Lepton Identification In Proton-Proton Collisions With Trilepton Final State Model Using Machine Learning

Kristoffer Langstad

Thesis submitted for the degree of
Master in Computational Science: Physics
60 credits

Department of Physics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Spring 2021

Lepton Identification In Proton-Proton Collisions With Trilepton Final State Model Using Machine Learning

Kristoffer Langstad

Lepton Identification In Proton-Proton Collisions With Trilepton Final State Model Using Machine Learning

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

Abstract

...

Acknowledgements

...

Contents

List of Figures

Listings

Chapter 1

*Introduction

In particle physics, one of the big focuses is colliding particles at high energies is to produce known and possibly unknown particles. When two particles collide, they will produce new particles that move through the detectors which are built around the collision points. At the Large Hadron Collider (Large Hadron Collider (LHC)), $\sim 10^4$ protons collide every 25 ns and produces huge amount of data each second which is captured by detectors and stored for further analysis. Many particles are produced in each such particle collision, and it is not always trivial to identify all these particles. There are also cases where some particles are not detected at all, like neutrinos. By using machine learning (ML) algorithms, which is a study in computer science and mathematics involving, among others, pattern recognition, we can try to computationally identify these newly produced particles from the collision decays.

Particle physics takes a closer look at the building blocks of the universe, the fundamental particles in the Standard Model (SM). This is a theory that fits well with most observations. However there are several observations in the universe that cannot fully be explained by the SM, like dark matter and dark energy, meaning that the SM is incomplete and have to be extended. One of the methods to complete or expand the SM is to find new particles. This is done at large laboratories like CERN, where one of the things they do is to collide particles at high energies to produce new particles. After colliding particles, the new particles are detected using big detectors like A Toroidal LHC ApparatuS (ATLAS). One of the major discoveries at CERN is the discovery of the Higgs boson[1][2], which was the last missing piece of the SM. In this thesis, we will analyze both "*truth*"[1] and more realistic simulations after taking into account hadronization, showers and detector

[1]Meaning we have simulated data of particle collisions where we know exactly which particles have been produced and their origin.

inefficiency of particle collisions.

The goal of this thesis is to use machine learning algorithms to automatically classify the origin of the final state particles from many events of proton-proton collisions. We will study a few different scenarios of a beyond the SM particle physics model that gives three final state leptons and a neutrino, and compare them with each other.

1.1 Motivation for Machine Learning

Machine learning and data science has had a huge growth in the past years. There are now a bigger variety of algorithms and approaches better suited for data analysis and different types of analyses depending on both the data sets and the desired goals. The data sets are as well a lot bigger than before, with samples ranging up to billions. This is both good and bad, since most machine learning models need a lot of training data to perform and predict on a good level. But with larger data sets, more time is needed to do the analyses since there are more data, obviously. This means that the models have to be fast and good.

Machine learning models have pattern recognition as one of the main focuses. The idea is to automatize and learn what normally is complex and difficult for humans to do. This can include image analysis or to learn the rules of a game. The more data the learning models have, the better they can do their tasks. Even though the algorithms can do quite complex tasks, the fundamental methods in these algorithms include normally simple methods. Many of the most used algorithms have several hyperparameters that are used to optimize the models.

This thesis looks at the same models as Das et al. [3], with a focus on the trilepton channel, and Pascoli et al. [4] to produce the final trilepton plus missing transverse energy states. By applying machine learning techniques on simulated data that force this type of trilepton final state, we want to see if we can automatically identify the final state leptons and if we then can say something about the MET. If successful, then we can export the best model to other similar scenarios later.

1.2 Structure of Thesis

In the first chapters of part I, we take a look at an introduction to particle physics and further theories connected with the model we study. The next chapters involve the particle kinematics of particle collisions, and how they

are detected by instruments, followed by a short explanation of the model to be analyzed. Then follows theory of machine learning, the learning models and evaluation metrics to be used in this thesis.

Part II starts by looking at the data and generation of the data prior to our analysis. Then we look at more detailed implementation of the machine learning algorithms. The analysis is chosen to be done in the programming language **Python**, which has many useful libraries for doing machine learning.

In part III we present the results of the analysis done with the machine learning algorithms, and compare the different scenarios of the particle physics model we use.

Part IV consists of discussions of the results, concluding remarks of the thesis and a short look into future research based on this thesis.

Notation and Conventions

- $e = 1.6 \cdot 10^{-19}$ C : The elementary charge.
- $c = 2.998 \times 10^8$ m/s: Speed of light in vacuum.
- 1 GeV $= 10^9$ eV $= 10^9 \times 1.602 \times 10^{-19}$ J: Approximately the rest mass energy of the proton.
- $m_e = 9.109 \times 10^{-31}$ kg $= 0.511$ MeV/c^2: Mass of an electron.
- 1 barn (b) $\equiv 10^{-28}$ m^2: Interaction cross sections (dimension of area).
- $h = 6.626 \times 10^{-34}$ J·s: Planck's constant, a fundamental physical constant.
- $\hbar = \frac{h}{2\pi} = 1.055 \times 10^{-34}$ J·s: Unit of action in quantum mechanics (also called the reduced Planck constant).
- Einstein energy-momentum formula: $E^2 = p^2c^2 + m_o^2c^4$
- Coulomb force between two charged particles: $F = \frac{q_1 q_2}{4\pi\epsilon_0 r^2}$
- Natural units (from S.I. units):
  - Replace [kg, m, s] with [$\hbar$, c, GeV].
  - $\hbar c = 197$ MeV fm.
  - Use $\hbar = c = \epsilon_0 = \mu_0 = 1$.
- 1D time-dependent Schrödinger equation:

$$i\frac{\partial\psi(\mathbf{x},t)}{\partial t} = -\frac{1}{2m}\frac{\partial^2\psi(\mathbf{x},t)}{\partial x^2} + \hat{V}\psi(\mathbf{x},t)$$

- Planck scale $\sim 10^{19}$ GeV.
- GUT scale $\sim 10^{16}$ GeV.
- Magnetic fields are measured in Tesla (T).

Part I

Theory

Chapter 2

The Standard Model of Particle Physics

Throughout the years, there have been many theories in physics of what the universe is made up of and how everything fits together. For now, the best theory/model is the Standard Model (SM) of particle physics. This theory has many times through the years proven to successfully predict and explain particles and their interactions. This model has lead to the discovery of what we now call elementary particles and fundamental forces, and they are the building blocks of the universe.

In this chapter we look closer at the contents of the SM and the underlying theories and models. Much of the information in this chapter is based upon Thomson [5] and some Elert [6].

2.1 Particle and Force Contents

The known elementary particles can be categorized into two main categories according to their spins; fermions and bosons. Fermions have half-integer spins, while bosons have integer spins. The Higgs boson is categorized as a boson but has 0 spin. In Figure 2.1 we see the categorization of the elementary particles, and the fundamental forces, in the SM. The individual categorizations will be explained in the upcoming sections. The interactions between the SM particles can be seen in Figure 2.2.

2.1.1 Gauge Bosons

From what we know of, there exists four fundamental forces. Three of these can be explained by the SM through exchange of (gauge) bosons. That is

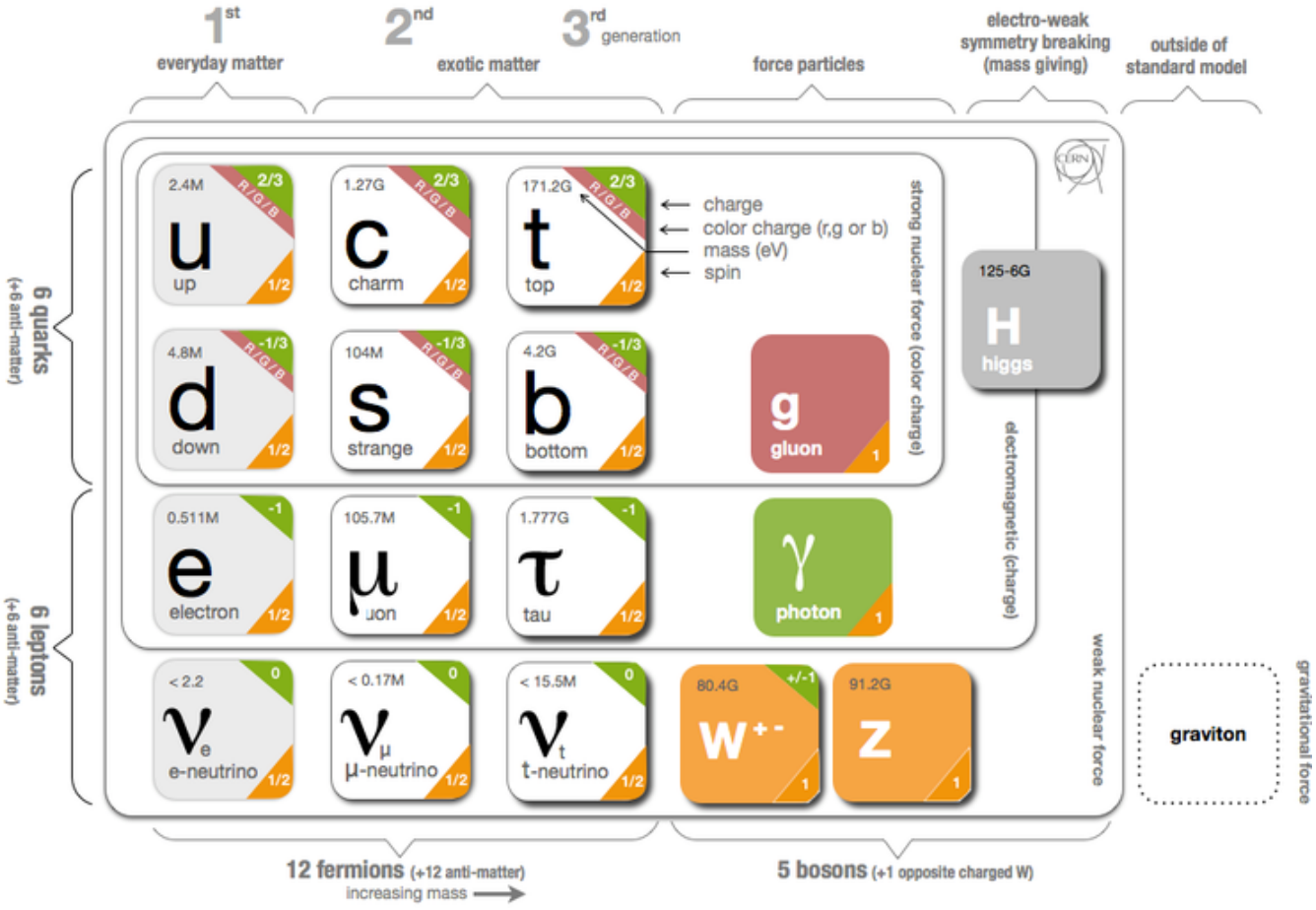


Figure 2.1: The Standard Model contents, source [7].

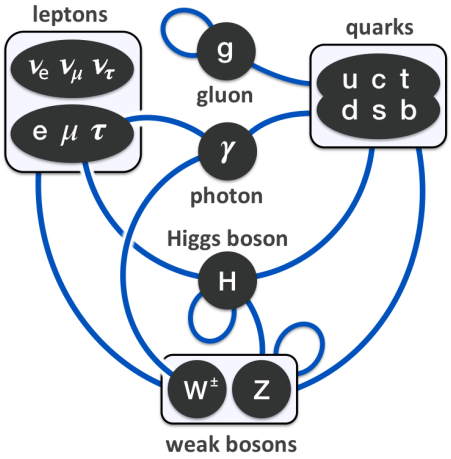


Figure 2.2: The interaction between the particles in the Standard Model. Credit: Wikipedia.

why bosons also are called force-carrier particles. The three forces are the electromagnetic, strong and weak nuclear forces, where each force has its own connected boson(s). There are five different bosons that mediates these forces, and they all have integer spins. This means that they go with vector fields, along a direction.

Strong nuclear force

The strong nuclear force is mediated by eight massless gluons (g). They only affect the (r,g,b) color charged quarks, and come in combinations of color and anti-color charges. Since the six gluons carry a different variation of color and anti-color combinations, they come in an octet of colored states. The color assignments of these eight physical gluon variations can be written as:

$$b\bar{g}, b\bar{r}, g\bar{b}, g\bar{r}, r\bar{b}, r\bar{g}, \frac{1}{\sqrt{2}}(r\bar{r} - g\bar{g}), \frac{1}{\sqrt{6}}(r\bar{r} + g\bar{g} - 2b\bar{b})$$

It is this strong interaction force that binds the quarks together to make e.g. protons and neutrons. The gluons can also self-interact with each other. This makes the interaction range of the strong nuclear force short, keeping the gluons within the nucleus. The exchange of gluons by interactions of colored particles is a mathematical model known as quantum chromodynamics (QCD, sect.2.4.2).

Electromagnetic force

The electromagnetic force is mediated by the massless photon (γ). Photons have interact with electrically charged particles. Since the photon is massless and electrically neutral, it has an infinite range. The electromagnetic force is responsible for holding electrons in place around the nucleus, and is not as strong as the strong nuclear force. Electrically charged particles are either attracted to each other or repelled away for each other, dependent on if the charges of the particles have the same sign or not. The exchange of photons by interactions of charged particles is a mathematical model known as quantum electrodynamics (QED, sect.2.4.2).

Weak nuclear force

The weak nuclear force is mediated by the $W^{\pm}$ and Z^0 bosons. There are two charged variants of the W with charge $+e$ or $-e$. The Z boson is electrically neutral. They er all massive which gives a short lifetime and short range. Because of the difference in charges, they act on different particles. The W boson couples the electromagnetic interactions. The W boson can decay to all flavors of quarks, except the top quark which is too massive, leptonic final states or hadronic final states. The weak interaction force can change the flavor of quarks. The exchange of W and Z bosons is explained with a more complex mathematical model that unifies both the weak and electromagnetic interactions, and it is known as electroweak theory (EWT, sect.2.4.2).

Gravitational force

The last force of nature is gravity. We have not yet found the hypothetical graviton (G) particle which should carry the gravitational force. All the other forces seem to be well explained in the SM, except for gravity. So the gravitational force is not included in the SM. There has been a lot of theories about the graviton and a lot of experiments. LIGO and Virgo discovered in 2015 gravitational waves from observing the merging of two black holes with $\sim$30 solar masses each [8], which might give insight into gravitons in the future. So far, we have nothing conclusive if the graviton exists, yet. It is thought that the graviton would have spin two, since the gravitational waves is described in general relativity as a propagating tensor disturbance. When looking at small objects (micro size), gravity does not seem to have any noticeable effect. But when we look at bigger objects of mass like humans or planets (macro size), then gravity has a much bigger effect and is well described by Einstein's General Theory of Relativity. Since gravity has more

or less a negligible effect on particles energies so far probed in experiments, particle physicists do not have to take gravity into consideration.

2.1.2 Higgs Boson

The Higgs boson (H) is a "recently" discovered particle (2012) [1][2] theorized by Peter Higgs in 1964. This particle has intrinsic no spin, which makes it a scalar particle, and the only scalar particle discovered so far. It's electrically neutral and massive ($m_H \approx$125 GeV), and interacts with itself. Since it is so massive, the lifetime of the Higgs boson is very short and it's hard to detect directly. It can in principle decay to all massive SM particles. The heavier particle, the stronger is the coupling to the Higgs.

The discovery of the Higgs boson was a major contribution to the SM since it can explain the origin of the masses of the other elementary particles. It also confirmed the existence of the Higgs (scalar) field, which gives the other elementary particles mass when they interact with this field. This field is thought to be everywhere in the universe with a non-zero vacuum expectation value. Here, Higgs bosons appear and disappear and interact with other particles in the field giving them their masses. The gluons and photons do not interact with this field, hence they are massless.

2.1.3 Fermions

The fermion group in the SM consists of 12 elementary particles with half-integer spins. These particles are also known as matter-particles, since these particles are the building blocks of the matter in the universe. Each fermion has its own antiparticle. The antiparticles have the same mass as their particle partner, but has opposite electric charges and different quantum numbers. Fermions which acts as their own antiparticles are called Majorana particles.

The 12 fermions can be split into two groups of six quarks and six leptons. The fermions can then be categorized into three generations, which goes from lighter and more stable to heavier and less stable. As seen in the SM figure (2.1), the first generation is called the "everyday matter". This is because most of the stable (baryonic) matter is made from the first generation particles. The reason for this is that the first generation particles do not decay. Second and third generation particles are only observed in high-energy environments. None of the neutrinos decay, but they oscillate and scatter, and they rarely interact with baryonic matter.

Quarks

The six quarks are up, down, charm, strange, top and bottom. A characteristic property for the quarks is that they all have color and electric charges, and they interact through the strong nuclear force. The colors charges are denoted red, green and blue and they all have an anti-color. Quarks cannot exist as free particles. As explained in section 2.1.1, the quarks have a strong binding force between them since they are acted upon by the strong nuclear force. From this strong binding force, the quarks form particles called hadrons, like protons and neutrons. They are made up of either three quarks (baryons) or a quark and an anti-quark (mesons). A proton is made up of one down quark and two up quarks. The hadrons are color-neutral particles. Since quarks have electric charges, they also interact via the electromagnetic force and the weak nuclear force.

Leptons

The six leptons are electron (e), electron neutrino (ν_e), muon (μ), muon neutrino (ν_μ), tau (τ) and tau neutrino (ν_τ). The electron, muon and tau leptons have electric charges and are influenced by electromagnetism. They all carry a $-1e$ electric charge, while their respective antiparticle having electric charge $+1e$. Every lepton carrying a lepton number, which is conserved in all known interactions. The leptons also interact weakly. Both leptons and antileptons have their respective lepton number +1 and -1, and each flavor has its own lepton flavor number with the same values as the lepton numbers. There are three generations where the three charged leptons are paired with their respective neutrino, and the masses of these three leptons increases with the generation. Only the electron (1st gen) is stable and doesn't decay, while the muon and tau leptons decay via the weak interaction.

2.2 Neutrinos

The three neutrinos (electron, muon, tau) are a little more special than the other elementary particles. They are classified as leptons with half-integer spins, but they do not carry any charge and are thus neutral. They only interact via the weak nuclear force, making them very hard to observe since they go through almost everything without interacting much with anything. If the neutrinos are Majorana, they are the only Majorana fermions of the SM since all the other fermions have an non-zero electric charge. By detection of neutrinos and antineutrinos, only left-handed neutrinos and right-handed antineutrinos are observed. From the weak nuclear force mediator particles,

the $W^{\pm}$ bosons, we know that they only couple to left-handed particles and right-handed antiparticles. This means that interaction of the right-handed neutrinos is not covered in the SM. Since mass terms couple both left- and right-handed states, the neutrinos are considered as massless in the SM. Through the discovery of neutrino oscillations [9], we know that the neutrinos can change flavor meaning they cannot be massless. We know that they have to have mass since the neutrinos oscillate, but the mechanism behind the masses are not known. So, one type of neutrino can in fact change flavor to another type of neutrino when it travels over a large distance. Neutrino oscillation describes the difference between the neutrino flavor eigenstates and the neutrino mass eigenstates. This type of physics is not covered by the SM and will be looked more into later in this thesis.

2.2.1 Neutrino Oscillations

From the SM we know that for all interactions the lepton number is conserved for both the total and each lepton flavor separately. The lepton number is conserved when a $W^{\pm}$ boson decays into a lepton neutrino pair. We will in this thesis have a $W^{\pm}$ boson that decays into leptons

The discovery of neutrino oscillations was done by two experiments. Namely the Super-Kamiokande Observatory[10] and the Sudbury Neutrino Observatories (SNO)[11] experiments. They got the Nobel Prize in physics in 2015 for their contributions by detecting solar neutrinos from the Sun [12]. The Super-Kamiokande detected electron neutrinos using a big water Čerenkov detector, but they got a too low electron neutrino flux than what was expected to be produced in the Sun. The SNO experiment showed that the atmospheric neutrinos and the neutrino flux from β-decay in the Sun had strong muon and tau components by using heavy water. Since only electron neutrinos are produced by nuclear fusion in the Sun, the neutrinos must have the ability to change their flavor when moving over large distances.

The neutrino oscillation is a quantum-mechanical phenomenon, where the neutrino flavor (weak) eigenstates $(\nu_e, \nu_\mu, \nu_\tau)$ can be related to the mass eigenstates (ν_1, ν_2, ν_3) by an unitary transformation matrix U as

$$\begin{pmatrix} \nu_e \\ \nu_\mu \\ \nu_\tau \end{pmatrix} = \begin{pmatrix} U_{e1} & U_{e2} & U_{e3} \\ U_{\mu 1} & U_{\mu 2} & U_{\mu 3} \\ U_{\tau 1} & U_{\tau 2} & U_{\tau 3} \end{pmatrix} \begin{pmatrix} \nu_1 \\ \nu_2 \\ \nu_3 \end{pmatrix}.$$

The flavor eigenstates are linear combinations of the mass eigenstates. The 3×3 unitary matrix is the Pontecorvo-Maki-Nakagawa-Sakata (PMNS) matrix, and it's expressed with three mixing (rotation) angles and a complex

Dirac CP violation phase if the neutrinos are Dirac particles. The unitary of the PMNS matrix implies that $U^{-1} = U^{\dagger} \equiv (U^{*})^{T}$ and $UU^{\dagger} = I$.

If the neutrino mass eigenstates are not the same, we get neutrino oscillations from the phase differences in components of the wavefunction. Since we already know that the neutrinos change flavor from the discovery of neutrino oscillations, we know that the neutrinos need some mass, differing by flavor, to being able to change flavor. That is why the neutrinos need non-zero masses and not equal to each other for neutrino oscillations to be true. From experimental measurements, like long baseline accelerators, for the neutrino masses there is only found upper limits to the masses. The best upper limits on the neutrino masses was found to be

$$\sum_{i=1}^{3} m_{\nu_i} \lesssim 1.1\text{eV} \tag{2.1}$$

by the Karlsruhe Tritium Neutrino (KATRIN)[13] experiment in Germany. The reason why the neutrino masses seems to be so much smaller than the other fundamental particles is not known.

2.3 Symmetries

Particle dynamics are heavily influenced by symmetries and laws of conservation. From classical Newtonian physics, we know that energy (E), three-momentum ($\vec{p}$) and total angular momentum (J) are conserved quantities. This is also the case in the SM. A quantity that is not conserved is the (rest) mass (m). This is something we know according to Einstein's Special Relativity. This enables production of heavier particles than the colliding particles.

Another fundamental symmetry of physical laws is the CPT theorem. The CPT theorem is one of the results concluded by quantum field theory (QFT), and states that all physical processes are symmetric under CPT-transformation [14]. C is charge conjugation, where every particle can be replaced by its antiparticle. P is parity reflection, where everything in the universe is mirrored along the three physical axes. T is time reversal, where the direction of time is reversed in the sense of looking at the local properties of the SM. The combination of these three symmetries is predicted by the SM to be a symmetry, while each symmetry alone is only a near-symmetry. The CPT symmetry explains why particles and antiparticles have identical masses, magnetic moments, etc. The CPT is also thought to be an exact symmetry in the Universe. Only the weak interactions of quarks and lep-

tons seems to violate the C-, P-, T- and CP-symmetries out of the three fundamental forces explained by the SM.

A topic to be further discussed later is gauge theory (sect. 2.4.2). From the connected gauge symmetry in the SM, we get a conservation of certain quantum numbers during the different interactions with the fundamental forces based on the $SU(3) \times SU(2) \times U(1)$ group. The quantities that are conserved are: the color charge for the strong nuclear interaction ($SU(3)$), the electric charge for electromagnetic interactions ($U(1)$) and the weak isospin for the weak nuclear interaction ($SU(2)$).

Other important conservation laws are the conservation of baryon number, B, and lepton number, L_x, in an interaction. x is the lepton flavor. The only case where the lepton number is not conserved is for neutrino oscillation. As we have explained earlier, neutrinos can change flavor when traveling large distances. But, this is not something we have to be concerned about in our case since we look at particles in particle detectors over short distance. This distance is not big enough for neutrino oscillations to occur.

2.4 Quantum Field Theory

The Standard Model is based on the framework of quantum field theory (QFT). This is a theory that combines quantum mechanics, special relativity and field theory. In other words, quantum field theory tries to explain the little things in the universe, like the elementary particles, that move very fast, close to or with light speed c. This also means that every elementary particle has its own associated field. These fields can then be explained in terms of the Lagrangian density, $\mathcal{L}$, to explain the dynamics and kinematics of the fields.

The combination of quantum mechanics and special relativity does give some problems. The most important equation in quantum mechanics is the Schrödinger equation, and it's not Lorentz invariant. The problem with this is that Schrödinger's equation is not the same for two observers in different reference frames. Other problems this leads to is that we get violation of causality, negative energy states and there is no possibility for new particle creations. The good thing is that these problems can be fixed by exchanging the Schrödinger equation (see Notation and Conventions) by the Dirac equation [15][16] for $\frac{1}{2}$-spin particles and the Klein-Gordon equation [15][16] for scalar particles. With the Dirac and Klein-Gordon fields, this leads to specific (gauge) theories for different particles and associated interactions, which we have briefly mentioned earlier and will explain more soon.

2.4.1 The Lagrangian

For more simple classical mechanics cases the Lagrangian is just given as the difference between the kinetic energy, K, and the potential energy, V, $L = K - V$. This is also a baseline for the QFT. By using the Lagrangian of a system with a set of generalized coordinates q_i and their time derivatives $\dot{q}_i$, we can find the equation of motion that describes the system by using the Euler-Lagrange equation,

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = 0. \tag{2.2}$$

A difference for QFT is that instead of kinetic and potential energies, or the generalized coordinates, we use fields with four space-time coordinates. This changes the Lagrangian L to the Lagrangian density $\mathcal{L}$ as a continuous system. This is a function of the fields, $\phi_i(t, x, y, z)$, and their derivatives, $\partial_\mu \phi_i(t, x, y, z)$. Since L is the spatial integral over $\mathcal{L}$,

$$L = \int \mathcal{L} d^3\mathbf{x}, \tag{2.3}$$

and using the principle of least action [17], the new Euler-Lagrange equation becomes

$$\partial_\mu \left(\frac{\partial \mathcal{L}}{\partial(\partial_\mu \phi_i)}\right) - \frac{\partial \mathcal{L}}{\partial \phi_i} = 0. \tag{2.4}$$

For simplicity we will just denote the Lagrangian density the Lagrangian from now on. From this new Euler-Lagrange equation, we can derive both the free-particle Dirac and the Klein-Gordon equations by imposing the Lagrangian with a free fermion field[1] and free theory[2] respectively. The Lagrangian for the spin-half (spinor) field, ψ, is

$$\mathcal{L}_D = i\bar{\psi}\gamma^\mu \partial_\mu \psi - m\bar{\psi}\psi, \tag{2.5}$$

and the Lagrangian for the non-interacting scalar field, ϕ, is

$$\mathcal{L}_S = \frac{1}{2}(\partial_\mu \phi)(\partial^\mu \phi) - \frac{1}{2}m^2\phi^2. \tag{2.6}$$

Both of these two equations for the Lagrangian contain a kinematic term and a mass term.

With perturbation theory in quantum mechanics, the Lagrangian can also be used to describe the behavior and interaction of elementary particles with Feynman diagrams for simpler visualization of usually complex particle interactions.

[1] Relativistic spin-half fields, Chapter 17.2.2 in Thomson [5]
[2] Relativistic scalar fields, Chapter 17.2.2 in Thomson [5]

2.4.2 Qauge Theories

From the new Lagrangian we now know that we need some new theory to explain the interactions between the elementary particles, since these interactions vary depending on the particles and associated interactions involved. In this theory we need to require that the Lagrangian stays invariant under local transformations using symmetry or gauge groups. In special relativity, this global symmetry group is called the Poincaré group which includes spacetime symmetries.

To describe the SM we need an internal gauge invariant symmetry that represents the different elementary interactions and is independent of spacetime coordinates. This is the local $SU(3) \times SU(2) \times U(1)$ gauge symmetry group. Here each special unitary group with degree n (the number in the parenthesis) is connected to its own gauge theory and the three elementary interactions in the SM, and n is a n-dimensional space. If a symmetry group is commutative, meaning that regardless of what the order of the elements are applied the result will be the same, then it is called an Abelian group. If the group is non-commutative, it is then a non-Abelian gauge theory which implies the existence of gauge boson self-interaction.

Quantum chromodynamics (QCD)

The gauge theory that defines the strong interaction between the quarks and (eight) gluons (color charged particles) is the quantum chromodynamics sector [18]. The QCD conserves the separately conserved color charges red, green and blue, and thus works in a three dimensional color space. Another quantity which is conserved in QCD is parity. This comes from that the QCD interaction Hamiltonian is invariant under parity transformations (sect. 11.2.2 in Thomson [5]). The antiquarks carry the opposite color charge to the quarks of red, green and blue. The color states consists of color isospin and color hypercharge. It also ensures invariance under the local gauge transformation. The gauge symmetry group for this sector is $SU(3)_C$ and is represented by 3×3 matrices, where the C stands for the conserved color. This symmetry group does not commute and is a non-Abelian gauge theory, or more precise it is a Yang-Mills gauge theory [19]. By using this gauge theory, we can derive a new invariant Lagrangian which does not have a mass term for the gluons:

$$\mathcal{L}_{QCD} = \bar{\psi}(i\gamma^\mu \partial_\mu - m)\psi - \frac{1}{2} g_s \bar{\psi}\gamma^\mu \lambda_a \psi G^a_\mu - \frac{1}{4} G^a_{\mu\nu} G^{\mu\nu}_a \tag{2.7}$$

ψ is a fermion (quark) field, g_s is a coupling constant of the strong interaction, γ^μ are Dirac matrices, $a = 1, ..., 8$ are the eight gluons, λ_a is one of the eight

Gell-Mann matrices and $G^a_{\mu\nu}$ is a gauge invariant gluon field strength tensor. The last term of the Lagrangian in equation 2.7 implies that the gluons should be massless and can self-interact.

In Figure 2.3 we see the QCD vertices for quark and gluon interactions (and self-interacting gluons).

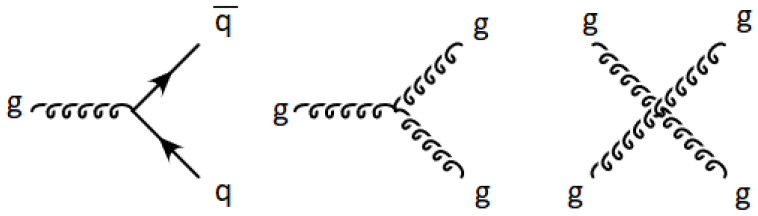


Figure 2.3: Here we see Feynman diagrams of the basic QCD vertices. From left to right we see, the coupling of gluon fields (g) interaction with quark fields (q), a triple gluon vertex and a quartic gluon vertex. Source Fig. 10.1 in Thomson [5].

Quantum electrodynamics (QED)

The gauge theory that defines the electromagnetic interaction for the electrically charged particles and photons is the quantum electrodynamics sector [20]. The QED conserves the electric charge of the particles. Like in QCD, parity is conserved in QED (sect. 11.2.2 in Thomson [5]). The gauge symmetry group for QED is $U(1)$ which is an Abelian group. By starting with a free fermion field for the Lagrangian (eq.2.5, invariant under *global* $U(1)$ transformation) and require invariance under a local phase transformation, leads to a Lagrangian with a Lorentz-invariant description where there is an electromagnetic interaction between fermions and the gauge field of the massless photon:

$$\mathcal{L}_{QED} = \bar{\psi}(i\gamma^\mu \partial_\mu - m_e)\psi + e\bar{\psi}\gamma^\mu \psi A_\mu - \frac{1}{4}F_{\mu\nu}F^{\mu\nu}. \tag{2.8}$$

ψ is the field of the spin half particles, e is a coupling constant of the electromagnetic interaction, γ^μ are Dirac matrices, A_μ is a covariant four-potential (gauge field), and $F_{\mu\nu}$ is the electromagnetic field strength tensor.

From the Lagrangian in equation 2.8, we can construct the Feynman diagram of a QED interaction vertex between a single photon and two spin-half fermions, seen in Figure 2.4.

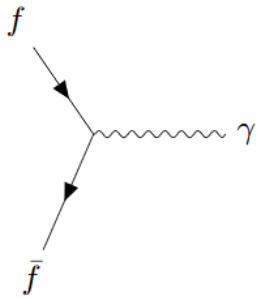


Figure 2.4: A Feynman diagram of the basic QED vertex for the interaction between fermions (f) and a massless photon (γ). Source Fig. 5.6 (and 10.10a) in Thomson [5].

Electroweak theory (EWT)

The gauge theory which defines the weak interaction for the 3rd component of isospin particles and the W and Z bosons/fields is the unified theory known as electroweak theory (EWT) [21] or Glashow-Weinberg-Salam (GWS) theory. This theory (from the 1960's) earned the three contributors Glashow[22], Weinberg[23] and Salam[24] the Nobel Prize in Physics in 1979 [25][26].

Unlike QCD and QED, it is found experimentally that parity is not conserved in the weak interaction (sect. 11.2.3 in Thomson [5]). This parity-violation makes the weak interaction treat left-handed and right-handed particles differently. The charge-current weak interaction is invariant under $SU(2)$ local phase transformations and includes weak isospin. The cross-section of W-pairs produced at higher energies, violates quantum mechanical unitarity such that particle probability is no longer conserved. This is solved because the couplings of the γ (QED), $W^{\pm}$ and Z EWT are related to each other in the unified electroweak model.

In the EWT theory fermions exists as left- and right-handed chirality states, while W-bosons only couple to left-handed fermions. The EWT conserves the flavor charge and weak isospin of the particles. It is the weak isospin quantum number that accounts for the W-boson coupling, since left-handed fermions have half-isospin and appears as isospin doublets while right-handed fermions appear as isospin singlets. Something to take notice of here is that, the weakly interacting quarks are superpositions of the mass eigenstates while the strongly interacting quarks are mass eigenstates.

The electroweak theory is based on the $SU(2)_L \times U(1)_Y$ symmetry group, where L is left-handed interaction and Y is the weak hypercharge expressed by the electric charge Q and the third component of the weak isospin I_3, $Y = 2(Q - I_3)$. This new $U(1)_Y$ local gauge symmetry is used instead of that

in QED, where the charge now has been replaced by the weak hypercharge. Each gauge invariant transformation in this theory, introduce new gauge fields which as linear combinations corresponds to the photon and the W and Z bosons of the weak interaction. With these new gauge fields, we can derive yet another new preliminary (electroweak) Lagrangian that is associated with the EWT theory:

$$\begin{aligned}\mathcal{L}_{EWT} =&\bar{\psi}_L\gamma^\mu\left[i\partial_\mu - \frac{1}{2}g\boldsymbol{\sigma}\mathbf{W}_\mu - \frac{1}{2}g'YB_\mu\right]\psi_L + \bar{\psi}_R\gamma^\mu\left[i\partial_\mu - \frac{1}{2}g'YB_\mu\right]\psi_R \\ &- \frac{1}{4}B_{\mu\nu}B^{\mu\nu} - \frac{1}{4}\mathbf{W}_{\mu\nu}\mathbf{W}^{\mu\nu}\end{aligned} \tag{2.9}$$

$\psi_{L,R}$ are the fields for left- and right-handed fields respectively, g and g' are coupling constants related to the elementary charge, γ^μ are the Dirac matrices, $\boldsymbol{\sigma}$ are the Pauli matrices, B_μ is a field strength tensor for the weak hypercharge gauge field for $U(1)_Y$, $\mathbf{W}_{\mu\nu}$ is a field strength tensor for the three weak isospin gauge fields for $SU(2)_L$.

The EWT gauge symmetry group is non-Abelian. In Figure 2.5 and 2.6 we see Feynman diagrams of the electroweak interaction vertices including fermions and gauge boson self-interactions. The photon and the Z-boson couple with both left- and right-handed fermions, while the W-bosons do not.

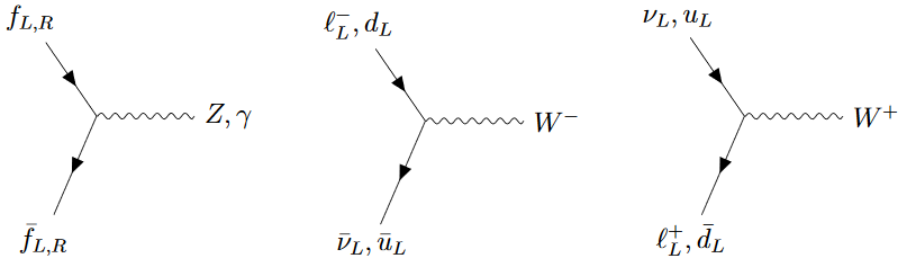


Figure 2.5: Here we see Feynman diagrams of the electroweak interaction vertices that includes fermions.

By introducing the BEH mechanism we get, in addition to the coupling in Figure 2.5 and 2.6, couplings between the Higgs boson and the massive gauge boson as well as Higgs self-interaction. These couplings can be seen in Figure 2.7.

Fermion masses: The Higgs mechanism can also be used to give masses to the fermions. The Higgs isospin doublet has a lower and an upper element. The lower element is used to give masses to down-type quarks and charged leptons, while the masses of the up-type quarks are constructed from the

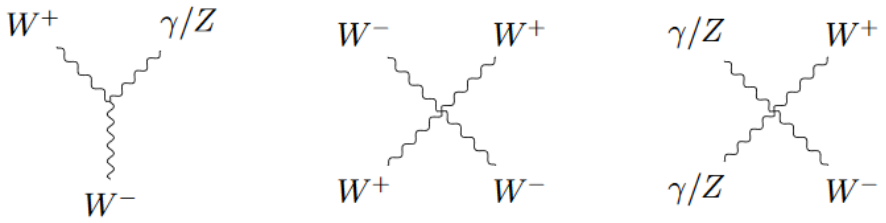


Figure 2.6: Here we see Feynman diagrams of the electroweak interaction vertices for gauge boson self-interaction.

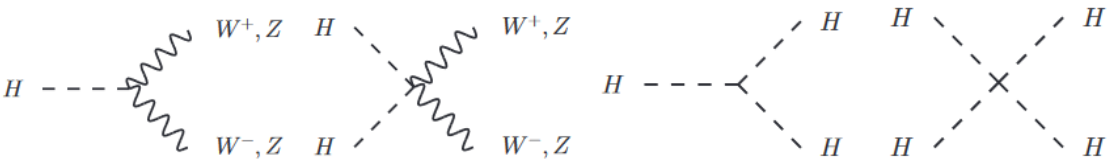


Figure 2.7: Here we see Feynman diagrams of the couplings between the Higgs boson and the massive gauge bosons and Higgs self-interaction.

conjugate doublet. The gauge invariant mass terms of the Dirac fermions are then described as

$$m_f = \frac{g_f v}{\sqrt{2}}, \tag{2.10}$$

where g_f is the Yukawa coupling constant of the fermions to the Higgs field, as shown in Figure 2.8.

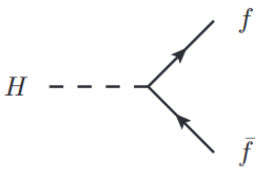


Figure 2.8: Here we see Feynman diagrams of the coupling between the Higgs boson and fermions.

Full EWT Lagrangian

The complete Lagrangian for the EWT is given by:

$$\begin{aligned}\mathcal{L}_{EWT} =& \overline{\psi}_L \gamma^\mu \left[i\partial_\mu - \frac{1}{2} g \boldsymbol{\sigma} \mathbf{W}_\mu - \frac{1}{2} g' Y B_\mu \right] \psi_L + \overline{\psi}_R \gamma^\mu \left[i\partial_\mu - \frac{1}{2} g' Y B_\mu \right] \psi_R \\ & - \frac{1}{4} B_{\mu\nu} B^{\mu\nu} - \frac{1}{4} \mathbf{W}_{\mu\nu} \mathbf{W}^{\mu\nu} + \left| \left(i\partial_\mu - \frac{1}{2} g \boldsymbol{\sigma} \mathbf{W}_\mu - \frac{1}{2} g' Y B_\mu \right) \phi \right|^2 \\ & - V(\phi) - (g_f \overline{\psi}_L \phi \psi_R + G'_f \overline{\psi}_L \phi_c \psi_R + h.c.) \end{aligned} \tag{2.11}$$

The first line is the couplings between the fermions and the gauge fields and kinetic terms for the fermion fields. The second line is the kinetic terms for the gauge fields and the Higgs field, the couplings between the gauge field and the Higgs field, and the couplings between the gauge fields. The third line contains the scalar potential, the Yukawa coupling terms and the fermion mass terms, and $h.c$ stands for the corresponding Hermitian conjugate.

Chapter 3

Neutrinos Beyond the Standard Model

The SM explains most of the physics we measure in experiments. The SM has several free parameters which are chosen to match observations. Nevertheless, the SM does not explain everything. For theorists the ultimate goal is to construct a Theory of everything, which explain all the physical phenomena in a unified way (including also gravity). Particle physicists try to address the shortcomings of the SM by extending it and construct more complete models which can explain e.g. gravity or the masses of the neutrinos.

A major problem in today's particle physics is that the SM can only explain about 5% of the total energy density of the Universe. This 5% of the matter in the Universe is called baryonic matter, while the rest is something yet unknown. One theory is that about 25% is something called dark matter, that acts as matter, but we can't see it, and has a gravitational pull in the Universe. The remaining 70% is then thought to be dark energy, which has a pushing affect on the galaxies in the Universe making it expanding faster and faster with time. A dark matter candidate is neutrinos. We will not go into the dark matter aspect, but look closer at neutrinos and the neutrino masses.

From the discovery of neutrino oscillations, we know from observations and experiments, that the neutrinos need to have mass since they have the ability to change flavor over very large distances. Why the neutrinos have mass and what gives them mass, on the other hand, are not explained in the SM. The only place in the SM that allows CP-violation, is in the weak interaction domain where left-handed neutrinos are affected through neutrino mixing. Since it is not observed right-handed neutrinos nor left-handed antineutrinos, C- and P-symmetry should be violated. It has not yet been observed if CP-violations occur in neutrino oscillations, since neutrinos seem

to uphold the CP-symmetry with the existence of right-handed antineutrinos. This means that some new physics is required to explain this breaking of CP-violation.

For the neutrinos to acquire mass, we have to go beyond the SM neutrino knowledge, and introduce some new theories. We will look more into the neutrino masses and the model for this thesis in this chapter.

3.1 Neutrino Masses

According to the SM, neutrinos do not have mass because only left-handed (LH) neutrinos are covered by the SM and thus, right-handed (RH) neutrinos are not involved in any of the fundamental interactions and have not yet been observed. As mentioned earlier, we know from observations and experiments of neutrino oscillations that neutrinos have a tiny, but non-zero mass. to being able to change flavor when moving over large distances. The neutrino masses are something we need to look more into.

3.1.1 Dirac Neutrinos

Dirac particles are particles which can be distinctively separated from its antiparticle. The Dirac field is described by a four-component Dirac spinor ψ and can be divided into a left-handed ψ_L and a right-handed ψ_R part as two component Weyl spinors:

$$\psi = \begin{pmatrix} \psi_L \\ \psi_R \end{pmatrix}. \tag{3.1}$$

The left-handed neutrinos in the SM are described by this left-handed Weyl field. Since the Dirac mass term require both left- and right-handed fields in the SM, there is no Dirac mass term for the neutrinos.

If we assume neutrinos as Dirac particles, the neutrino mass is added similarly to the up-type quarks as the conjugate Higgs doublet. The gauge invariant Dirac neutrino mass term after spontaneous symmetry breaking becomes

$$\mathcal{L}_D = -m_\nu(\overline{\nu_R}\nu_L + \overline{\nu_L}\nu_R), \tag{3.2}$$

with the neutrino mass still determined by the Yukawa coupling constant as for Dirac fermions (eq. 2.10):

$$m_\nu = \frac{g_\nu v}{\sqrt{2}} \tag{3.3}$$

The neutrino masses have been found to be several orders of magnitude smaller than the charged lepton masses. This leads to a Yukawa coupling constant $g_\nu \leq 10^{-12}$ for neutrino masses that are less than 1.1 eV (sect. 2.2.1). There are no reasons why the Yukawa constants should be so small, which gives reason to believe that there must be some other mechanism giving neutrinos their masses. The right-handed neutrino in the SM would be sterile and only interact with the Higgs boson.

3.1.2 Majorana Neutrinos

Another option for the neutrinos, is that they can be Majorana neutrinos. This means that they can be their own antiparticles. The result of this would mean that the lepton number no longer is conserved, which it is in the SM. To not break the gauge invariance of the SM when adding the fields for RH neutrinos and LH antineutrinos in the Lagrangian, the LH antineutrinos appear as the CP conjugate field of the RH neutrino[5] defined by

$$\psi_L^c = \hat{C}\hat{P}\psi = C\overline{\psi}_R^T, \tag{3.4}$$

where C is the charge conjugation matrix.

For a Majorana neutrino we have $\psi^c = \psi$, which means that the neutrino field can be expressed with a Majorana spinor

$$\psi_\nu = \begin{pmatrix} \overline{\nu_R^c} \\ \nu_R \end{pmatrix} \tag{3.5}$$

for LH and RH neutrino fields and the CP conjugate of the RH field (or the LH antineutrino) $\overline{\nu_R^c}$. The local gauge invariant Majorana neutrino mass term, with Majorana mass M, becomes

$$\mathcal{L}_M = -\frac{1}{2}M(\overline{\nu_R^c}\nu_R + \overline{\nu_R}\nu_R^c). \tag{3.6}$$

This means that the Majorana mass term is not constrained by gauge symmetry and can be arbitrary large. The global baryon number minus the lepton number $(B-L)$ symmetry of the SM would be broken if the neutrino is a Majorana neutrino. From observations of the asymmetry between matter and antimatter in the Universe, it actually looks like the baryon number is not conserved.

A generic Majorana mass matrix, $\mathcal{M}$, with three neutrinos can also be expressed as

$$\mathcal{M} = \begin{pmatrix} M_L & m_D \\ m_D^T & M_R \end{pmatrix} \tag{3.7}$$

m_D is the mass for a Dirac neutrino, M_L is the Majorana mass for a LH neutrino (ν_L) and M_R is the Majorana mass for a RH neutrino (ν_R).

3.1.3 Pseudo-Dirac Neutrinos

A pseudo-Dirac neutrino[27][28] mass matrix is similar to the Majorana mass matrix in equation 3.7, except that the M_L and M_R masses are the lepton number violating Majorana masses of light neutrinos[1]. When the Dirac mass is $m_D >> M_L, M_R$, we get a pseudo-Dirac mass matrix where the eigenvalues of the resulting mass eigenstates are close to each other. This means that the two light neutrinos can form a Dirac-like/pseudo-Dirac neutrino.

3.1.4 The Seesaw Mechanism

One of many theories for the light masses of the neutrinos is to add RH neutrinos that couple to the LH neutrinos. However, this would lead to a disparity problem regarding mass scale. To solve this, a seesaw mechanism is introduced where the observed (light Dirac) LH neutrinos couple with very heavy (sterile) Majorana RH neutrinos. This would explain the small masses of the observed SM left-handed neutrinos and the absence of observation of RH neutrinos. The problem is that the mass scale of the RH neutrinos is unknown, since the masses of the Dirac neutrinos are still uncertain. So they could be somewhere between a few keV, and possibly be light dark matter particle candidates, or have higher masses near the unification energy (GUT scale), where the electromagnetic, weak and strong forces have equal strength.

Type-I seesaw mechanism

There are several varieties of the seesaw mechanism which extends the SM, but the simplest one is the **Type-I seesaw mechanism**[29]. This involves the mix of LH Dirac neutrinos and RH Majorana neutrinos. In this theory, a right-handed neutrino is added for each of the SM LH neutrinos, in total three. When involving neutrinos as Majorana, we get that $\overline{\nu_L}\nu_R$ is equivalent to $\overline{\nu_R^c}\nu_L^c$. The Lagrangian after the spontaneous electroweak symmetry breaking with both the Dirac and Majorana mass terms becomes:

$$\mathcal{L}_{DM} = -\frac{1}{2}\left(m_D\overline{\nu_L}\nu_R + m_D\overline{\nu_R^c}\nu_L^c + M\overline{\nu_R^c}\nu_R\right) + h.c. \tag{3.8}$$

m_D is the Dirac mass and M is the Majorana mass. This seesaw mechanism is characterized by $M_L << m_D << M_{(R)}$. This equation can also be written in terms of a 2×2 mass matrix ($\mathcal{M}$) for the neutrinos:

$$\mathcal{L}_{DM} = -\frac{1}{2}(\overline{\nu_L}\,\overline{\nu_R^c}) \begin{pmatrix} 0 & m_D \\ m_D & M \end{pmatrix} \begin{pmatrix} \nu_L^c \\ \nu_R \end{pmatrix} + h.c. \tag{3.9}$$

[1] A RH neutrino is also called a sterile neutrino, ν_s.

By looking at the eigenvalues (λ) of the mass matrix $\mathcal{M}$ we get the physical masses of the neutrinos (in this model) as (sect.17.8.1 in Thomson [5])

$$m_{\pm} = \lambda_{\pm} = \frac{M \pm M\sqrt{1 + 4m_D^2/M^2}}{2}. \tag{3.10}$$

If we assume the Majorana mass much larger than the Dirac mass, $M >> m_D$, we get a light LH neutrino state (ν) and a heavy RH neutrino state (N) with masses

$$|m_\nu| \approx \frac{m_D^2}{M} \quad \& \quad m_N \approx M. \tag{3.11}$$

The physical neutrino states are in this case

$$\nu \approx (\nu_L + \nu_R) - \frac{m_D}{M}(\nu_R + \nu_R^c) \quad \& \quad N \approx (\nu_R + \nu_R^c) + \frac{m_D}{M}(\nu_L + \nu_L^c). \tag{3.12}$$

By looking at equation 3.11, we see that the lightness of the SM neutrinos are explained by the existence of much heavier right-handed neutrinos.

Inverse seesaw mechanism

The model we will be studying in the following section involves a slightly different seesaw (ISS) theory, namely the so-called **Inverse seesaw mechanism** [4][3]. This is a low-scale Type-I neutrino mass model and yields heavy neutrino masses and allows large Yukawa couplings. While the ordinary (Type-I) seesaw predict very heavy RH neutrinos ($\sim 10^{14}$ GeV), from the ISS predicts TeV-scale RH neutrinos. Masses of 10^{14} GeV is out of range for experiments, which is not so attractive.

Besides the addition of three right-handed neutrinos, this model also adds three LH singlet fermions as well as three light LH neutrinos. These three added particle "groups" make a 3×3 matrices for each group. The ISS Lagrangian is a 9×9 matrix given as:

$$\mathcal{L}_{\text{ISS}} = -\nu_L m_D N_R - S_L M N_R - \frac{1}{2}\bar{S}_L \mu S_L^c + h.c. \tag{3.13}$$

ν_L is the (SM) LH neutrino, N_R is the RH neutrino, S_L is a new light singlet neutrino and μ is a lepton violating parameter ($\mu << m_D, M$). The light neutrino mass matrix can be written as a 3×3 matrix:

$$m_\nu = m_D^T (M^T)^{-1} \mu M^{-1} m_D.^{2} \tag{3.14}$$

These nine neutrinos form three heavy pseudo-Dirac neutrino pairs with small lepton number violations in the singlet mass terms. This comes from

[2]Scales: $m_\nu \sim$ eV, $m_D \sim$ eV, $\mu \sim$ keV, $M \sim$ TeV.

the decay of a $W_R^{\pm}$ to a pseudo-Dirac neutrino, since a neutrino coupled to a $W_R^{\pm}$ is a pseudo-Dirac fermion. It is during this process that the lepton number is approximately conserved, and accounts for missing same-sign electron events.

Our base model is the $SU(2)_L \times SU(2)_R \times U(1)_{B-L}$ left-right symmetry group which involves the ISS mechanism, and is based on the

$$SU(3)_C \times SU(2)_L \times SU(2)_R \times U(1)_{B-L} \tag{3.15}$$

gauge symmetry. The main difference from the Type-I seesaw mechanism is that instead of a heavy Majorana mass eigenstate neutrino, we have a heavy pseudo-Dirac neutrino mass eigenstate. The mass difference (mixing) between the left- and right-handed neutrinos probe small neutrino masses. This leads to Left-Right symmetric models with the same final state as for a heavy Majorana neutrino.

3.2 The Charge Current Drell-Yan Process

The model in this thesis is based on the works of Pascoli et al. [4] with the inverse seesaw mechanism. Here, two protons are accelerated and collided to produce a heavy pseudo-Dirac neutrino, and a left-right symmetric model. Since the inverse seesaw mechanism allows a large left-right neutrino mixing, while keeping the neutrino masses tiny, the W boson may decay into a charged lepton l and a heavy pseudo-Dirac neutrino N_m. The pseudo-Dirac neutrino then decays into another lepton with opposite sign and another W, which then decays into another lepton and MET/a (light) neutrino:

$$q\overline{q'} \to W^{\pm(*)} \to l_1^{\pm} N_m \to l_1^{\pm} l_2^{\mp} W^{\pm(*)} \to l_1^{\pm} l_2^{\mp} l_3^{\pm} \bar{\nu}_l \tag{3.16}$$

The final state is then three charged leptons (trilepton) plus a neutrino which goes undetected through the detector, and is observed indirectly through large missing transverse energy in the event (like ATLAS). This decay process can be seen in Figure 3.1, and is produced through the charged current Drell-Yan (CCDY) process [4]. In this model, the lepton number is almost conserved. This is set by the mixing parameter μ. For the mixing of N1's couplings to electrons and muons, the mixings are set to $\mu = |V_e N| = |V_\mu N| = \frac{1}{\sqrt{2 \cdot 10^{-2}}}$ and $|V_\tau| = 0$ for no mixing to tau in the simulation models for charged lepton flavor violation (LFV)[3]. This means that the amount of opposite-sign and same-sign events for the first two leptons may differ from e.g. the normal seesaw model. The mixings allow LFV between vertex 1 and 2, i.e. an

[3] Equation 3.18 in Pascoli et al. [4].

electron at vertex 1 and a muon at vertex 2 or vice versa, while the W boson decays according to the SM. As seen in equation 3.16, the leptons in the lepton pairs 1 and 2 and 2 and 3 must always have opposite sign (OS) while 1 and 3 always have same sign (SS).

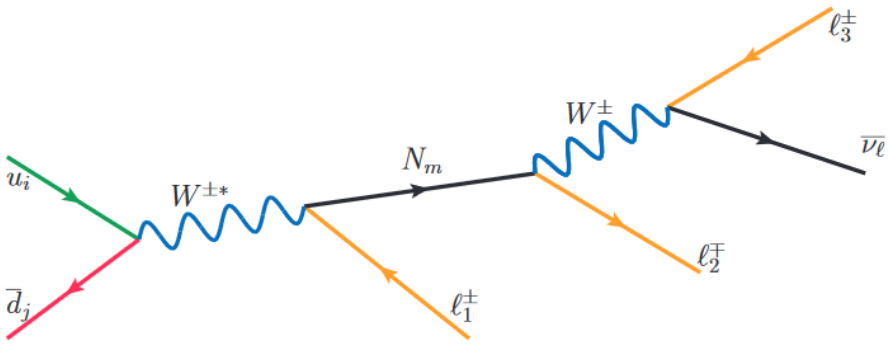


Figure 3.1: The Born diagram for the charged current Drell-Yan process of the proton-proton collision producing a heavy pseudo-Dirac neutrino N in the inverse seesaw mechanism model, leading to a trilepton plus missing transverse energy (a light neutrino) final state. Figure is taken from ref. [4].

The decay products of such particle collisions can be detected in experiments like the LHC and ATLAS (sect. 5.3). These events can also be simulated, meaning that we can simulate proton-proton collision events and the decay processes. For each decay final state product, we can measure many properties like momentum, the transverse momentum, the polar angle and the azimuthal angle. We can also detect which final state particles are produced. With these particle properties we can calculate the angles and angular distances between each produced particle, and for truth we have all the information about the neutrino (MET). In a real detector, we only have the transverse information[4]. We can also calculate the invariant masses of pairs of combined final state particles. We should then be able to find out which lepton comes from which decay branch in the decay process in Figure 3.1 computationally.

The end goal is to identify each lepton, and decay vertex (according to Fig. 3.1), by utilizing the particle properties in various machine learning algorithms. We will look more into machine learning in chapter 6. The computational setup and input parameters for the CCDY process are covered in chapter **??**.

[4]I.e. no p_z and no θ.

Chapter 4

Proton-Proton Collisions

In this thesis we study the proton-proton (p-p) collisions from LHC (sect.5.2). Protons consists of quarks and this makes proton-proton collisions somewhat complex. When two hadrons collide, it is the constituents of the hadrons[1] which collide. The colliding partons only carry fractions of the total momentum of the protons. We use the center-of-mass (CM) frame of the p-p collision system and not the CM frame of the patrons that collide. This chapter explains the basics of high energy proton-proton collisions.

4.1 Particle Kinematics

To describe the kinematics of what happens in p-p collisions, we need the momentum, energy and rest mass of the particles. The Einstein energy-momentum relation in natural units becomes

$$E^2 = p^2 + m^2. \tag{4.1}$$

Since the protons will reach very high velocities when they collide, we need to include special relativity into the equations[2]:

$$E = \gamma m \quad \text{and} \quad \mathbf{p} = \boldsymbol{\beta}\gamma m \tag{4.2}$$

These equations depend on the Lorentz factor

$$\gamma = \frac{1}{\sqrt{1-\beta^2}} \quad \text{and} \quad \beta = \frac{v}{c}.$$

[1] The partons, i.e. quarks and gluons.

[2] In natural units.

We then introduce the momentum as a four-vector momentum

$$P^\mu = (E, \mathbf{p}) = (E, p_x, p_y, p_z).$$

The scalar product of the four-momentum is then a Lorentz-invariant quantity

$$\begin{aligned} P^2 = P^\mu P_\mu &= E^2 - \mathbf{p}^2 \\ &= \gamma^2 m^2 - \beta^2 \gamma^2 m^2 \\ &= m^2, \end{aligned} \tag{4.3}$$

since the momentum and energy are conserved separately, the four-momentum is also conserved. By rearranging this equation, we just end up with the Einstein energy-momentum relationship in equation 4.1. This is a very useful relation in particle collisions.

4.1.1 Colliding Particles

The reference frame of choice for colliding particles, is as mentioned the CM frame of the two colliding particles. This is defined where the sum of the three-momenta $\mathbf{p}$ is zero. When two particles collide, this means that $\mathbf{p}_1 = -\mathbf{p}_2$. And when these two particles have the same rest mass $E_1 = E_2 = E$, we get

$$(P_1 + P_2)^\mu = (2E, \mathbf{0}). \tag{4.4}$$

Now we introduce what is called a Mandelstam variable[5], s , which is defined as the squared sum of the four-momenta

$$s = (P_1 + P_2)^2. \tag{4.5}$$

This we have already found out is a Lorentz-invariant quantity. We can then draw two conclusions; 1) s is a Lorentz-invariant quantity as well, and 2), the $\sqrt{s} = 2E$ can be interpreted as the total energy of the CM system. This is a key quantity in particle physics for particle colliders.

From equation 4.3, we got that $P^2 = m^2$. This means that if the colliding particles were elementary particles, $\sqrt{s}$ could be interpreted as the possible energy available for heavier particle production. This would then be an upper limit for producing a heavy particle with mass M, as $M \leq \sqrt{s}$. But since protons are not elementary particles and the p-p collisions are really collisions between partons, this limit changes. We denote the momenta carried by the two partons colliding as $\mathbf{q}_1$ and $\mathbf{q}_2$. The associated four-momenta for the partons are Q_1^μ and Q_2^μ. Since we mentioned that the partons only carry

fractions of the momenta, these fractions will be defined as x_1 and x_2 for the two colliding partons. By using what is called the Drell-Yan process[3] (explained and derived in Thomson [5]) for a quark and an antiquark, we get the fractions given as

$$x_1 = \frac{q_1}{E} \quad \text{and} \quad x_2 = \frac{q_2}{E}. \tag{4.6}$$

To get the mass M of a produced particle from the collision with the partons, we use the same limit as for an elementary particle collision and equation 4.5 for s:

$$\begin{aligned} M &\leq \sqrt{s} \\ M^2 &\leq s \\ M^2 &\leq (Q_1 + Q_2)^2 = E^2 \left[(x_1 + x_2)^2 - (x_1 - x_2)^2\right] \\ &= 4x_1 x_2 E^2 \\ &= x_1 x_2 s \end{aligned}$$

This leads to that the produced invariant mass is equal to the CM energy of the colliding partons.

The actual values of the fractions are described by the parton distribution functions (PDFs). These PDFs can be interpreted as the probability of a parton with a special flavor to carry the fraction x of the proton momentum when the parton participates in a hard scattering process.

From this section, we can see that the event kinematics in hadron-hadron collisions have to be explained by the three independent kinematic variables, Q^2, x_1 and x_2.

4.1.2 Products of Particle Collisions

In particle colliders, like at the LHC, the direction of the particle beams are normally defined in the z-direction which gives $\mathbf{p} = (0, 0, p)$. This plane is the longitudinal plane. The positive y-direction is defined upwards, and the positive x-direction is defined towards the center of the ring. We can then define the transverse momentum p_T perpendicular to the z-axis as

$$p_T = \sqrt{p_x^2 + p_y^2}. \tag{4.7}$$

The corresponding transverse energy is given as

$$E_T = \sqrt{p_T^2 + m^2}. \tag{4.8}$$

[3]This is not restricted to Drel-Yan processes, but yields for any 2-¿1 process.

The total momentum can then be derived as

$$p = \sqrt{p_T^2 + p_z^2}. \tag{4.9}$$

The reason for working in the transverse (xy) plane of the initial beam direction, is that the initial momentum is zero in this direction. We want to express the kinematics in spherical coordinates in terms of the polar angle θ and the azimuthal angle ϕ.

After the collisions, not just the parton jets, but the whole system will get a boost along the beam direction. That is why we introduce a *rapidity* variable y that is used to express the jet angles:

$$y = \frac{1}{2} \ln \left(\frac{E + p_z}{E - p_z} \right) \tag{4.10}$$

What is useful with this rapidity variable, is that the rapidity differences are invariant under Lorentz boosts along the beam direction. This does not apply for the polar angle θ.

If the particle mass is small compared to the particle energy, $p_z \approx E \cos\theta$. We can then rewrite the rapidity as

$$y \approx \frac{1}{2} \ln \left(\frac{1 + \cos\theta}{1 - \cos\theta} \right) = \frac{1}{2} \ln \left(\cot^2 \frac{\theta}{2} \right) = -\ln \left(\tan \frac{\theta}{2} \right) \equiv \eta \tag{4.11}$$

This new variable η is called the *pseudorapidity*. The pseudorapidity also has the following relation with the polar angle: $\eta(\theta) = -\eta(180^\circ - \theta)$. We now have the most used set of variables (p_t, ϕ, θ) for describing the kinematics of particles in a detector. In Figure 4.1 we see the illustration of the transverse and longitudinal planes. The cylindrical shape shows how particle accelerators will be situated around the collision point.

Another useful variable associated with hadron colliders, is the angular distance between two particles

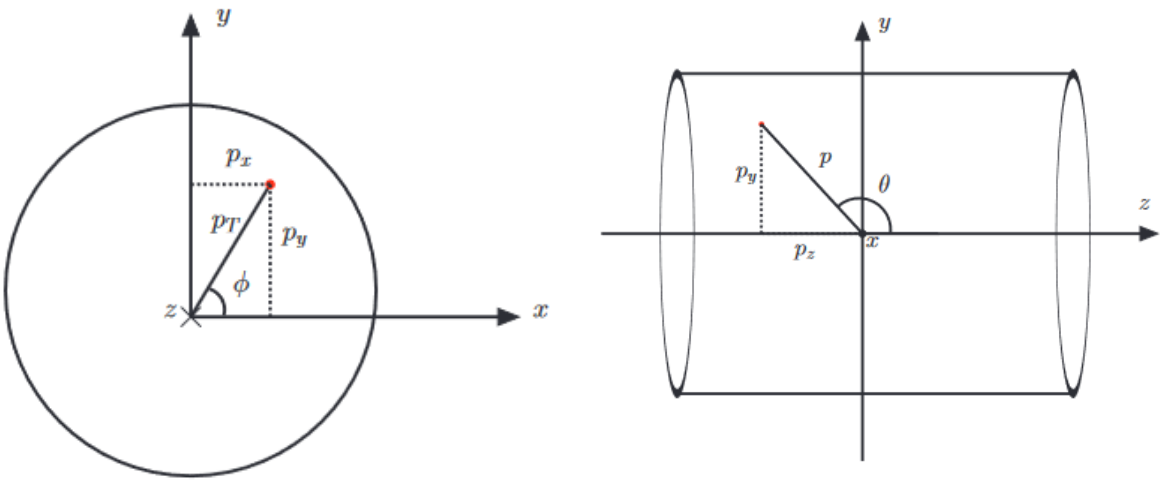
$$\Delta R = \sqrt{(\Delta\eta)^2 + (\Delta\phi)^2}. \tag{4.12}$$

The angular distance defines how much two particles are moving in the same direction or as the separation in the $\phi\eta$-space, and is invariant under longitudinal boosts.

4.2 Proton-Proton Interactions

When proton-proton collisions take place in colliders, the interactions can roughly be divided into three groups:

i) elastic (el) ii) diffractive (di) iii) non-diffractive (nd)



(a) The transverse plane.

(b) The longitudinal plane.

Figure 4.1: Illustrations of the (a) transverse plane and the (b) longitudinal plane. The collision point is at the origin. Figures are both from ref. [30].

These three groups are also components that make up the total cross-section at proton-proton colliders:

$$\sigma_{\text{total}} = \sigma_{el} + \sigma_{di} + \sigma_{nd} \tag{4.13}$$

For elastic processes, both the colliding protons remain unchanged. For the diffractive processes (di and nd), the collisions/interactions are inelastic and one or both protons will be fragmented. This leads to multi-particle final states.

The elastic and diffractive interactions have cross-sections that can not be calculated using perturbation theory, meaning they are non-perturbative processes. In these cases we get so-called *pomerons*, which are color singlet states that do not exchange color between the protons. These interaction processes at high-p_T proton-proton collisions are normally not interesting, since they will produce particles with low transverse momentum close to the beam line. They are thus difficult to detect, but important for luminosity measurements since they contribute to the total p-p cross-section. These events are detected in special experiments that use *minimum bias* events, where the final state has no requirements or special triggers.

4.2.1 Hard Scattering Events

The more interesting events to look at in high-p_T p-p collisions, are the non-diffraction events. With non-diffractive events, there is an exchange of color between the partons in the interaction. These are called hard scattering events. Hard scattering events with high momentum transfers, Q^2, may create heavy particles. This is the main interest in particle colliders.

A hard scattering event can be expressed as

$$A + B \to c + X, \tag{4.14}$$

where the collision between the partons are expressed as

$$a + b \to c. \tag{4.15}$$

A and B are the two colliding protons, and a and b are the corresponding colliding partons. c are the interesting high p_T objects. X are underlying products which are mostly remnants after the original collision.

In Figure 4.2 we see how a hard scattering p-p collision may look like, with outgoing partons, underlying events, initial- and final-state radiation. The initial-state radiation is mean radiation of gluons or photons from partons before the hard scattering. Final-state radiation is the mean radiation from the produced partons after the hard interaction. The underlying events are the further interactions between partons beyond the hard scattering. These interactions will often go out of reach of the detector, and is another reason why we look at the transverse plane.

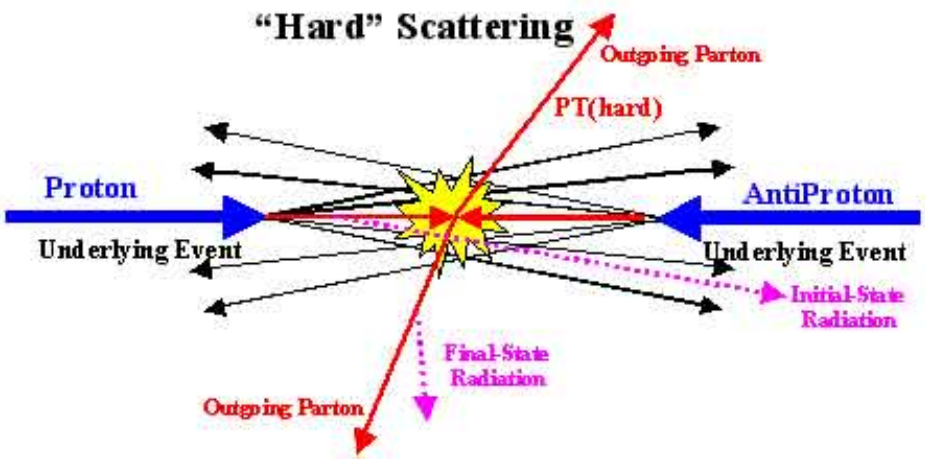


Figure 4.2: Illustration of a hard scattering proton-proton collision. Figure is taken from ref. [31].

4.2.2 Parton Distribution Function

The parton distribution function (PDF)[4] is used to describe the probability density of the two partons, a in proton A and b in proton B, to carry the proton momentum fractions x_a and x_b. These PDFs are also dependent on the squared of the momentum scale indicating the total four-momentum transfer in the collisions Q^2 as $F_{a/A}(x_a, Q^2)$ and $F_{b/B}(x_b, Q^2)$. These PDFs must be found experimentally in Deep Inelastic Scattering (DIS) experiments of leptons against hadrons, since they cannot be calculated from QCD theory. The PDFs are also used to get the cross-section of the collisions.

With the measured PDFs $f(x, Q^2)$, a structure function $F_2^{ep}(x, Q^2)$ can be determined

$$F_2^{ep}(x, Q^2) = 2xF_1^{ep}(x, Q^2) = x \sum_i Q_i^2 f_i(x), \tag{4.16}$$

where i is a quark in the proton and Q_i is the charge of the quark. The interesting here are the $f(x)$ of each of the partons. So results of measurements from several DIS experiments of varying structure functions, which are superpositions of the same $f_i(x)$'s, are combined to get the $f(x)$ for each parton.

4.2.3 Hadronization

We already have covered that quarks and gluons carry color charge (sect. 2.1), and that they are not observed as free particles[5]. They can only be found in colorless objects like hadrons.

We also talked about the strong force, which increases in strength when increasing the distance between (elementary) particles. So if we separate a quark from a hadron, the color field will increase and the emerged energy will enable creation of new quark-anitquark pairs or gluons. These will be observed as jets of colorless particles. As this production of partons continue, the energy will decrease until it is low enough to produce hadrons. This process of high-energy quarks (and gluons) that produce new jets until we get hadrons, is called *hadronization*. The jets can also be called hadronic showers, since many hadrons are usually produced in hadronization processes.

Jets are not only produced in p-p collisions with hard scattering, but also in the underlying events and from initial- and final-state radiation. This makes p-p collisions very complicated and messy when trying to study them, compared to electron-positron collisions.

[4] See chapter 8 in Thomson [5] for more in depth explanations.

[5] Only exception is the top quark with shorter lifetime than the QCD interaction time scale.

Chapter 5

Particle Accelerators and Collider Experiments

To fully understand the physics of the particles around us and what the Universe is made of, we need some way of looking at the subatomic world. This is done in huge particle accelerators where particles are accelerated to high velocities and energies, and collided with each other to make other particles. Here the aftermath of the collisions result in new particles with new energies that are detected as they move through detectors.

There are various accelerators and detectors which produce and accelerate different particles in the world. In this chapter we will look at the biggest particle physics laboratory in the world, namely the European Organization for Nuclear Research (CERN[1]), and some of its components like particle accelerators and detectors.

5.1 CERN

The CERN laboratory lies near Geneva, on the border between France and Switzerland, and was founded in 1954 [32]. It is a multinational collaboration between 23 (mostly) European countries. They also have several international relations with other countries both inside and outside of Europe. CERNs main focuses today is particle physics and particle accelerator experiments. Many of the biggest discoveries in particle physics have come from particle experiments at CERN. This includes, among others, the discovery of the Higgs boson and discovery of the W and Z bosons. At the main site of CERN in Meyrin, and in the World LHC Computing Grid (WLCG) scat-

[1]The name CERN is originally from French; Conseil Européen pour la Recherche Nucléaire.

tered around the world, data of simulations of particle collisions are stored. CERN is the place where Tim Berners-Lee invented the World Wide Web in the late 1980s [33].

CERN consists of several particle accelerators, experiments and facilities in different shapes and sizes. The two main types of accelerators are linear and circular. They are located at various sites, and they accelerate particles to high energies before they send the particles to be collided with other accelerated particles or particles with stationary targets, or are sent to more powerful accelerators. They are built differently to accelerate different kinds of particles with different masses. In Figure 5.1 we see the CERN accelerator complex. Some of the accelerators are mostly used to pre-accelerate the particles before they are sent to another accelerator where they are accelerated even more. This repeats until the particles reach the desired energy to collide with at one of the detectors.

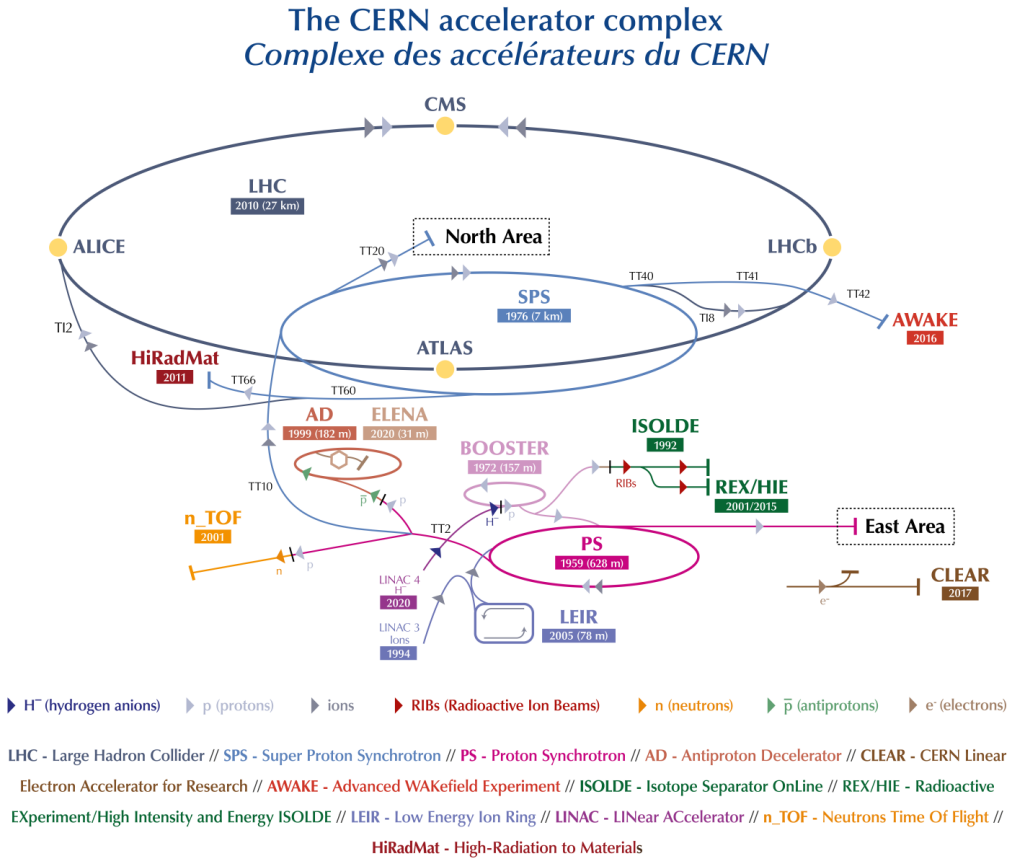


Figure 5.1: The CERN accelerator complex as of 2019. Credit: CERN[34].

For the more important discoveries, like the ones we have mentioned above, the W and Z bosons were discovered by the Super Proton Synchrotron (SPS) in 1983. The SPS delivered an energy between 300-450 GeV. It was then later used to accelerate high energy electrons and positrons into the Large Electron-Positron Collider (LEP). LEP is the largest and most powerful lepton collider built to this date, and was functional between 1989 and 2000. LEP was then replaced by the Large Hadron Collider (LHC) in 2008 to collide protons and heavy ions. We will look more into the LHC later (sect. 5.2).

There are also many plans for the future regarding both upgrades to existing accelerators and building new ones. The two biggest projects is the Compact Linear Collider (CLIC), which is a linear electron-positron collider at higher energies, and the Future Circular Collider (FCC), which is a even larger version of the LHC. This does not only apply at CERN, but several other places in the world like in China (CEPC [35] [36]) and Japan (ILC [37][38]).

5.2 The LHC and Accelerator Experiments

Today's largest and most powerful particle accelerator is the Large Hadron Collider (LHC) [39], which we easily can see in Figure 5.1 as the biggest gray circle around the North Area. The particles are sent in bunches up to 10^{11} protons and are accelerated using radio frequency cavities in a 27 km ring consisting of superconducting magnets, where the particles are boosted in several structures along the ring to the desired energies. The LHC is designed to have 2808 bunches at the same time traveling in the ring. The ring lies 100 m underground in a tunnel beneath the French-Swiss border. Along the ring, there are 4 main crossing points (ATLAS, CMS, ALICE, LHCb) with detectors that register the particle collisions and the following particle decays. At these collision points, the total collision energy, or center-of-mass energy $\sqrt{s}$, can reach 13 TeV[2]. There are in total seven detectors along the ring, each designed for different experiments.

The LHC was first used for proton-proton (hadron) collisions in 2010 (run 1), where it reached a record high energy of 3.5 TeV per beam. After upgrades, during run 2, it reached an even higher energy of 6.5 TeV per beam. It is currently stopped for another upgrade, which started in 2018 and is during operation. The accelerator sends two high-energy beams, in separate tubes and directions, near the speed of light before they collide at one of the detectors. To reach these high energies, the particle beams are accelerated

[2]The LHC is theorized to a limit of 14 TeV.

in several systems which increase the energies before injected into the main LHC ring [40]. Inside the tubes, there is an ultrahigh vacuum. To make sure that the particles are directed correctly through the ring, superconducting electromagnets are used to bend the particle trajectories. The magnets vary in strengths and sizes to direct the beams properly. Since the particles are incredibly tiny, the precision of the magnets have to be extremely good to make the particles hit each other at the collision points. That is also why beams of 10^{11} protons are accelerated and not single particles. Since the construction of the accelerator is a ring they can continue around again when some of them do not collide. A beam can typically go around in the ring for about 10 hours before the beam has lost too much intensity.

As mentioned earlier, there are seven detector experiments at the LHC [41]. The four main, and biggest, detectors in the LHC, have different objectives. The ATLAS and CMS experiments are two large and similar general-purpose particle detectors that looks for new physics and more precise study of the SM. The ALICE and LHCb experiments have more specific roles, and study the quark-gluon plasma from heavy ion collisions and missing antimatter connected to CP-violation after the Big Bang, respectively. The remaining detectors are much smaller and are used in more specialized research. We will look more at the ATLAS detector later (sect.5.3).

The LHC is used to explore many different open questions in physics, like to further study the SM and theories beyond it. In addition to proton-proton collisions, the LHC can also collide heavy ion collisions at some of the detectors.

5.2.1 Important Parameters

One of the most important parameters of measurements at particle accelerators, is the CM energy $\sqrt{s}$ we already have mentioned. For two particles colliding, the Lorentz invariant quantity s (the squared invariant mass) is formed as

$$s = \left(\sum_{i=1}^{2} E_i\right)^2 - \left(\sum_{i=1}^{2} \mathbf{p}_i\right)^2 . \tag{5.1}$$

There are also other important parameters used to describe the performance of particle colliders:

Luminosity

Another important parameter in particle collider performance is the *luminosity*, $\mathcal{L}$. The design luminosity of the LHC is $\mathcal{L} = 10^{34}$ cm^{-2}s^{-1}. The bunches

at the LHC are separated by 25 ns, which corresponds to a frequency of $f = 40$ MHz. The (instantaneous) luminosity is used to describe the number of collisions per area per second as[3]

$$\mathcal{L} = f \frac{n_1 n_2}{4\pi\sigma_x\sigma_y}, \tag{5.2}$$

where f is the frequency of the particle beam bunches colliding (bunch crossing rate), n_1 and n_2 are the number of particles in the colliding bunches and σ_x and σ_y are the root-mean-square (rms) horizontal and vertical beam sizes.

The complete collider luminosity at the LHC can be written in terms of colliding beam parameters [42]

$$\mathcal{L} = f \frac{n_1 n_2 n_b}{4\pi\sigma_x\sigma_y} F(\sigma_x, \sigma_y, \sigma_s, \Phi). \tag{5.3}$$

This equation has the same parameters as in equation 5.2, except for two additional parameters. n_b is the number of proton bunches. F is a geometrical reduction factor accounting for the non-zero-crossing angle at the interaction point, depending on the two rms beam sizes, the beam length σ_s and the crossing angle Φ.

Rate

The cross-section, σ, for a given collision process is given by the SM (or any other new model). The cross-section can be used to compute the (event) *rate*, R, after accumulating many such collisions. The rate is calculated as

$$R = \sigma\mathcal{L}. \tag{5.4}$$

Number of interactions

The total number of expected events of a given process with cross-section, σ, over a given time, is the time integration of the event rate

$$N = \sigma \int \mathcal{L} dt. \tag{5.5}$$

The time-integral of the luminosity, $\int \mathcal{L} dt$, is often called the *integrated luminosity*, and is given in inverse femtobarns $[fb^{-1}]$.

[3] With the assumption of Gaussian profile beams and head-on collisions.

Pile-up

In particle collisions, we want a high instantaneous luminosity. This means that the intensity of the proton beam need to be high. But with high intensity proton beams, the probability of having more than one proton undergoing an inelastic interaction per bunch crossing is increased. This leads to what is called *pile-up* events, where there are several collisions from the same bunch crossing. This means we need very accurate measurements in detection of the particle tracks to distinguish which new particles comes from which collisions. The main event that is normally used in detection, and this corresponding vertex is called the *primary vertex*.

Since we want higher and higher luminosity to get more collisions, we also get more pile-ups. This need to be controlled to be able to use the data efficiently. The additional collisions do normally have smaller momentum transfers, which means we can characterize them as minimum bias events.

5.3 The ATLAS Experiment and Particle Detection

To detect the particles produced at particle colliders, we need different instruments that can detect the various types of particle interactions. The largest detector at the LHC is the ATLAS (A Toroidal LHC ApparatuS) experiment. It is 25 m in diameter, 46 m long and weights about 7000 tons. The cylindrical shape of ATLAS is optimized to detect as many particles as possible, and covers almost a 4π angle with detectors. Like we mentioned earlier for particle collisions in the LHC, ATLAS uses the same Cartesian coordinate system with the z-direction in the direction of the beam, y-direction is upward and x-direction is towards the center of the accelerator circle. It also uses a spherical coordinate system with the azimuthal angle ϕ in the xy-plane around the beam axis, and the polar angle θ being the angle from the beam axis. To measure the distance between the particles, the angular distance ΔR (eq.4.12) in the $\phi\eta$-plane is used.

The detector can be divided into three parts; the central part is called the *barrel*, and the two end parts are called *end-caps*. In Figure 5.2 we see a computer generated image of the ATLAS detector with pointers to the main components.

The ATLAS detector is designed to be a general-purpose detector, covering a wide range of signals. The particle properties the ATLAS detector can detect is the mass, momentum and energies of the particles. For ATLAS to detect these properties, it has a layered design of detectors that is optimized

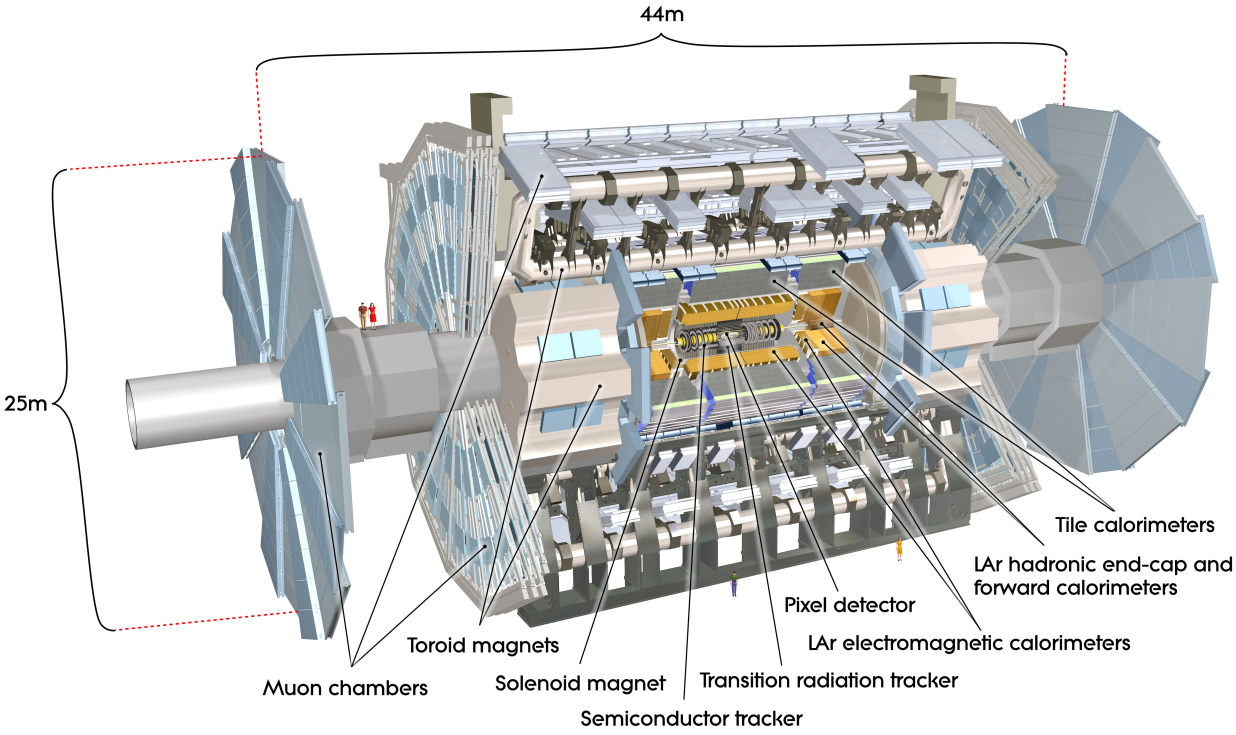


Figure 5.2: The ATLAS detector and its components. Credit: CERN [43].

in observing specific properties of the various particles. The ATLAS detector consists of several main systems; the inner detector (ID), calorimeters, a muon spectrometer (MS), a magnet system and a trigger and data acquisition system. The main systems consists of smaller sub-systems, which we will take a brief look at next. In Figure 5.3, we see a sketch of the detector layout systems and how some particles behave in these different systems. Only the neutrinos should now go undetected through the detectors, in principle, and they are normally identified as missing momentum, or MET. This comes from the energy conservation law, where the sum of the measured transverse momenta of the all particles produced should be zero.

5.3.1 Inner Detector

The inner detector tracks charged particles that leaves traces of ionized atoms when traveling through a medium. The tracks, momentum and charges of the particles can be traced in a 2 T magnetic field that makes the charged particles curve. The degree of the curvature is used to determine the charge

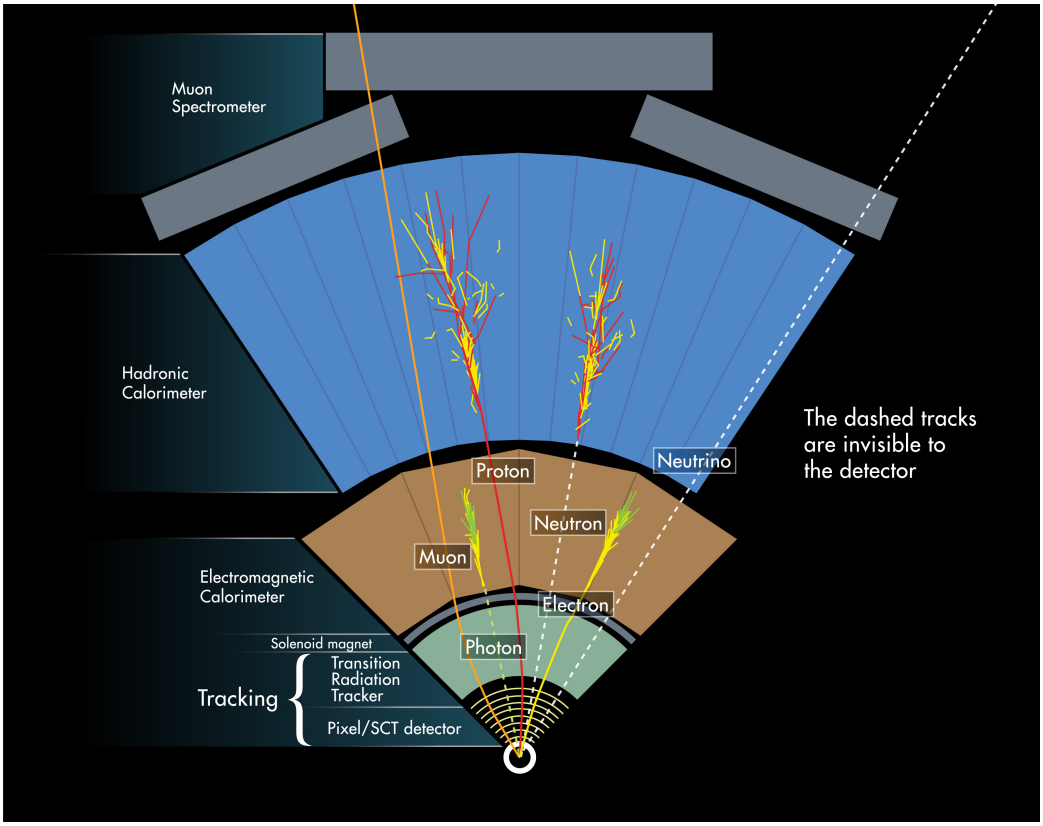


Figure 5.3: An illustration of the main tracking systems in the ATLAS detector, including how some particles behave in the various systems. Credit: ref. [44].

and the momentum.

The inner detector consists of three sub-systems. The inner most part is a silicon Pixel Detector that is used for extremely precise tracking near the interaction point of the particle collisions, which covers $|\eta| < 2.5$. The second part is a Semiconductor Tracker (SCT) that covers a bigger area than the pixel detector for the particle tracking and uses long and narrow strips instead of pixels. The SCT provides detection in the range of $|\eta| < 2.5$ as well. The third part is a Transition Radiation Tracker (TRT) that covers an even larger area with lower spatial resolution, and can detect transition radiation photons by using gas filled drift/straw tubes. The TRT provides the capability of electron identification for a variety of energies since the transition radiation gives out a stronger signal than ionization signal. It has a coverage of $|\eta| < 2$.

5.3.2 Calorimeters

Outside the ID and the solenoid magnet system, there follows two types of calorimeters; an (inner) electromagnetic calorimeter and a (outer) hadronic calorimeter. Their purpose is to measure the energy of the passing particles and particle showers especially. They both consist of a barrel part and two end-cap parts.

The **electromagnetic** calorimeter (ECal) measures particles that interact electromagnetically, like charged leptons and photons. The ECal is made of layers of lead absorbing plates and liquid argon, and covers the whole ϕ angle around the beam axis. The energy is measured in the liquid argon, and free electrons are picked up by electrodes. The ECal is covered by cryostats to keep it at the correct low temperature. The thickness of the ECal is measured in radiation lengths X_0, which is the mean length required to reduce the energy of a particle by $1/E$ in a material. The thickness of the barrel part is $\geq 22X_0$, while the end-caps are $\geq 24X_0$.

The **hadronic** calorimeter (HCal) measures hadrons and hadronic showers[14]. The HCal is made of several layers of steel absorbers and plastic scintillator tiles that alternates. The HCal is a lot bigger than the ECal. The HCal consists of three parts. The iron in the detector both slows down and traps hadrons. The HCal is not as precise as the ECal. The thickness of the HCal is measured in interaction lengths λ, which is the mean distance a particle travels before interacting strongly with the material. The detector is 9.7 interaction lengths thick [45].

[14]It measures the energy of particles that interact via the strong force, which is mainly hadrons.

5.3.3 Muon Spectrometer

Outside the calorimeters, we find the muon spectrometer. Here high-energy muons are detected. This detector is very large, 11 m radius [46], and consist of three parts as well as a barrel and two end-caps; a magnetic field with several toroidal magnets, a set of chambers measuring the tracks of the muons and a set of triggering chambers with accurate time-resolution. The detection of the muons happens the same way as before, by measuring their momentum as they are bent in the detector. They should also be simpler to identify since all other identifiable particles should not reach this far out from the interaction point.

5.3.4 Magnet System

ATLAS uses two types of superconducting magnet systems to measure the momentum from the bending of the particles through the Lorentz force. The magnet system consists of a central solenoid, a barrel toroid and two end-cap toroids. The central solenoid is located between the inner detector and the electromagnetic calorimeter, which produces the 2 T magnetic field for the ID. The barrel toroid produces a magnetic field of 0.6 T, and is located around the middle cylinder of the MS barrel outside the calorimeters. The two end-cap toroids produce magnetic fields of 1 T, and located at the end-cap regions of the Muon System.

5.3.5 Trigger System

The detector produces a huge amount of data, which need to be stored and processed. The output event storage rate have to be reduced from an initial bunch crossing of 40 MHz to ∼200 Hz . To only get the most interesting data for further analysis, a trigger system is used to extract these relevant events. The ATLAS Trigger and Data Acquisition system (TDAQ) has three levels for reducing the amount of stored data [47]; the Level 1 (LVL1) trigger is hardware-based and makes quick decisions of which events to store, the Level 2 (LVL2) and the Event filters (EF) are software-based and are often combined to and referred to as the High Level Triggers (HLT). Only the events passing both the LVL1 and HLT are stored for further analysis.

The LVL1 trigger uses information from the calorimeters and the muon spectrometer to choose interesting events. These interesting events passed on to the next trigger. The LVL1 trigger also defines regions based on the ϕ and η coordinates from the interesting events.

The LVL2 trigger uses all the information within the regions of interest

(ROIs) defined by the LVL1 trigger to further reduce the amount of event data. The accepted events are then assembled put together into a full event. The EF uses an offline analysis to even further reduce the data used to store and further analysis at the WLCG.

Chapter 6

Machine Learning

6.1 Introduction

Machine learning (ML) has recently become widely used in many fields of research. The meaning of machine learning is to train computational algorithms to automatically determine an outcome from specific patterns in data the algorithms have not seen before by using pre-trained algorithms with a given input set of hyperparameters. When training an algorithm, one tries to teach patterns using large amounts of data. ML goes in under what is called artificial intelligence, which is where the computer takes its own decisions to produce and predict solutions to problems.

The machine learning algorithms build a model based on some given data and general rules. The data may often need to be processed in some way, like when there are missing values in the data set. The models are then fit and trained on sample data, which is a subset of the full data set. The remaining data, which is a smaller part than the training data, are used to make predictions and do an evaluation of the trained model. There is a huge variety of different evaluation metrics which are used to check the performance of the algorithms on data. When we have a good enough trained model, we can save it and use it later on similar unseen data.

There is a plethora of usages for machine learning, and it is often divided into estimation or prediction problems. An example of a machine learning problem can be to identify objects in images of animals, which may be easy to humans. Algorithms can be trained to identify various animals by the algorithms given some features to best distinguish the animals from each other. This may be the shape of ears or the tail of the animals. Computationally this means we choose some observable quantity $\mathbf{x}$ in the data we look at which are related to some parameter θ. The model $p(\mathbf{x}|\theta)$ is describing

the probability of observing $\mathbf{x}$ given θ. A data set $\mathbf{X}$, also called a design matrix, is produced to fit the model. The design matrix only consists of feature data, while the class variables are stored in a target vector $\mathbf{y}$. These two datasets are often split into training and test sets, and sometimes even into training, test and validation sets. The fitting of the model then tries to find the parameters $\hat{\theta}$ which best explains the data. In this thesis, it is the accuracy of the model that we want to optimize and focus on. Optimizing the accuracy of $\hat{\theta}$ is often the concern with estimation problems, where as prediction problems focuses more on how the model makes new predictions.

Most machine learning problems consists of the same ingredients, starting with a data set $\mathcal{D} = (\mathbf{X}, \mathbf{y})$, where $\mathbf{X}$ is the matrix containing the independent variables $\mathbf{x}$ and $\mathbf{y}$ is a vector containing the dependent variables. Then there is a model as a function $\mathbf{f} : \mathbf{x} \to \mathbf{y}$ with the parameters θ. The function is used to predict the outputs given vectors of input variables. For the predictions to take place, we need a cost function $\mathcal{C}(\mathbf{y}, \mathbf{f}(\mathbf{X}; \theta))$ that judges how well the model performs on the observations. When fitting the model, we want the $\hat{\theta}$ which best explains the data. When considering a linear regression case with the sum of least squares as the cost function,

$$\mathcal{C}(\mathbf{y}, f(\mathbf{X}; \theta)) = \sum_{i}^{N} (y_i - f(\mathbf{x}_i; \theta))^2, \tag{6.1}$$

we get the best fit with the set of parameters that minimize the cost function:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \{\mathcal{C}(\mathbf{y}, f(\mathbf{X}; \theta))\} \tag{6.2}$$

The ML approaches are usually divided into supervised, unsupervised and reinforcement learning [1]. **Supervised learning** already has the answers or outputs before we do anything to the model. The data set needs to be labeled and have the answers to the problem such that the algorithms know what is correct. During training, the algorithm predicts the answers from what it has learned. If we are not satisfied with the accuracy the algorithm provides, we change the hyperparameters or the algorithm until we are satisfied with the results. **Unsupervised learning** does not have any labeled data or correct answers, meaning that it has to find its own structure in the inputs. The algorithms can only use predefined metrics to make a conclusion. This can then be used to discover hidden data patterns or to reproduce the given input. **Reinforcement learning** uses a dynamic environment that has a

[1] There exists other approaches that goes beyond these three mentioned approaches. The most dominant approach today of these is called deep learning. See Goodfellow et al. [48] for more on deep learning and other possible machine learning tasks.

specific goal. As the problem is solved through trial, error and experience, the program tries to maximize its rewards from feedback during the problem solving. The program then trains itself to make decisions.

This chapter takes a closer look at the supervised learning category in machine learning and some of the basics of statistical learning, as well as classification and multiclass classification, which is used in this thesis. The theory is mostly based on the works of Hastie et al. [49] and Mehta et al. [50].

6.2 Supervised Learning

For supervised learning, we already mentioned that we need the outputs, labeled data and the need to tune hyperparameters for optimization. The inputs may also be called independent variables, while the outputs can be called dependent variables. Supervised learning can be divided into different learning algorithms; classification, regression and active learning. **Active learning** algorithms uses a source with information to label data points with some desired output. **Regression** algorithms uses a given set of features and inputs, and estimates the relationship between the features and an outcome variable. Regression is mostly used for problems with a variation of outcome values, or a continuous output, within a range of values. **Classification** algorithms has a limited set of values as outputs, which can be categories, numbers or names. Classification uses pattern recognition in sets of categories of discrete variables to identify new observations or to group unseen data based on the inputs. We will take a closer look into the basics of statistical learning with a focus on supervised learning next.

6.2.1 Basics of Statistical Learning

In statistical learning, the goal is to find a function h in a hypothetical set $\mathcal{H}$ such that $h \in \mathcal{H}$ approximates an unknown function $y = f(x)$ as best as possible. $\mathcal{H}$ consists here of all possible functions that are defined in the domain of f and are of interest for the problem at hand. With the newly developed function $h(x)$, we would then get $h \approx f$. The *expected error* for a particular function h over all inputs x and outputs y is given by the cost function $\mathcal{C}$ and the joint probability distribution for x and y as:

$$\mathbb{E}[h] = \int_{X \times Y} \mathcal{C}(h(x), y)\rho(x, y)dxdy. \tag{6.3}$$

In this case we need knowledge of the probability distribution, which we in most cases do not. For n data points, we can instead use the *empirical error*:

$$\mathbb{E}_E[h] = \frac{1}{n}\sum_i^n \mathcal{C}(h(x_i), y_i). \tag{6.4}$$

With the expected and empirical errors, we can compute the *generalization error* as the difference between those two:

$$G = \mathbb{E}[h] - \mathbb{E}_E[h]. \tag{6.5}$$

In the limit of the generalization error goes towards zero,

$$\lim_{n\to\infty} G = 0,$$

we say that an algorithm can learn or generalize from the data. In general, we cannot compute the generalization error since we in general cannot compute the expectation error. To solve this we can divide our data set into training and test sets, and then use cross-validation to estimate the generalization error. The values on the cost function on the training and test sets are called the *in-sample* error, E_{in}, and *out-of-sample* error, E_{out}, respectively. The in-sample error can be an appropriate approximation to the generalization error if the data set is large enough and is representative of the function f.

In Figure 6.1 we see how the errors in general behave when the training set size, or number of data points, increases. We have assumed here that the number of data points is large and that the true function $f(x)$ can't be exactly fit. As the number of data points increase, we see that the in-sample error increases while the out-of-sample error decreases. The sampling noise decreases since the error difference between the two errors decreases. The out-of-sample error we get from this sampling noise is called the *variance*, which goes towards zero in the infinite data limit. As the training data set approaches the infinity limit, we can conclude that the two errors must go to the same value. This is called the model *bias*. The bias is a representation of the best our model can do with infinite data size.

6.2.2 Bias-Variance Decomposition

We will now go a bit further into the bias and variance that is an important aspect of machine learning. Lets consider a data set $\mathcal{D}(\mathbf{X}, \mathbf{y})$ with N pairs of independent and dependent variables. We then assume that the true data is created from a noise model

$$y = f(x) + \epsilon, \tag{6.6}$$

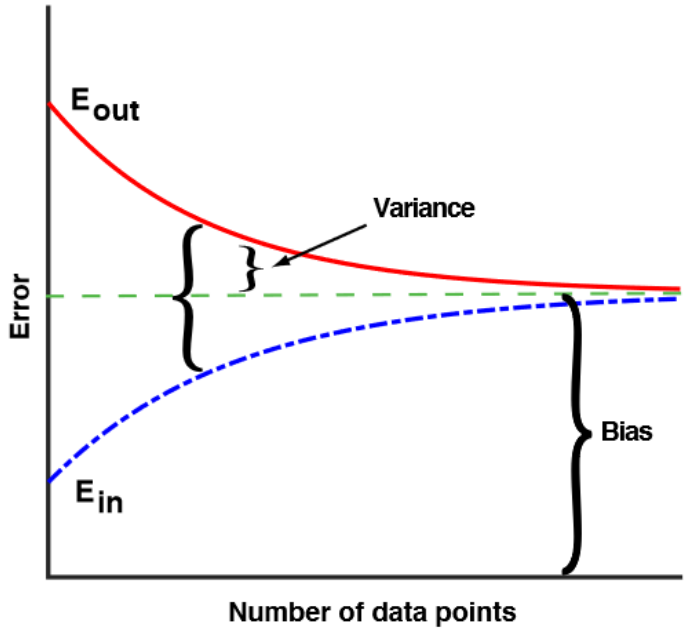


Figure 6.1: Illustration of the in-sample error, E_{in}, out-of-sample error, E_{out}, variance, bias and difference of errors as function of the training set size. It is assumed that the number of data points is not small, and that we cannot exactly fit the true function $f(x)$. The training error increases while the test error decreases as the training set size increases. Figure is taken from ref. Mehta et al. [50].

where ϵ is a normally distributed noise with mean zero and standard deviation σ_ϵ. A chosen estimator $f(\mathbf{x};\hat{\theta})$ is trained by minimizing the cost function, lets say the sum of squared errors[2],

$$\mathcal{C}(\mathbf{y}, f(\mathbf{X};\theta)) = \sum_i (y_i - f(\mathbf{x}_i;\theta))^2. \tag{6.7}$$

Our best estimates for the model parameters,

$$\hat{\theta}_{\mathcal{D}} = \underset{\theta}{\text{argmin}} \, \{\mathcal{C}(\mathbf{y}, f(\mathbf{X};\theta))\}, \tag{6.8}$$

are functions of the data set $\mathcal{D}$. Then we make another set of data sets $\mathcal{D}_n = (\mathbf{y}_n, \mathbf{X}; n)$, where all sets have N samples. We want the expectation value, $\mathbb{E}_{\mathcal{D}}$, of the cost function of all these data sets. We also want the

[2]This is used in regression cases. For classification we could use cross-entropy for instance.

expectation value of the average over different noise instances $\mathbb{E}_\epsilon$. The expected generalization error can be found to be (full derivation can be seen in Appendix A):

$$\begin{aligned}\mathbb{E}_{\mathcal{D},\epsilon}[\mathcal{C}(\mathbf{y}, f(\mathbf{X}; \hat{\theta}_\mathcal{D}))] = &\sum_i (f(\mathbf{x}_i) - \mathbb{E}_\mathcal{D}[f(\mathbf{x}_i; \hat{\theta}_\mathcal{D})])^2 \\ &+ \sum_i \mathbb{E}_\mathcal{D}[\{f(\mathbf{x}_i; \hat{\theta}_\mathcal{D}) - \mathbb{E}[f(\mathbf{x}_i; \hat{\theta}_\mathcal{D})]\}^2] \\ &+ \sum_i \sigma_\epsilon^2 \end{aligned} \tag{6.9}$$

The first term in equation 6.9 is the bias

$$\text{Bias}^2 = \sum_i (f(\mathbf{x}_i) - \mathbb{E}_\mathcal{D}[f(\mathbf{x}_i; \hat{\theta}_\mathcal{D})])^2, \tag{6.10}$$

and is a measure of the deviation of the expectation value of the model estimator from the true value. This is the best we can do in the infinity limit as we have already discussed. The second term is the variance

$$\text{Var} = \sum_i \mathbb{E}_\mathcal{D}[\{f(\mathbf{x}_i; \hat{\theta}_\mathcal{D}) - \mathbb{E}[f(\mathbf{x}_i; \hat{\theta}_\mathcal{D})]\}^2], \tag{6.11}$$

and measures the fluctuation in the estimator due to finite-sample effects. The last term is just a noise term $\text{Noise} = \sum_i \sigma_\epsilon^2$. By combining these three terms we can decompose the out-of-sample error as

$$E_{\text{out}} = \text{Bias}^2 + \text{Var} + \text{Noise}. \tag{6.12}$$

It is often much simpler to train a very complex model than it is to obtain sufficient good data. Therefore it is normally more useful to use a less complex model with higher bias, since it is less sensitive to noise in the sampling data from having a finite-sized training data set.

6.2.3 Bias-Variance Tradeoff

Before we look into classification, we need to be aware of a few problems with supervised learning. First is the balance of variance and bias. This is called the **bias-variance tradeoff** in statistics and machine learning. We want to minimize both the variance and bias such that our model both works well on unseen data and captures the relations between the features and classes, but when one of them is lowered the other has a tendency to increase. High bias may lead to underfitting between the features and the

classes, while high variance may lead to overfitting. When a model is overfit, it is excessively complex and will then model noise in the data as well. Overfit models will then do a great job during fitting, but worse on data outside of the training domain. Underfit models do not have the power to capture important variations in the data. With today's improved machinery, it is often easier to make a model too complex rather than to not.

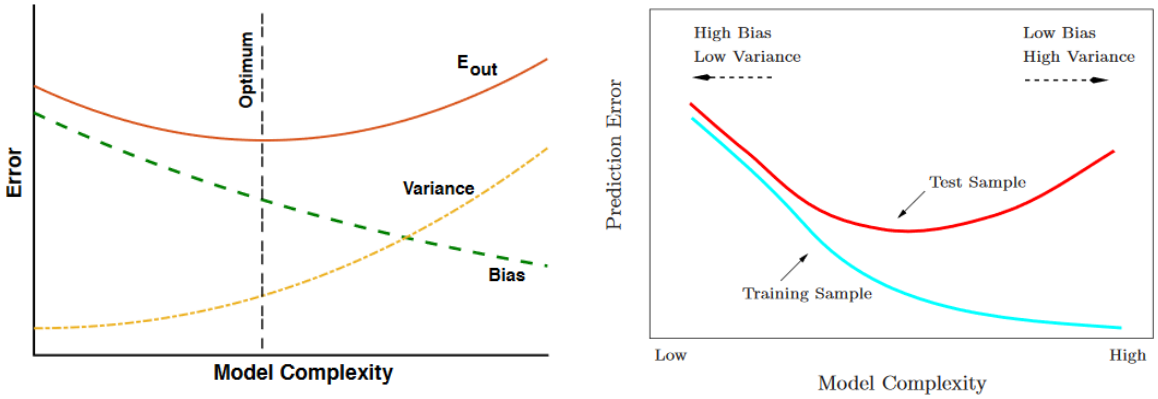
Second is the amount of training data that is available depending on the real function. For a more simple real function, the model does not need that much training data to learn on. While for a more complex[3] real function, the model needs a lot of training data.

Third is the dimensionality of the features. If there are a lot of features with high dimensionality, the model may be confused and cannot separate out the most important features that defines the output. One way to fix this is to manually remove irrelevant features in the data that can confuse the model. The method for doing this is called **dimensionality reduction**, and there are several strategies for doing this.

The fourth and final major concern is noise or incorrect values in the desired output values. This often comes from human error or errors in sensors which can lead to overfitting. This can be fixed by e.g., remove noise training data or use early stopping. There also exists other factors that one need to consider, but these four bias-variance related issues are some of the biggest.

In Figure 6.2 we see illustrations of the bias-variance tradeoff for training error, E_{in}, and test error, E_{out}, as the model complexity increases. In Figure 6.2a we see that as the model complexity increases, the model fits the training data well leading to high variance. For a low complexity model the bias is high. This is exactly as we have already look at above. So we want a model that has a compromise between the variance and the bias, as seen by the optimal line in Figure 6.2a. This optimal line is also where we have a minimum in E_{out}. For the prediction error for test and training samples in Figure 6.2b as function of the model complexity, we see the variance and bias areas for low and high model complexities. From the gap between the two prediction error samples we see the same argument for choosing a optimal compromise between variance and bias. This will lead to a predicted error difference between training and test samples that is not to0 big and not to0 similar to each other. Often we want to use a more biased model with small variance to minimize E_{out} and maximize the predictions.

[3]When we talk about simple and complex real function, we mean the complexity of interactions between the features and the number of features we use to approximate the true function.



(a) (b)

Figure 6.2: Illustrations of the bias-variance tradeoff as function of model complexity. From these two illustrations we see that we want to find the optimal compromise between variance and bias that gives the best model, which does not underfit nor overfit the data. Figures are taken from ref. Hastie et al. [49] and Mehta et al. [50].

6.2.4 Regularization

With increasing data power and amount of data collected, the data sets we can gather can be quite complex. This means that we need better machine learning models. With these better models we can solve more complex problems than before. As we mentioned earlier, this also gives rise to more problems, especially overfitting models. Overfitting is a more common issue than underfitting, since overfitting comes from models fitting functions and training data too well, making it perform worse on unseen (test) data. This is not something we desire to get, since machine learning is all about training model to analyze new data.

Finding good methods to reduce overfitting has been an important aspect in machine learning a long time. That is the reason for developing *regularization* techniques that reduces overfitting problems without significantly worsen the performance on the training data. Regularization techniques try to improve the generalization error of the test set. There are several different regularization methods that can be used, depending on the type of models which are used.

One way is to tune the model complexity to be better at predicting. This is done by introducing a penalty for individual weights, w . There are two

types of norms of regularization that is often used; L1 and L2:

$$L_{1,\text{norm}} = \sum_i |w_i| \tag{6.13}$$

$$L_{2,\text{norm}} = \sum_i ||w_i||^2 \tag{6.14}$$

The L1 penalty will yield sparse feature vectors from the fact that most features weights will be zero. That means that the L1 norm can be seen as a kind of feature selection that removes irrelevant features in data sets with higher dimensionality that would only confuse the model when training. This feature reduction can also be done manually by removing the irrelevant features that makes the model underperform, making it less complex. The L2 norm also acts on the weights of the loss function. These two regularization norms are set in the models as *hyperparameters*.

Other ways to avoid overfitting is to *prune* the models which use *trees*[4], affecting the splitting of trees. *Sampling* and *early stopping* are other ways to control overfitting, by making boosted trees less correlated or stop training when a chosen training metric of a model no longer improves. All these ways to control overfitting are controlled by various input parameters numerically.

6.2.5 Hyperparameters

As we have already mentioned, hyperparameters are something which need to be manually chosen before fitting a model. Hyperparameters help to tune and optimize the models in order to do a better fit of the data, and used to control the algorithms. These hyperparameters have no strict solution and change depending on the data set we are looking at. The same type of parameter may not have the same value in different models. For a small set of hyperparameters we could simply use trial and error to test the parameters. Most modern models require a lot of different hyperparameters. When there are a lot of parameters to tune, we may want to use some learning algorithm that searches through some given sets of hyperparameter values. An efficient method for doing this is to do a random search that uses the fact that not all hyperparameters are equally important. Searching for parameters are often computationally expensive since they require that the model is re-trained each time we change a configuration of hyperparameters.

During the hyperparameter optimization, we want the test set to be isolated until the model is fully optimized. This is where the validation set becomes useful. The purpose of the validation set is to be used when training

[4]We will come back to what this is later.

the model and optimize the hyperparameters. The first split of the original data set is into training and test sets. The training set can be further split into a smaller training set and a validation set. This means that we loose some training data which we need to take into consideration. The evaluation of the validation set will not be the exact same as evaluating the test set. The generalization error of the test set will be underestimated by the validation set error since the hyperparameters are trained on the validation set.

6.3 Classification

Classification is one of the most used and successful tasks in machine learning. Classification uses algorithms to decide which category the input belongs to. The function that produces an output value can be used to produce a probability distribution over the different outcomes. The simplest and probably most common classification problems are binary outcomes like True or False, Yes or No, Cat or Dog etc, where the outcomes are either the one or the other. When there are more than two outcomes, or classes, we use multiclass classification algorithms. Not all classification algorithms are made to classify instances with more than two outcomes, and cannot be used to classify problems other than binary outcomes. On the other hand, they can be turned into multiclassifiers by using various strategies. There are also other types of classifiers that are similar to multiclass classifiers, like multilabel and multioutput classification. They are similar, but are used in different cases with different outcomes. For example, multiclass classification labels a sample as one class only, meaning that it cannot be classified with two classes. This means that an image of a cat can only be classified as either a "cat" or a "dog" by the algorithm. The other two may categorize the image as both a "cat" and as "small" for instance.

In this thesis, we use multiclass classification to classify different particles in event decay chains produced by colliding protons at the LHC. In this thesis we will test different classification models and algorithms with various values for the hyperparameters for the respective models, to try and optimize and find the most accurate model. We will also study various evaluation metrics used to both find and evaluate the performance of the best model.

6.4 Classification Models

A so-called "hard" classifier will assign each datapoint to a category, while a "soft" classifier will give the probability of a given category. The sim-

plest classification algorithm is the "perceptron". It is given by the same transformation as linear regression with a weight matrix $\mathbf{w}$,

$$\mathbf{y} = \mathbf{X}\mathbf{w}. \tag{6.15}$$

The classes are then determined by the sign of the predictions by using sign functions or boundary thresholds. The perceptron is an example of a "hard" classifier. Sometimes it may be useful to use a "soft" classifier yielding category probabilities instead.

There are a lot of different classification models in machine learning with their own strengths and weaknesses. This is why we in this thesis will test a few different approaches and algorithms to find the best model for the analysis. In this section, we will briefly look at the classification methods we will test in this thesis.

6.4.1 Logistic Regression

A simple "soft" model in statistical analysis for classifying discrete outcomes is logistic regression (LR)[50]. It uses linear regression to fit data and a logistic function[5], usually the Sigmoid function

$$\sigma(s) = \frac{1}{1 + e^{-s}}, \tag{6.16}$$

to predict the outcomes into categories using probabilities. A threshold for the predicted values is chosen which determines which classes the data belongs to. These boundary thresholds can be complex and doesn't have to be linear. The cost function is the usually cross-entropy with added L_1 (eq.6.13) and L_2 (eq.6.14) regularization terms. The cross-entropy is the negative log-likelihood of the prediction being in the data set. The cross-entropy is derived from the fact that the Maximum Likelihood Estimator (MLE) is the set of parameters that maximize the log-likelihood.

The most basic model is a Binary Logistic Regression that yields two possible outcomes. However, it can be extended to more than two outcomes by using Multinomial Logistic Regression (MLR. Both LR and MLR can be combined with cross-validation, using various optimization solvers supporting the regularization parameters as input.

6.4.2 Multi-Layer Perceptron

A Multi-Layer Perceptron (MLP)[51] is an artificial feed-forward neural network (FFNN) model consisting of interconnected nodes, and is similar to LR

[5]It can also be called an activation function.

in that it has an *input* and an *output layer*, but differs in that between these layers, the MLP can have several non-linear layers called *hidden layers*. In a FFNN the information only goes one way. The inputs are called neurons and are transformed in the hidden layers by a weighted linear summation of the inputs and a non-linear activation function to determine the outputs for each layer. The hidden layers often have some bias to ensure non-zero values. The output layer transforms the values from the last hidden layer into output values. In Figure 6.3a we see how each node in a neural network is connected to all the nodes in the previous layer with a weight value. Then it goes to a non-linear activation function that transforms the node to an output either to a new node in a hidden layer or to the output layer. The nodes will have some bias term individually connected to them. In Figure 6.3b we see a fully connected neural network since all nodes are connected to all nodes in the next layer.

The MLP trains the model using *backpropagation* with initial guesses for the biases and weights. Backpropagation is a method used to optimize the weights and biases to minimize he cost function. The backpropagation iterates backwards from the last layer to the first layer using gradient descent of the weights and biases to start a new feed-forward process from the input layer. This process is repeated until the cost function is sufficiently minimized.

Typical choices of activation functions are the hyperbolic tangent function, the sigmoid and the rectified linear unit function (ReLU). The choice of cost function also needs to be considered. The MLP library in *Scikit-Learn* only supports the cross-entropy loss function as the cost function.

Neural networks typically have a large amount of parameters which often leads to overfitting. That is why we add a L_2 regularization penalty to the weights. This hyperparameter have to be tuned. Another hyperparameter that is needed is the *learning rate*. This parameter is used to control the step length in the optimization of the cost function with a gradient descent method. In neural networks the weights and biases are the parameters to be adjusted, while it can be different in other models. There are several gradient descent methods, e.g. the stochastic gradient descent with minibathces, which can be used to avoid interpreting a local minimum as a global minimum. A more modern method is the *adam* solver proposed by Kingma and Ba [53]. It is a stochastic gradient-based optimizer which combines an adaptive learning rate[6] with other functions, and thus adds a few more hyperparameters to be tuned, i.e. β_1, β_2 and ϵ.

For multiclass classification, the softmax function is used as the last hid-

[6]It adjusts the learning rate as it iterates towards the minimum.

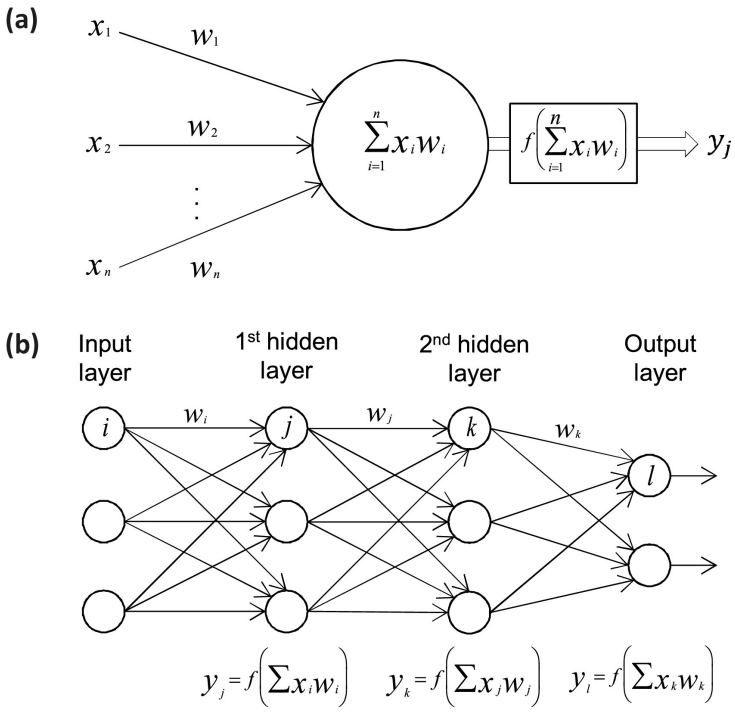


Figure 6.3: (a) Each node in a neural network has some input acted on by some associated weights. Then the weighted inputs are summed together inside the node and passed to a non-linear activation function which transforms it to an output. (b) This is a FFNN/MLP with 3 inputs, 2 hidden layers and 2 classes. All the nodes in one layer is connected to all the nodes in the next layer. This is then a fully connected neural network. Figure is taken from ref. Vieira et al. [52].

den layer activation function. It normalizes the output of the network into a probability distribution over the predicted output classes.

6.4.3 Decision Tree

Decision Trees (DTs)[54] tries to learn simple decision rules from the data features by constructing tree-like models to predict the target values. The models simply break down a complex decision into several simpler decisions. Each tree starts of with a single *root* node containing all the class labels. The root node then splits into several smaller *internal* nodes, which then splits into *leaf* nodes in the end representing the class targets. The splits are decided by some chosen *criterion* function that uses a certain strategy to do the splits. The root node is chosen as the feature with the highest

information gain value by the criterion function. The path from the root node to an internal node or a leaf is always unique, and the leaves do not have any descendants.

The DT uses a cost function to determine the the most homogeneous branch when splitting. The stopping point for splitting is something we can set as an input parameter to the model by choosing a maximum depth of the tree from the root to the leaves, or by setting the maximum number of leaves at the end. Other parameters for controlling the size and splitting of the tree should be considered since the DT is prone to overfitting with many features. This can be fixed by pruning the tree, i.e. to remove nodes with low importance features, use dimensionality reduction with e.g. principal component analysis (PCA)[55] or decrease some of the controlling parameters.

There are a few different DT algorithms to generate the optimal trees. The algorithm that is implemented in Scikit-Learn is an optimized version of the Classification and Regression Trees (CART) algorithm that construct binary trees from the features and thresholds for giving the highest information gain at each node. The Scikit-Learn DT classifier automatically supports multiclass classification.

6.4.4 Random Forest

Another classification ensemble method is the Random Forest (RnF)[56] algorithm. It produces a number of DT classifiers on bootstrapped training samples with a low correlation to each other, and uses their average like the bagging method. This improves the accuracy score and helps control overfitting. One can also choose to bootstrap samples.

Like with the DT algorithm, we can control the size and splitting of the tree. The DT and RnF algorithms are very similar and have many of the same input parameters and same procedure for building the trees. The main difference is that with the RnF algorithm, we produce many trees with sources of randomness. This randomness is very important in that it decreases the variance when combining and taking the average of many trees, and can cancel out some prediction errors. This normally yields a better model.

6.4.5 AdaBoost

Instead of using average ensembles where we use many independent bootstrap samples, we can use something called *boosting*. This is a type of method which keeps the weight for each iteration, and the base estimators are built sequentially. The boosting model then builds a combined estimator that

reduces the bias and the variance. The result is to get a powerful ensemble from several weaker models combined.

One such boosting method is the AdaBoost classifier[57][58]. The AdaBoost uses adaptive boosting and weaker classifiers as estimators to sequentially combine them into a single better classifier with a weighted majority. It will fit weaker classifiers sequentially such that the next classifier will have a different weight than the previous to adjust for incorrect classification in the previous. Data which are difficult to predict will then have an increasing influence since the next classifier will learn from the mistakes of the previous weaker classifier. The final prediction is the result of a weighted majority vote of the combination of the weaker classifier predictions. A weaker classifier can then be boosted to a stronger classifier that is more accurate.

The AdaBoost algorithm in Scikit-Learn takes a weaker classification algorithm as input together with the maximum number of estimators to be boosted before stopping. If the model is to be perfectly fit, the executing will also be stopped. It also takes a learning rate parameter for the shrinking of the classifiers. The AdaBoost algorithm can naturally detect and adapt to a multiclass problem.

6.4.6 Gradient Boosting

Gradient Boosting Decision Tree (GBDT)[59] is another boosting method like AdaBoost. The GBDT is an additive model that tries to identify the shortcomings of the weak classifiers. While AdaBoost uses high weight data points, the GBDT uses the same for gradients in the loss function. This allows the cost function to become better for optimizing the fitting. The K number of regression trees[7] at each stage are fit on the negative gradient of the binomial (binary class) or multinomial (multiclass) deviance loss function.

The algorithm is well suited for both binary and multiclass classification, and takes the maximum number of estimators and the learning rate as input parameters. Since it is a boosted tree method, it can also take the maximum depth of the trees and maximum number of leaves as inputs. It is also quite robust against overfitting. In a multiclass problem, the algorithm will create K trees for each iteration when we have K classes. The loss function for multiclass also have to be "deviance" to give probabilistic outputs (similar to LR).

When dealing with larger data sets (n_samples>10 000) or a large number of classes, a histogram-based gradient estimator can be more useful. Scikit-learn has an experimental implementation of GBDTs called **HistGradient-**

[7]For a binary classification case, only a single tree is fit.

BoostingClassifier which is inspired by Ke et al. [60] on a **LightGBM** algorithm. This estimator can be orders of magnitude faster than the original GBDT estimator. To reduce the computation time and number of splitting points, the algorithm bins the input samples into integer-valued bins. They share most of the same parameter inputs which controls the models, except that the histogram-based estimator gets a parameter for controlling the number of bins. This can act as another regularization parameter.

6.4.7 Extreme Gradient Boosting

Another highly efficient, flexible and portable tree boosting method is the Extreme Gradient Boosting (XGBoost)[61]. It is a scalable end-to-end tree boosting system using an optimized distributed gradient boosting algorithm and provides fast and accurate parallel tree boosting. It is one of the most used and highly recognized machine learning algorithms today together with deep neural networks. The XGBoost algorithm won the Kaggle Higgs ML challenge in 2014[62]. One of the most important aspects of the XGBoost is its scalability, making it several times faster than other algorithms combined with parallelization.

The XGBoost algorithm uses the GBDT framework as its core. It looks at distributions of the features for all data points in a leaf to build trees using potential loss for the possible splits to make a new branch. This decreases the space of possible feature splits search. The algorithm chooses features and split-points based on the criteria to maximize the gain. The splits are binary such that it splits according to if a value is bigger or lower than a threshold set by the algorithm. The gain is different depending on the type of loss function which is used. With a small data set, the XGBoost algorithm tries all split points gained by the data values for each feature. The feature and threshold combination with the highest gain is then chosen. For a larger data set, the algorithm uses fewer candidate splits given by the quantiles of the data.

Since XGBoost is more complex than other algorithms, it also requires more parameters to be tuned to control the model properly. The parameters can be sorted into *general parameters* for choosing the booster method, *booster parameters* which are dependent on the boosting method and *task parameters* which specify learning task parameters and learning objectives. We are using a tree booster which has many of the same tree boosting parameters as the DT and RnF algorithms, i.e. regularization terms, hyperparameters for tree controlling, pruning and others. The task parameters include the type and size of the classes we have, e.g. multiclass classification, and types of evaluation metrics to use.

6.4.8 Light Gradient Boosting Machine

Light Gradient Boosting Machine (LGBM)[60] is a distributed gradient boosting framework for machine learning. It is similar to the XGBoost algorithm, but made to be faster, around 7 times faster, with higher efficiency, lower memory usage and better accuracy. This is a huge advantage when dealing with larger data sets. The LGBM algorithm uses a gradient based one-side sampling and exclusive feature bundling for filtering the data samples to find the split value in the trees, while the XGBoost uses a histogram based algorithm to find the best splits. This means that the LGBM algorithm will keep features with higher absolute values, regarding information gain, than a pre-defined threshold and drop the features with small absolute values. This will improve the accuracy. The features that rarely have non-zero values simultaneously will be combined into a single feature, to reduce the number of features in the data set. XGBoost and LGBM have very similar input parameters.

6.4.9 Multiclass Classification Models

To do multiclass classification, there are several existing techniques. We will look more into two of those techniques[8]; transformation to binary and extension from binary. These are all meta-estimators. This means that they all need a base estimator, most often a binary classifier, which is extended to do multiclass classification when they are implemented in the constructors.

The extension from binary technique is rather trivial. We simply use already existing binary classifiers and modify them to do multiclass classification. Not all binary classifiers can be extended to multiple classes. The classification models we have looked at, this far, can either do this automatically, or have input parameters and constraints in the models to tell the models to do multiclass classification.

Transformation to binary reduces our multiclass problem down to several binary classification problems. This technique can also be split into more strategies, which we will look more into.

One-Vs-Rest Classifier

The first strategy is the one-vs-rest (OvR) classifier. Each class in this model has its own classifier which does the fitting, and the classifier fits the single class against the rest of the classes. This means we only need n classifiers

[8]There is also a third technique, hierarchical classification, that we will not cover.

for the n classes. This also improves interpretability, since we can get information about a specific class by looking at its classifier.

The OvR takes a input a binary classifier along with samples and targets and outputs a list of the classifiers for each class. When doing predictions, it uses all the classifiers on unseen data and picks the class with the highest confidence score.

One-Vs-One Classifier

The second strategy is the one-vs-one (OvO) classifier. This takes one classifier and a pair of classes at a time. For each pair of classes, the classifier trains on data containing these classes and learns to distinguish them. This happens between all the classes. It then uses a voting scheme to select the class with the most votes. For n number of classes in the multiclass problem, the OvO trains $n(n-1)/2$ binary classifiers. All the classifiers that are trained will be applied when doing the prediction on unseen data, and the one with the highest number of predictions will be predicted by the combination of classifiers.

This method is slower than the OvR since it has a $\mathcal{O}(n^2)$ complexity. Both the OvO and OvR methods suffer from the fact that there may be regions where the input space can get the same number of votes.

6.5 Evaluation Metrics

To evaluate the performance of the classification models properly and decide which model best fits the data, we need to have some evaluation metrics. In this section we will take a look at the evaluation metrics used for the classification[9].

6.5.1 Mutual Information

To look closer at the correlations in the data set, we can use the entropy and information gain. The entropy can be calculated using the probability $P(j)$ of a value j occurring, where j is a value which a feature group x_i can take;

$$H(x_i) = -\sum_{j \in x_i} P(j) \log_2 P(j) \tag{6.17}$$

[9]See Scikit-Learn[65] for more details on metrics.

With a given target $\mathbf{y}$, we can calculate the conditional entropy of a feature x_i:

$$H(x_i|\mathbf{y}) = -\sum_{y\in\mathbf{y}} P(y) \sum_{j\in x_i} P(j|y) \log_2 P(j|y) \tag{6.18}$$

Now we compute the information gain, or *mutual information* in the context of variable selection, for a given feature as the difference between these two entropies:

$$I(x_i : \mathbf{y}) = H(x_i) - H(x_i|\mathbf{y}) \tag{6.19}$$

With the information gain we get a measure of the correlation between a feature and the target, which shows dependencies between features and the amount of information that one feature provides about others.

6.5.2 Accuracy Score

To measure the performance of the models, we use the the accuracy score for classification. This is a measure on how well the models can predict the classes. It is defined as the number of correct predictions divided by the total number of predictions, giving a value between 0 and 1.

$$\text{Accuracy} = \frac{\sum_{i=1}^{n} I(\tilde{y}_i = y_i)}{n}, \tag{6.20}$$

where $\tilde{y}_i$ is the predicted target by the model, y_i is the actual class target, n is the total number of predictions and I is and indicator function

$$I = \begin{cases} 1, & \text{if } \tilde{y}_i = y_i \\ 0, & \text{if } \tilde{y}_i \neq y_i \end{cases} \tag{6.21}$$

When the model prediction fits the data perfectly we get an optimal score of 1.

The accuracy score can be computed for all data sets, i.e. training validation and test sets. If there is a big difference between the accuracy score for either validation and training or test and training, we might under- or overfit the data. When the training score is much better, we most likely overfit the data.

Another way to balance out the accuracy scores is to use *cross-validation*. It's a very useful technique against overfitting, and can be used to tune hyperparameters. There are several cross-validation techniques, but the main idea of cross-validation is to divide samples into subsets. The cross-validation will do the analysis on one subset and compute the accuracy on that subset. Then it will do another analysis with another subset and compute the

accuracy again. After many iterations, dividing the data into subsets and computing several accuracy scores, the average score is used as an estimate of the model performance. The Scikit-Learn library has a function called *cross_val_score* which does this, and we can choose how many folds of subsets we want the data to be split into.

6.5.3 Cohen Kappa Score

Another scoring statistic is the Cohen Kappa Score (CKS)[66]. The CKS accounts for uncertainties in the predictions, comparing a random classifier against a more accurate and tuned classifier. The CKS is calculated by using the rate of agreement for random guessing, p_e, and the rate of agreement for the actual prediction, p_a. The CKS ranges from -1 to 1, where 1 is the optimal score representing perfect agreement, 0 represents agreement that can be expected by random guess and -1 represents no agreement, and is calculated as

$$\kappa = \frac{p_a - p_e}{1 - p_e} \tag{6.22}$$

6.5.4 Error Evaluation

We will use several different error metrics to get a good overall error estimate of the classification models. These will also help to discover any over- or underfitting of the data.

Error Rate

With the accuracy score, we can compute the *error rate*. The error rate is defined as the fraction of misclassifications:

$$\text{error} = 1 - \text{accuracy} \tag{6.23}$$

This is an often used metric in classification. Both the error and the accuracy score can be computed in multiclass classification cases.

Log Loss

Instead of using discrete predictions, we can evaluate probability outputs of classifiers. We can use the *log loss* function, also called the cross-entropy or logistic regression loss, to evaluate the probabilities. When dealing with a binary case with a probability estimate $p = P(y = 1)$, the log loss is defined

as the negative log-likelihood given a true output for each sample. It is computed as

$$L_{\log} = -\log P(y|p) = -(y \log(p) + (1-y)\log(1-p)). \tag{6.24}$$

For a multiclass case, the log loss is taken over a whole set of size n with K labels, a binary indicator matrix $\mathbf{Y}$ and a matrix $\mathbf{Pr}$ of probability estimates as

$$L_{\log}(\mathbf{Y}, \mathbf{Pr}) = -\log P(\mathbf{Y}, \mathbf{Pr}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K} y_{i,k} \log p_{i,k}. \tag{6.25}$$

Variance

Previously in section 6.2.2, we defined the variance and the bias of a model. These two are used to check for possible under- and overfitting. The variance is a measure of how far the spread of our predictions are from their average values. Given the predictions, $\tilde{\mathbf{y}}$, of a model, the variance is calculated as

$$\text{Var}(\tilde{\mathbf{y}}) = \frac{1}{n}\sum_{i=1}^{n}(\tilde{y}_i - \frac{1}{n}\sum_{j=1}^{n}\tilde{y}_j)^2. \tag{6.26}$$

Bias

The bias error is a measure of the difference between the true values, $\mathbf{y}$, and the average of the predicted values. To get the out-of-sample error in equation 6.12. The bias squared can be calculated as

$$\text{Bias}^2(\mathbf{y}, \tilde{\mathbf{y}}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \frac{1}{n}\sum_{j=1}^{n}\tilde{y}_j)^2. \tag{6.27}$$

6.5.5 Classification Report

With Scikit-Learn, we can easily build what is called a *classification report*. This is a text report containing some useful classification metrics using the true targets and predictions of the model.

First we will look at some useful prediction results used to compute some of the report metrics:

1. Positive (P) - The observation is positive.
2. Negative (N) - The observation is negative.

3. True Positive (TP) - Observation is positive, and the prediction is positive.

4. True Negative (TN) - Observation is negative, and the prediction is negative.

5. False Positive (FP) - Observation is negative, but the prediction is positive.

6. False Negative (FN) - Observation is positive, but the prediction is negative.

With the last four outcomes above (3-6), we can compute some useful metrics in the report:

Precision - The fraction of a sample classified correctly as positive of all positive predicted samples by the model:

$$\frac{TP}{TP + FP}$$

Recall - The fraction of a sample classified correctly as positive of all positive observations (true positive rate):

$$\frac{TP}{TP + FN}$$

The recall of the positive class is also called the sensitivity. The recall of the negative class (true negative rate) is called the specificity.

F1-score - A weighted average of the precision and recall:

$$2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

In the multiclass case, these metrics are computed for each class independently.

The classification report also includes for various classification cases:

Support - The number of true classes in the data set for each class.

Accuracy - The accuracy score of the model (binary case).

Macro avg - Average of the unweighted mean for each class.

Micro avg - Average of the total true positives, false negatives and false positives (multiclass or multilabel cases).

Weighted avg - Average of the support-weighted mean for each class.

Sample avg - Average of samples (multilabel case).

6.5.6 Confusion Matrix

With the four outcomes in the classification report (3-6), we compute a *confusion matrix*. For a binary case, the confusion matrix looks like Figure 6.4. Here we see the predictions versus the true values. It gives a better understanding of the accuracy of a classification model. The accuracy score shows the overall accuracy, whereas the confusion matrix shows the predictions and accuracy of each class. It is easily extended for multiclass classification as a matrix with dimension $k \times k$ for k classes. When the confusion matrix is normalized the total values of the rows are equal to 1. In the optimal case with all predictions correctly guessed, we should have 1's along the diagonal and 0 elsewhere.

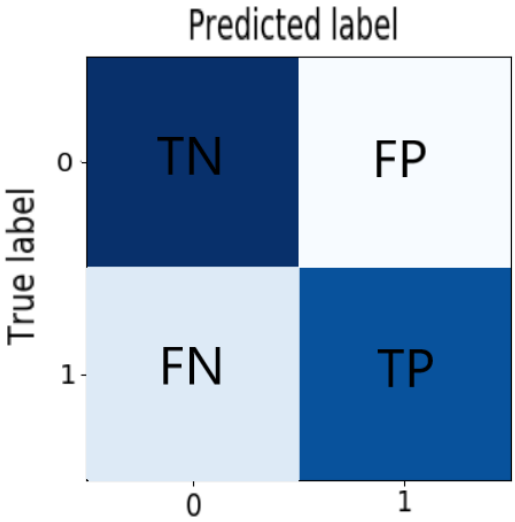


Figure 6.4: The confusion matrix is used to evaluate the accuracy of a classification model by using the four true and false observation and prediction outcomes (TP, TN, FP, FN. See sect. 6.5.5.).

6.5.7 Precision-Recall Curve

Scikit-Learn provides a useful function for plotting precision versus recall. In Figure 6.5 we see an example of how a precision-recall curve can look like in a multiclass case with 10 classes. This lets us see how the precision and recall behaves for different thresholds. A large area under the curve is the result of both high precision and high recall, which is preferable. The range of values will be between 0 and 1, as for accuracy. When the area under the curve

of a class is close to 1, the classification model can predict this class with a good accuracy. For a multiclass-case, the precision and recall are computed for each class as binary cases. A large area for each class is the optimal case here as well.

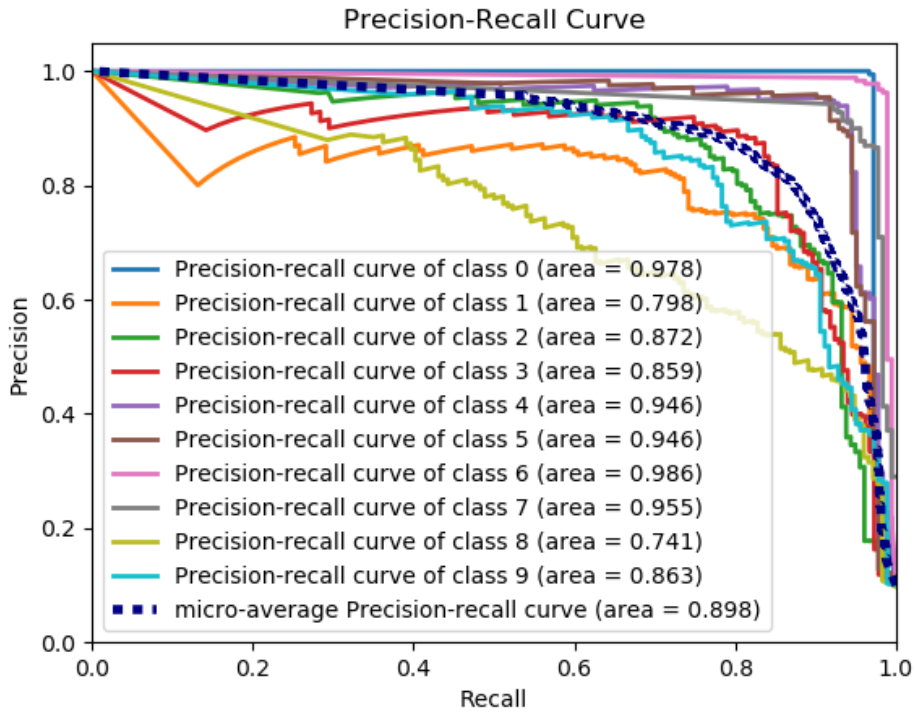


Figure 6.5: Example of a precision-recall curve for a multiclass classification case with 10 classes and a micro-average curve plotted. Most of the classes have a large area under the curve, showing that the classifier can predict these classes with good accuracy. Credits Scikit-Learn [65].

6.5.8 Balanced Accuracy

If we are dealing with imbalanced data sets, we can use *balanced accuracy*. It uses a macro-average of the recall for each class. When we have a balanced data set, this just becomes the standard classification accuracy. It is computed as the mean of the sensitivity and the specificity:

$$\text{Balanced-accuracy} = \frac{1}{2}\left(\frac{TP}{TP+FN} + \frac{TN}{TN+FP}\right) \tag{6.28}$$

The balanced accuracy ranges from 0 to 1.

6.5.9 ROC Curve

The Receiver Operating Characteristic (ROC) curve utilizes the area under the curve to summarize the overall performance of classification models. An example of a ROC curve plot can be seen in Figure 6.6 with a multiclass case with 10 classes. This results in 10 ROC curves, a random model curve and two different average curves, as seen in the figure. The ROC curve function in Scikit-Learn plots the sensitivity versus the specificity for a model. A totally random model would result in an area under the curve of 0.5, showing as the straight dashed line from the left bottom corner to the right top corner in the figure. The optimal model would show an infinitely quick incline in the ROC curve at the beginning, before flattening out with area under the curve close to 1. A good classifier would typically have an area under the curve larger than 0.8.

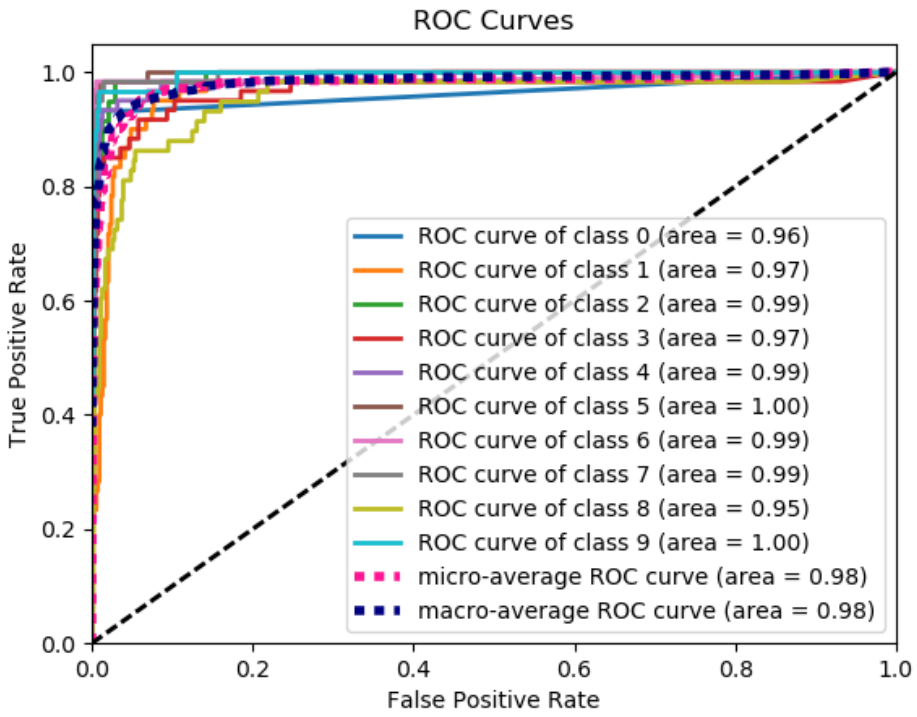


Figure 6.6: Example of a ROC curve for a multiclass classification case with 10 classes, a micro-average curve, a macro-average curve and a random model (black dashed line) plotted. All classes and averages have large areas under the curves, showing that the classifier can predict the classes with good accuracy. Credits Scikit-Learn [65].

Part II

*Implementation

Chapter 7

*Methods-Overview

In this part we will look at the framework for the classification exploiting a range of different models. First we will take a look at the data we are using, how it is made and how the it is converted to fit our purpose. The data is already produced beforehand as ROOT files (more in sect. **??**), it will be converted into data frames with Python and we will make new features as well as the targets. The data will then be analyzed and preprocessed (sect. 9.3) with different methods before it is split into training, validation and test sets. We will then go through some of the tuning which is done with the models and the validation set, and use the evaluation metrics from section 6.5, before we use the best fit model on unseen data. The best model will be used to classify the vertexes of the leptons in background and signal data. All the results will be presented in part III and discussed in part IV.

Python is used for easy implementation of machine learning libraries with Scikit-Learn and plotting using the Matplotlib library, as well as other useful libraries. In the following section, we will give a brief presentation of the most important Python libraries we use in our code.

7.1 Python Libraries

Many of the libraries we use require other libraries to be installed, but they do not have to be explicitly imported in the code itself. The code and necessary software requirements are found in the GitHub repository[1]. It contains explanations on how to setup and run the code.

When we present code snippets in this thesis, we will leave out some parts, noted by "\\...", since it will only be used for visualization of the code. The full source codes can be found in the GitHub repository.

[1] https://github.com/krilangs/ComPhys—Master

- **NumPy**: NumPy, or Numerical Python, is one of the most used packages in Python. It handles arrays, matrices, has functions for working with high-level mathematics, can dump data to files and more.
- **Pandas**: Pandas is a powerful and easy Python made library for handling data manipulation and analysis. This library is very useful with machine learning for handling the data for visualization, since it creates data structures that are flexible, efficient, customizable and easy to use and read.
- **Matplotlib**: Matplotlib is a plotting library for Python and NumPy which creates graphs and visualizations.
- **Seaborn**: Seaborn is a more high-level visualization library, and is based upon Matplotlib. It is most used for statistical graphics to understand data better, and is closely connected to pandas.
- **ROOT**: ROOT, or PyROOT, is ROOT's Python C++ bindings. It lets us use ROOT in Python. This is very much used in the particle physics community. This also lets us use Python libraries like NumPy and Pandas combined with ROOT.
- **Uproot**: Uproot is a library for converting ROOT files to e.g. data frames by combining Uproot and Pandas.
- **Scikit-Learn**: Scikit-Learn[65] is a library used for data analysis and machine learning in Python. It contains a lot of useful tools for statistical modeling and machine learning. Most of the classification models we use, are imported from this library.
- **Imblearn**: Imblearn, or Imbalanced-learn, is used with Scikit-Learn to handle imbalanced data sets in machine learning.
- **XGBoost**: XGBoost is a library that provides a powerful, scalable and distributed gradient boosting framework for machine learning.
- **LightGBM**: LightGBM is another distributed gradient boosting library for machine learning. It is made to be efficient and faster than XGBoost for larger data sets.

7.2 *Data*

What to have here? Thoughts:

Short explanation of what the backgrounds are?

Explanation of how the data sets looks like?

Chapter 8

Analyzing the Data

We will now use the data for Monte Carlo (MC generated data, several backgrounds and neutrino signals with a few different scenarios for $N1$ mass, masses of 150 GeV and 450 GeV. We use several Python scripts for plotting different property variables of the final state leptons as histograms. We will plot some of the already existing variables in the files, as well as making new variables for $d\phi$, dR and m_{ll}. Then we study the variables for the different backgrounds and signal samples. The background and signal samples will later be used to classify the leptons with ML.

The scripts for making the histograms uses ROOT in Python. The overall layout of the scripts are to make histogram plots of property variables of the final state leptons in many events for several background samples, neutrino signals and Monte Carlo simulated data, which all are stored as ROOT files. We can then look at the differences between the different periods the data was taken, the different backgrounds, the difference between the two neutrino signals and between the leptons.

The main script imports different functions from the other scripts, which have different jobs, and can take multiple ROOT-files as input, depending on how many backgrounds we want to plot as well as MC data and signal samples. The scripts can be found at the same GitHub repository as before.

Chapter 9

Preparing the Data

9.1 Making New Variables

After downloading the desired data set with the proton-proton collision events, we start by converting it from ROOT to Python syntax with the Uproot library. The file is then stored as a dataframe with Pandas. This is done in the script called *Trilepton_read_root.py*. By using the existing data for momentum and energy for the leptons in each event, we compute new useful variables to add to a new dataframe. The new variables we make are the angular variables for each lepton (θ, ϕ, η), angular variables between pairs of leptons ($d\phi$, dR) and the invariant masses of pairs of leptons (m_{ll}). Then we classify the events to construct a *target* variable as permutations of the vertexes the leptons come from by using their identity traits. The leptons are ordered by decreasing p_T. Since the neutrinos are so small, they will always have the lowest p_T and always come from the fourth vertex in the decay chain and are not considered in the targets. This leads to the following permutations:

$$[123, 132, 213, 231, 312, 321]$$

The function for making the new variables can be seen in Listing 9.1. The new dataframe is then exported as a .h5-file.

```
    # Method for flattening and adding additional variables
    def lepaugmentation(df, nlep):
        px  = awkward.fromiter(df['px'])
        py  = awkward.fromiter(df['py'])
        pz  = awkward.fromiter(df['pz'])
        E   = awkward.fromiter(df['E'])
        vtx = awkward.fromiter(df['vtxid'])
        pid = awkward.fromiter(df['pdgid'])

        # Make tlv - handy when computing angular variables
```

```
        tlv = uproot_methods.classes.TLorentzVector.TLorentzVectorArray.↩
            from_cartesian(px, py, pz, E)

        df["tlv"] = tlv[:]

        df["pt"]  = tlv[:].pt
        pt  = awkward.fromiter(df['pt'])
        pt_org  = awkward.fromiter(df['pt'])
        df["phi"] = tlv[:].phi
        phi = awkward.fromiter(df['phi'])

        df["theta"] = tlv.theta
        theta  = awkward.fromiter(df['theta'])

        df["eta"] = tlv.eta
        eta  = awkward.fromiter(df['eta'])

        \\...

        # Make the lepton variables
        for i in range(1,nlep+1):
            df['lep%i_pt'%i]  = pt[pt.argmax()].flatten()
            df['lep%i_phi'%i]  = phi[pt.argmax()].flatten()
            df['lep%i_eta'%i]  = eta[pt.argmax()].flatten()
            df['lep%i_theta'%i]  = theta[pt.argmax()].flatten()
            df['lep%i_px'%i]  = px[pt.argmax()].flatten()
            df['lep%i_py'%i]  = py[pt.argmax()].flatten()
            df['lep%i_pz'%i]  = pz[pt.argmax()].flatten()
            df['lep%i_E' %i]  = E[pt.argmax()].flatten()
            df['lep%i_vtx'%i]   = vtx[pt.argmax()].flatten()
            df['lep%i_pid'%i]   = pid[pt.argmax()].flatten()
            df['lep%i_tlv'%i]   = tlv[pt.argmax()].flatten()

        \\...

        # Compute variables for all combinations of 2 leptons
        pairs = pt_org.argchoose(2)
        print("pairs:", pairs)
        left  = pairs.i0
        right = pairs.i1

        \\...

        for ilep in range(len(left[0])):
            i = left[0][ilep]
            j = right[0][ilep]
            print('i = %i, j = %i'%(i,j))
            idx1 = left[0][i]
            idx2 = right[0][i]

            df['mll_%i%i'%(i+1,j+1)]    = (df['lep%i_tlv'%(i+1)]+df['lep%↩
                i_tlv'%(j+1)]).apply(get_invmass)
            df['dphi_%i%i'%(i+1,j+1)] = df.apply(lambda x : get_deltaPhi(x↩
                ['lep%i_tlv'%(i+1)],x['lep%i_tlv'%(j+1)]), axis=1)
            df['dR_%i%i'%(i+1,j+1)]    = df.apply(lambda x : get_deltaR(x['↩
                lep%i_tlv'%(i+1)],x['lep%i_tlv'%(j+1)]), axis=1)


        df['target'] = df.apply(lambda x : classify_event(x['lep1_vtx'],x[↩
            'lep2_vtx'],x['lep3_vtx'],x['lep4_vtx'],x['lep1_pid'],x['↩
            lep2_pid'],x['lep3_pid'],x['lep4_pid']), axis=1)
```

```
        df = df.drop(['px', 'py', 'pz', 'pt', 'E', 'vtxid', 'pdgid', '↩
            evnum', 'tlv', 'phi', 'theta', 'eta'], axis=1)
        return df
```

Listing 9.1: Making new variables.

9.2 Plotting New Variables

The new dataframe can now be imported by other scripts using Pandas to be used further. With the *Trilepton_plotter.py* script, we import the dataframe for visualization of the variables by using the Matplotlib library. We make two different functions that plots different features. One function plots the individual variables for each lepton in one plot. E.g. one plot can show the p_T for each lepton in a figure. The other function plots the *pair* variables (m_{ll}, $d\phi$, dR). In this way, it is easier to study the variables and properties of the leptons. The figures can then be saved to a folder. In Algorithm 1 we have a pseudocode of these two functions.

Algorithm 1 Plotting particle properties.

Function:
if *input variable is in dataframe* **then**
  Make histogram and figure
  if *save is True* **then**
    Save the figure to a folder
  end if
else
  print "Variable not in dataframe"
  print Dataframe
  break
end if
End function
Plot figure(s)

9.3 Preprocessing of the Data

After importing all the necessary libraries in a new script, *Trilepton_classifier.py*, we load in the dataframes and drop unnecessary features and events we will not consider in the classification. With some useful properties of Pandas

dataframes, and utilizing Seaborn and Scikit-Learn, we need to do some pre-processing of the data. One thing we have to consider for our data set is NaN, or NULL, values. Not all classification models can deal with NaN values, so these have to be dealt with. We can check for NaN values and drop them easily from the dataframe by doing the following in Listing 9.2:

```
df.isnull()   # Returns a boolean matrix, if the value is NaN then↩
    True otherwise False.
df.isnull().sum()   # Returns the column names along with the ↩
    number of NaN values in that particular column.
df.dropna(inplace=True)  # Removes rows in the dataframe ↩
    containing NaN values.
```

Listing 9.2: Check NULL values.

9.3.1 Inspect Data

Pandas dataframes lets us easily print a few lines and a summary of the dataframe. We then get a quick overview of what the data looks like. The $N1 = 150$ GeV data summary:

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 66885 entries, 0 to 67773
Data columns (total 55 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   lep1_pt     66885 non-null  float32
 1   lep1_phi    66885 non-null  float32
 2   lep1_eta    66885 non-null  float32
 3   lep1_theta  66885 non-null  float32
 4   lep1_px     66885 non-null  float32
 5   lep1_py     66885 non-null  float32
 6   lep1_pz     66885 non-null  float32
 7   lep1_E      66885 non-null  float32
 8   lep1_tlv    66885 non-null  object
 9   lep2_pt     66885 non-null  float32
 10  lep2_phi    66885 non-null  float32
 11  lep2_eta    66885 non-null  float32
 12  lep2_theta  66885 non-null  float32
 13  lep2_px     66885 non-null  float32
 14  lep2_py     66885 non-null  float32
 15  lep2_pz     66885 non-null  float32
 16  lep2_E      66885 non-null  float32
 17  lep2_tlv    66885 non-null  object
 18  lep3_pt     66885 non-null  float32
 19  lep3_phi    66885 non-null  float32
 20  lep3_eta    66885 non-null  float32
 21  lep3_theta  66885 non-null  float32
 22  lep3_px     66885 non-null  float32
 23  lep3_py     66885 non-null  float32
 24  lep3_pz     66885 non-null  float32
 25  lep3_E      66885 non-null  float32
 26  lep3_tlv    66885 non-null  object
```

```
 27  lep4_pt      66885 non-null  float32
 28  lep4_phi     66885 non-null  float32
 29  lep4_eta     66885 non-null  float32
 30  lep4_theta   66885 non-null  float32
 31  lep4_px      66885 non-null  float32
 32  lep4_py      66885 non-null  float32
 33  lep4_pz      66885 non-null  float32
 34  lep4_E       66885 non-null  float32
 35  lep4_tlv     66885 non-null  object
 36  mll_12       66885 non-null  float64
 37  dphi_12      66885 non-null  float64
 38  dR_12        66885 non-null  float64
 39  mll_13       66885 non-null  float64
 40  dphi_13      66885 non-null  float64
 41  dR_13        66885 non-null  float64
 42  mll_23       66885 non-null  float64
 43  dphi_23      66885 non-null  float64
 44  dR_23        66885 non-null  float64
 45  mll_14       66885 non-null  float64
 46  dphi_14      66885 non-null  float64
 47  dR_14        66885 non-null  float64
 48  mll_24       66885 non-null  float64
 49  dphi_24      66885 non-null  float64
 50  dR_24        66885 non-null  float64
 51  mll_34       66885 non-null  float64
 52  dphi_34      66885 non-null  float64
 53  dR_34        66885 non-null  float64
 54  target       66885 non-null  object
dtypes: float32(32), float64(18), object(5)
memory usage: 20.4+ MB

             lep1_pt  lep1_phi  lep1_eta  ...    dphi_34     dR_34 ↩
                target
entry                                     ...
0       364078.281250  1.312494 -1.321615  ...  0.891000  0.892144 ↩
                (1, 2, 3)
2        43565.238281  1.124601  1.340168  ...  2.664658  3.030723 ↩
                (1, 3, 2)
3        62504.234375 -3.002433 -0.343577  ...  0.209667  1.262884 ↩
                (3, 2, 1)
4        77743.296875 -1.776769 -1.809337  ... -2.731673  3.427304 ↩
                (2, 3, 1)
5        65388.453125  2.266318  2.015514  ...  2.228343  2.480432 ↩
                (1, 2, 3)

[5 rows x 55 columns]
```

The data summary for the $N1 = 450$ GeV signal can be found in Appendix B. The two data summaries are very similar, the 450 GeV signal has more data and different values. The prints show short looks at the features in the dataframes, like the values for a few events, the names and index dtypes of each feature in the dataframes. They also count and print the number of non-null values and the memory usage.

Then we make a design matrix $\mathbf{X}$, containing all the particle variables for each event, and a target vector $\mathbf{Y}$, containing all the targets for each event. The targets are at this point of type *tuples*. This makes classification more difficult, which is why we convert each event target in $\mathbf{Y}$ into an *integer* and

make a new target vector $\mathbf{y}$. Another useful thing to print is the individual target counts in the target vector $\mathbf{y}$ to check the number of each target in the dataframe. With this check, we quickly get an overview to see if we have a balanced or imbalanced data set. This can be very important to check, since it might lead to problems later. The target counts for both signals are seen in Table 9.1. We easily see that there is an imbalance in both the data sets, and that the 450 GeV signal has a lot more events.

$N1$	150 GeV	450 GeV
123	26801	34303
132	9716	10863
213	12871	65308
231	8454	139686
312	4013	3938
321	5030	5338

Table 9.1: The target counts for both signal samples without altering the data sets.

Correlations

An important descriptive statistic for data analysis with multi-variable data is the *correlation matrix* using Seaborn. It is a symmetric table of size $k \times k$, for k features (this includes the targets as well), with pairwise correlations between the features in the data. It summarizes the relationships between the features that we most likely would not have seen by just looking at them. With machine learning, it is also an early preprocessing step that can give some information to whereas dimensionality reduction might come in handy when dealing with high-dimensionality data. The closer to 1 the correlation coefficients are, the more correlated are they. We don't want a value close to -1, since this indicates a strong negative correlation. The diagonal will of course always be 1, since it is the correlation between the feature itself. When a feature has a strong correlation to the target, is has a high significance for predicting the output than a feature close to -1. This helps us exclude features that worsen the predictions. The optimal case is then a high positive correlation between the independent and dependent variables, while any strong correlation between the independent variables (causing redundancy) is not.

By using the correlations between the features, we print the pairs with strong correlation (magnitude greater than 0.7) for the $N1 = 150$ GeV signal:

```
lep1_theta  lep1_eta     −0.982589
lep2_theta  lep2_eta     −0.980482
lep3_eta    lep3_theta   −0.979638
lep4_eta    lep4_theta   −0.967329
lep2_py     lep1_py      −0.861290
lep2_px     lep1_px      −0.832358
lep1_pz     lep1_theta   −0.821220
lep2_pz     lep2_theta   −0.798809
lep3_theta  lep3_pz      −0.788853
lep4_pz     lep4_theta   −0.701025
lep3_phi    lep3_py       0.707593
lep1_pt     lep3_pt       0.712439
mll_13      lep2_pt       0.719664
            lep3_pt       0.753440
mll_12      mll_13        0.784307
lep4_pz     lep4_eta      0.812571
lep1_pt     mll_13        0.827285
lep2_pz     lep2_eta      0.863876
lep3_eta    lep3_pz       0.868294
lep1_pz     lep1_eta      0.872948
lep1_pt     mll_12        0.900048
mll_12      lep2_pt       0.900361
lep2_pt     lep1_pt       0.914949
dtype: float64
```

We print the same for the 450 GeV signal yielding similar results:

```
lep1_eta    lep1_theta   −0.991685
lep4_eta    lep4_theta   −0.987881
lep3_eta    lep3_theta   −0.984183
lep2_eta    lep2_theta   −0.984020
lep1_pz     lep1_theta   −0.894602
lep1_py     lep2_py      −0.890322
lep2_px     lep1_px      −0.851323
lep2_pz     lep2_theta   −0.849920
lep3_pz     lep3_theta   −0.805985
lep4_theta  lep4_pz      −0.778867
dR_14       dR_13        −0.759559
dR_13       dR_23        −0.715678
            dphi_23      −0.709584
lep2_py     lep1_phi     −0.707875
lep4_phi    lep4_py       0.704739
lep3_py     lep3_phi      0.718642
dR_24       mll_24        0.730663
lep2_py     lep2_phi      0.735340
lep3_pt     mll_13        0.752684
mll_14      dR_14         0.775010
lep1_phi    lep1_py       0.783622
dphi_23     dphi_12       0.783714
dR_13       mll_13        0.805792
lep4_eta    lep4_pz       0.806726
lep1_pt     lep2_pt       0.850018
mll_12      lep1_pt       0.862141
            lep2_pt       0.864872
lep3_pz     lep3_eta      0.872818
```

```
lep2_pz     lep2_eta     0.898086
lep1_pz     lep1_eta     0.924052
dtype: float64
```

The full correlation matrices can be seen in Figures *Correlation_150.png* and *Correlation_450.png* in the **Plots**-folder at the GitHub repository.

To take a closer look at the correlations in the data set, we use the mutual information of the features from section 6.5.1. By using the *mutual_info_classif* function by Scikit-Learn, we can easily compute the information gain of the features and the targets. We want the features that maximizes the information gain. This helps us single out unnecessary features for classification. The full information gains for all features can be seen in Appendix C.

From the strong correlation pairs, correlation matrices and the mutual information, we choose to remove the eta features for all the four particles (lep(1,2,3,4)_eta) in both signals since they show high correlations to other features and have low mutual information, as seen in Table 9.2. lep1_pt shows a high correlation with other features, but has one of the highest values for mutual information with the target, lep1_pt ≈ 0.2766 for the 150 GeV signal. This is why we do not remove it. In the case of decision trees, the information gain is used for the splitting in the trees.

$N1$	150 GeV	450 GeV
lep1_eta	0.061662610215777125	0.061662610215777125
lep2_eta	0.06035343605141508	0.06035343605141508
lep3_eta	0.05594756636754683	0.05594756636754683
lep4_eta	0.0648977837020408	0.0648977837020408

Table 9.2: Table for the mutual information of the eta variables for the leptons for both $N1 = 150$ GeV and $N1 = 450$ GeV signals. They all show very low values, indicating low correlations between these features and the target.

9.3.2 Resampling

After the feature selection, we make a function using the Imblearn library for imbalanced data. This allows us to choose between the options of both oversampling and undersampling, or only one of them by using boolean input parameters for activating these techniques. This function can be seen in Listing 9.3. This function will balance the data by first sampling the information

we already have, then resample the data set. Undersampling is used to decrease the size of the samples for one or more classes, while the oversampler increase the size of the samples for one or more classes. The *random_state* is used to reproduce the data when necessary, since the sampling algorithms will differ each time they are run otherwise.

```
    """Resample the data to make the datasets more balanced."""
    def Resample(X, y, under=False, over=False):
        if under == True:
            print("Undersample")
            undersample = RandomUnderSampler(sampling_strategy="↩
                majority", random_state=42)
            X, y = undersample.fit_resample(X, y)

        if over == True:
            print("Oversample")
            oversample = ADASYN(sampling_strategy="not majority", ↩
                random_state=42)
            X, y = oversample.fit_resample(X, y)

        #print(y.target.value_counts()) # Print the counts of the ↩
            different classes after resampling
        return X, y
```

Listing 9.3: Resample data.

By looking at the target counts in section 9.3.1, we see that both data sets are imbalanced. For the 150 GeV data we use only oversampling with the ADASYN algorithm in Scikit-Learn to create more data depending on the distribution of the classes we will oversample. All the classes except the class with highest count, the majority class, will be sampled with ADASYN. The new target counts after resampling, for one run, are seen in Table 9.1.

For the 450 GeV signal we have a lot more data than the other signal, but the data are still very imbalanced. We balance the data by first undersample the majority class with a RandomUnderSampler algorithm and then oversample the minority classes with the ADASYN algorithm. The RandomUnderSampler takes random samples from the majority class to produce a subset of the data with approximately the size of the biggest minority class. We then get two majority classes with approximately the same size before the ADASYN algorithm oversample the minority classes. The new target counts after resampling, for one run, are seen in Table 9.3.

9.3.3 Train, validation and test sets

Regardless of resampling or not, one important thing we have to do with our data when doing classification is to split the data into multiple sets. We split the design matrix $\mathbf{X}$, containing the features, and the target vector

$N1$	150 GeV	450 GeV
123	26801	60667
132	25198	65352
213	26088	65308
231	26122	65263
312	27109	64924
321	27527	64600

Table 9.3: The target counts for both signal samples after using resampling techniques on the data sets. Oversampling is used for the 150 GeV signal, while both undersampling and oversampling is used on the 450 GeV signal.

y into three new sets each. This is done by using a Scikit-Learn function called *train_test_split*, as seen in Listing 9.4. First we split **X** and **y** into training and test sets. The training sets are then further split into new smaller training sets and validation sets. We chosoe the splits to have 60% of the data as training data, 20% are validation data and 20% are test data. The validation set is used to tune the classification models, while the test set is only used as unseen data in the end when we have a good enough trained model.

```
    """Split events into training, validation and test sets."""
    X_train, X_test, y_train, y_test = train_test_split(X, y, ↩
        test_size=0.2, random_state=42, stratify=y)

    X_train, X_val, y_train, y_val = train_test_split(X_train, y_train↩
        , test_size=0.25, random_state=42)
```

Listing 9.4: Splitting the data.

9.3.4 Scaling

The next technique we will apply is scaling of the data. We will use *standardization* of the data, which means we transform the values with a mean of 0 and a standard deviation of 1. This will fix any unwanted weighting favoring some features. Scikit-Learn has a function for doing this called *StandardScaler*. We will both fit and transform the training data, meaning that we both compute the mean and standard deviation to standardize the training set. We have to transform the validation and test sets as well with the scaler, but we don't fit them. In Listing 9.5 we use the *fit_transform* on the training set, while only using *transform* on the validation and test sets. Note that we

only scale the features, since scaling the targets will assign a distribution to the categorical features. We do not want to do that.

```
        """Scale the data when called."""
        def scaler(X_train, X_val, X_test):
            sc = StandardScaler()
            X_train = sc.fit_transform(X_train)
            X_val = sc.transform(X_val)
            X_test = sc.transform(X_test)
            return X_train, X_val, X_test
```

Listing 9.5: Scaling.

Chapter 10

Model Classification

In this chapter we will use the two signals to train several classification models, before we choose the best performing model for each of the signals and store them in separate files. Then we use these fully trained models to do the same classification for several background and signal Ntuples. New Ntuples with the predicted classes are converted back in to ROOT, and used to make new distributions with a selection of cuts applied.

10.1 Training the Classification Models

With all the data prepared and ready for classification, we start with the training of the classification models on the validation set with various hyperparameters. We create a useful function that uses the *RandomizedSearchCV* function in Scikit-Learn to test several different values of hyperparameters for some chosen model using a randomized search with cross-validation. This is much easier than changing one hyperparameter at a time for each run, since this randomized search function can test several hyperameters in one run. Our function in Listing 10.1 prints the results of the randomized search given some set(s) of hyperparameters. The results include the mean test scores for each combination of hyperparameter set given, and the best mean test score with the corresponding hyperparameters.

```
        def getTrainScores(gs):
            # Function that prints the RandomizedSearchCV best parameters ↩
                and mean scores
            print("Start getTrainScores:")
            gs.fit(X_train, np.ravel(y_train))
            results = {}
            runs = 0
            for x,y in zip(list(gs.cv_results_['mean_test_score']), gs.↩
                cv_results_['params']):
```

```
            results[runs] = 'mean:' + str(x) + 'params' + str(y)
            runs += 1
        best = {'best_mean': gs.best_score_, "best_param":gs.↩
            best_params_}
        print(results)
        print(best)
        return results, best
```

Listing 10.1: Training models.

After training all the models with the randomized search function and implementing the best hyperparameters for each model, we use the validation set to test and compare the classification models. To evaluate and compare the models, we use the accuracy score on both the training and validation sets. This lets us see if we have any overfitting when the training accuracy score is much higher than the validation accuracy score. It is the accuracy score and the confusion matrices we will use as the main evaluation methods to check the model performances with the validation set. We also look at the variance and bias of the models. The XGBoost and LGBM models lets us plot the errors and log losses to see the convergence and fitting of these two models. The best overall performing model for each signal sample will be chosen for further use.

10.2 Classification with Test Set

After analyzing the models with the validation set by looking at the prints and plots, we choose the best performing model for each signal. We now do a new evaluation using the test set, and use the evaluation metrics from section 6.5 that evaluates classification model performance, to check that the performances of the best models are satisfying.

When the performances of the best models are good enough, we use a Scikit-Learn module called *Pickle* to save the models to separate .pkl-files. These files can be loaded and exported to be used on new unseen data with the same features we have trained on. This quick and easy way of loading already trained models, lets us skip the training of the models, such that we can go straight to predicting the outcomes on the new data.

10.3 Ntuple Classification

The models have been trained on two samples of signal data with truth variables for classifying leptons. Now we want to classify and predict the lepton vertices in various background data. The background data are originally ROOT-files which have to be converted to dataframes with the same

features like the truth signals. We use a new script for classification of the backgrounds and signal Ntuples, called *Trilepton_classify_Ntuples.py*. This time we do not create the target variables since the backgrounds are not truth data. We also do this with two signal samples, similar to the samples used in classification but containing more data and more variables.

After converting the backgrounds and signals to dataframes, we load the files containing the trained best models for the two signals and use them to classify the background and signal lepton vertices. For each dataframe, the predicted outcomes are saved in the dataframe before the dataframe is converted back into ROOT. The ROOT-files now containing all original variables, as well as the variables we produced and used for classification and the predicted outcomes.

10.4 Signal Region Distributions

We can then plot the background and signal variable distributions again using the first ROOT-plotting scripts. We will define some signal regions and apply some cuts to the variables. The outcome classes (lepton vertices) from the classification are used to define different cuts. We will use the lepton flavors, same flavor (SF) and different flavor (DF), and signs of the leptons, opposite sign (OS), for lepton 1 and 2 as cuts as well. These cuts are seen in Table 10.1. Then we will use some (benchmark) cuts for a more "standard" analysis at $\sqrt{s} = 14$ TeV from Pascoli et al. [4]. These cuts are seen in Table 10.2.

Signal region cuts:	
SF & OS	lep1_Flavor == lep2_Flavor & lep1_Charge != lep2_Charge
DF & OS	lep1_Flavor != lep2_Flavor & lep1_Charge != lep2_Charge
Lepton vertices	classes == [123, 132, 213, 231, 312, 321]

Table 10.1: Signal region cuts used for plotting variable distributions of Ntuples for backgrounds and signals with classification variables and predicted lepton vertices. Cuts to be applied leptons 1 and 2 to have same flavor (SF) and opposite sign (OS) and leptons 1 and 2 to have different flavor (different flavor (DF)) and OS. Combine them with the cuts for lepton vertices.

Benchmark "Standard" Analysis at $\sqrt{s} = 14$ TeV:
$m_{l_i,l_j} > 10$ GeV, $\quad \lvert m_{l_i,l_j} - M_Z \rvert > 15$ GeV, $\quad \lvert m_{3l} - M_Z \rvert > 15$ GeV, $p_T^{\text{b-Tagged}} < 25$ GeV, $\quad p_T^{l_1} > 55$ GeV, $\quad p_T^{l_2} > 15$ GeV, $\quad m_{3l} > 80$ GeV

Table 10.2: Cuts used for a benchmark analysis to be compared with our cuts from Table 10.1. Reference: Table 6 in Pascoli et al. [4].

Part III

*Results

Chapter 11

Variable Distributions

Chapter 12

*Classification Results

To classify leptons in background and signal Ntuples, we first train classification models on two signals samples with truth data and choose the best performing model for each signal sample. Then we evaluate the performance of these models with classification evaluation metrics, before we use these models to classify backgrounds and full signal Ntuples.

We present the classification results in this chapter with plots and printed results used to evaluate the classification models.

12.1 *Choosing the Best Performing Models*

In Table 12.1 we see the evaluation metrics used to evaluate the performance of the classification models on the 150 GeV signal validation set after tuning each model. There we see the accuracy score of both the validation and training sets, the balanced accuracy score, the variance and the bias for each classifier.

12.2 *Evaluation of the Best Models*

N1_150 best model LGBM on test data:

```
        Assess final best LGBM model evaluation with test set:
        Best iter: 1600
        Best score: defaultdict(<class 'collections.OrderedDict'>, {'↩
            training': OrderedDict([('multi_logloss', 0.01564531706499713)↩
            , ('multi_error', 0.0)]), 'valid_1': OrderedDict([('↩
            multi_logloss', 0.33348715748131275), ('multi_error', ↩
            0.12071516257987346)])})
```

Model	Score	Score_train	BAcc	Var	Bias
LogRegCV	0.409676	0.412992	0.408724	5932.8642	6099.5482
DecisionTree	0.608518	0.795671	0.607949	6033.3890	6086.2548
AdaBoost	0.851900	1.000000	0.850790	5796.7615	6088.7001
RandomForest	0.765558	0.911612	0.764617	5751.0957	6136.3113
OvR	0.774592	0.930866	0.773828	5819.3619	6123.2995
OvO	0.778810	0.929900	0.777738	5841.6059	6135.7178
MLP	0.822657	0.949238	0.822251	6045.2875	6044.7911
HGBC	0.786301	0.899881	0.785906	6023.6668	6058.7670
XGBoost	0.863106	0.999769	0.862487	6016.7689	6053.3081
LGBM	0.877868	0.999926	0.877134	6046.5238	6055.6849

Table 12.1: Table containing evaluation values with the 150 GeV signal validation set of the classification models in section 6.4. From left to right: The classification model, accuracy score of validation set, accuracy score of training set, balanced accuracy score, variance and bias.

Model	Score	Score_train	BAcc	Var	Bias
LogRegCV	0.696451	0.696825	0.694335	5980.4339	5974.8244
DecisionTree	0.850938	0.916329	0.848865	6048.3255	5953.8820
AdaBoost	0.938490	1.000000	0.937000	6107.4315	5948.8834
RandomForest	0.899343	0.922821	0.896808	6057.5101	5972.7990
OvR	0.906038	0.931635	0.903435	6045.4265	5972.2990
OvO	0.908771	0.931885	0.906208	6046.0539	5971.0928
MLP	0.934993	0.960599	0.933578	5984.1467	5950.7039
HGBC	0.928027	0.957107	0.925760	5980.3233	5961.7443
XGBoost	0.950883	0.999879	0.949377	6005.1208	5951.0907
LGBM	0.954081	0.999922	0.952588	5999.1799	5950.8842

Table 12.2: Table containing evaluation values with the 450 GeV signal validation set of the classification models in section 6.4. From left to right: The classification model, accuracy score of validation set, accuracy score of training set, balanced accuracy score, variance and bias.

```
Precision: 0.8795608937023826
Classification report LGBM:
precision    recall  f1-score   support

123       0.88      0.84      0.86      5360
132       0.90      0.85      0.87      5040
213       0.86      0.85      0.85      5218
231       0.89      0.88      0.88      5224
312       0.87      0.94      0.91      5422
321       0.89      0.91      0.90      5505

accuracy                           0.88     31769
macro avg       0.88      0.88      0.88     31769
weighted avg       0.88      0.88      0.88     31769

Cross-validation score:
[0.76786276 0.77478754 0.77447277 0.78155493 0.77396506]
Highest accuracy score from cross-val:
0.7815549260308468
Mean accuracy score from cross-val:
0.7745286107981076
Standard deviation of cross-val:
0.004340285207078167

Conf matrix:
[[4497  342  273   16  201   31]
[ 408 4268   96   78  158   32]
[ 124   65 4428  191  222  188]
[   4   28  159 4610   87  336]
[  68   54  106   29 5114   51]
[   2    8  111  278   89 5017]]

Model              LGBM
Score          0.879285
Score_train         1.0
CKS            0.855092
BAcc           0.878572
LogLoss        0.333487
Var           6033.5629
Bias          6054.7185
```

N1_450 best model LGBM on test data:

```
Assess final best LGBM model evaluation with test set:
Best iter: 1540
Best score: defaultdict(<class 'collections.OrderedDict'>, {'↩
    training': OrderedDict([('multi_logloss', ↩
    0.005884714604339098), ('multi_error', 0.0)]), 'valid_1': ↩
    OrderedDict([('multi_logloss', 0.11199173258105581), ('↩
    multi_error', 0.043018271758413947)])})
Precision: 0.9586760984764053
Classification report LGBM:
precision    recall  f1-score   support

123       0.97      0.83      0.89     12133
132       0.92      0.99      0.96     13070
213       0.99      0.92      0.95     13062
231       1.00      1.00      1.00     13053
312       0.93      1.00      0.97     12985
321       0.94      0.99      0.97     12920
```

```
accuracy                           0.96     77223
macro avg       0.96      0.96      0.96     77223
weighted avg       0.96      0.96      0.96     77223

Cross-validation score:
[0.92826157 0.92606021 0.92923276 0.92994043 0.92832168]
Highest accuracy score from cross-val:
0.9299404299404299
Mean accuracy score from cross-val:
0.9283633313319296
Standard deviation of cross-val:
0.0013084027937776625

Conf matrix:
[[10064  1100   108     2   831    28]
[   42 12991     2     3    31     1]
[  255    39 11967    19    20   762]
[    0     0     0 13052     0     1]
[    6     3     0     0 12972     4]
[    3     0    37     0    25 12855]]

Model               LGBM
Score           0.956982
Score_train          1.0
CKS             0.948356
BAcc            0.955582
LogLoss         0.111992
Var            5997.1512
Bias           5948.8764
```

12.3 *Classify Background and Signal Ntuples*

Predicted outcomes

```
\underline{LFCMN1150}: (150)
123    7358
132    4879



\underline{LFCMN1450}: (450)
123    7856
132    5057
213       1
```

Chapter 13

Post-Classification Distribution Analysis

Part IV

Discussion, Conclusion and Future Prospects

Chapter 14

Discussion

14.1 ?

Chapter 15

Conclusion and Future Work

15.1 Future Work

Part V

Appendices

Appendix A

Bias-Variance Decomposition

Here we do the full derivation of the expected generalization error in equation 6.9:

$$\begin{aligned}
\mathbb{E}_{\mathcal{D},\epsilon}[\mathcal{C}(\mathbf{y}, f(\mathbf{X}; \hat{\theta}_{\mathcal{D}}))] &= \mathbb{E}_{\mathcal{D},\epsilon}\left[\sum_i (y_i - f(\mathbf{x}_i; \hat{\theta}_{\mathcal{D}}))^2\right] \\
&= \mathbb{E}_{\mathcal{D},\epsilon}\left[\sum_i (y_i - f(\mathbf{x}_i; \hat{\theta}_{\mathcal{D}}) - f(\mathbf{x}_i) + f(\mathbf{x}_i))^2\right] \\
&= \sum_i \mathbb{E}_{\epsilon}[(y_i - f(\mathbf{x}_i))^2] + \mathbb{E}_{\mathcal{D},\epsilon}[(f(\mathbf{x}_i) - f(\mathbf{x}_i; \hat{\theta}_{\mathcal{D}}))^2] \\
&\quad + 2\mathbb{E}_{\epsilon}[y_i - f(\mathbf{x}_i)]\mathbb{E}_{\mathcal{D}}[f(\mathbf{x}_i; \hat{\theta}_{\mathcal{D}})] \\
&= \sum_i \sigma_\epsilon^2 + \mathbb{E}_{\mathcal{D}}[(f(\mathbf{x}_i) - f(\mathbf{x}_i; \hat{\theta}_{\mathcal{D}}))^2]
\end{aligned}$$

Here we have used the fact that the noise has zero mean and variance σ_ϵ^2. We also further decompose the second expectation term:

$$\begin{aligned}
\mathbb{E}_{\mathcal{D}}[(f(\mathbf{x}_i) - f(\mathbf{x}_i; \hat{\theta}_{\mathcal{D}}))^2] &= \mathbb{E}_{\mathcal{D}}\left[(f(\mathbf{x}_i) - f(\mathbf{x}_i; \hat{\theta}_{\mathcal{D}}) - \mathbb{E}_{\mathcal{D}}[f(\mathbf{x}_i; \hat{\theta}_{\mathcal{D}})] + \mathbb{E}_{\mathcal{D}}[f(\mathbf{x}_i; \hat{\theta}_{\mathcal{D}})])^2\right] \\
&= (f(\mathbf{x}_i) - \mathbb{E}_{\mathcal{D}}[f(\mathbf{x}_i; \hat{\theta}_{\mathcal{D}})])^2 + \mathbb{E}_{\mathcal{D}}[\{f(\mathbf{x}_i; \hat{\theta}_{\mathcal{D}}) - \mathbb{E}[f(\mathbf{x}_i; \hat{\theta}_{\mathcal{D}})]\}^2]
\end{aligned}$$

Putting these two equation together leads to the expected generalization error:

$$\begin{aligned}
\mathbb{E}_{\mathcal{D},\epsilon}[\mathcal{C}(\mathbf{y}, f(\mathbf{X}; \hat{\theta}_{\mathcal{D}}))] &= \sum_i (f(\mathbf{x}_i) - \mathbb{E}_{\mathcal{D}}[f(\mathbf{x}_i; \hat{\theta}_{\mathcal{D}})])^2 \\
&\quad + \sum_i \mathbb{E}_{\mathcal{D}}[\{f(\mathbf{x}_i; \hat{\theta}_{\mathcal{D}}) - \mathbb{E}[f(\mathbf{x}_i; \hat{\theta}_{\mathcal{D}})]\}^2] \\
&\quad + \sum_i \sigma_\epsilon^2
\end{aligned}$$

Appendix B

N1=450 GeV Data Summary

The $N1 = 450$ GeV data summary:

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 259436 entries, 0 to 261331
Data columns (total 55 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   lep1_pt      259436 non-null  float32
 1   lep1_phi     259436 non-null  float32
 2   lep1_eta     259436 non-null  float32
 3   lep1_theta   259436 non-null  float32
 4   lep1_px      259436 non-null  float32
 5   lep1_py      259436 non-null  float32
 6   lep1_pz      259436 non-null  float32
 7   lep1_E       259436 non-null  float32
 8   lep1_tlv     259436 non-null  object
 9   lep2_pt      259436 non-null  float32
 10  lep2_phi     259436 non-null  float32
 11  lep2_eta     259436 non-null  float32
 12  lep2_theta   259436 non-null  float32
 13  lep2_px      259436 non-null  float32
 14  lep2_py      259436 non-null  float32
 15  lep2_pz      259436 non-null  float32
 16  lep2_E       259436 non-null  float32
 17  lep2_tlv     259436 non-null  object
 18  lep3_pt      259436 non-null  float32
 19  lep3_phi     259436 non-null  float32
 20  lep3_eta     259436 non-null  float32
 21  lep3_theta   259436 non-null  float32
 22  lep3_px      259436 non-null  float32
 23  lep3_py      259436 non-null  float32
 24  lep3_pz      259436 non-null  float32
 25  lep3_E       259436 non-null  float32
 26  lep3_tlv     259436 non-null  object
 27  lep4_pt      259436 non-null  float32
 28  lep4_phi     259436 non-null  float32
 29  lep4_eta     259436 non-null  float32
 30  lep4_theta   259436 non-null  float32
 31  lep4_px      259436 non-null  float32
 32  lep4_py      259436 non-null  float32
 33  lep4_pz      259436 non-null  float32
```

```
 34  lep4_E       259436 non-null  float32
 35  lep4_tlv     259436 non-null  object
 36  mll_12       259436 non-null  float64
 37  dphi_12      259436 non-null  float64
 38  dR_12        259436 non-null  float64
 39  mll_13       259436 non-null  float64
 40  dphi_13      259436 non-null  float64
 41  dR_13        259436 non-null  float64
 42  mll_23       259436 non-null  float64
 43  dphi_23      259436 non-null  float64
 44  dR_23        259436 non-null  float64
 45  mll_14       259436 non-null  float64
 46  dphi_14      259436 non-null  float64
 47  dR_14        259436 non-null  float64
 48  mll_24       259436 non-null  float64
 49  dphi_24      259436 non-null  float64
 50  dR_24        259436 non-null  float64
 51  mll_34       259436 non-null  float64
 52  dphi_34      259436 non-null  float64
 53  dR_34        259436 non-null  float64
 54  target       259436 non-null  object
dtypes: float32(32), float64(18), object(5)
memory usage: 79.2+ MB

             lep1_pt  lep1_phi  lep1_eta  ...    dphi_34     dR_34 ↩
                 target
entry                                      ...
0       247239.234375  1.705486 -0.009060  ... -0.059092  0.872886 ↩
                (2, 3, 1)
1       242870.343750 -2.694518 -1.187190  ...  0.105023  0.717894 ↩
                (2, 1, 3)
2       275632.000000 -0.628263 -0.185416  ...  1.024500  1.044483 ↩
                (2, 1, 3)
3       203956.265625  2.971124 -2.185891  ... -0.364194  0.678069 ↩
                (2, 1, 3)
4       106095.476562 -0.685928 -0.315220  ... -0.983589  1.397503 ↩
                (2, 1, 3)

[5 rows x 55 columns]
```

Appendix C

Correlations

Signal $N1 = 150$ GeV:

```
Information gain of the features:
OrderedDict([('lep2_phi', 0.05178672925582184), ('lep1_phi', ↩
    0.053745432520311276), ('dphi_24', 0.05485490369918944), ('↩
    lep3_phi', 0.05502894645503176), ('lep3_eta', ↩
    0.05594756636754683), ('lep3_theta', 0.05598787413359707), ('↩
    lep4_phi', 0.05641456053531346), ('lep4_pz', ↩
    0.05947799347697558), ('lep2_eta', 0.06035343605141508), ('↩
    lep2_theta', 0.060373423669804804), ('dphi_14', ↩
    0.06083513368874938), ('dphi_34', 0.06115565193619199), ('↩
    lep1_eta', 0.061662610215777125), ('lep1_theta', ↩
    0.06168403392066146), ('lep3_pz', 0.06422467081537953), ('↩
    lep4_theta', 0.06485782851335387), ('lep4_eta', ↩
    0.0648977837020408), ('lep4_py', 0.06604010400547766), ('↩
    lep4_px', 0.06752198284777089), ('lep4_E', ↩
    0.06793095808433014), ('dR_34', 0.06849304962655012), ('↩
    lep1_pz', 0.06885166365937634), ('lep2_pz', ↩
    0.06906061765196592), ('dR_13', 0.0716082982162769), ('dR_14',↩
     0.07197872976190567), ('dphi_13', 0.07423421063493052), ('↩
    dR_24', 0.07595752322276583), ('lep3_E', 0.08410378910700622),↩
     ('lep4_pt', 0.09082301709001017), ('dphi_23', ↩
    0.09258874841582809), ('dphi_12', 0.09770848996184833), ('↩
    lep1_E', 0.10512713284596842), ('lep3_px', ↩
    0.10532072166635142), ('lep2_E', 0.10629293534281459), ('↩
    lep3_py', 0.10734840100318799), ('dR_12', 0.10964280256859293)↩
    , ('dR_23', 0.11751795736407011), ('mll_24', ↩
    0.12528976189890173), ('lep1_px', 0.14667014422688363), ('↩
    lep2_px', 0.14672888551516206), ('lep2_py', ↩
    0.15222854675813702), ('lep1_py', 0.1559645954915414), ('↩
    lep3_pt', 0.17443874057539732), ('mll_14', ↩
    0.19302957284767652), ('mll_34', 0.20736617389081324), ('↩
    mll_23', 0.2081857239040863), ('lep2_pt', 0.27355582731666006)↩
    , ('lep1_pt', 0.276609594149106), ('mll_13', ↩
    0.28691694495451703), ('mll_12', 0.3932658736116754)])
```

Signal $N1 = 450$ GeV:

```
Information gain of the features:
```

```
        OrderedDict([('lep4_theta', 0.4191552527318376), ('lep4_phi', ↩
            0.42027977972758146), ('lep3_eta', 0.42045370659465386), ('↩
            lep1_phi', 0.4210809806293345), ('lep2_pz', ↩
            0.42162772292849215), ('lep2_eta', 0.42201102378344824), ('↩
            lep2_phi', 0.4230067197296721), ('lep1_pz', 0.424541501741019)↩
            , ('lep4_eta', 0.4257280226520528), ('lep3_pz', ↩
            0.42672816312478656), ('lep3_phi', 0.427599991411618), ('↩
            lep3_theta', 0.42907751026540675), ('lep4_pz', ↩
            0.4301374641613722), ('lep2_theta', 0.4312103916746548), ('↩
            lep1_E', 0.43129937565133747), ('lep4_px', ↩
            0.43132544346123725), ('dR_12', 0.4322203904822728), ('lep4_E'↩
            , 0.43372185011808995), ('dphi_12', 0.43389037653643614), ('↩
            lep4_py', 0.43395756196715096), ('dR_13', 0.4347988629887223),↩
             ('dphi_13', 0.43616113224927466), ('lep1_eta', ↩
            0.43616248827916504), ('lep1_theta', 0.4386034683797735), ('↩
            lep3_E', 0.44167047831807293), ('lep2_E', 0.44415728982148406)↩
            , ('lep3_px', 0.4471424012534033), ('dphi_24', ↩
            0.4487297203828642), ('lep3_py', 0.44887857656083185), ('↩
            lep1_px', 0.45032798796079376), ('lep1_py', ↩
            0.4515757089823731), ('lep4_pt', 0.45213792012843435), ('↩
            lep2_px', 0.4580357692816419), ('lep2_py', 0.4610110981533644)↩
            , ('dphi_14', 0.4611574842208519), ('dphi_23', ↩
            0.462426274698871), ('dR_24', 0.4692019081992844), ('dphi_34',↩
             0.4784639532967272), ('dR_14', 0.4809026477868572), ('lep3_pt↩
            ', 0.48225046197217947), ('lep1_pt', 0.49983354913044953), ('↩
            dR_34', 0.5095008908298231), ('lep2_pt', 0.5120865029463617), ↩
            ('dR_23', 0.5122830173546382), ('mll_24', 0.5187574240316617),↩
             ('mll_13', 0.540453212175724), ('mll_14', 0.5424036353399868)↩
            , ('mll_12', 0.5602705982085625), ('mll_34', ↩
            0.6049717229179872), ('mll_23', 0.6204488569318036)])
```

Acronyms

ALICE A Large Ion Collider Experiment

ATLAS A Toroidal LHC ApparatuS

BEH Brout-Englert-Higgs

CART Classification and Regression Trees

CCDY charged current Drell-Yan

CERN European Organization for Nuclear Research (EN)

CKS Cohen Kappa Score

CLIC Compact Linear Collider

CM center-of-mass

CMS Compact Muon Solenoid

DIS Deep Inelastic Scattering

DT Decision Tree

ECal electromagnetic calorimeter

EF Event filter

EWT electroweak theory

FCC Future Circular Collider

FFNN Feed-Forward Neural Network

GBDT Gradient Boosting Decision Tree

GWS Glashow-Weinberg-Salam

HCal hadronic calorimeter

HLT High Level Trigger

ID inner detector

ISS Inverse Seesaw

KATRIN Karlsruhe Tritium Neutrino

LEP Large Electron-Positron Collider

LFV lepton flavor violation

LGBM Light Gradient Boosting Machine

LH left-handed

LHC Large Hadron Collider

LHCb LHC-beauty

LR Logistic Regression

MC Monte Carlo

MET missing transverse momentum

ML machine learning

MLE Maximum Likelihood Estimation

MLP Multi-Layer Perceptron

MLR Multinomial Logistic Regression

MS muon spectrometer

OS opposite sign

OvO one-vs-one

OvR one-vs-rest

PCA Principal Component Analysis

PDF parton distribution function

PMNS Pontecorvo-Maki-Nakagawa-Sakata

QCD quantum chromodynamics

QED quantum electrodynamics

QFT quantum field theory

ReLU Rectified Linear Unit

RH right-handed

RnF Random Forest

ROC Receiver Operating Characteristic

ROI region of interest

SCT Semiconductor Tracker

SM Standard Model

SNO Sudbury Neutrino Observatories

SPS Super Proton Synchrotron

SS same sign

TDAQ The ATLAS Trigger and Data Acquisition system

TRT Transition Radiation Tracker

WLCG World LHC Computing Grid

XGBoost Extreme Gradient Boosting

Bibliography

[1] Georges Aad, Tatevik Abajyan, B Abbott, J Abdallah, S Abdel Khalek, Ahmed Ali Abdelalim, R Aben, B Abi, M Abolins, OS AbouZeid, et al. Observation of a new particle in the search for the standard model higgs boson with the atlas detector at the lhc. *Physics Letters B*, 716(1):1–29, 2012.

[2] Serguei Chatrchyan, Vardan Khachatryan, Albert M Sirunyan, Armen Tumasyan, Wolfgang Adam, Ernest Aguilo, Thomas Bergauer, M Dragicevic, J Erö, C Fabjan, et al. Observation of a new boson at a mass of 125 gev with the cms experiment at the lhc. *Physics Letters B*, 716(1): 30–61, 2012.

[3] Arindam Das, Natsumi Nagata, and Nobuchika Okada. Testing the 2-TeV resonance with trileptons. *Journal of High Energy Physics*, 2016 (3):49, 2016.

[4] Silvia Pascoli, Richard Ruiz, and Cedric Weiland. Heavy neutrinos with dynamic jet vetoes: multilepton searches at $\sqrt{s}$= 14, 27, and 100 TeV. *Journal of High Energy Physics*, 2019(6):49, 2019.

[5] Mark Thomson. *"Modern particle physics*. Cambridge University Press, 2013.

[6] Glenn Elert. The physics hypertextbook, 1998-2020. URL https://physics.info/standard/.

[7] *Why is the Higgs discovery so significant?* Science and Technology Facilities Council, 2017. URL https://stfc.ukri.org/research/particle-physics-and-particle-astrophysics/peter-higgs-a-truly-british-scientist/why-is-the-higgs-discovery-so-significant/.

[8] Benjamin P Abbott, Richard Abbott, TD Abbott, MR Abernathy, Fausto Acernese, Kendall Ackley, Carl Adams, Thomas Adams, Paolo

Addesso, RX Adhikari, et al. Observation of gravitational waves from a binary black hole merger. *Physical review letters*, 116(6):061102, 2016.

[9] Y Fukuda, T Hayakawa, E Ichihara, K Inoue, K Ishihara, Hirokazu Ishino, Y Itow, T Kajita, J Kameda, S Kasuga, et al. Evidence for oscillation of atmospheric neutrinos. *Physical Review Letters*, 81(8): 1562, 1998.

[10] Christopher W Walter and Super-Kamiokande collaboration. The super-kamiokande experiment. In *Neutrino Oscillations: Present Status and Future Plans*, pages 19–43. World Scientific, 2008.

[11] Alain Bellerive, JR Klein, AB McDonald, AJ Noble, AWP Poon, SNO Collaboration, et al. The sudbury neutrino observatory. *Nuclear Physics B*, 908:30–51, 2016.

[12] Elizabeth Gibney and Davide Castelvecchi. Neutrino flip wins physics prize, 2015.

[13] Max Aker, K Altenmüller, M Arenz, M Babutzka, J Barrett, S Bauer, M Beck, A Beglarian, J Behrens, T Bergmann, et al. Improved upper limit on the neutrino mass from a direct kinematic method by katrin. *Physical review letters*, 123(22):221802, 2019.

[14] Alan Kostelecky. The status of cpt. *arXiv preprint hep-ph/9810365*, 1998.

[15] Franz Mandl and Graham Shaw. *Quantum field theory*. John Wiley & Sons, 2010.

[16] M. Bellac. Quantum and statistical field theory. 1991.

[17] Richard P Feynman. The principle of least action in quantum mechanics. In *Feynman's Thesis—A New Approach To Quantum Theory*, pages 1–69. World Scientific, 2005. doi: https://doi.org/10.1142/9789812567635_0001.

[18] Gerhard Ecker. Quantum chromodynamics. *arXiv preprint hep-ph/0604165*, 2006.

[19] C. N. Yang and R. L. Mills. Conservation of isotopic spin and isotopic gauge invariance. *Phys. Rev.*, 96:191–195, Oct 1954. doi: 10.1103/PhysRev.96.191. URL https://link.aps.org/doi/10.1103/PhysRev.96.191.

[20] Richard Phillips Feynman. *QED: The strange theory of light and matter*. Princeton University Press, 2006.

[21] Guido Altarelli. The standard electroweak theory and beyond. *arXiv preprint hep-ph/9811456*, pages 27–93, 2000.

[22] Sheldon L Glashow. Partial-symmetries of weak interactions. *Nuclear physics*, 22(4):579–588, 1961. doi: https://doi.org/10.1016/0029-5582(61)90469-2.

[23] Steven Weinberg. A model of leptons. *Physical review letters*, 19(21): 1264, 1967. doi: https://doi.org/10.1103/PhysRevLett.19.1264.

[24] A Salam. N. svartholm, ed. elementary particle physics: Relativistic groups and analyticity. In *Eighth Nobel Symposium. Stockholm: Almquvist and Wiksell*, page 367, 1968.

[25] *Press release: The Nobel Prize in Physics 1979*. Nobel Media AB 2020, 1979. URL https://www.nobelprize.org/prizes/physics/1979/press-release/.

[26] Abdus Salam and Steven Weinberg. Nobel Prize for Physics, 1979. *CERN Courier*, page 395, 1979.

[27] Darwin Chang and Otto CW Kong. Pseudo-dirac neutrinos. *Physics Letters B*, 477(4):416–423, 2000.

[28] KRS Balaji, Anna Kalliomäki, and Jukka Maalampi. Revisiting pseudo-dirac neutrinos. *Physics Letters B*, 524(1-2):153–160, 2002.

[29] Rabindra N Mohapatra. Seesaw mechanism and its implications. In *SEESAW 25*, pages 29–44. World Scientific, 2005.

[30] Eirik Gramstad. Searches for Supersymmetry in di-Lepton Final States with the ATLAS Detector at $\sqrt{s}$= 7 TeV. 2013.

[31] Richard D Field. The underlying event in hard scattering processes. *arXiv preprint hep-ph/0201192*, 2002.

[32] About CERN. CERN, (10.12.20). URL https://home.cern/about.

[33] Stephanie Sammartino McPherson. *Tim Berners-Lee: Inventor of the World Wide Web*. Twenty-First Century Books, 2009.

[34] Esma Mobs. The CERN accelerator complex - 2019. Complexe des accélérateurs du CERN - 2019. Jul 2019. URL https://cds.cern.ch/record/2684277. General Photo.

[35] XinChou Lou. The Circular Electron Positron Collider. *Nat. Rhev. Phys.*, 1:232–234, 2019. doi: https://doi.org/10.1038/s42254-019-0047-1.

[36] Jie Gao. China's bid for a circular electron-positron collider. *CERN Courier*, 2018.

[37] Shinichiro Michizono. The International Linear Collider. *Nat. Rhev. Phys.*, 1:244–245, 2019. doi: https://doi.org/10.1038/s42254-019-0044-4.

[38] Philip Bambade, Tim Barklow, Ties Behnke, Mikael Berggren, James Brau, Philip Burrows, Dmitri Denisov, Angeles Faus-Golfe, Brian Foster, Keisuke Fujii, et al. The international linear collider: a global project. *arXiv preprint arXiv:1903.01629*, 2019.

[39] The Large Hadron Collider. CERN, (11.12.20). URL https://home.cern/science/accelerators/large-hadron-collider.

[40] Accelerators. CERN, (11.12.20). URL https://home.cern/science/accelerators.

[41] Experiments: LHC experiments. CERN, (11.12.20). URL https://home.cern/science/experiments.

[42] A Airapetian, V Dodonov, L Micu, D Axen, V Vinogradov, D Akerman, B Szeless, P Chochula, C Geich-Gimbel, P Schacht, et al. *ATLAS detector and physics performance: Technical Design Report, 1*, volume 1. 1999.

[43] CERN. Computer generated image of the whole ATLAS detector. CERN Document Server, (01.01.21). URL https://cds.cern.ch/record/1095924?ln=en.

[44] How ATLAS detects particles: diagram of particle paths in the detector . CERN Document Server, (01.01.21). URL https://cds.cern.ch/record/1505342?ln=en.

[45] ATLAS Collaboration, Georges Aad, JM Butterworth, J Thion, U Bratzler, PN Ratoff, RB Nickerson, JM Seixas, I Grabowska-Bold, F Meisel,

S Lokwitz, et al. The atlas experiment at the cern large hadron collider. *Jinst*, 3:S08003, 2008.

[46] CERN. Overall detector concept. *ATLAS Technical Proposal*, 1994.

[47] DA Scannicchio. Atlas trigger and data acquisition: Capabilities and commissioning. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 617(1-3):306–309, 2010.

[48] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[49] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[50] Pankaj Mehta, Marin Bukov, Ching-Hao Wang, Alexandre GR Day, Clint Richardson, Charles K Fisher, and David J Schwab. A high-bias, low-variance introduction to machine learning for physicists. *Physics reports*, 810:1–124, 2019.

[51] Dennis W Ruck, Steven K Rogers, and Matthew Kabrisky. Feature selection using a multilayer perceptron. *Journal of Neural Network Computing*, 2(2):40–48, 1990.

[52] Sandra Vieira, Walter HL Pinaya, and Andrea Mechelli. Using deep learning to investigate the neuroimaging correlates of psychiatric and neurological disorders: Methods and applications. *Neuroscience & Biobehavioral Reviews*, 74:58–75, 2017.

[53] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[54] S Rasoul Safavian and David Landgrebe. A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics*, 21(3):660–674, 1991.

[55] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

[56] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[57] J Ross Quinlan et al. Bagging, boosting, and c4. 5. In *Aaai/iaai, Vol. 1*, pages 725–730, 1996.

[58] Gunnar Rätsch, Takashi Onoda, and K-R Müller. Soft margins for adaboost. *Machine learning*, 42(3):287–320, 2001.

[59] Jerome H Friedman. Stochastic gradient boosting. *Computational statistics & data analysis*, 38(4):367–378, 2002.

[60] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30:3146–3154, 2017.

[61] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[62] Abha Eli Phoboo. Machine learning wins the higgs challenge. Technical report, 2014.

[63] Dymitr Ruta and Bogdan Gabrys. Classifier selection for majority voting. *Information fusion*, 6(1):63–81, 2005.

[64] Jingjing Cao, Sam Kwong, Ran Wang, Xiaodong Li, Ke Li, and Xiangfei Kong. Class-specific soft voting based multiple extreme learning machines ensemble. *Neurocomputing*, 149:275–284, 2015.

[65] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[66] Mary L McHugh. Interrater reliability: the kappa statistic. *Biochemia medica*, 22(3):276–282, 2012.