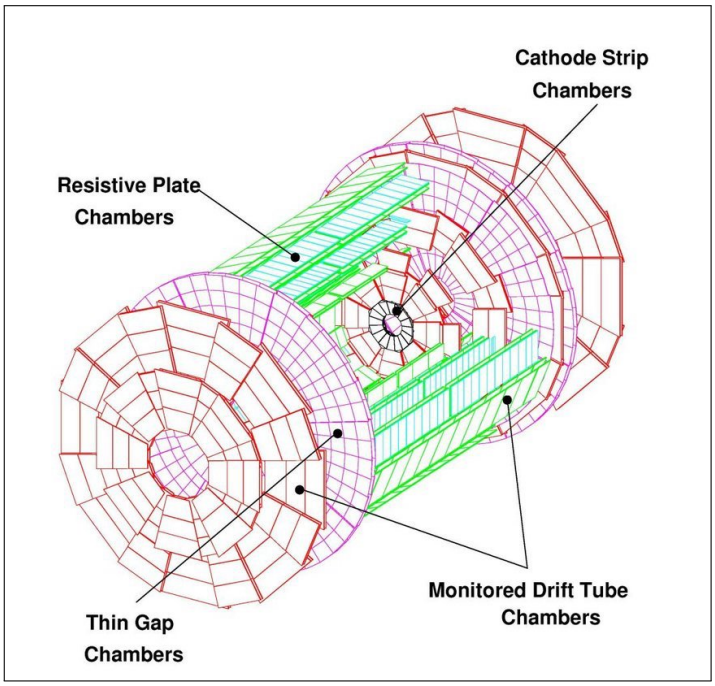

Figure 1.10: The Muon Spectrometer

The Monitored Drift Tubes and the Cathode Strip Chambers

The Monitored Drift Tubes ($|\eta| < 2.7$) measure the muons tracks with a resolution of 80 µm. They are composed of 1171 chambers with a total of 354 240 tubes, each with a diameter of 3 cm and a length between 0.85 and 6.5 m.

The Cathode Strip Chambers ($2 < |\eta| < 2.7$) consist of 70 000 channels with a 60 µm resolution. Their function is to measure precise coordinates at the ends of the detector.

The Thin Gap and the Resistive Plate Chambers

The Thin Gap Chambers ($1.05 < |\eta| < 2.7$) and the Resistive Plate Chambers ($|\eta| < 1.5$) are made up of respectively 440 000 and 380 000 channels. They provide fast trigger signals for the central ATLAS trigger and the drift time measurement in the drift chambers.

1.4.5 The Magnet System

The Magnet System's function is to bend particles around the various detectors' layers to allow momentum measurement of charged particles. There are three parts in the Magnet System: the Central Solenoid Magnet, the Barrel Toroid, and the End-Cap Toroids.

The Central Solenoid Magnet

The Central Solenoid Magnet bends charged particles for momentum measurement in the Internal Detector volume. It is 5.3-meter-long and 4.5-centimeter-thick, for a diameter of 2.4 m and a weight of 5 t. 7.73 kA of current flow in 9 km of superconducting wire, generating a magnetic field of 2 T with a stored energy of 38 MJ.

The Barrel Toroid

The Barrel Toroid has a length of 25.3 m and an outer diameter of 20.1 m, weighing 830 t. A magnetic field of 0.5 T is created by eight separate coils, with a total stored energy of 1.08 GJ. The 100-kilometer-long superconducting wire works at a temperature of 4.7 K, with a nominal current of 20.5 kA.

The End-Cap Toroid

The End-Cap Toroid is made up of eight coils, each storing an energy of 0.25 GJ. It is 5-meter long, with an outer diameter of 10.7 m and a weight of 240 t. A current of 20.5 kA flowing through a superconductor at the temperature of 4.7 K generates a field of 1 T.

1.5 High Luminosity LHC

In 2026 LHC will enter the High Luminosity (HL-LHC) phase [7], in which an integrated luminosity of up to 4000 fb^{-1} will be delivered in about ten years, as shown in Fig. 1.11. Instantaneous luminosity will reach $7.5 \times 10^{34}\,\text{cm}^{-2}\text{s}^{-1}$, more than three times the current value. The interaction region will spread over 50 mm along the axis, for an average of 1.6 collisions per millimeter, compared to the current 0.24.

To cope with these conditions, ATLAS will need a series of important upgrades: in general, both data acquisition and trigger systems are being upgraded, especially in terms of faster electronics, to achieve the desired output rate. The Inner Tracker will cover up to $|\eta| < 4$, and muon coverage will be extended. Tracking, vertexing, muon triggering, and timing will also be improved. The large amount of available data will grant the opportunity to explore Standard Model physics with unprecedented precision and boosting the search for new physics and dark matter.

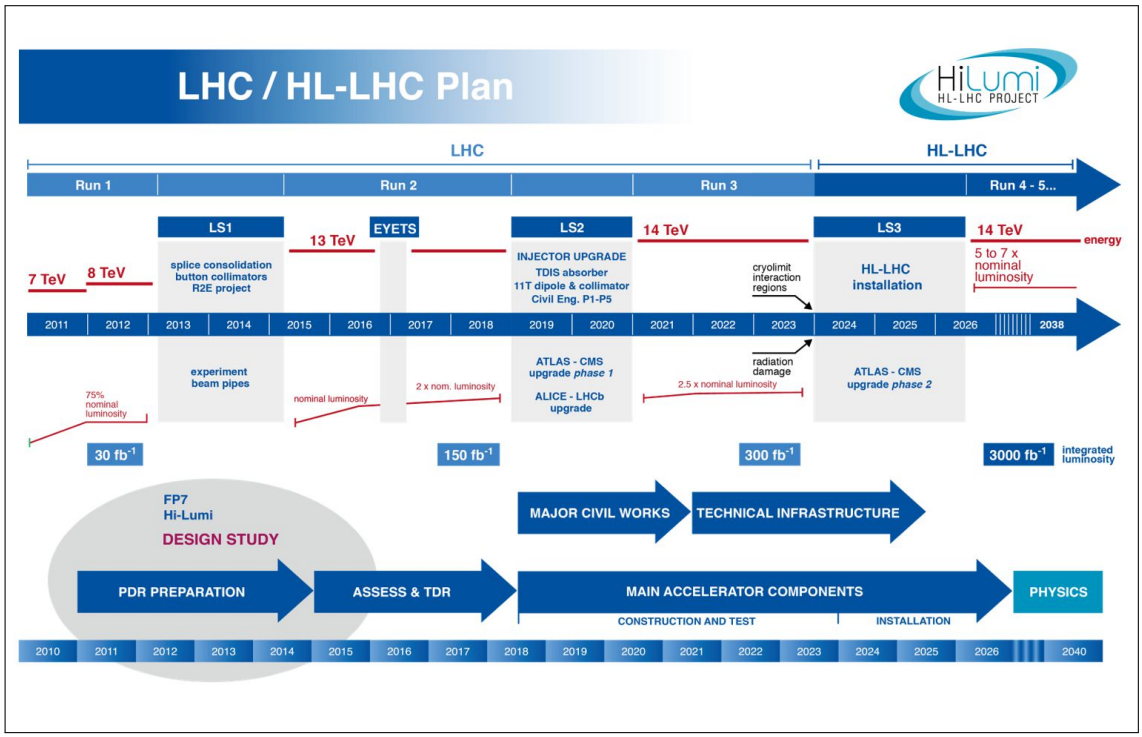

Figure 1.11: The LHC timeline

1.5.1 Physics motivations of HL-LHC

One of the opportunities offered by HL-LHC in the field of electroweak physics will regard a more precise determination of the W boson mass, pushing uncertainty down to 10 MeV. Moreover, vector boson scattering processes, which are essential to test electroweak symmetry-breaking mechanisms, are expected to be observed.

Another measurement that HL-LHC will make possible is that of the top mass value from $J/\psi \rightarrow \mu\mu$ decays, which is orthogonal to jet-based measurements. Besides, the high amount of data will allow rare Standard Model processes to be observed, such as the production of four-top quarks.

After having been observed by LHC in 2012, the properties of the Higgs boson will be investigated as one of the main goals of HL-LHC. In particular, exploiting the large amount of produced Higgs bosons, all couplings will be measured with a percent-level precision. Rare decay models such as $H \rightarrow \mu\mu$ or $H \rightarrow Z\gamma$ could also be observed. ATLAS and CMS will extract the self-coupling of the boson with a significance of around 3σ each, for an overall significance of 4σ.

Other research fields might include dark matter, the flavor problem, neutrino masses, the strong CP problem, and baryogenesis. The new particles predicted for these events will be searched for thanks to the much larger statistics, slightly higher energy, and upgraded detectors.

1.6 The High Granularity Timing Detector

Since high luminosity is correlated with a higher occurrence of background events, the main challenge of HL-LHC will be to reject the particles produced by pileup. To face this issue, the High Granularity Timing Detector (HGTD), a device with a time resolution of 30 ps in the forward region [8], has been proposed (Fig. 1.12).

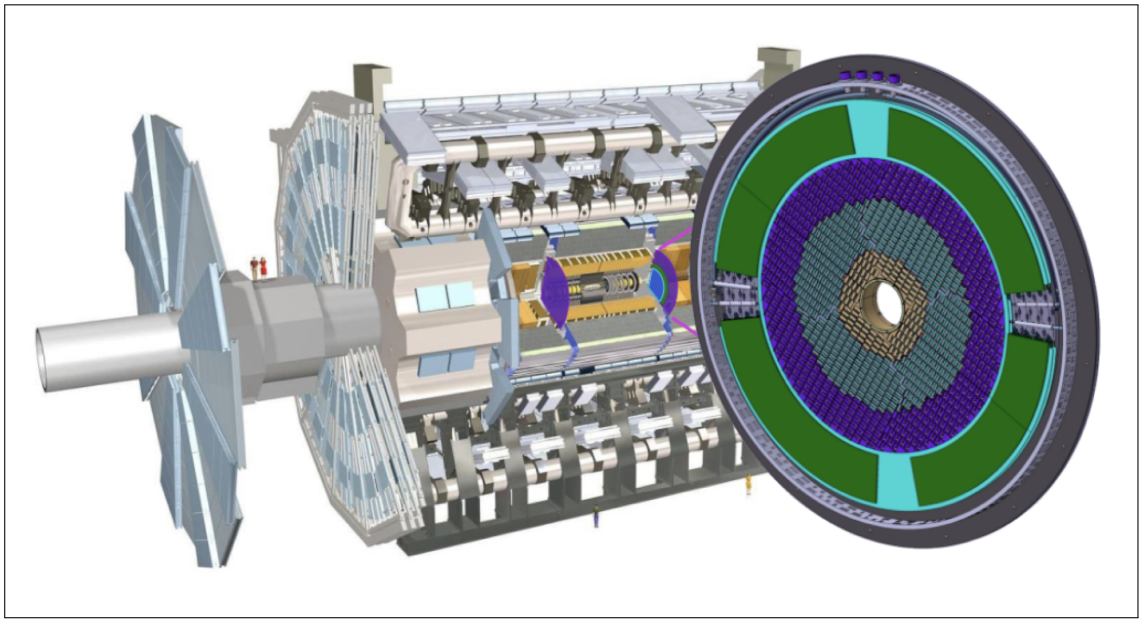

Figure 1.12: The High Granularity Timing Detector

The idea behind HGTD is to assign a time measurement to charged particle tracks and then using the timing information together with the already available longitudinal impact parameter to exclude all the tracks which are not compatible in time with the hard-scatter interaction. HGTD will also improve luminosity measurements and the general performance of the ATLAS detector in the forward region.

1.6.1 The detector

The detector will be placed in front of the End-Cap Calorimeters, covering the $|\eta|$ region from 2.4 to 4. The active radius will range between 120 mm to 640 mm, with a thickness not over 125 mm. Low Gain Avalanche Detectors with an active thickness of 50 µm will be used, with pixels of $1.3 \times 1.3\,\mathrm{mm}^2$. Because of the high particle rate in the region, radiation will damage the sensors' electronics, decreasing the timing resolution as a function of integrated luminosity. Therefore, some of the sensors will probably have to be replaced after half of the detector's lifetime.

Low Gain Avalanche Detectors

The HGTD will be made of Low Gain Avalanche Detectors (LGAD's), shown in Fig. 1.13. LGAD's are silicon diode sensors with an internal gain provided by an extra p-type layer added below the p-n junction. To obtain the desired 30-picosecond timing resolution, the time walk and the time jitter need to be accounted for. Both of these effects are inversely proportional to the signal's rise time; thus, a high slope and a high gain need to be used. Therefore, sensors will have an active thickness of 50 µm and a gain of 20.

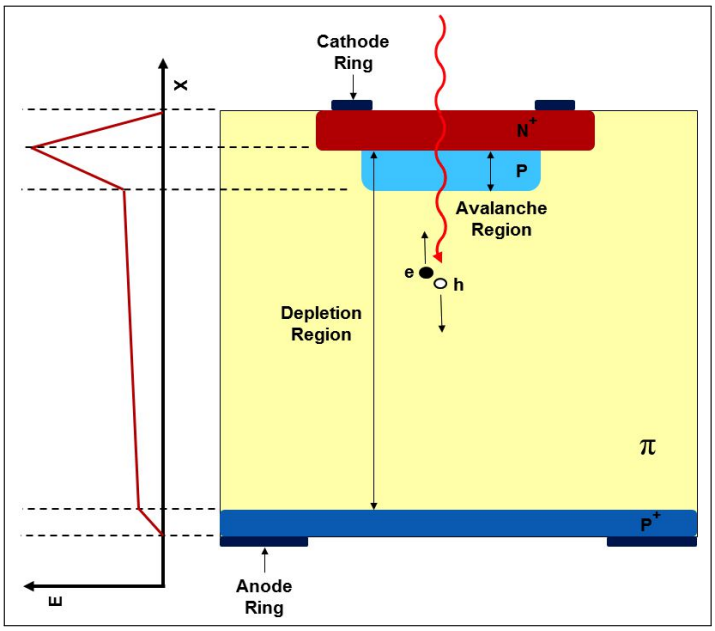

Figure 1.13: The Low Gain Avalanche Detector

1.6.2 Suppression of pileup jets

As already discussed, suppression of pileup jets is one of the main reasons why HGTD will be installed in the ATLAS detector. A simple variable to effectively discriminate the jets is $R_{p_{\mathrm{T}}}$, defined as:

$$R_{p_{\mathrm{T}}} = \frac{\sum p_{\mathrm{T}}^{\mathrm{trk}}\left(\mathrm{PV}_0\right)}{p_{\mathrm{T}}^{\mathrm{jet}}} \tag{1.9}$$

where $\sum p_{\mathrm{T}}^{\mathrm{trk}}\left(\mathrm{PV}_0\right)$ is the sum of the transverse momenta of all the tracks inside the jet cone and originating from the hard-scatter vertex PV_0, while $p_{\mathrm{T}}^{\mathrm{jet}}$ is the fully calibrated jet transverse momentum. In general, small values of $R_{p_{\mathrm{T}}}$ correspond to jets with a small fraction of p_{T} originating from the vertex PV_0, probably indicating a pileup jet. However, at high pileup conditions, the discriminating power

of this variable is reduced, because of the worsening of the longitudinal impact parameter resolution, which results in pileup tracks being incorrectly included in the numerator of $R_{p_{\mathrm{T}}}$, thus the need to use timing information obtained from HGTD.

Chapter 2

Object reconstruction

As already anticipated, particles produced in collisions cannot be directly detected. The partons undergo fragmentation and recombine to form a collimated jet of hadrons with a total momentum pointing approximately in the same direction as the initial parton. In other words, jets are a tool to represent groups of hadrons in a detector and need to be reconstructed and calibrated (Fig. 2.1).

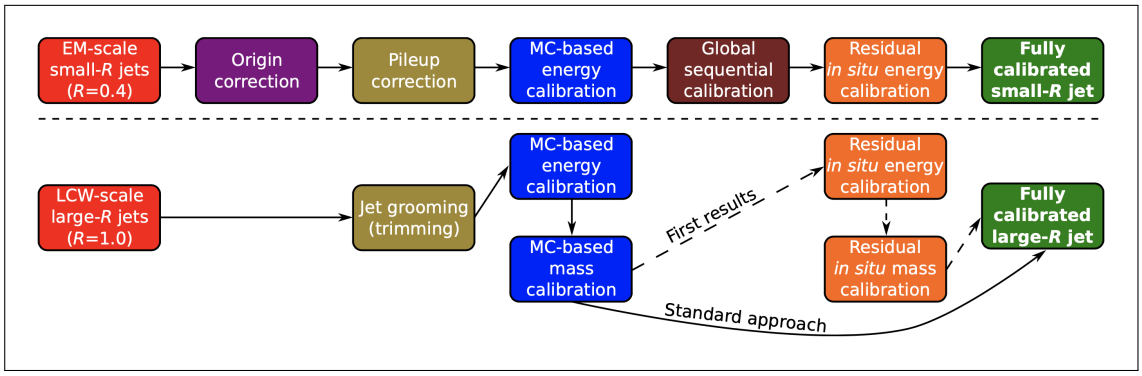

Figure 2.1: Jet reconstruction and calibration overview

2.1 Jet reconstruction

Jet reconstruction consists of three primary stages: input definition, parameter definition, and jet grooming.

2.1.1 Jet inputs

ATLAS primarily uses topologically-grouped noise-suppressed clusters (topo-clusters) of calorimeter cells as inputs [9]. Topo-clusters are formed from seed cells with more than 4σ of energy, where σ represents the average noise expected in the cell. Noise

can be of two kinds: electronic and pileup, with pileup noise dominating in current data-taking. All cells adjacent to the seed, provided they have at least 2σ of energy, are then grouped, repeating the process until there are no more suitable cells. In the end, all calorimeter cells adjacent to the topo-cluster are added as well.

2.1.2 Jet algorithms

For jet reconstruction, ATLAS uses the FastJet software library, in particular the anti-k_t algorithm [10]. The algorithm works as follows:

1. For each pair of particles, the k_t distance is calculated as:

$$d_{ij} = \min\left(\frac{1}{k_{ti}^2}, \frac{1}{k_{ij}^2}\right)\frac{\Delta R_{ij}^2}{R^2} \tag{2.1}$$

   where $\Delta R_{ij}^2 = (y_i - y_j)^2 + (\varphi_i - \varphi_j)^2$, with k_{ti}, y_i and φ_i as the transverse momentum, the rapidity, and the azimuth of particles. The beam distance is also calculated as $d_{iB} = \frac{1}{k_{ti}^2}$.

2. The minimum $d_{\min}$ is calculated for all the d_{ij} and d_{iB}. If it is a d_{ij}, the particles i and j are merged; if it is a d_{iB}, the particle i is declared as a final jet.

3. The procedure is repeated for all of the particles.

The distance parameter R is used to distinguish between jets with small R ($R = 0.4$, for jets representing quarks and gluons) and large R ($R = 1.0$, for jets representing hadronically decaying massive particles).

2.1.3 Jet grooming

Grooming refers to a class of algorithms that discard some constituents with a defined strategy. It is only applied to large-R jets to damp the increased sensitivity to pileup effects caused by the larger fraction of the calorimeter enclosed within the jet volume.

2.2 Jet calibration

The need for this step comes from the fact that a large fraction of the hadronic shower energy is not visible to the detector, for example by ending up in volumes without active sensors. Since this energy is not observed, it must be recovered

to calculate the Jet Energy Scale (JES) accurately. As the name suggests, this is usually done by comparing the reconstructed jets and an MC particle simulation, known as "truth jets." The response $\frac{X_{reco}}{X_{truth}}$ is then used to calibrate the average reconstructed jet, usually in fine bins of both pseudorapidity and energy.

Large-R jets ideally have a well-defined mass; therefore, it is useful for these jets to have a mass corresponding to the parent massive particle to help identify and interpret events. Thus, truth is again matched to reconstructed jets, this time with mass response rather than energy, resulting in a so-called Jet Mass Scale (JMS) calibration.

Small-R jets undergo an additional step, applying the Global Sequential Calibration (GSC), which further refines the algorithm by considering shower development and flavor differences. This step does not change the central values of the JES, but rather reduces the differences between different populations.

Besides, known physical processes can be used to perform an *in situ* calibration. This kind of technique provides very precise results: small-R jets have a JES uncertainty of around 1 % while large-R jets have an uncertainty of 1–2 %.

Chapter 3

Boosted Decision Trees

3.1 Supervised learning

The locution "supervised learning" refers to a class of algorithms that learn the function that maps an input to an output from a labeled training sample [11].

3.1.1 Models and parameters

When talking about "models" in supervised learning, one usually refers to the mathematical structure which predicts y_i starting from x_i. The simplest example is the linear model where the prediction is in the form $\hat{y}_i = \sum_j \theta_j x_{ij}$, basically a weighted linear combination of the inputs.

The parameters need to be learned from the training dataset: for example, in linear regression problems, the parameters are the coefficients $\boldsymbol{\theta}$.

3.1.2 The Objective Function

The term "training" refers to the process of finding the best parameters $\boldsymbol{\theta}$ for the training data x_i and labels y_i. The way the model is trained is through the usage of an Objective Function (Obj), which is made up of two parts: the training loss L and the regularization term Ω:

$$\text{Obj}\left(\boldsymbol{\theta}\right) = L\left(\boldsymbol{\theta}\right) + \Omega\left(\boldsymbol{\theta}\right) \tag{3.1}$$

Training loss

The training loss is a measure of the predictivity of the model with respect to the training data. Training loss is often defined as the mean squared error:

$$L(\boldsymbol{\theta}) = \sum_i (y_i - \hat{y}_i)^2 \tag{3.2}$$

Another common alternative is the logistic loss:

$$L(\boldsymbol{\theta}) = \sum_i \left[y_i \log\left(1 + e^{-\hat{y}_i}\right) + (1 - y_i) \log\left(1 + e^{\hat{y}_i}\right) \right] \tag{3.3}$$

Regularization term

The regularization term is a tool to control the model complexity and helps to avoid overfitting. The best way to understand it is by looking at Fig. 3.1.

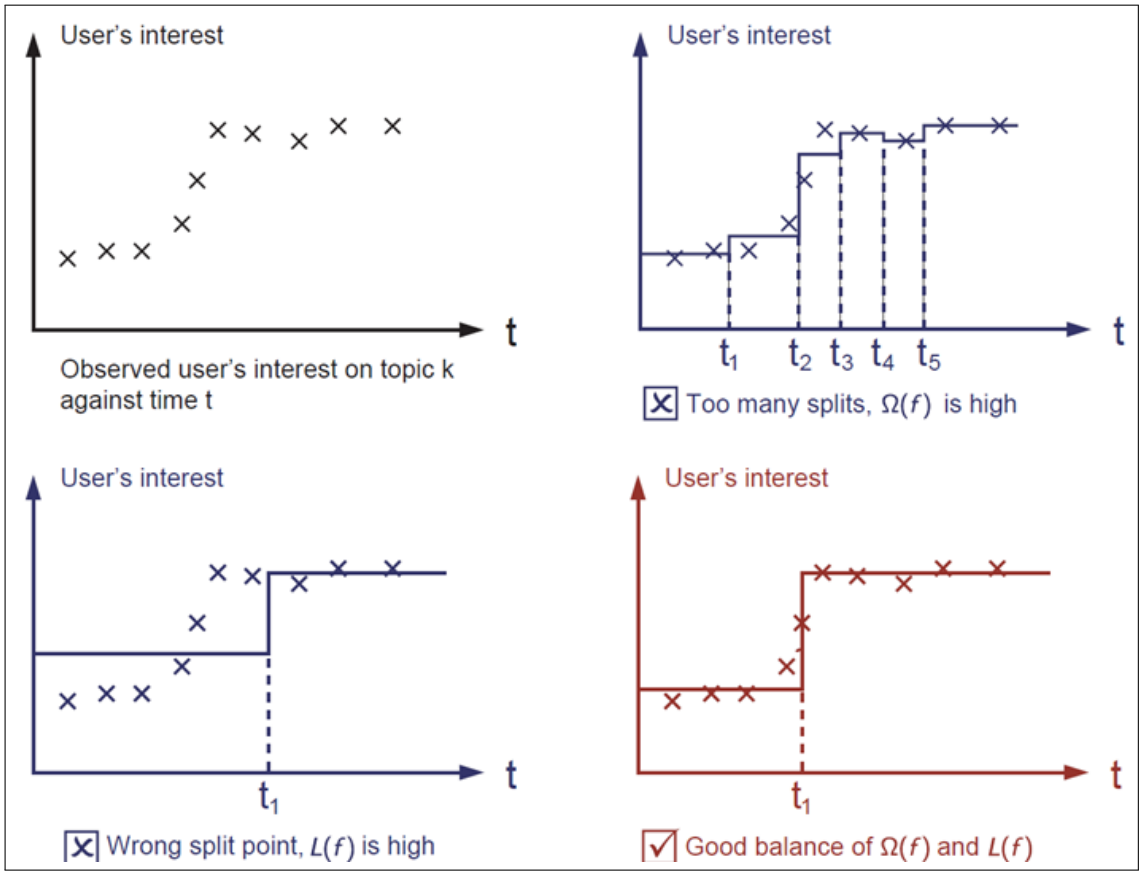

Figure 3.1: Bias-variance tradeoff

The starting data are shown in the top-left graph. The top right plot shows an overtraining, meaning that L is very low, but at the expense of a very high Ω. The bottom left plot shows the opposite situation: Ω is low since only one

split point is used, but the wrong point position results in a high L. Both these situations are not desirable: the first one requires too much computational power, and the model might become poorly predictive on a different dataset; in contrast, the second one gives inaccurate results. The optimal situation is shown in the red plot, representing the ideal balance between Ω and L, resulting in a minimization of the Objective Function. In other words, the model needs to be both simple and predictive, with a balance which is referred to as "bias-variance tradeoff." Of course, this situation is overly simple and is only intended to illustrate the idea. In general, reaching the right equilibrium is not so easy, and it is why Decision Tree algorithms are used.

3.2 Decision tree ensembles

A decision tree ensemble consists of a set of Classification And Regression Trees (CART's) [11]. A very simple example of a CART is represented in Fig. 3.2.

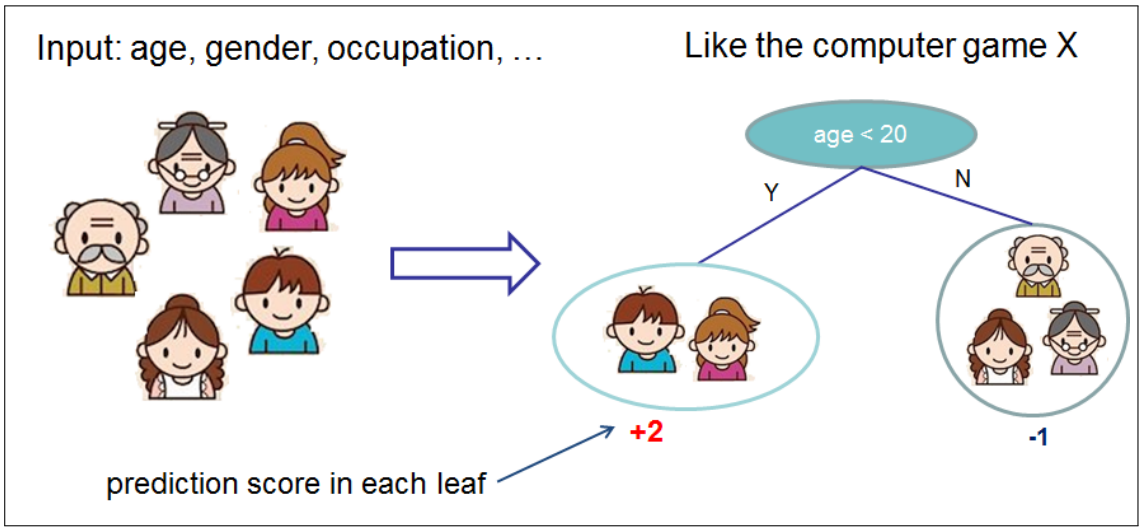

Figure 3.2: A simple CART example

In the example, to find whether a family member will like a computer game, only age is considered: if the age is less than 20 the CART will return a prediction score of 2, returning -1 otherwise. It is clear that a single CART typically results in a weak classifier, particularly inadequate for complex models. A simple way to advance the very primitive model of Fig. 3.2 consists in adding a second CART, as shown in Fig. 3.3. In this case, another variable is taken into consideration: daily computer usage. This tree returns ± 0.9, depending on whether a member of the family uses a computer daily or not. The total prediction score of the CART ensemble is then computed by simply adding together each person's scores.

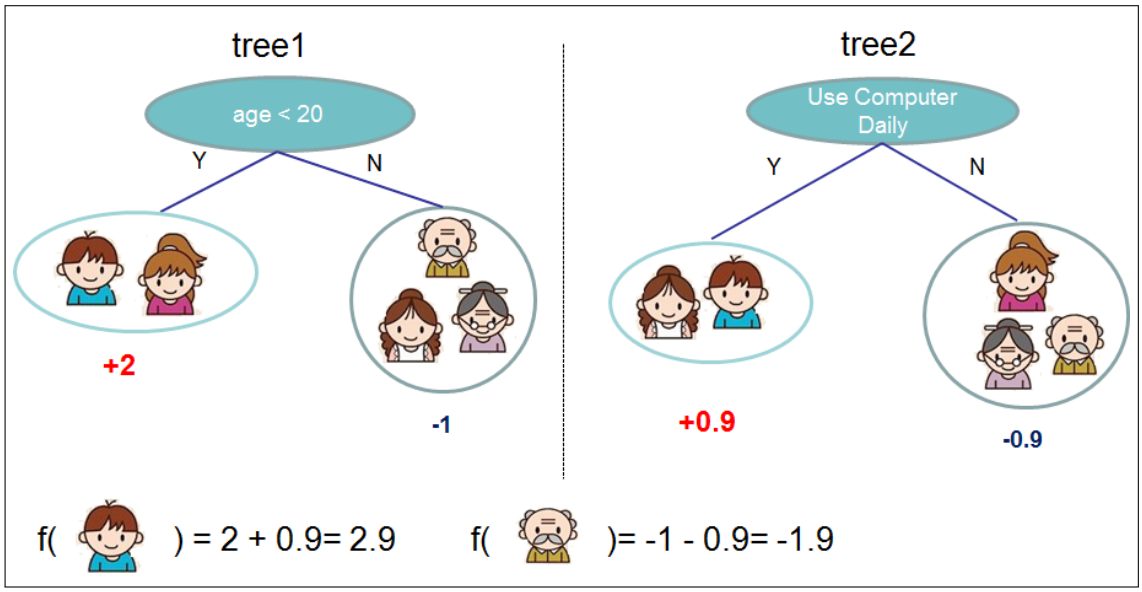

Figure 3.3: A less simple CART example

In a more mathematically precise form:

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), \quad f_k \in \mathcal{F} \tag{3.4}$$

where K is the number of trees, f is a function in the functional space $\mathcal{F}$, and $\mathcal{F}$ is the set of all possible CART's.

3.3 Tree boosting

Tree boosting conceptually happens in a simple way: defining an Objective Function and then optimizing it [11]. Let the Objective Function be in the form:

$$\text{Obj} = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t)}\right) + \sum_{i=1}^{t} \Omega(f_i) \tag{3.5}$$

where n is the number of events in the training sample and t is the number of trees in the ensemble.

3.3.1 Additive training

Ideally, one would like to learn the trees structures, which means the functions f_i, each containing the structure of a tree and the leaf scores. This is, however, much harder than a traditional gradient-based optimization problem. The strategy

which is commonly used is an additive one: adding one tree at a time, keeping fixed what has already been learned. If $\hat{y}_i^{(t)}$ is the prediction value at step t, then:

$$\begin{aligned}
\hat{y}_i^{(0)} &= 0 \\
\hat{y}_i^{(1)} &= f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i) \\
\hat{y}_i^{(2)} &= f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i) \\
&\dots \\
\hat{y}_i^{(t)} &= \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)
\end{aligned} \tag{3.6}$$

The last thing that remains to be decided is which tree to add at each step. The most natural choice appears to be to add the one which optimizes Obj:

$$\begin{aligned}
\text{Obj}^{(t)} &= \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t)}\right) + \sum_{i=1}^{t} \Omega(f_i) \\
&= \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) + \text{constant}
\end{aligned} \tag{3.7}$$

Using for example the mean squared error as the loss function [see Equation (3.2)], Obj becomes:

$$\begin{aligned}
\text{Obj}^{(t)} &= \sum_{i=1}^{n} \left(y_i - \left(\hat{y}_i^{(t-1)} + f_t(x_i)\right)\right)^2 + \sum_{i=1}^{t} \Omega(f_i) \\
&= \sum_{i=1}^{n} \left[2\left(\hat{y}_i^{(t-1)} - y_i\right) f_t(x_i) + f_t(x_i)^2\right] + \Omega(f_t) + \text{constant}
\end{aligned} \tag{3.8}$$

In general, since not all losses are as friendly as the mean squared error, it is useful to take the Taylor expansion of the loss function up to the second order:

$$\text{Obj}^{(t)} = \sum_{i=1}^{n} \left[\left(y_i, \hat{y}_i^{(t-1)}\right) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)\right] + \Omega(f_t) + \text{constant} \tag{3.9}$$

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

After having removed all the constants, the Objective Function at step t becomes:

$$\text{Obj}_{\text{net}}^{(t)} = \sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)\right] + \Omega(f_t) \tag{3.10}$$

3.3.2 Model complexity and tree structure score

Until now the regularization term Ω has only been naively explained, and it is time to introduce a more accurate description. Before doing so, the definition of a tree $f(x)$ must be better formalized:

$$f_t(x) = w_{q(x)},\ w \in R^T,\ q : R^d \to \{1,\ 2,\ \ldots,\ T\} \tag{3.11}$$

where w is the vector of scores, q is a function which assigns each data element to the corresponding leaf, and T is the number of leaves. Complexity can be defined in uncountable ways, here a simple definition will be used only as a demonstration:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \tag{3.12}$$

where γ and λ are additional parameters of the model (hyperparameters).

The Objective Function can then be reformulated as:

$$\begin{aligned} \mathrm{Obj}^{(t)} &\approx \sum_{i=1}^{n} \left[g_i w_{q(x_i)} + \frac{1}{2} h_i w_{q(x_i)}^2 \right] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \\ &= \sum_{j=1}^{T} \left[\left(\sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left(\sum_{i \in I_j} h_i + \lambda \right) w_j^2 \right] + \gamma T \end{aligned} \tag{3.13}$$

where $I_j = \{i | q(x_i) = j\}$ is the set of indices of data assigned to the jth leaf. It can be useful to define $G_j = \sum_{i \in I_j} g_i$ and $H_j = \sum_{i \in I_j} h_i$, so that the Objective Function becomes:

$$\mathrm{Obj}^{(t)} = \sum_{j=1}^{T} \left[G_j w_j + \frac{1}{2} (H_j + \lambda) w_j^2 \right] + \gamma T \tag{3.14}$$

It is easy to see that the optimal values for w_j and Obj are:

$$w_j^* = -\frac{G_j}{H_j + \lambda} \tag{3.15}$$

$$\mathrm{Obj}^* = -\frac{1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T \tag{3.16}$$

An example of how the process works can be seen in Fig. 3.4: the scores are obtained in a way similar to those of Fig. 3.3, but they now take into account the complexity of the model.

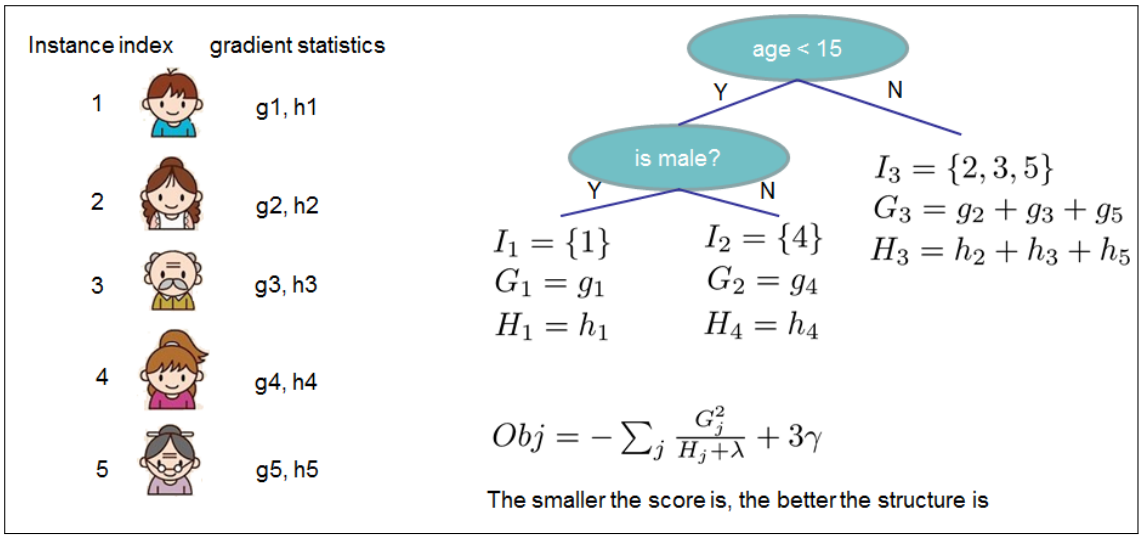

Figure 3.4: An example of score calculation

3.3.3 Learning the tree structure

After having found a way to measure the "goodness" of a tree, the ideal procedure would be to enumerate all possible trees and pick the best one. In practice, the algorithm tries to split each leaf into two leaves and calculates the score gain as:

$$\text{Gain} = \frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right] - \gamma \tag{3.17}$$

where L and R stay for left and right leaf: for example, the first term in the parenthesis represents the left leaf's score, and the second term is the score of the right leaf. The gain equation shows that the new branch should be added if the gain is larger than γ. This kind of technique is known as "pruning." In the simple situation shown in Fig. 3.5, a left to right scan can be used to calculate the scores of all possible splits and decide which position optimizes the problem. This kind of strategy, albeit simple, works pretty well in most of the cases.

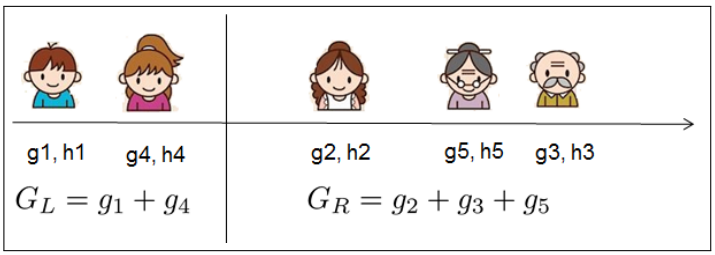

Figure 3.5: Research of optimal split

3.4 LightGBM

LightGBM is a gradient boosting framework which uses tree based algorithms [12]. It aims to offer the following advantages with respect to other competitors such as XGBoost or TMVA:

- Faster training speed and higher efficiency
- Lower memory usage
- Better accuracy
- Support of parallel and GPU learning
- Capability of handling large-scale data

3.4.1 Optimization in speed and memory usage

Contrary to the simple procedure presented in Section 3.3, LightGBM does not implement pre-sort-based algorithms, as they are not easy to optimize. On the contrary, it uses histogram-based algorithms in order to speed up training and reduce memory usage. Advantages of this kind of algorithms include:

- Reduced cost of calculating the gain for each split
  - Pre-sort-based algorithms have time complexity $O(n_{\text{data}})$
  - Histogram computation also has a time complexity $O(n_{\text{data}})$, but this involves only a fast sumup operation; after the histogram construction the algorithm has a time complexity $O(n_{\text{bins}})$, with $n_{\text{bins}} \ll n_{\text{data}}$
- Usage of histogram subtraction for further speedup
  - To get the histograms of one leaf in a binary tree, the histogram subtraction of its parent and its neighbor is used
  - There is the need to construct the histogram for only one leaf (the one with smaller n_{data}); the histogram of the neighbor can then be obtained by histogram subtraction with a small cost of $O(n_{\text{bins}})$
- Reduced memory usage
  - Continuous values are replaced with discrete bins; if n_{bins} is small, a small data type can be used

  - There is no need to store additional information for pre-sorting feature values

- Reduced communication cost for parallel learning

3.4.2 Optimization in accuracy

Leaf-wise tree growth

The majority of tree learning algorithms add leaves by level, as shown in Fig. 3.6. On the contrary, LightGBM uses leaf-wise (also known as best-first) tree growth, as shown in Fig. 3.7. This strategy consists in choosing the leaf with the maximum delta loss to grow: by keeping the number of leaves fixed, algorithms of this kind tend to result in lower loss than level-wise ones. Since leaf-wise may result in overfitting if n_{data} is too small, LightGBM includes a parameter `max_depth` to limit the tree growth in depth.

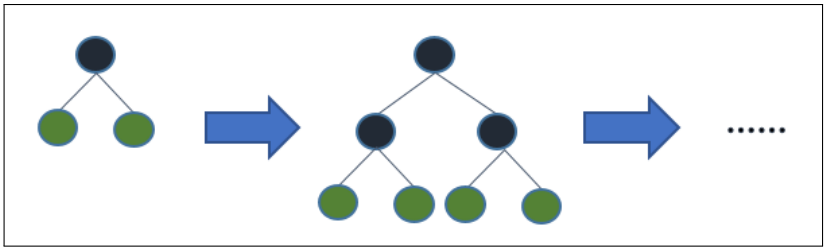

Figure 3.6: Level-wise tree growth

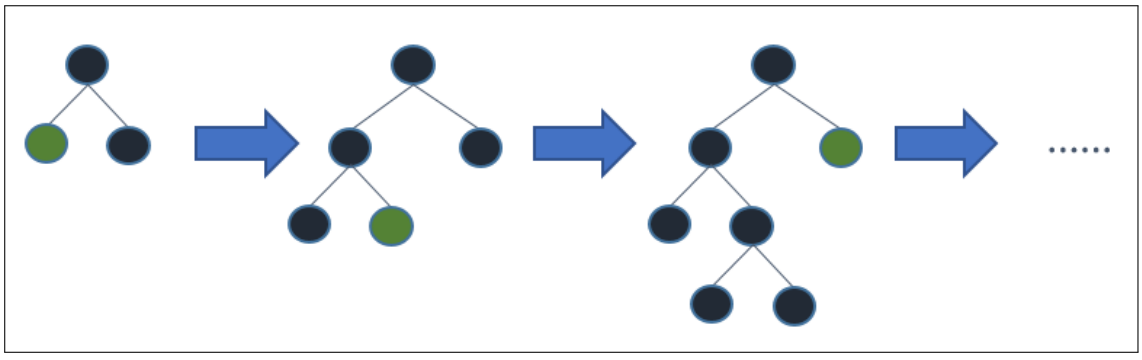

Figure 3.7: Leaf-wise tree growth

Optimal split for categorical features

Categorical features are usually represented with one-hot[1] encoding. However, for high-cardinality categorical features, a one-hot tree needs to grow very deep,

[1] A one-hot is a group of bits among which a single bit is high, and all the others are low.

therefore being suboptimal. A good solution can be partitioning the categories into two subsets: for a feature with k categories, the number of possible partitions is $2^{k-1} - 1$. Then the idea is, at each split, to sort the categories based on the training objective. In particular, LightGBM sorts the histogram according to its accumulated values and then finds the best split on the sorted histogram.

3.4.3 Applications and metrics

LightGBM can be applied in a number of ways:

- Regression
- Binary classification
- Multi classification
- Cross-entropy
- Lambdarank

It also support various metrics:

- L1 loss
- L2 loss
- Log loss
- Classification error rate
- Auc
- Ndcg
- Map
- Multi-class log loss
- Multi-class error rate
- Fair
- Huber
- Poisson
- Quantile

- MAPE
- Kullback-Leibler
- Gamma
- Tweedie

3.4.4 Hyperparameters

Some tuning parameters need to be defined to control the learning: they are known as "hyperparameters," to contrast them with the other parameters derived via training. LightGBM's hyperparameters include the following:

- `objective`: the application chosen, such as regression or binary classification
- `metric`: the metric to be evaluated on the evaluation set
- `boosting`: the kind of boosting used, such as Gradient Boosting Decision Tree or Random Forest
- `num_leaves`: the maximum number of leaves in one tree
- `feature_fraction`: the fraction of features which LightGBM will select on each iteration
- `bagging_fraction`: the fraction of data which LightGBM will select on each iteration
- `min_data_in_leaf`: the minimum number of data in each leaf

Part II

ANALYSIS AND RESULTS

Chapter 4

Jet variables

The study presented here on the classification of pileup and hard-scattering jets is performed using events of Higgs Bosons produced in vector boson fusion process and decaying into four neutrinos, as shown in Fig. 4.1.

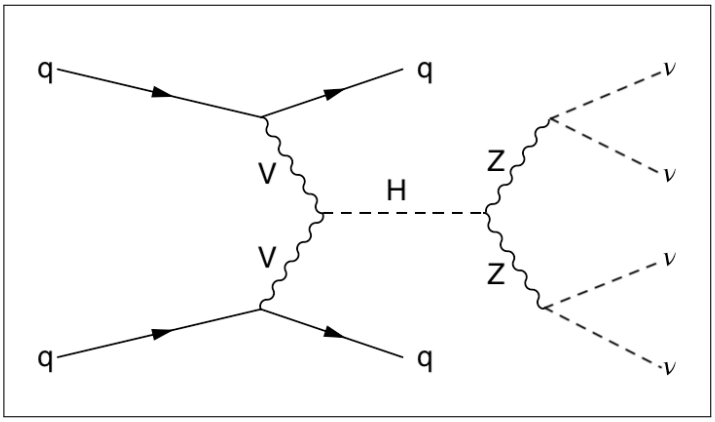

Figure 4.1: Feynman diagram representing the decay of a Higgs boson into four neutrinos

These events are simulated using a Geant4 [13] simulation of the interaction of jets with the detectors, including the simulation of the HGTD detector. The events have been simulated in the HL-LHC conditions overlaying up to 200 pileup interactions. Jets from the hard scattering are considered signal, while jets from secondary interactions are considered background.

A typical classification study always begins by comparing the signal and background features.

4.1 Jet variable distributions

Three kinds of plots will be shown for each variable (where possible): its distribution, a two-dimensional distribution of the variable and of jet p_T (or η), and a plot of the variable average as a function of the transverse momentum, with error bars showing standard deviation.

4.1.1 jet_pt

The first variable to be considered is `jet_pt` (Fig. 4.2). This variable represents the jet's transverse momentum, which is the component of the jet momentum projected on the transverse plane. As shown in Fig. 4.2, on average, signal jets tend to have higher `jet_pt`, since much of the pileup comes from soft collisions.

4.1.2 jet_eta

The next variable is `jet_eta` (Figs. 4.3 to 4.5), which describes the pseudorapidity of the jet. All the plots show the absolute value of the variable. The first thing worth noting in Fig. 4.3 is that the signal tends to be more concentrated at low `jet_eta` values, as expected from the hard-scattering process kinematics. What can be noted, especially from Fig. 4.5, is that the average transverse momentum for signal tends to decrease as `jet_eta` increases, corresponding to smaller angles. On the contrary, it is approximately constant for background jets.

4.1.3 jet_phi

The `jet_phi` variable (Figs. 4.6 to 4.8) is simply the azimuthal angle of the jet in spherical coordinates. Indeed, its distribution is expected to be uniform between $\varphi = -\pi$ and $\varphi = \pi$; moreover, it should not depend on the value of `jet_pt`. Both these expectations are confirmed by Fig. 4.6 and 4.7. An important thing to notice, for Fig. 4.8 and for all the other average plots, is that background jets cover a smaller `jet_pt` range compared to signal ones, since the `jet_pt` spectrum is much softer for these jets.

4.1.4 jet_Rpt

As already mentioned in Section 1.6.2, `jet_Rpt` (Figs. 4.9 to 4.11) is defined as:

$$R_{p_{\mathrm{T}}} = \frac{\sum p_{\mathrm{T}}^{\mathrm{trk}}\left(\mathrm{PV}_0\right)}{p_{\mathrm{T}}^{\mathrm{jet}}} \tag{4.1}$$

Since it is designed to distinguish signal and background jets efficiently, its distributions should be very different for the two data groups. This is shown in Fig. 4.9: while the signal distribution is quite flat with a maximum at around $R_{p_T} = 0.5$, the background data are very strongly peaked at $R_{p_T} = 0$. This is also visible in Fig. 4.10, where the bottom background plot is almost white outside of the bottom bin. In Fig. 4.11 the averages of `jet_Rpt` are shown in function of `jet_pt` and what can be seen is that the `jet_Rpt` does not depend on `jet_pt`.

4.1.5 jet_t0comp

The `jet_t0comp` variable (Figs. 4.12 to 4.14) is defined as:

$$t_0^{\text{comp}} = \frac{\sum_i p_{T,i} \frac{\sqrt{\sigma_{\text{lead}}^2 + \sigma_i^2}}{|t_{\text{lead}} - t_i|}}{\sum_i p_{T,i}} \tag{4.2}$$

where t_i are the timing information for each track i of the event, t_{lead} is the timing information of the leading (highest p_T) track, and σ_i and σ_{lead} are the corresponding uncertainties. This variable is expected to have a good discriminating power, since it compares the leading track time with a sort of t_0 (the primary collision time): if the difference is small, which means a high `jet_t0comp` value, it is probable that the jet comes from the hard-scatter collision.

4.1.6 jet_sign

`jet_sign` (Figs. 4.15 to 4.17) is another time-related variable, defined as:

$$\text{Sign} = \frac{|t_{\text{lead}} - t_{\text{sublead}}|}{\sqrt{\sigma_{\text{lead}}^2 + \sigma_{\text{sublead}}^2}} \tag{4.3}$$

where t_{lead} and t_{sublead} are the timing information of the leading (highest p_T) and subleading (second highest p_T) tracks, and σ_{lead} and σ_{sublead} are the corresponding uncertainties. This variable is expected be a good discriminating feature, since it basically measures the difference between the times of the leading and the subleading tracks: if these times are very different, it is probable that the tracks come from different (secondary) vertices.

4.1.7 jet_NtrkTime

The last variable which has been studied is `jet_NtrkTime` (Figs. 4.18 to 4.20), which represents the number of tracks with timing information in the jet. It appears

clear from Fig. 4.18 that the distributions for signal and background are extremely different. Moreover, as shown in Fig. 4.20, there seems to be a quasi-linear correlation between `jet_NtrkTime` and `jet_pt`, meaning that excluding `jet_pt` from the training variables might potentially result in some loss in discrimination power.

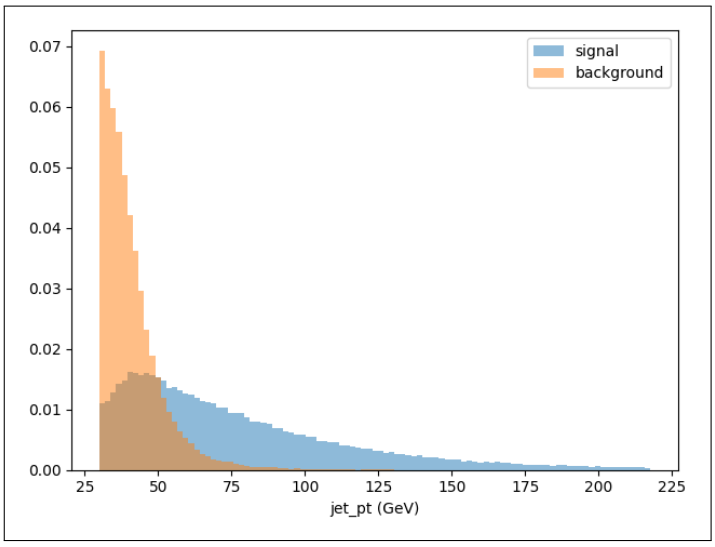

Figure 4.2: Distributions of `jet_pt` for signal and background

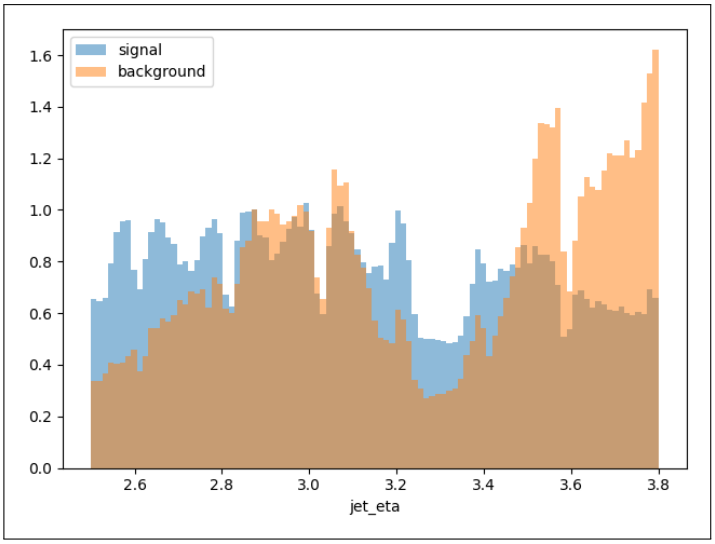

Figure 4.3: One-dimensional distributions of `jet_eta` for signal and background

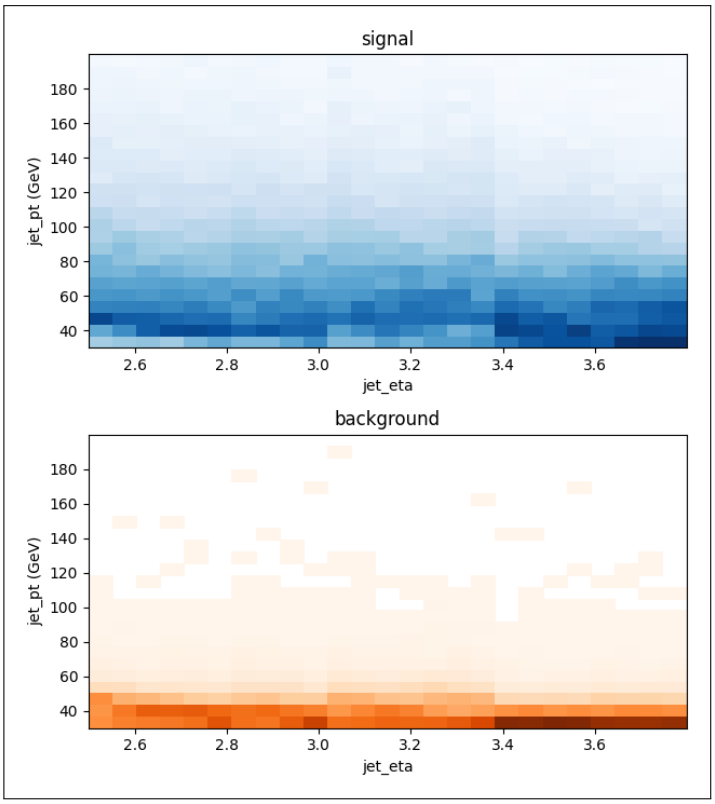

Figure 4.4: Two-dimensional distributions of `jet_eta` for signal and background

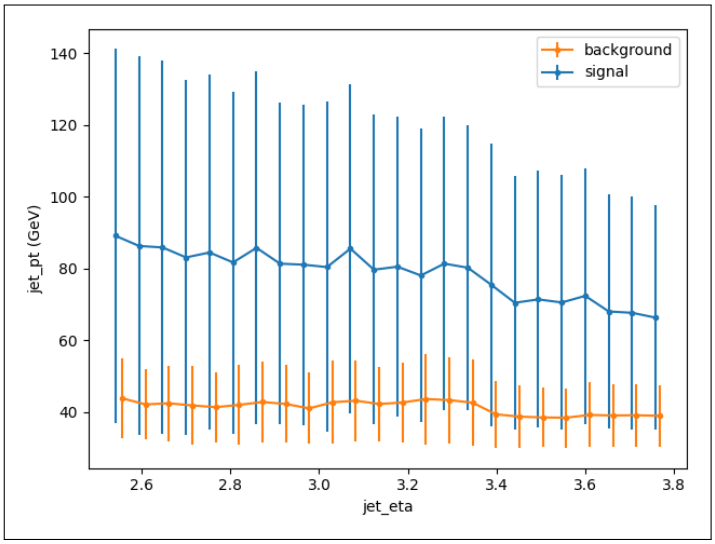

Figure 4.5: `jet_pt` as a function of `jet_eta` for signal and background. On the y axis, the `jet_pt` averages for each `jet_eta` bin are shown, with error bars representing standard deviation.

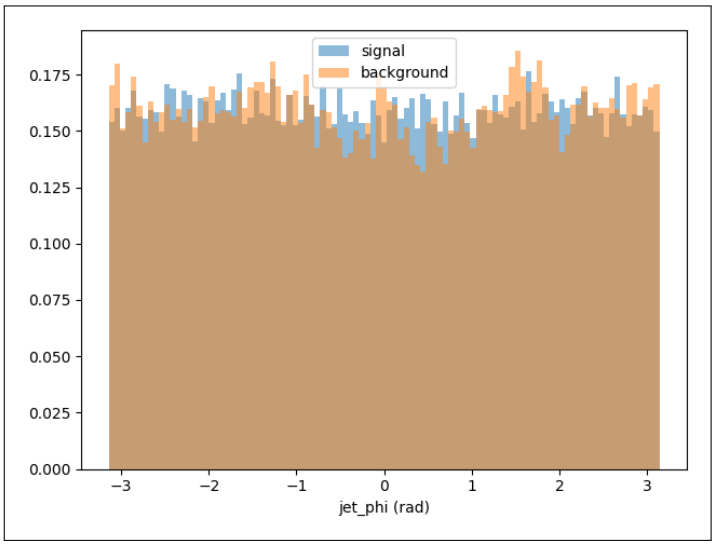

Figure 4.6: One-dimensional distributions of `jet_phi` for signal and background

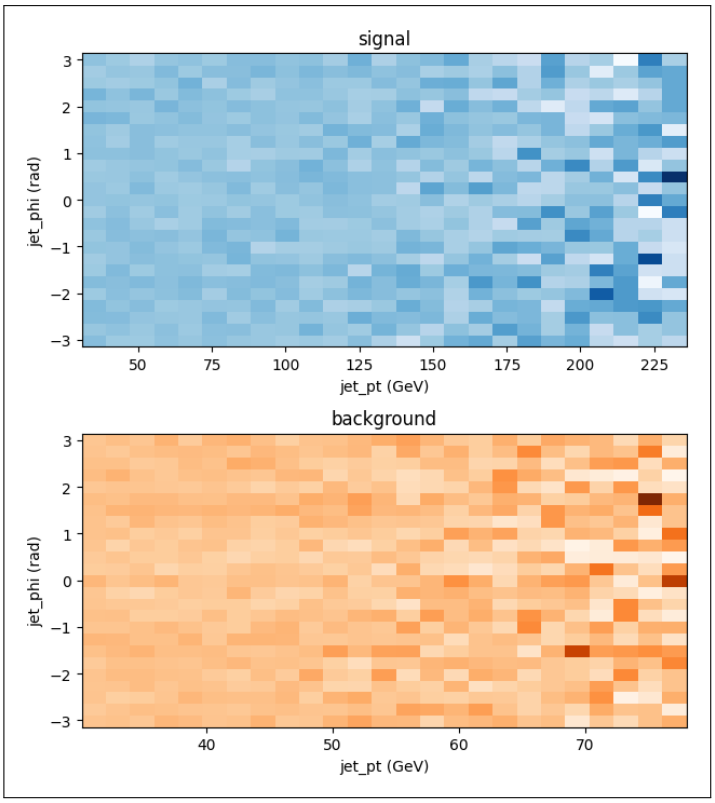

Figure 4.7: Two-dimensional distributions of `jet_phi` for signal and background

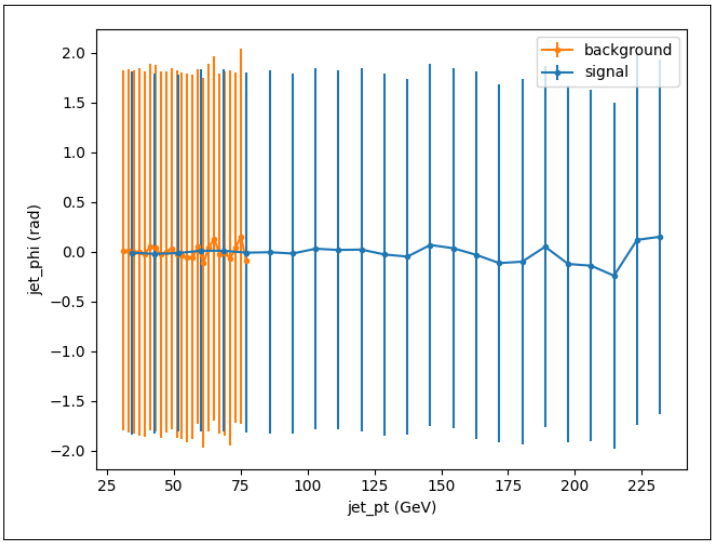

Figure 4.8: `jet_phi` as a function of `jet_pt` for signal and background. On the y axis, the `jet_phi` averages for each `jet_pt` bin are shown, with error bars representing standard deviation.

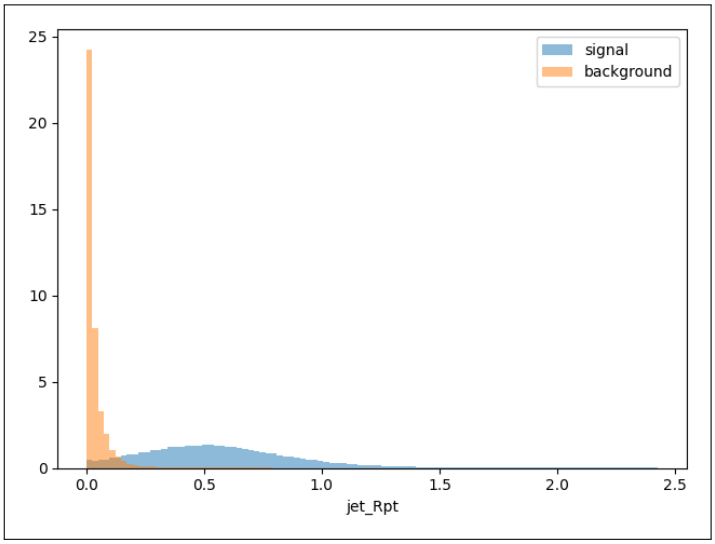

Figure 4.9: One-dimensional distributions of `jet_Rpt` for signal and background

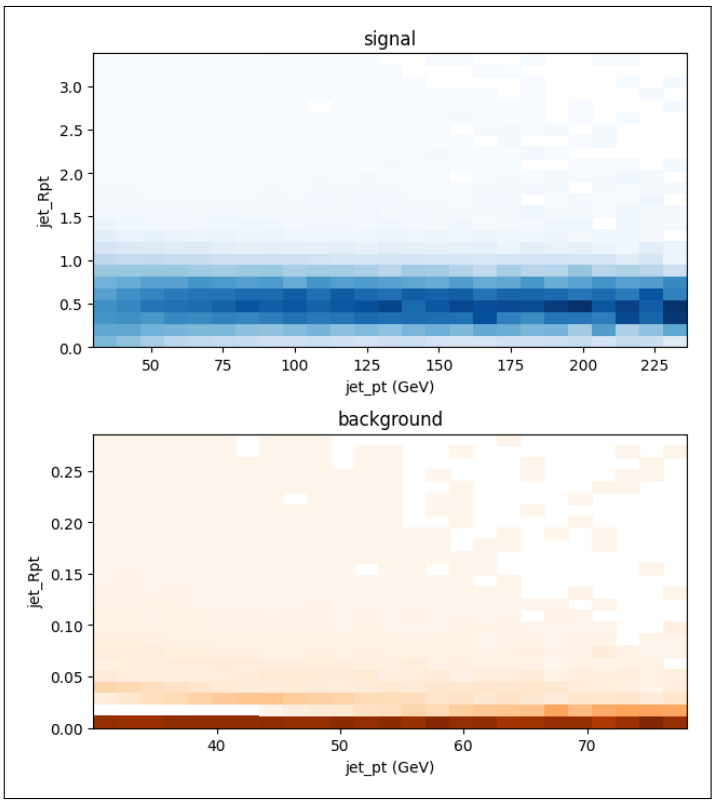

Figure 4.10: Two-dimensional distributions of `jet_Rpt` for signal and background

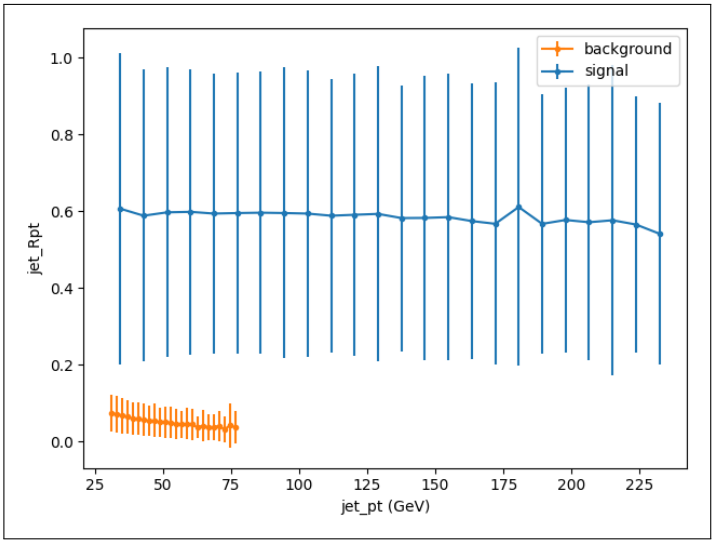

Figure 4.11: `jet_Rpt` as a function of `jet_pt` for signal and background. On the y axis, the `jet_Rpt` averages for each `jet_pt` bin are shown, with error bars representing standard deviation.

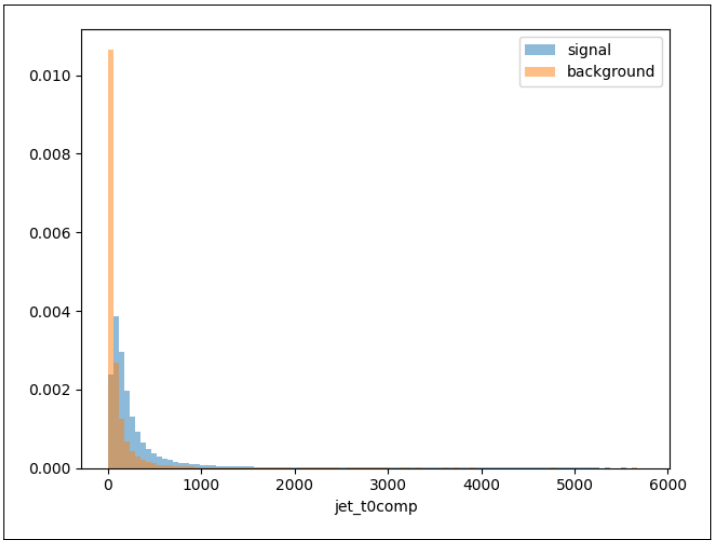

Figure 4.12: One-dimensional distributions of `jet_t0comp` for signal and background

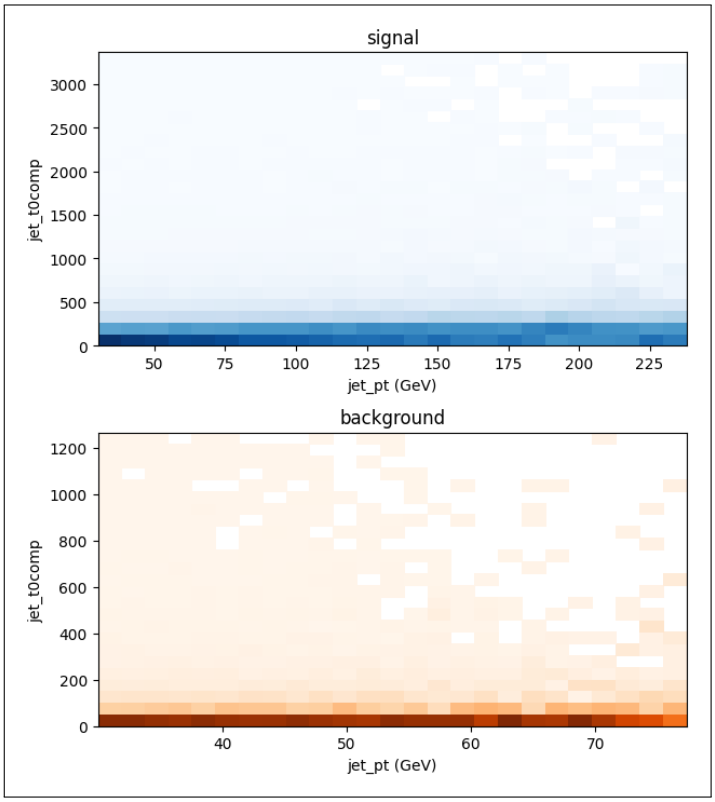

Figure 4.13: Two-dimensional distributions of `jet_t0comp` for signal and background

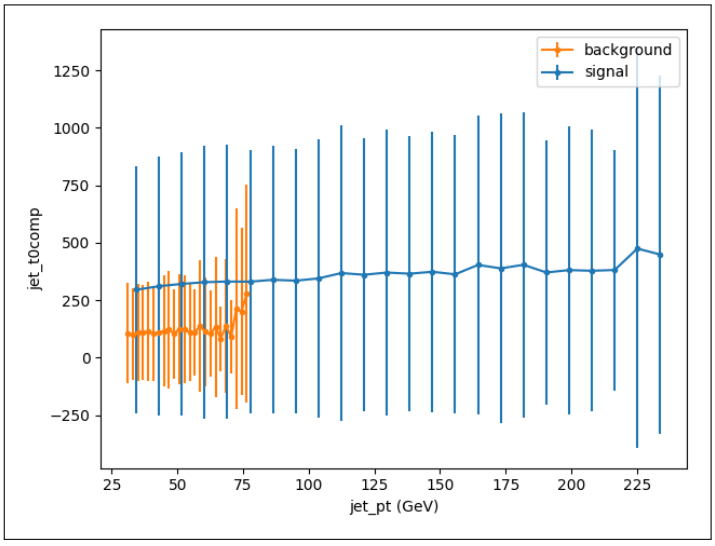

Figure 4.14: `jet_t0comp` as a function of `jet_pt` for signal and background. On the y axis, the `jet_t0comp` averages for each `jet_pt` bin are shown, with error bars representing standard deviation.

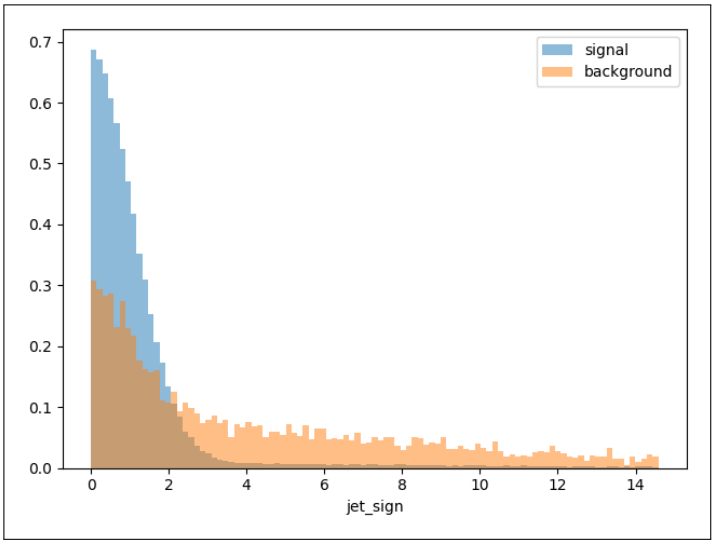

Figure 4.15: One-dimensional distributions of `jet_sign` for signal and background

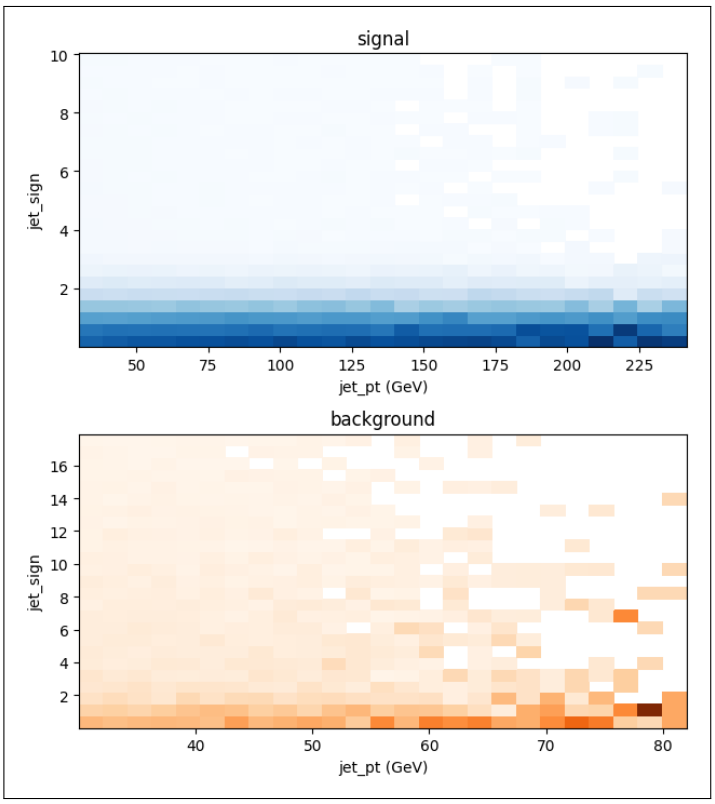

Figure 4.16: Two-dimensional distributions of `jet_sign` for signal and background

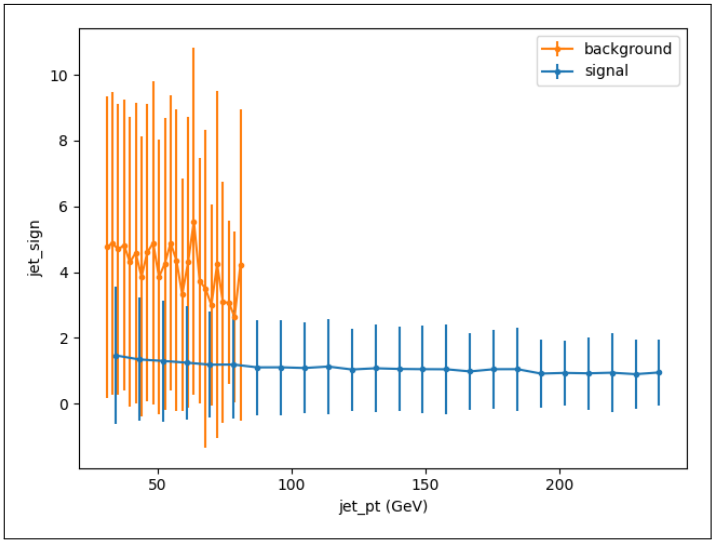

Figure 4.17: `jet_sign` as a function of `jet_pt` for signal and background. On the y axis, the `jet_sign` averages for each `jet_pt` bin are shown, with error bars representing standard deviation.

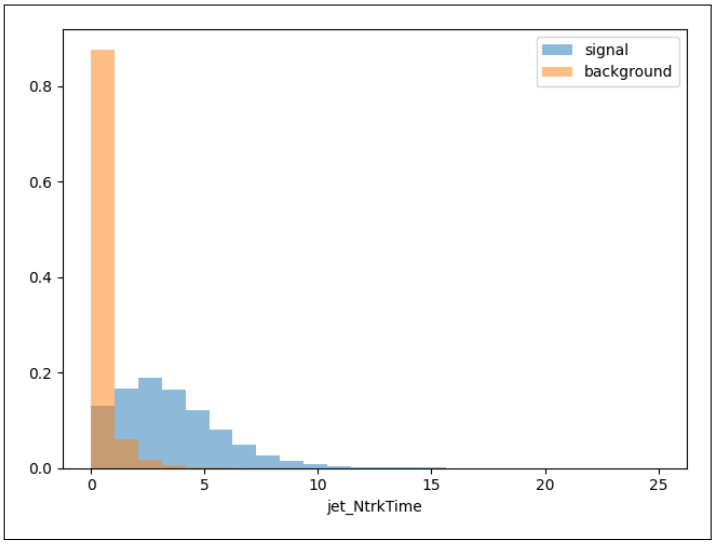

Figure 4.18: One-dimensional distributions of `jet_NtrkTime` for signal and background

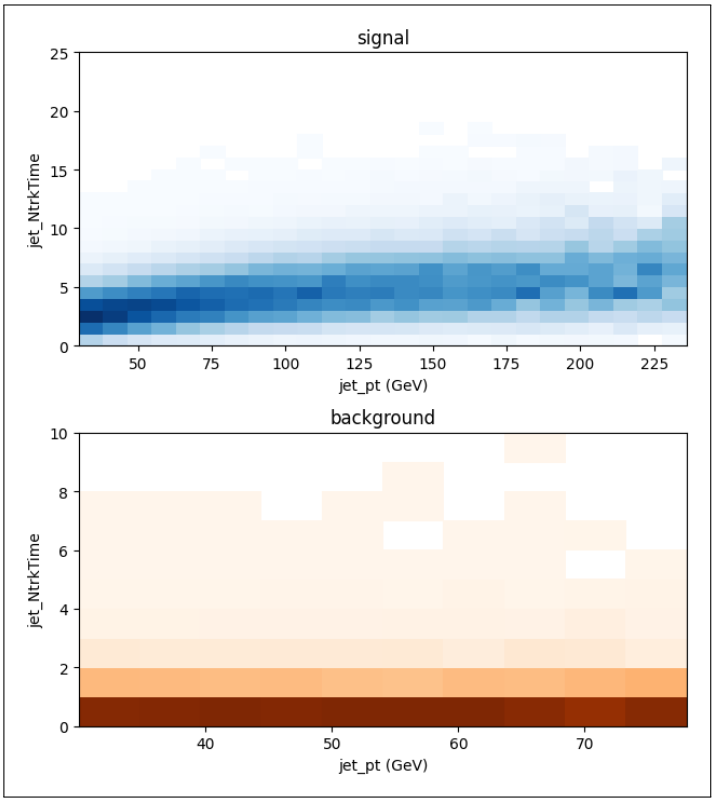

Figure 4.19: Two-dimensional distributions of `jet_NtrkTime` for signal and background

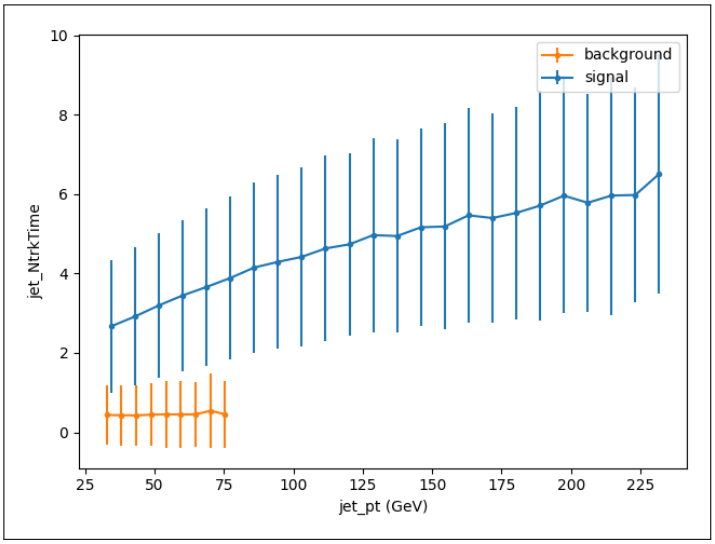

Figure 4.20: `jet_NtrkTime` as a function of `jet_pt` for signal and background. On the y axis, the `jet_NtrkTime` averages for each `jet_pt` bin are shown, with error bars representing standard deviation.

4.2 BDT training

A first classification using the BDT implementation of LightGBM was performed with different combinations of `jet_Rpt`, `jet_t0comp`, `jet_sign`, and `jet_NtrkTime`. `jet_pt` and `jet_eta` were not included because they strongly depend on the physical process in question, therefore their usage might result in some generality loss for the algorithm.

First of all, all the graphs shown are repeated for two p_T ranges of the jets, 30–50 GeV and 50–80 GeV, since the separation power of the variables vary for different p_T ranges. Out of the 90 695 jets for the low p_T range (35 759 signal and 54 936 background) and the 51 732 jets for the high p_T range (43 488 signal and 8244 background), a random 50 % was used for training the algorithm and the other 50 % for testing. The hyperparameters cited in Section 3.4.4 were also optimized using the `optuna` package [14], as shown in Section A.2.1.

As discussed in Chapter 3, the training is performed optimizing the Objective Function. The loss part of the Objective Function is represented in Fig. 4.21 as a function of the number of iterations. Since it is used mainly to check the algorithm's correct functioning, only one representative plot will be shown; all the others are qualitatively identical. When the training does not encounter any improvement on the test set for 100 rounds, as shown in Fig. 4.21, the algorithm stops and records the current results.

In Figs. 4.22 and 4.23 the relative importance of all the features is shown in terms of the number of times a feature is used in a tree split. Even though this graph is created for each of the variable combinations, only the one with all the variables used is shown, as it is the most significant to understand the importance of the features.

As expected, `jet_Rpt` is the most important one, with `jet_t0comp`, `jet_sign`, and `jet_NtrkTime` progressively less useful: therefore, adding `jet_t0comp` to the situation where only `jet_Rpt` is used will likely have more impact than adding `jet_NtrkTime` to the case where the other three features are already considered.

Fig. 4.24 shows a one-dimensional histogram of the predictions distributions: again, only one graph is shown as an example, as the other ones are qualitatively similar.

4.2.1 ROC curves

A Receiver Operating Characteristic (ROC) curve is a way to represent the diagnostic ability of a binary classifier graphically by plotting each data point with an

x value corresponding to the false positive rate and a y value corresponding to the true positive rate. These points are obtained by imposing a selection on the BDT output and then integrating the distributions for signal and background from the value to the right and left ends. In other words, if the classification is perfect, the graph will be perfectly vertical going from 0 to 1 on the y axis and then perfectly horizontal going from 0 to 1 on the x axis; on the other hand, if the classification is completely random the graph will go up and right diagonally. An example of a generic ROC curve is shown in Fig. 4.25. For the current work, though, the true positive rate is represented on the x axis, corresponding to the efficiency for signal jets, and the reciprocal values of the false positive rates are shown on the y axis, corresponding to the rejection of background jets (Fig. 4.26).

To conclude, the Area Under the Curve (AUC) values are shown in Table 4.1: it is clear from the definition of the ROC curve that a perfect graph will have an AUC value of 1, therefore this parameter is a good indicator of the quality of the BDT. As expected, the values for the high p_T range are higher.

Features	$p_T < 50\,\text{GeV}$	$p_T > 50\,\text{GeV}$
`Rpt`	0.972	0.983
`Rpt+t0comp`	0.974	0.984
`Rpt+t0comp+sign`	0.974	0.984
`Rpt+t0comp+sign+NtrkTime`	0.974	0.984

Table 4.1: The AUC values for trainings using jet variables

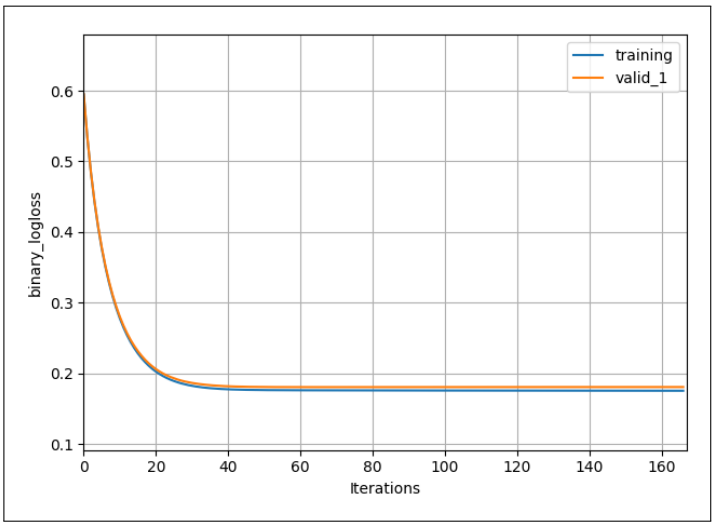

Figure 4.21: The loss part of the Objective Function as function of the number of iterations

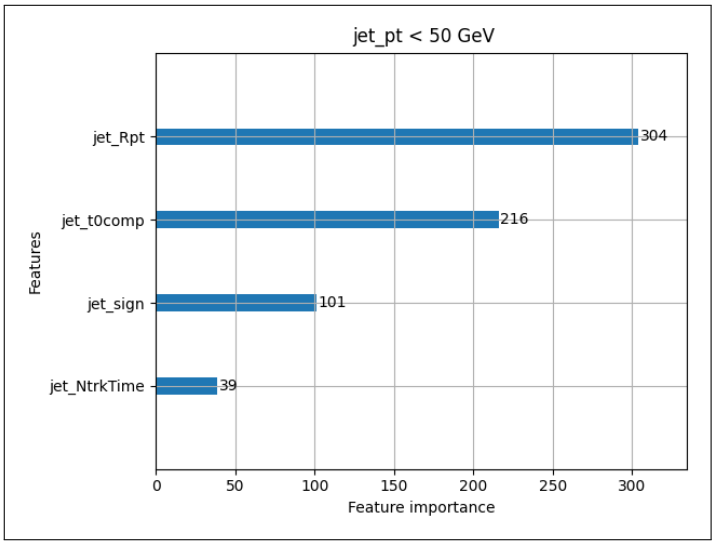

Figure 4.22: Number of splits associated with each feature for the low p_T range

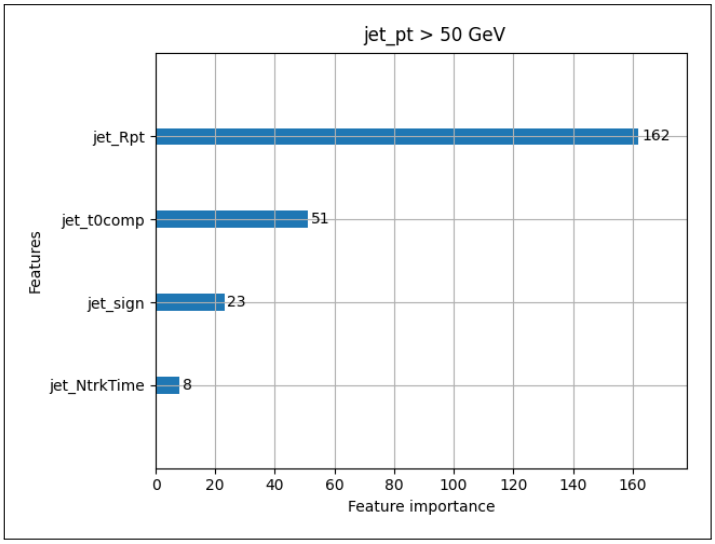

Figure 4.23: Number of splits associated with each feature for the high p_T range

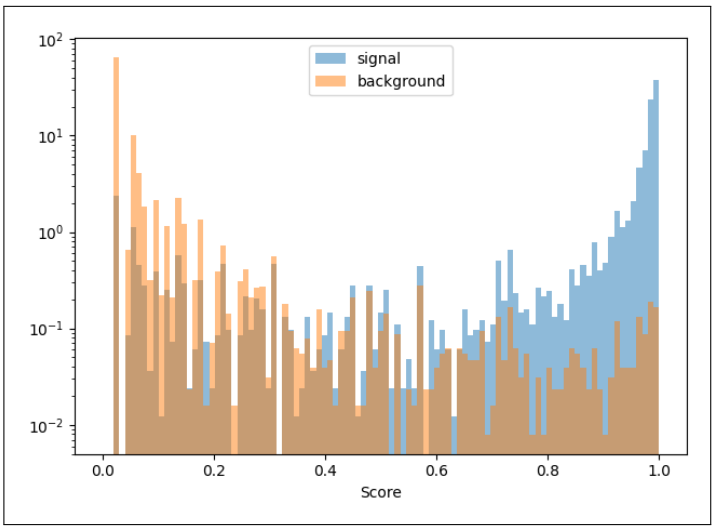

Figure 4.24: One of the prediction distributions (low p_T range)

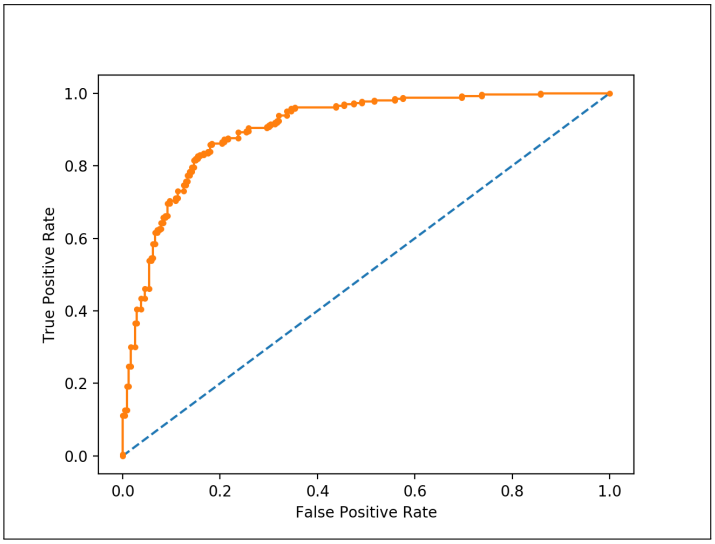

Figure 4.25: An example of a generic ROC curve

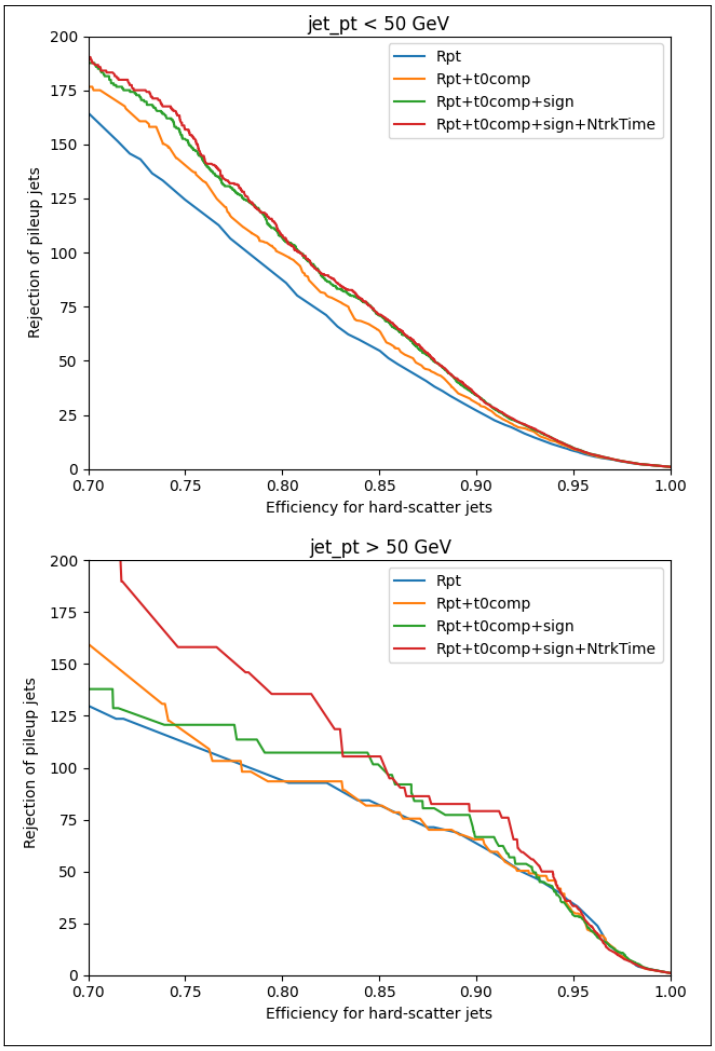

Figure 4.26: The ROC curves for trainings using jet variables

Chapter 5

Track variables

After having checked the classification performance using high-level jet variables, the potential of individual track information usage is explored. Of course, not all jets have the same number of tracks when the data are recorded. Usually, a preliminary filter, checking if some standard conditions regarding the space variables are fulfilled, is used, resulting in a decrease in most of the jets' track number.

5.1 Track variable distributions

5.1.1 Ntrks

The distributions of the number of tracks for signal and background jets are shown in Fig. 5.1: as expected, signal jets tend to have a higher number of tracks per jet. For each jet, the tracks are first ordered according to p_T, then for each track a set of features is recorded. Since showing all the distributions would be redundant, only the distributions for each jet's first track will be shown.

5.1.2 trk_pt

This variable represents the transverse momentum of the track. As expected, the distribution of `trk_pt` is very different between signal and background, following the different `jet_pt` spectrums shown in Fig. 4.2.

Since the `trk_pt` distribution might depend on the physical process chosen, it will not be used as such in the BDT training, but as a normalized `trk_pt_norm` (Fig. 5.3), defined as:

$$p_T^{\text{norm}} = \frac{p_T^{\text{track}}}{p_T^{\text{jet}}} \tag{5.1}$$

5.1.3 trk_eta

The `trk_eta` is the pseudorapidity of the tracks. Again, the absolute values are shown in Fig. 5.4. Qualitatively, the distribution of `trk_eta` is similar to that of `jet_eta`.

5.1.4 trk_phi

Being `trk_phi` the azimuthal angle of the track, it is expected to be uniformly distributed, as confirmed in Fig. 5.5.

5.1.5 trk_d0

The `trk_d0` variable (Fig. 5.6) represents the transverse impact parameter, which is the projection of the impact parameter on the transverse plane. As one may expect, the signal tracks, being on average more energetic, have a narrower distribution than the softer background tracks.

5.1.6 trk_z0

The `trk_z0` variable (Fig. 5.7) represents the longitudinal impact parameter, which is the projection of the impact parameter on the longitudinal direction. Contrary to `trk_d0`, this variable is distributed slightly more narrowly for background tracks.

5.1.7 trk_time

The last variable analyzed for each track is `trk_time`, which provides information about the time at which each track is detected. The values in Fig. 5.8 do not represent the absolute time values, but rather the differences between the time of the track and the t_0 of the event. The t_0 is an event variable that represents the time when the collision of interest takes place: in order to obtain it, a different BDT is run, and the event time is computed.

As one might expect, the signal jets tend to have a `trk_time` closer to 0 (which means an absolute time closer to t_0) compared to background jets, since a significant amount of background jets do not originate from the primary collision vertex but from secondary collisions, which usually happen at a different t_0.

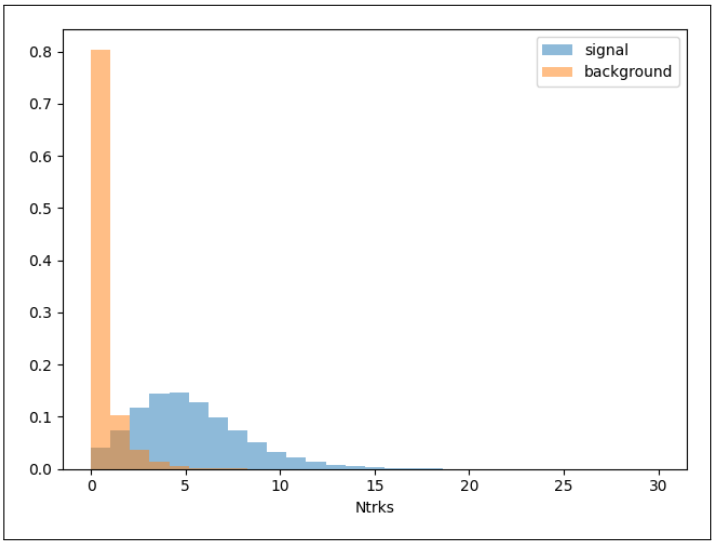

Figure 5.1: Distributions of `Ntrks` for signal and background

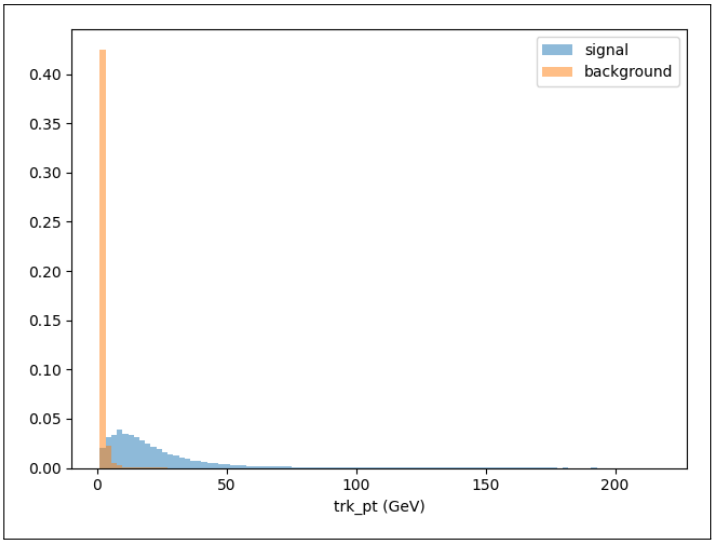

Figure 5.2: Distributions of `trk_pt` for signal and background

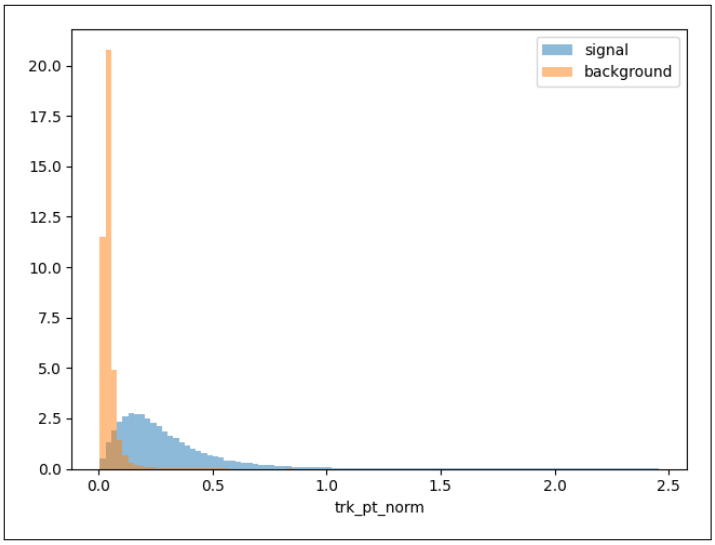

Figure 5.3: Distributions of `trk_pt_norm` for signal and background

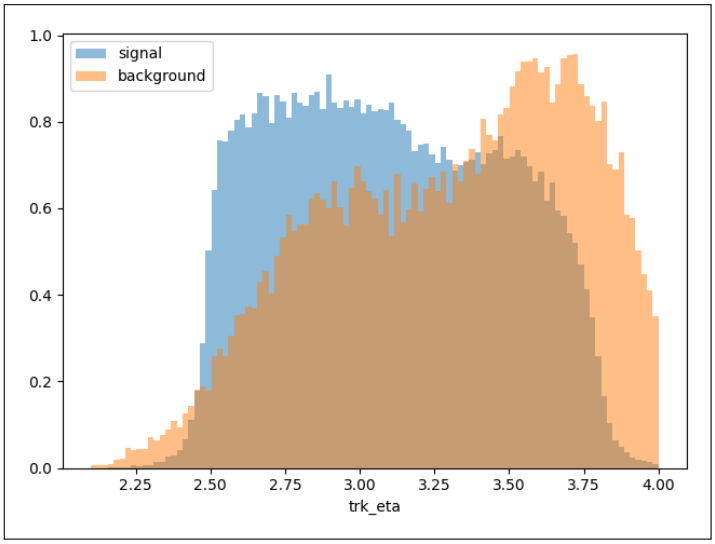

Figure 5.4: Distributions of `trk_eta` for signal and background

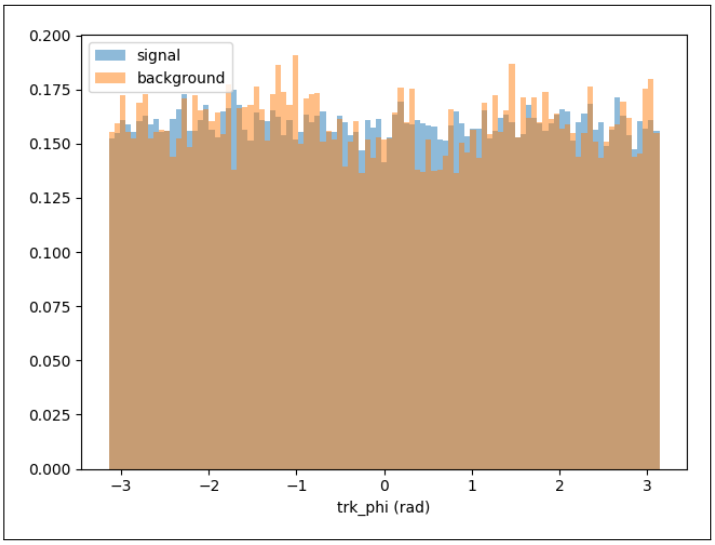

Figure 5.5: Distributions of `trk_phi` for signal and background

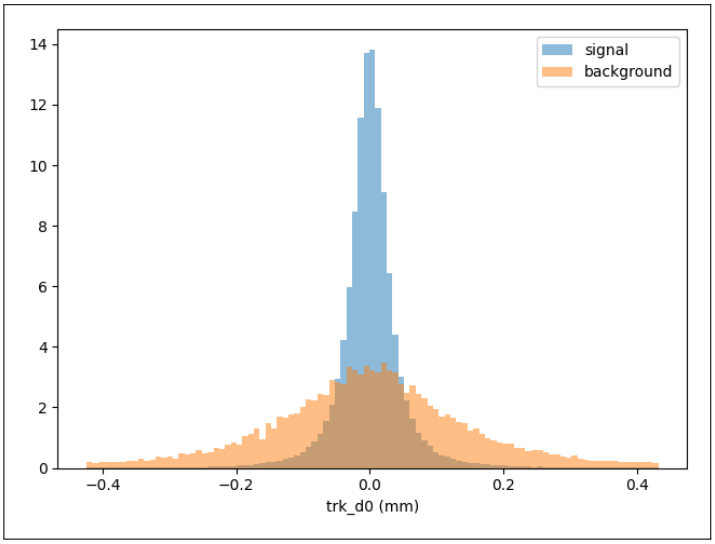

Figure 5.6: Distributions of `trk_d0` for signal and background

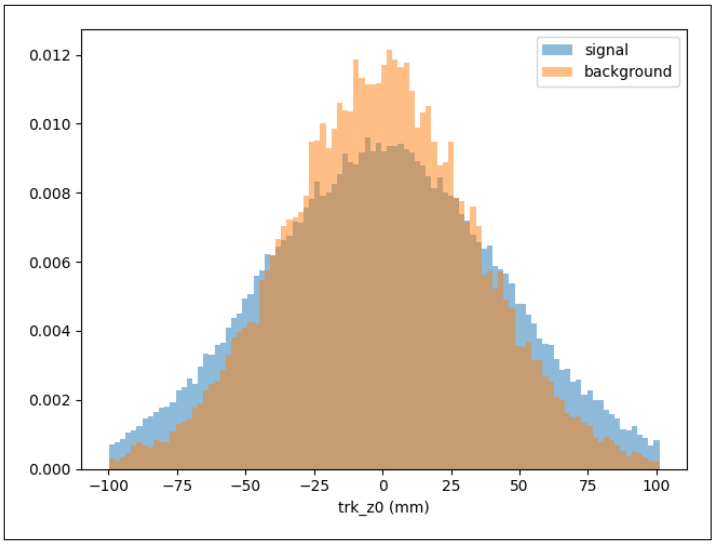

Figure 5.7: Distributions of `trk_z0` for signal and background

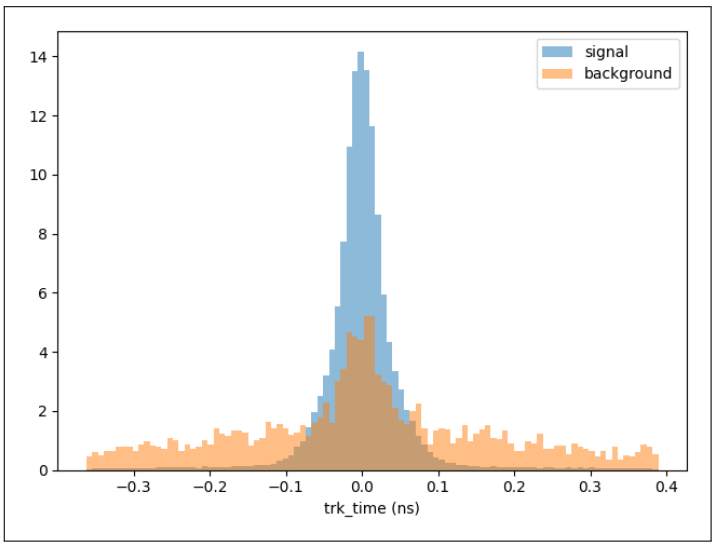

Figure 5.8: `trk_time`

5.2 BDT training

After having explored the BDT training using only high-level variables and having studied the performance of the configurations with `Rpt`, `t0comp`, `sign` and `NtrkTime`, the next step consists in exploring the potential of using low-level track variables directly in the training. As a first default configuration, `trk_pt_norm`, `trk_phi`, `trk_d0`, and `trk_z0` are used.

5.2.1 Effect of trk_eta

The first thing that one wants to check is the impact of `trk_eta` on the BDT performance. It is crucial to evaluate the importance of `trk_eta` because it is a cinematic variable whose distributions for signal and background might depend on the physical process chosen. Therefore, if the BDT relies too heavily on it, it means that its application to other physical processes may result in suboptimal performance.

Two different setups will be used: the first one includes the variables `trk_pt_norm`, `trk_phi`, `trk_d0`, and `trk_z0`; the second one uses the same ones adding `trk_eta` as well. For both configurations, the first ten tracks will be taken into consideration.

As shown in Fig. 5.9 and in Table 5.1, even though `trk_eta` actually helps in training an efficient BDT, its impact is not particularly big. Therefore, considering the potential bias that using `trk_eta` would introduce, it is safe to assume that it is better not to include it while training the BDT.

Features	$p_T < 50\,\text{GeV}$	$p_T > 50\,\text{GeV}$
Without `trk_eta`	0.976	0.986
With `trk_eta`	0.976	0.986

Table 5.1: The AUC values for trainings using low-level variables, with and without using `trk_eta`

5.2.2 Effect of the number of tracks

After having decided which variables to use, the optimal number of tracks has been investigated. By looking at Fig. 5.1, it can be noted that most of the jets have at least five tracks, but only a few arrive at ten. Therefore, it may seem reasonable to cut some of the tracks in order to simplify the algorithm. The BDT is therefore tested using the information from a maximum of five or ten tracks per jet.

It is clear from Fig. 5.10 and from Table 5.2 that using five or ten tracks does not make much difference, as one may expect by looking at the `Ntrks` variable distribution. In order to make the algorithm lighter, only the first five tracks will then be used.

Features	$p_T < 50\,\text{GeV}$	$p_T > 50\,\text{GeV}$
5 tracks	0.976	0.986
10 tracks	0.976	0.986

Table 5.2: The AUC values for trainings using low-level variables, with five and ten tracks

5.2.3 Timing information

The final step consists in adding track-level timing information (`trk_time`) to the optimal track configuration found in the previous sections.

As shown in Fig. 5.11, the addition of timing information actually improves the BDT performance, as expected. This improvement is shown in Table 5.3 as well.

Features	$p_T < 50\,\text{GeV}$	$p_T > 50\,\text{GeV}$
Without `trk_time`	0.976	0.986
With `trk_time`	0.977	0.987

Table 5.3: The AUC values for trainings using low-level variables, with and without timing information

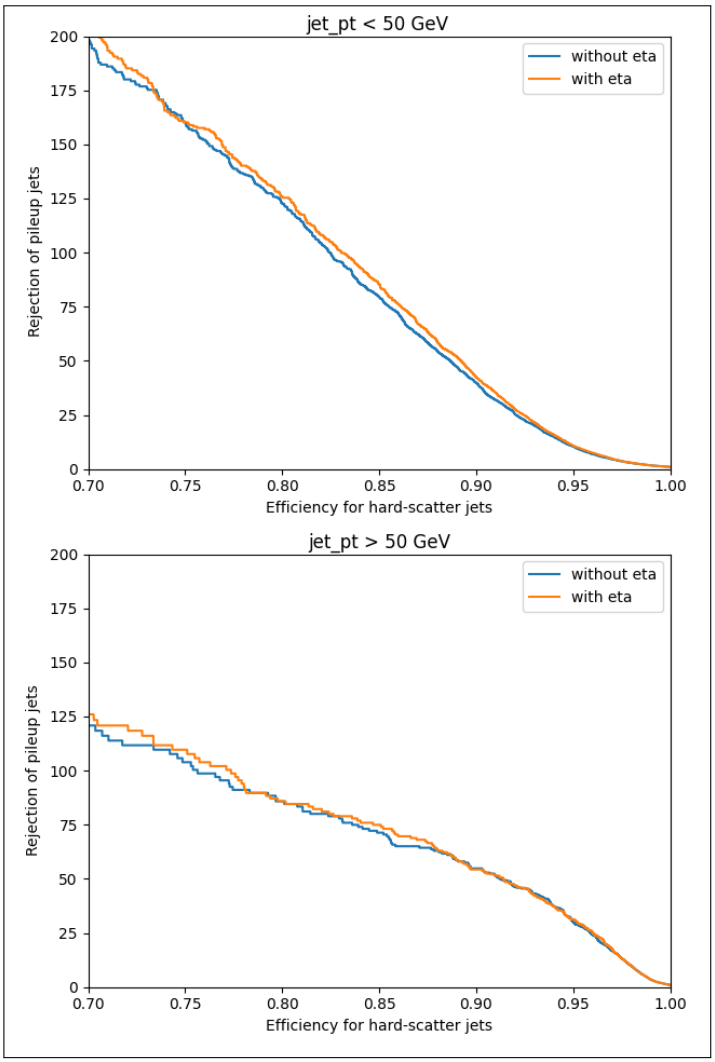

Figure 5.9: The ROC curves for trainings with and without using `trk_eta`

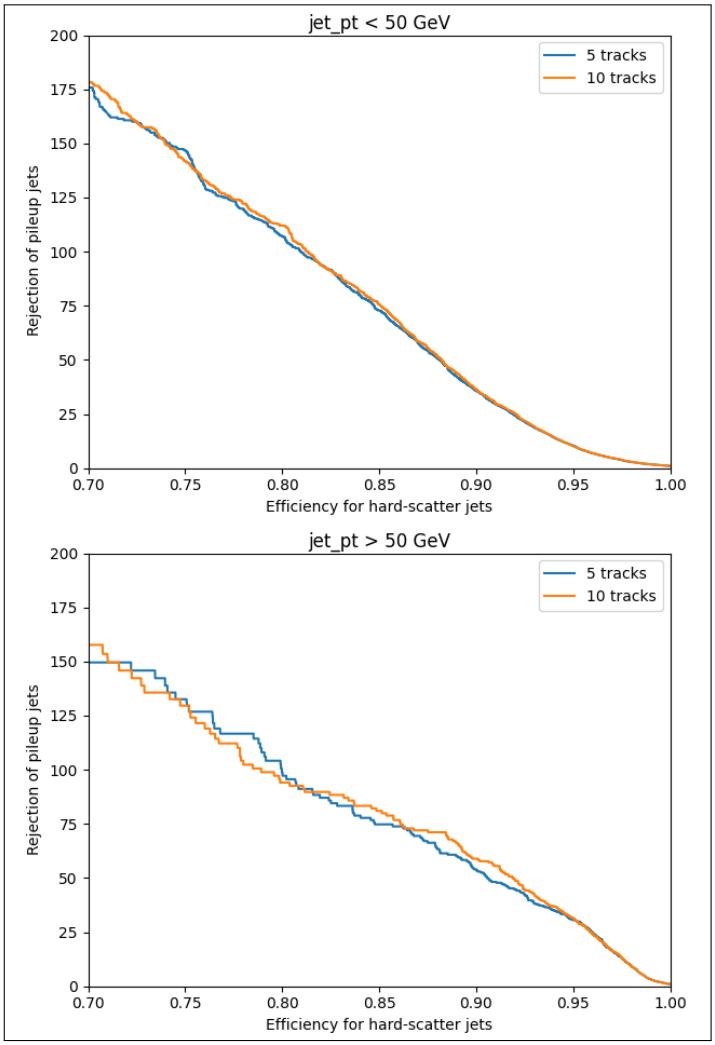

Figure 5.10: The ROC curves for trainings with five and ten tracks

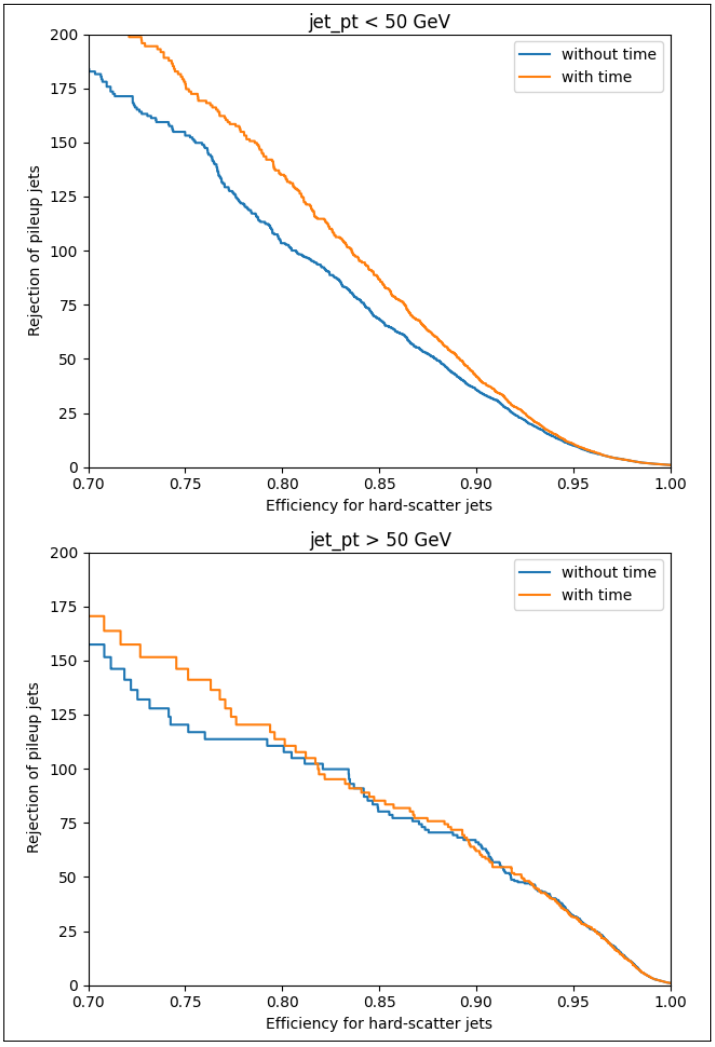

Figure 5.11: The ROC curves for trainings with and without timing information

Chapter 6

Conclusion

This thesis's aims were mainly two: studying if using track information in the BDT training may improve its performance compared to the case when only high-level variables are used and exploring the impact of the usage of timing information deriving from the HGTD detector.

In order to answer to these questions, Fig. 6.1 and Table 6.1 are studied: the results for the best case with high-level variables only (`jet_Rpt`, `t0comp`, `jet_sign`, and `NtrkTime`) are compared with the ones obtained in the best case with track variables (`trk_pt_norm`, `trk_phi`, `trk_d0`, and `trk_z0`), the last configuration both with and without using `trk_time`.

Features	$p_T < 50\,\text{GeV}$	$p_T > 50\,\text{GeV}$
`Rpt+t0comp+sign+NtrkTime`	0.974	0.984
Tracks without `trk_time`	0.976	0.986
Tracks with `trk_time`	0.977	0.987

Table 6.1: The AUC values for trainings with all the notable variable combinations

The first comparison is the one between the case with only high-level variables and the one with track variables without timing information. It is clear from Fig. 6.1 that for both low and high p_T ranges the improvement in BDT performance is remarkable. This result is not trivial, since the jet configuration already includes three variables with timing information (`t0comp`, `sign`, and `NtrkTime`), which are all unavailable in the track case. On the other hand, though, `Rpt` is defined as a function of the normalized `trk_pt`'s: therefore, it can be internally reconstructed by the BDT. The presence of extra track variables, unavailable at jet level, probably explains the further improvement over the best high-level variable configuration.

As expected, when `trk_time` is used as well, the performance is further im-

proved. This is clear from Fig. 6.1 as well as from Table 6.1. For example, in both p_T ranges, for a fixed 0.8 hard-scatter jet efficiency the pileup jet rejection improvement is around 30 %, while for a fixed 100 pileup jet rejection the hard-scatter jet efficiency improvement is around 5 %. This result is particularly valuable since it is produced using only five tracks per jet and ignoring pseudorapidity completely, making the outcome of this thesis probably valid even for different physical processes. Therefore, it really seems that the usage of timing information from HGTD can improve the suppression of jets from pileup in high-luminosity conditions, especially if low-level track information is exploited.

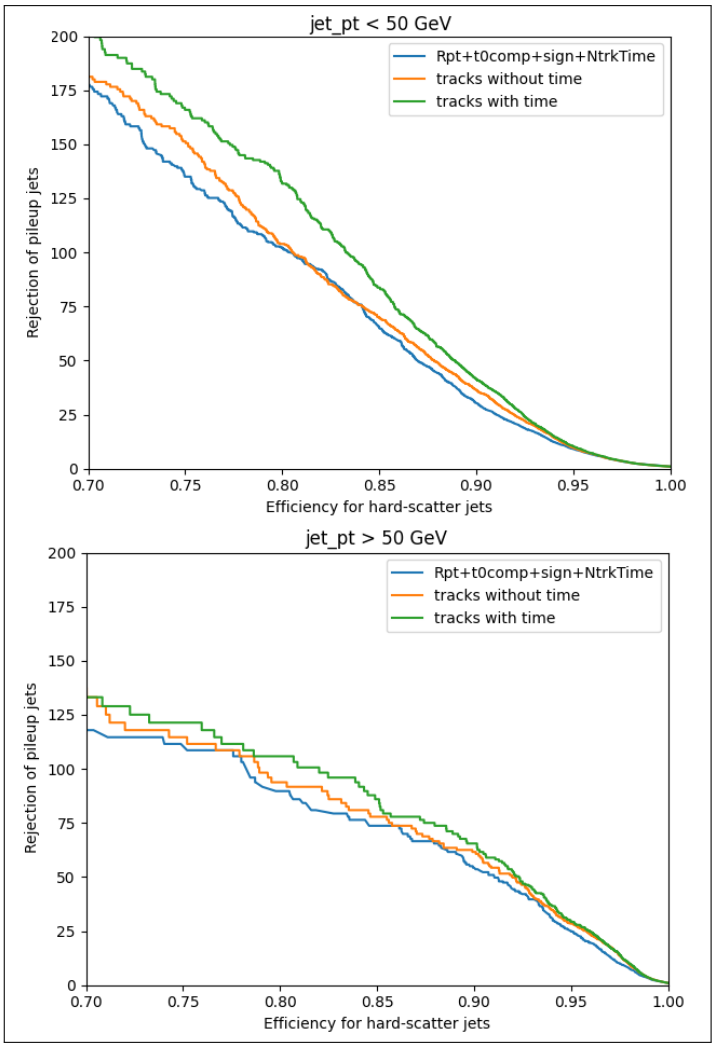

Figure 6.1: The ROC curves for trainings with all the notable variable combinations

Appendix A

The code

In this appendix, the main parts of the various chapters' codes will be shown and briefly explained.

A.1 Variables

First of all, various packages need to be introduced:

```
1 import numpy as np
2 import pandas as pd
3 from matplotlib import pyplot as plt
4 import uproot
```

The first one is the `numpy` package, which provides an N-dimensional array object with sophisticated functions: in the present code `numpy` is used for a variety of purposes, such as the creation of 2D-histograms or linear spaces. The second package is `pandas`, which provides fast, flexible, and expressive data structures: it is used as the main container for the jet data. The next one is `pyplot`, from the `matplotlib` package, a comprehensive library for creating static, animated, and interactive visualizations: it is used to plot the data graphs. The last one is `uproot`, a package to interact with ROOT[1] files: contrary to the standard ROOT implementation, this package is a mere input/output library, which is all that is needed in the present code.

Then, the data need to be imported and briefly processed:

```
1 f = uproot.open("FILE.root")
2 tree_s = f.get("myTreeS")
3 tree_b = f.get("myTreeB")
```

[1] ROOT is an open-source data analysis framework, used by high energy physics, in particular in the ATLAS experiment.

```
4  tree_s = tree_s.pd.df()
5  tree_b = tree_b.pd.df()
6
7  def normalize_input_df(tree):
8      for var_name in "EventNumber", "Nfwdjet", "Ncenjet":
9          tree[var_name] = tree[var_name].astype(int)
10     mask_nojet = np.isnan(tree["jet_pt"])
11     df = tree[~mask_nojet]
12     return df
13
14 df_s = normalize_input_df(tree_s)
15 df_b = normalize_input_df(tree_b)
16
17 df_s = df_s.assign(is_signal=True)
18 df_b = df_b.assign(is_signal=False)
19 df = pd.concat([df_s, df_b]).reset_index().drop(["subentry"
       , "entry"], axis=1).sort_values(["EventNumber", "jet_pt"
       ])
```

At line 1 the ROOT file is imported through `uproot`, then at lines 2–5 the two data trees (respectively the signal and the background ones) are extracted from the file and converted to `pandas`'s dataframes. After that, the `normalize_input_df` function is defined and then applied to both trees: this function first converts the variables `"EventNumber"`, `"Nfwdjet"`, `"Ncenjet"` to `int` data, then filters all those jets without the `"jet_pt"` variable, whose meaning has been explained in Section 5.1.2. In lines 17 and 18 a new `"is_signal"` variable is assigned to each entry of both trees, set to `True` for signal jets and to `False` for background ones. The last line concatenates both trees, creating a new dataframe with all the data.

A.1.1 One-dimensional histogram

The first graph is a one-dimensional histogram. In this plot, the variable to analyze is represented on the x axis, with the distribution density shown on the y axis:

```
1  fig, ax = plt.subplots(figsize=(6.4, 4.8), dpi=100,
       tight_layout=True)
2  min_v, max_v = df["VARIABLE"].quantile([0.01, 0.99])
3  bins = np.linspace(min_v, max_v, 100)
4  ax.hist(df[df["is_signal"]==True]["VARIABLE"], bins=bins,
       density=True, alpha=0.5, label="signal")
```

```
5 ax.hist(df[df["is_signal"]==False]["VARIABLE"], bins=bins,
      density=True, alpha=0.5, label="background")
6 ax.set_xlabel("VARIABLE")
7 ax.legend()
```

The first thing to do is create a figure `fig` and a graph `ax` with the `plt.subplots` function: the `figsize` argument provides width and height in inches, `dpi` is the resolution of the figure in dots per inch, and `tight_layout` automatically adjusts subplot parameters so that it fits into the figure area. The purpose of the second and third line is to create a linear space for binning on the x axis: first, a minimum and a maximum value are computed by trimming the ends of the variable distribution to exclude outliers, then the interval is divided into 100 equally spaced bins[2]. Lines 4 and 5 are the same code, one for the signal and one for the background. The function `ax.hist` creates a one-dimensional histogram, using the data filtered by `"is_signal"` value and the bins provided at line 3. The role of `density=True` is to normalize the histogram, `alpha` increases plot transparency, and `label` is used in the legend.

A.1.2 Two-dimensional histogram

The second graph is a two-dimensional histogram. In this plot, the variable to analyze is represented on the x axis, `"jet_pt"` is shown on the y axis, and the darkness of each square represents the distribution density:

```
1  fig, axx = plt.subplots(nrows=2, figsize=(6.4, 7.2), dpi
       =100, tight_layout=True)
2
3  for num in [0, 1]:
4      x=df[df["is_signal"]==num]["VARIABLE"]
5      y=df[df["is_signal"]==num]["jet_pt"]
6      my_range = [x.quantile([0.01, 0.99]), y.quantile([0.01,
           0.99])]
7      H, xedges, yedges = np.histogram2d(x, y, bins=25, range
           =my_range)
8      H = np.rot90(H)
9      H = np.flipud(H)
10     for i, _ in enumerate(H.T):
11         for j, _ in enumerate(H.T[i]):
```

[2]Various bin numbers were tried, but 100 appears to be the optimal value. The number of bins was changed only to deal with discrete variables, where trimming was also avoided.

```
12                 H.T[i][j] /= H.T[i].sum()
13         Hmasked = np.ma.masked_where(H==0, H)
14         axx[num-1].pcolormesh(xedges, yedges, Hmasked)
15
16 axx[0].set_title("signal")
17 axx[0].set_xlabel("VARIABLE")
18 axx[0].set_ylabel("jet_pt (GeV)")
19 axx[1].set_title("background")
20 axx[1].set_xlabel("VARIABLE")
21 axx[1].set_ylabel("jet_pt (GeV)")
```

Line 1 is the same as the one for the one-dimensional histogram, with the addition of `nrows`. This argument specifies the existence of two separate plots, one for the signal and one for the background[3]. The `for` loop is repeated in order to create the two plots. In lines 4 and 5 the dataframes for x and y axes are filled, filtering according to the `is_signal` value corresponding to `num`. In line 6, a list of two ranges is created, a range for the x axis and one for the y axis: this range is then used in line 7 to create the histogram object. The histogram `H` is then rotated and flipped in lines 8 and 9 to allow the proper plotting orientation. In lines from 10 to 12 each column of the histogram is normalized before creating a masked version of the histogram to render the empty bins accurately. The graph is then plotted in line 14, and the titles of axes are added.

A.1.3 Average plot

The third graph is an xy plot, with the variable to analyze represented on the y axis and `"jet_pt"` on the x axis. Error bars representing standard deviations of the distributions are also shown for each point:

```
1 fig, ax = plt.subplots(figsize=(6.4, 4.8), dpi=100,
     tight_layout=True)
2 labels=["signal", "background"]
3
4 for num in [0, 1]:
5         df1 = df[df["is_signal"]==num]
6         df1 = df1[(df1[VARIABLE]<df1["VARIABLE"].quantile(0.99)
             )]
```

[3]In this case, it is not wise to graph both the signal and the background on the same graph since it can become difficult to read because of color superimposition

```
7       df1 = df1[(df1[VARIABLE]>df1["VARIABLE"].quantile(0.01)
            )]
8       x = df1[df1["is_signal"]==num]["jet_pt"]
9       y = df1[df1["is_signal"]==num]["VARIABLE"]
10      xls = x.values.tolist()
11      yls = y.values.tolist()
12      bins = np.linspace(x.quantile(0), x.quantile(1), 25)
13      xmeans=[]
14      for i in range(len(bins)-1):
15          xmeans.append((bins[i+1]+bins[i])/2)
16      means=[]
17      errors=[]
18      for i in range(len(bins)-1):
19          temp=[]
20          for j in range(len(xls)):
21              if xls[j]>=bins[i] and xls[j]<bins[i+1]:
22                  temp.append(yls[j])
23          means.append(np.mean(temp))
24          errors.append(np.std(temp))
25      ax.errorbar(xmeans, means, xerr=0, yerr=errors, marker=
            ".", label=labels[num-1])
26
27  ax.set_xlabel("jet_pt␣(GeV)")
28  ax.set_ylabel("VARIABLE")
29  ax.legend()
```

Line 1 is the usual `subplots` function. Line 2 is a list of two labels that will be used for the plots. The first three lines of the `for` loop select only the data with the appropriate `"is_signal"` value and then trim the tails of the distribution to exclude outliers. Lines 8 and 9 then create two dataframes for the x ad y axes respectively, before converting them to a list of `float` values in lines 10 and 11. Line 12 creates a linear space for binning on the x axis. Lines from 13 to 15 create a list of the middle points of all the bins created earlier to allow proper plotting. Lines 16 to 24 create two other lists, one (`means`) for the average values on the y axis for each bin and one (`errors`) for standard deviations of each mean value. These values are then plotted using the `ax.errorbar` function, which takes two lists as inputs and has additional arguments for both x and y error bars, the marker shape, and the labels. As usual, the axes are labeled and a legend is created.

A.2 BDT training

As usual, the first things to look at are the included packages:

```
1 import numpy as np
2 import pandas as pd
3 from matplotlib import pyplot as plt
4 import uproot
5 import lightgbm as lgb
6 import optuna.integration.lightgbm as opt
7 from sklearn.metrics import roc_curve
8 from sklearn.metrics import roc_auc_score
9 from sklearn.model_selection import train_test_split
```

The first four lines are the same as those reported in Section A.1. Line 5 imports the `lightgbm` package, which is the one which actively performs the BDT training. Line 6 imports the `optuna` package, an automatic hyperparameter optimization software framework, which will be discussed soon. The last three lines import some functions from `sklearn`, a package created to deal with machine learning problems: these functions will be explained when encountered.

After this, the data need to be imported, in the same way shown in Section A.1. Once the data are imported, two steps need to be performed: first, the hyperparameters need to be tuned, and then the BDT training is performed.

A.2.1 Hyperparameters optimization

The following code will be shown for just one variable choice, but of course it needs to be repeated for all the combinations required.

```
1  params = {
2      "objective": "binary",
3      "metric": "binary_logloss",
4  }
5  df_train, df_test = train_test_split(df, test_size=0.5)
6  train_x = df_train[FEATURE].values
7  train_y = df_train["is_signal"].values
8  test_x = df_test[FEATURE].values
9  test_y = df_test["is_signal"].values
10 dtrain = lgb.Dataset(train_x, label=train_y)
11 dtest = lgb.Dataset(test_x, label=test_y)
```

```
12 model = opt.train(params, dtrain, valid_sets=[dtrain, dtest
    ], num_boost_round=1000, early_stopping_rounds=100)
13 best_params = model.params
```

In lines 1 to 4, the essential parameters of the BDT are defined, as they depend on data nature and desired result. The `"binary"` objective refers to the binary classification (signal or background), while the `"binary_logloss"` metric is the default way to evaluate efficiency for `"binary"` classifications; the other parameters will be optimised by the `optuna` package. At line 5, the `train_test_split` function is used: this function randomly splits the `df` variable into two dataframes, one for the training of the BDT and one for testing it: the default splitting sizes are 75 % of data for the training dataframe and the rest for the testing. Still, after various tests, the best splitting size appears to be 50 % for both sets. In lines 6 to 9, the dataframe data are split into an `x` and a `y`: `x` represents the features used, which means the data which will be given to the BDT to discriminate signal and background jets, while `y` corresponds to the true values, that is whether each jet is signal or background. Lines 10 and 11 create two LightGBM datasets using the lists created above. Line 12 is the line where the actual optimization takes place: `opt.train` takes as input the parameters defined at the beginning of the code, the training dataset `dtrain`, two valid sets (training and testing) which will be used to evaluate the optimal parameters, the maximum number of boosting iterations, and the number of iterations without improvement after which to stop the training. The last line saves the optimized parameters to a variable `best_params`.

A.2.2 Training

```
1 evals_result = {}
2 df_train, df_test = train_test_split(df)
3 train_x = df_train[FEATURE].values
4 train_y = df_train["is_signal"].values
5 test_x = df_test[FEATURE].values
6 test_y = df_test["is_signal"].values
7 dtrain = lgb.Dataset(train_x, label=train_y)
8 dtest = lgb.Dataset(test_x, label=test_y)
9 model = lgb.train(best_params, dtrain, valid_sets=[dtrain,
    dtest], evals_result=evals_result, early_stopping_rounds
    =100)
10 prediction = model.predict(test_x, num_iteration=model.
    best_iteration)
```

Line 1 creates a dictionary which will be used to plot some graphs, which will be explained in the next sections. Lines 2 to 9 are the same as those used for hyperparameters optimisation, with the only difference that now `model` is trained using the standard `lgb.train` function instead of the `optuna` version, using the `best_params` previously calculated. Line 10 creates a list of probabilities, which correspond to the level of confidence with which the algorithm can define each jet as background or signal: 0 corresponds to a 100 % background match, 1 to a 100 % signal match.

A.2.3 Training plots

Three graphs are plotted during the BDT algorithm to represent the evolution of the training visually.

```
1  lgb.plot_metric(evals_result, ax=ax)
2  lgb.plot_importance(model, ax=ax)
3  data = pd.DataFrame({"is_signal": test_y, "prediction":
       prediction})
4  fig, ax = plt.subplots(figsize=(6.4, 4.8), dpi=100,
       tight_layout=True)
5  min_v, max_v = [0, 1]
6  bins = np.linspace(min_v, max_v, 100)
7  ax.hist(data[data["is_signal"]==True]["prediction"], bins=
       bins, density=True, alpha=0.5, label="signal")
8  ax.hist(data[data["is_signal"]==False]["prediction"], bins=
       bins, density=True, alpha=0.5, label="background")
9  ax.set_xlabel("Score")
10 ax.set_yscale("log")
11 ax.legend()
```

The first graph is plotted in line 1: it is a plot of the metric used to evaluate che training loss in function of the number of iterations. The second one is a graph which shows the relative importance of each feature in creating the BDT. The rest of the lines plots the distribution of the BDT predictions, separately for signal and background jets. After having created a dataframe `data`, the usual `subplots` function is used: the procedure is the same for the one-dimensional histograms in Section A.1, a linear space is created and the two histograms are plotted. The only notable difference is at line 11, where the scale of the y axis is set to logarithmic, in order to better appreciate the distribution.

A.2.4 ROC curves

```
1  auc = roc_auc_score(test_y, prediction)
2  roc = ([], [])
3  roc[0], roc[1], _ = roc_curve(val_y, prediction)
4  graph = ([], [])
5  for elem in roc[1]:
6      graph[0].append(elem)
7  for elem in roc[0]:
8      graph[1].append(1/elem)
9  fig, ax = plt.subplots(figsize=(6.4,4.8), dpi=100,
       tight_layout=True)
10 ax.plot(graph[0], graph[1], label=label)
11 ax.set_xlim([0.7, 1])
12 ax.set_ylim([0, 200])
13 ax.set_xlabel("Efficiency for hard-scatter jets")
14 ax.set_ylabel("Rejection of pile-up jets")
15 ax.legend()
```

The first thing that is done is the calculation of the Area Under the Curve AUC. After this, a tuple of lists, representing the x and y coordinates of the points, is created at lines 2 and 3 using the `roc_curve` function. The plots provided by ATLAS are different though, as they show the true positive rate on the x axis and on the y axis the reciprocal of the false positive rate. Therefore, in order to compare the plots to the ATLAS ones, a new tuple of lists `graph` is created in lines from 4 to 8, with the proper variables in the x and y axes. After this, the plot is created as usual.

Bibliography

[1] *Luminosity? why don't we just say collision rate?*, CERN, (2020) https://home.cern/news/opinion/cern/luminosity-why-dont-we-just-say-collision-rate (visited on 09/30/2020).

[2] F. Gianotti, *Collider physics: LHC*, (2000) http://cds.cern.ch/record/458489/files/p219.pdf (visited on 09/30/2020).

[3] *The Large Hadron Collider*, CERN, (2020) https://home.cern/science/accelerators/large-hadron-collider (visited on 09/30/2020).

[4] *About the ATLAS experiment*, CERN, (2020) https://atlas.cern/discover/about (visited on 09/30/2020).

[5] *Detector & technology*, CERN, (2020) http://atlas.cern/discover/detector (visited on 09/30/2020).

[6] S. Manzoni, *Physics with photons with the ATLAS Run 2 data: calibration and identification, measurement of the Higgs boson mass and search for supersymmetry in di-photon final state* (Università degli studi di Milano, Milano, 2017).

[7] G. Apollinari, O. Brüning, T. Nakamoto, and L. Rossi, *High Luminosity Large Hadron Collider HL-LHC*, https://arxiv.org/pdf/1705.08830.pdf%5C%7D (visited on 09/30/2020).

[8] The ATLAS Collaboration, *Technical design report: a High-Granularity Timing Detector for the ATLAS phase-II upgrade*, (2020) http://cds.cern.ch/record/2719855/files/ATLAS-TDR-031.pdf (visited on 09/30/2020).

[9] S. Schramm, *ATLAS jet reconstruction, calibration, and tagging of Lorentz-boosted objects*, (2017) http://cds.cern.ch/record/2291608/files/ATL-PHYS-PROC-2017-236.pdf?version=1 (visited on 09/30/2020).

[10] M. Cacciari, G. P. Salam, and G. Soyez, *FastJet user manual*, http://fastjet.fr/repo/fastjet-doc-2.4.5.pdf (visited on 09/30/2020).

[11] *Introduction to boosted trees*, xgboost, (2020) https://xgboost.readthedocs.io/en/latest/tutorials/model.html (visited on 09/30/2020).

[12] *Features*, Microsoft Corporation, (2020) `http://lightgbm.readthedocs.io/en/latest/Features.html` (visited on 09/30/2020).

[13] *Overview*, `https://geant4.web.cern.ch` (visited on 09/30/2020).

[14] *Optuna - a hyperparameters optimization framework*, Preferred Networks, (2020) `https://optuna.org` (visited on 09/30/2020).