

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

© 2021 Kristoffer Langstad

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

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

Printed: Reprocentralen, University of Oslo

Abstract

...

Acknowledgements

...

Contents

List of Figures	4
List of Listings	5
1 *Introduction	7
1.1 Motivation for Machine Learning	8
1.2 Structure of Thesis	8
Notation and Conventions	10
I *Theory	11
2 The Standard Model of Particle Physics	12
2.1 Particle and Force Contents	12
2.1.1 Gauge Bosons	12
2.1.2 Higgs Boson	15
2.1.3 Fermions	16
2.2 Neutrinos	17
2.2.1 Neutrino Oscillations	17
2.3 Symmetries	19
2.4 Quantum Field Theory	20
2.4.1 The Lagrangian	20
2.4.2 Qauge Theories	21
3 *Neutrinos Beyond the Standard Model	27
3.1 *Neutrino Masses	27
3.1.1 Dirac Neutrinos	28
3.1.2 Majorana Neutrinos	28
3.1.3 The Seesaw Mechanism	29

4 Proton-Proton Collisions	32
4.1 Particle Kinematics	32
4.1.1 Colliding Particles	33
4.1.2 Products of Particle Collisions	34
4.2 Proton-Proton Interactions	35
4.2.1 Hard Scattering Events	36
4.2.2 Parton Distribution Function	38
4.2.3 Hadronization	38
5 Particle Accelerators and Collider Experiments	39
5.1 CERN	39
5.2 The LHC and Accelerator Experiments	41
5.2.1 Important Parameters	42
5.3 The ATLAS Experiment and Particle Detection	44
5.3.1 Inner Detector	45
5.3.2 Calorimeters	47
5.3.3 Muon Spectrometer	47
5.3.4 Magnet System	48
5.3.5 Trigger System	48
6 The Charge Current Drell-Yan Process	50
7 Machine Learning	52
7.1 Introduction	52
7.2 Supervised Learning	54
7.2.1 Basics of Statistical Learning	54
7.2.2 Bias-Variance Decomposition	55
7.2.3 Bias-Variance Tradeoff	57
7.2.4 Regularization	59
7.2.5 Hyperparameters	60
7.3 Classification	61
7.4 Classification Models	62
7.4.1 Logistic Regression	62
7.4.2 Ridge Classifier	63
7.4.3 Support Vector Machine	63
7.4.4 Multi-Layer Perceptron	63
7.4.5 Decision Tree	65
7.4.6 Bagging	66
7.4.7 Random Forest	66
7.4.8 AdaBoost	67
7.4.9 Gradient Boosting	67

7.4.10	Extreme Gradient Boosting	68
7.4.11	Light Gradient Boosting Machine	69
7.4.12	Voting Classifier	69
7.4.13	Multiclass Classification Models	70
7.5	Evaluation Metrics	71
7.5.1	Mutual Information	71
7.5.2	Accuracy Score	72
7.5.3	Cohen Kappa Score	72
7.5.4	Error Evaluation	73
7.5.5	Classification Report	74
7.5.6	Confusion Matrix	76
7.5.7	Precision-Recall Curve	76
7.5.8	Balanced Accuracy	77
7.5.9	ROC Curve	77
II	*Implementation	78
8	Methods-Overview	79
8.1	Python packages	79
9	*Data*	82
10	*Analyzing the Data	83
11	Preparing the Data	84
11.1	Making New Variables	84
11.2	Plotting New Variables	86
11.3	Inspect Data	86
11.4	Preprocessing of the Data	88
12	*Model Classification*	91
III	Results	92
13	**	93
IV	Discussion, Conclusion and Future Prospects	94
14	Discussion	95

14.1 ?	95
15 Conclusion and Future Work	96
15.1 Future Work	96
V Appendices	97
A Bias-Variance Decomposition	98
Acronyms	99
Bibliography	101

List of Figures

2.1	The Standard Model	13
2.2	Particle interactions in the SM	13
2.3	QCD vertex Feynman diagrams	22
2.4	QED vertex Feynman diagram	23
2.5	EWT fermion vertex Feynman diagrams	25
2.6	EWT gauge boson vertex Feynman diagram	25
2.7	Higgs-bosons coupling Feynman diagrams	25
2.8	Higgs-fermions coupling diagram	26
4.1	Collider geometry	36
4.2	Hard scattering	37
5.1	The CERN complex	40
5.2	The ATLAS detector components	45
5.3	The ATLAS detector tracking system	46
6.1	The charged current Drell-Yan process	51
7.1	In-sample and out-of-sample error as function of training set size	56
7.2	Bias-variance tradeoff and model complexity	59
7.3	Multi-Layer Perceptron illustration	64
7.4	Confusion matrix	76

Listings

11.1 Making new variables.	84
11.2 Check NULL values.	88
11.3 Resample data.	89
11.4 Splitting the data.	89
11.5 Scaling.	90

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

¹Meaning 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.
- $1 \text{ GeV} = 10^9 \text{ eV} = 10^9 \times 1.602 \times 10^{-19} \text{ J}$: Approximately the rest mass energy of the proton.
- $m_e = 9.109 \times 10^{-31}$ kg = $0.511 \text{ MeV}/c^2$: Mass of an electron.
- 1 barn (b) $\equiv 10^{-28}$ m²: 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: $F = \frac{q_1q_2}{4\pi\epsilon_0 r^2}$
- 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 what we 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

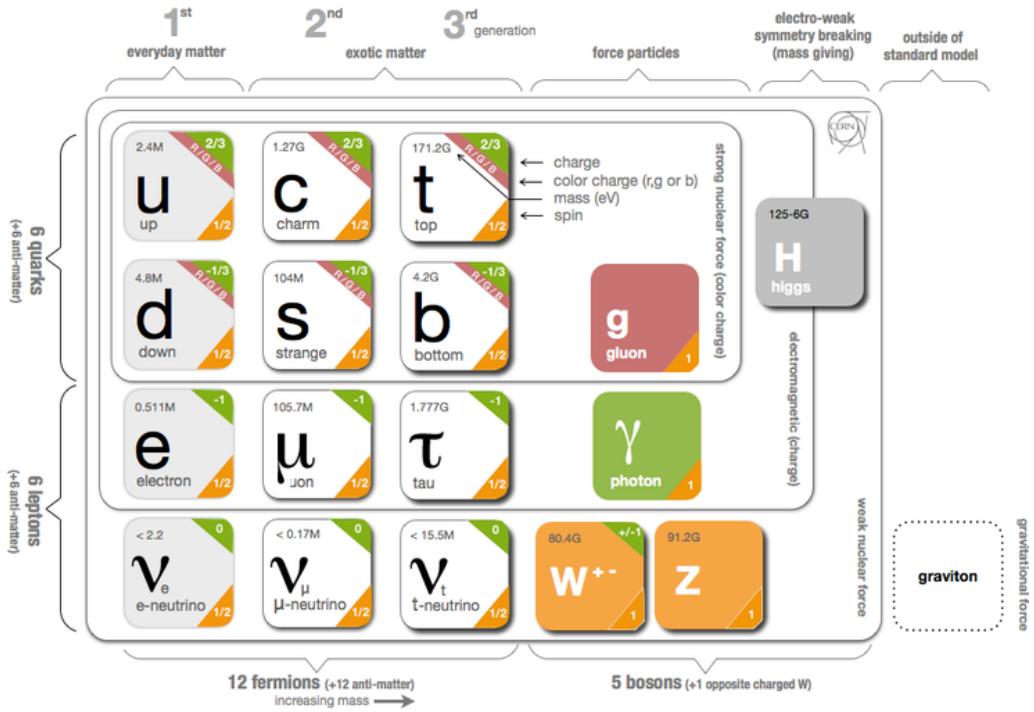


Figure 2.1: The Standard Model contents, source [7].

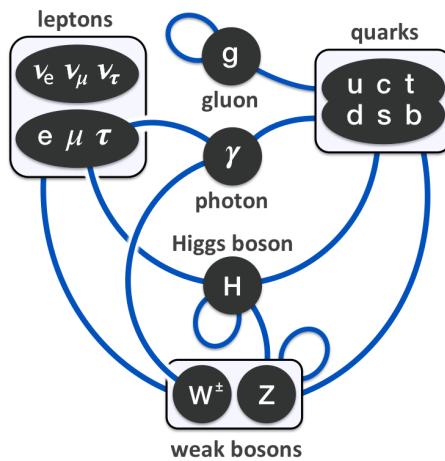


Figure 2.2: The interaction between the particles in the Standard Model.
Credit: Wikipedia.

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 mediate 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 from 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 are 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 ~ 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 a 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.

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.

The discovery of neutrino oscillations was done by two experiments. Namely

the Super-Kamiokande Observatory and the Sudbury Neutrino Observatories (**SNO**) experiments. They got the Nobel Prize in physics in 2015 for their contributions by detecting solar neutrinos from the Sun [10]. 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

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

by the Karlsruhe Tritium Neutrino (**KATRIN**)[11] 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 [12]. 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 leptons seem 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 conserved 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.

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 [13][14] for $\frac{1}{2}$ -spin particles and the Klein-Gordon equation [13][14] 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,

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

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} ,

$$L = \int \mathcal{L} d^3x, \quad (2.3)$$

and using the principle of least action [15], the new Euler-Lagrange equation becomes

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

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 field¹ and free theory² respectively. The Lagrangian for the spin-half (spinor) field, ψ , is

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

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

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

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.

2.4.2 Gauge Theories

To further explain the interactions between the elementary particles, which we now know from the new Lagrangian varies depending on the particles and associated interactions, we need a new theory. 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 space-time 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.

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

²Relativistic scalar fields, Chapter 17.2.2 in Thomson [5]

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 [16]. 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 [17]. By using this gauge theory, we can derive a new invariant Lagrangian which does not have a mass term for the gluons:

$$\mathcal{L}_{QCD} = \bar{\psi}(i\gamma^\mu \partial_\mu - m)\psi - \frac{1}{2}g_s \bar{\psi}\gamma^\mu \lambda_a \psi G_\mu^a - \frac{1}{4}G_{\mu\nu}^a G_a^{\mu\nu} \quad (2.7)$$

ψ is a fermion (quark) field, g_s is a coupling constant of the strong interaction, γ^μ are Dirac matrices, $a = 1, \dots, 8$ are the eight gluons, λ_a is one of the eight Gell-Mann matrices and $G_{\mu\nu}^a$ 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).

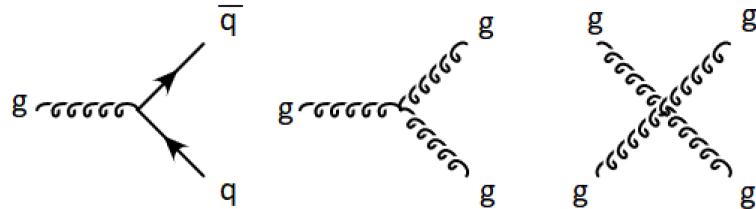


Figure 2.3: Here we see Feynman diagrams of the basic QCD vertices. From left to right we see, the coupling of gluon fields (g) interaction with quark fields (q), a triple gluon vertex and a quartic gluon vertex. Source Fig. 10.1 in Thomson [5].

Quantum electrodynamics (QED)

The gauge theory that defines the electromagnetic interaction for the electrically charged particles and photons is the quantum electrodynamics sector [18]. 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 quantum electrodynamics (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:

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

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.

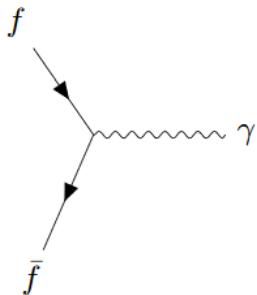


Figure 2.4: A Feynman diagram of the basic QED vertex for the interaction between fermions (f) and a massless photon (γ). Source Fig. 5.6 (and 10.10a) in Thomson [5].

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) [19] or Glashow-Weinberg-Salam (GWS) theory.

This theory (from the 1960's) earned the three contributors Glashow[20], Weinberg[21] and Salam[22] the Nobel Prize in Physics in 1979 [23][24].

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:

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

$\psi_{L,R}$ 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 complete **EWT** Lagrangian will be looked at soon.

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.

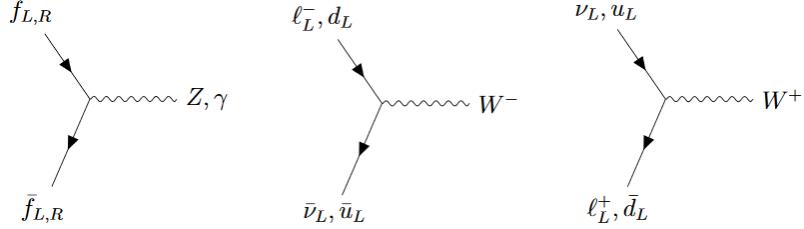


Figure 2.5: Here we see Feynman diagrams of the electroweak interaction vertices that includes fermions.

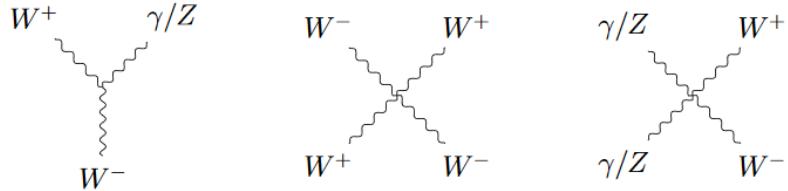


Figure 2.6: Here we see Feynman diagrams of the electroweak interaction vertices for gauge boson self-interaction.

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.

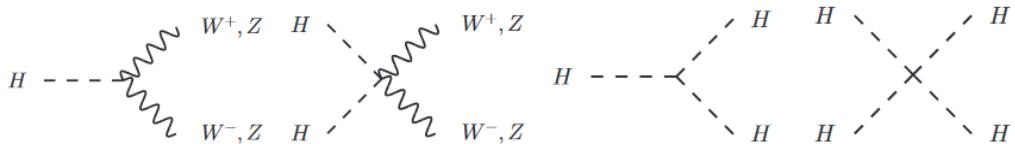


Figure 2.7: Here we see Feynman diagrams of the couplings between the Higgs boson and the massive gauge bosons and Higgs self-interaction.

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

$$m_f = \frac{g_f v}{\sqrt{2}}, \quad (2.10)$$

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

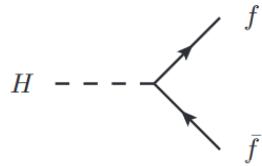


Figure 2.8: Here we see Feynman diagrams of the coupling between the Higgs boson and fermions.

Full EWT Lagrangian

The complete Lagrangian for the EWT is given by:

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

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 the about 25% is something called dark matter, that acts as matter 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 expand faster and faster with time as it seems now. A dark matter candidate is neutrinos. We will not go into the dark matter aspect, but look closer at neutrinos and the neutrino mass.

...
We will look more into this and a few other aspects and theories about the neutrino in this chapter which the classic **SM** does not cover.

3.1 *Neutrino Masses

According to the **SM**, neutrinos do not have mass. This happens because the **SM** only covers left-handed neutrinos, since the right-handed neutrinos

do not involve in any of the fundamental interactions in the model and have not been observed so far. As mentioned earlier, we know from observations and experiments of neutrino oscillations that neutrinos have to have some mass to be able to change flavor when moving over large distances. This neutrino masses is 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:

$$\psi = \begin{pmatrix} \psi_L \\ \psi_R \end{pmatrix}. \quad (3.1)$$

The left-handed neutrinos in the **SM** are then 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

$$\mathcal{L}_D = -m_\nu(\bar{\nu}_R\nu_L + \bar{\nu}_L\nu_R), \quad (3.2)$$

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

$$m_\nu = \frac{g_\nu v}{\sqrt{2}} \quad (3.3)$$

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 eV (sect. 2.2.1). It is not known why this Yukawa constant is so small, which gives reason to that there must be something else that gives the neutrinos their masses. From this 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

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

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

$$\psi_\nu = \begin{pmatrix} \overline{\nu}_R^c \\ \nu_R \end{pmatrix} \quad (3.5)$$

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

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

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.

3.1.3 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 remains. 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 particles or have more heavy 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**[25]. This involves the mix of **LH** Dirac neutrinos and **RH** Majorana neutrinos. In this theory there is added a right-handed neutrino for each of the SM **LH** neutrinos, in total three added **RH** neutrinos. When involving neutrinos as Majorana, we get that $\bar{\nu}_L \nu_R$ is equivalent to $\bar{\nu}_R^c \nu_L^c$. The Lagrangian after the spontaneous electroweak symmetry breaking with both the Dirac and Majorana mass terms becomes:

$$\mathcal{L}_{DM} = -\frac{1}{2} (m_D \bar{\nu}_L \nu_R + m_D \bar{\nu}_R^c \nu_L^c + M \bar{\nu}_R^c \nu_R) + h.c. \quad (3.7)$$

m_D is the Dirac mass and M is the Majorana mass. This equation can also be written in terms of a 2×2 mass matrix (\mathcal{M}) for the neutrinos:

$$\mathcal{L}_{DM} = -\frac{1}{2} (\bar{\nu}_L \bar{\nu}_R^c) \begin{pmatrix} 0 & m_D \\ m_D & M \end{pmatrix} \begin{pmatrix} \nu_L^c \\ \nu_R \end{pmatrix} + h.c. \quad (3.8)$$

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])

$$m_{\pm} = \lambda_{\pm} = \frac{M \pm M \sqrt{1 + 4m_D^2/M^2}}{2}. \quad (3.9)$$

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

$$|m_{\nu}| \approx \frac{m_D^2}{M} \quad \& \quad m_N \approx M. \quad (3.10)$$

The physical neutrino states are in this case

$$\nu \approx (\nu_L + \nu_R) - \frac{m_D}{M} (\nu_R + \nu_R^c) \quad \& \quad N \approx (\nu_R + \nu_R^c) + \frac{m_D}{M} (\nu_L + \nu_L^c). \quad (3.11)$$

By looking at equation 3.10, we see that the lightness of the **SM** neutrinos are explained when there exists much heavier right-handed neutrinos.

Inverse seesaw mechanism

The model we are studying involves a slightly different seesaw 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 allow large Yukawa couplings. Besides the addition of right-handed neutrinos, this model

also adds three singlet fermions. This then yields three light **LH** neutrinos and heavy pseudo-Dirac neutrinos[26][27] with small lepton number violations in the singlet mass terms. This comes from 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 (sect ??) which involves this inverse seesaw mechanism, and is based on the

$$SU(3)_C \times SU(2)_L \times SU(2)_R \times U(1)_{B-L} \quad (3.12)$$

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.

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 hadrons¹ 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 partons 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

$$E^2 = p^2 + m^2. \quad (4.1)$$

Since the protons will reach very high velocities when they collide, we need to include special relativity into the equations²:

$$E = \gamma m \quad \text{and} \quad \mathbf{p} = \beta \gamma m \quad (4.2)$$

These equations depend on the Lorentz factor

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

¹The partons, i.e. quarks and gluons.

²In 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, s [5], 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 process³ (explained and derived in Thomson [5]) for a quark and an antiquark, we get the fractions given as

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

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 :

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

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

$$p_T = \sqrt{p_x^2 + p_y^2}. \quad (4.7)$$

The corresponding transverse energy is given as

$$E_T = \sqrt{p_T^2 + m^2}. \quad (4.8)$$

³This is not restricted to Drell-Yan processes, but yields for any 2-to-1 process.

The total momentum can then be derived as

$$p = \sqrt{p_T^2 + p_z^2}. \quad (4.9)$$

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:

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

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

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

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

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

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)

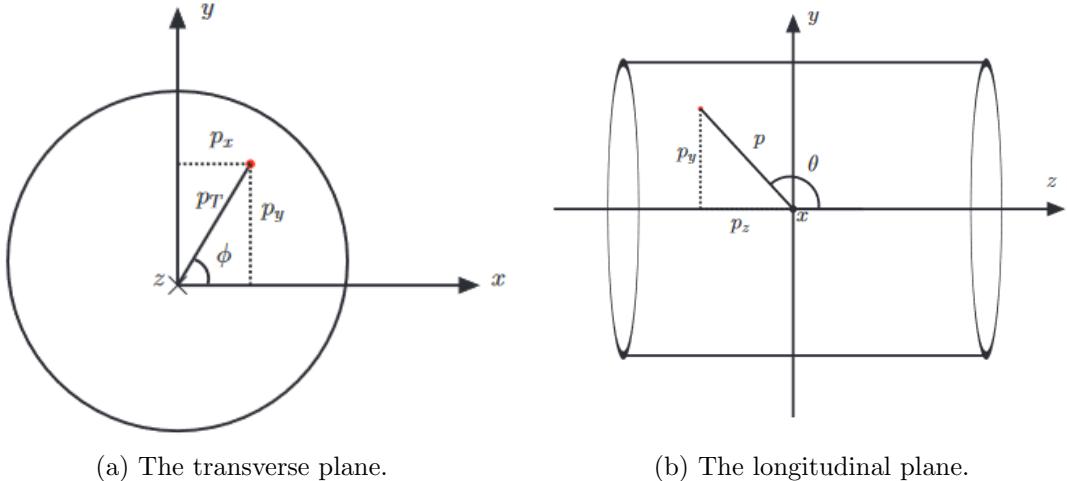


Figure 4.1: Illustrations of the (a) transverse plane and the (b) longitudinal plane. The collision point is at the origin. Figures are both from ref. [33].

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

$$\sigma_{\text{total}} = \sigma_{el} + \sigma_{di} + \sigma_{nd} \quad (4.13)$$

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-diffractive 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

$$A + B \rightarrow c + X, \quad (4.14)$$

where the collision between the partons are expressed as

$$a + b \rightarrow c. \quad (4.15)$$

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.

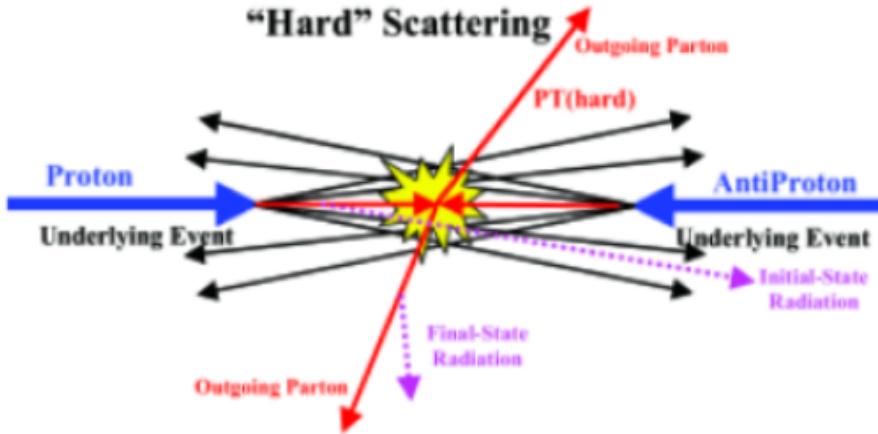


Figure 4.2: Illustration of a hard scattering proton-proton collision. Figure is taken from ref. [34].

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

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

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 particles⁵. 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-antiquark 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.

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

⁵Only 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 [35]. 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. **CERN**'s main involvement 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-

¹The 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 [36].

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 are sent to experiments that collides particles 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.

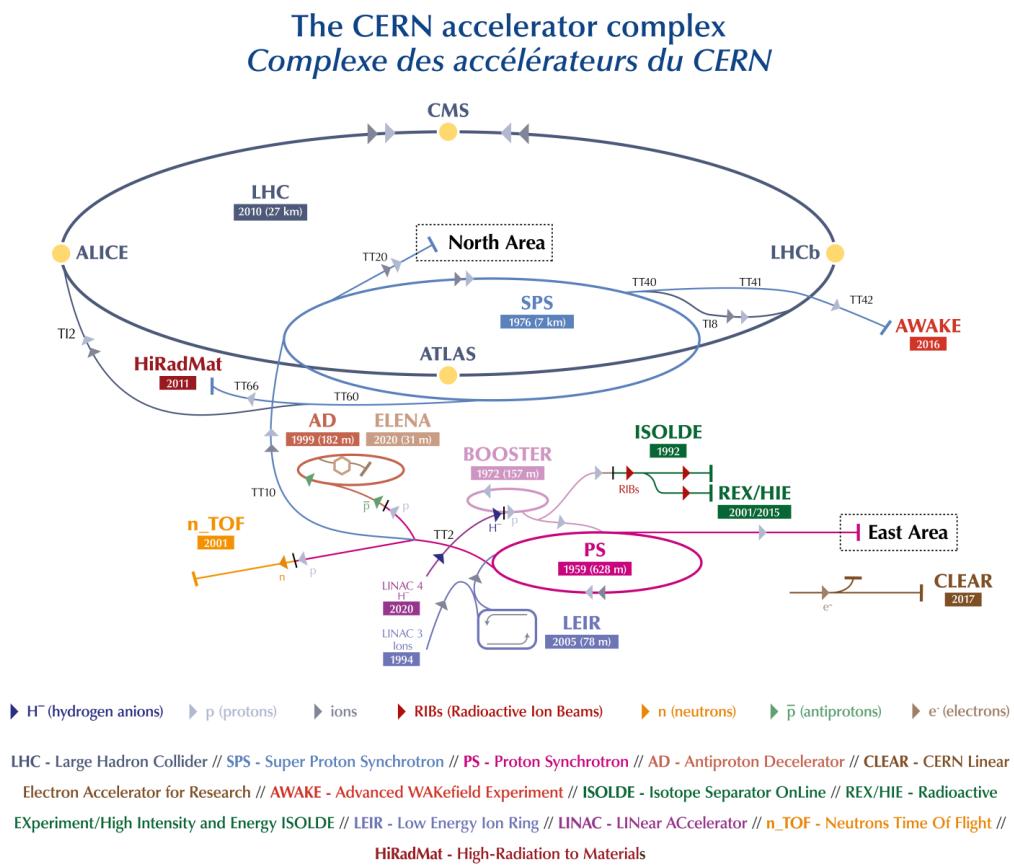


Figure 5.1: The CERN accelerator complex as of 2019. Credit: CERN[37].

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 [38] [39]) and Japan (ILC [40][41]).

5.2 The LHC and Accelerator Experiments

Today's largest and most powerful particle accelerator is the Large Hadron Collider (**LHC**) [42], 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**) which are 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 TeV². There are in total seven detectors along the ring, each designed for different experiments.

The first time, run 1, it was used for proton-proton (hadron) collisions in 2010, it reached a record high energy of 3.5 TeV per beam. After upgrades, 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. 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 in several systems which increase

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

the energies before injected into the main LHC ring [43]. 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 [44]. 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 ATLAS and the detector equipment later (sect.5.3).

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

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

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 $\mathcal{L} = 10^{34} \text{ cm}^{-2}\text{s}^{-1}$. 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 as³

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

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 [45]

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

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

$$R = \sigma \mathcal{L}. \quad (5.4)$$

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

$$N = \sigma \int \mathcal{L} dt. \quad (5.5)$$

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

³With 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

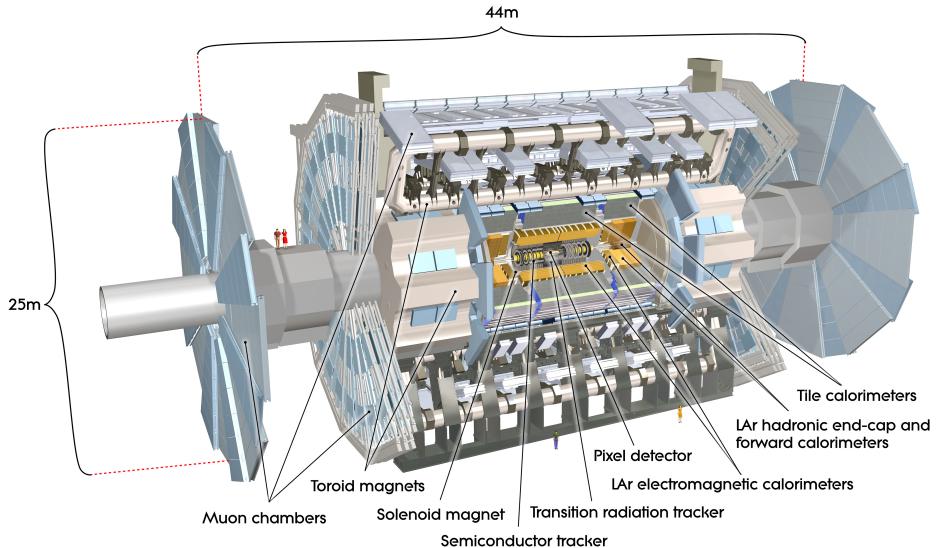


Figure 5.2: The ATLAS detector. Credit: ref. [47].

in observing specific properties of the various particles. The **ATLAS** detector consists of five main systems; the inner detector (**ID**), different 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.

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 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 longer 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

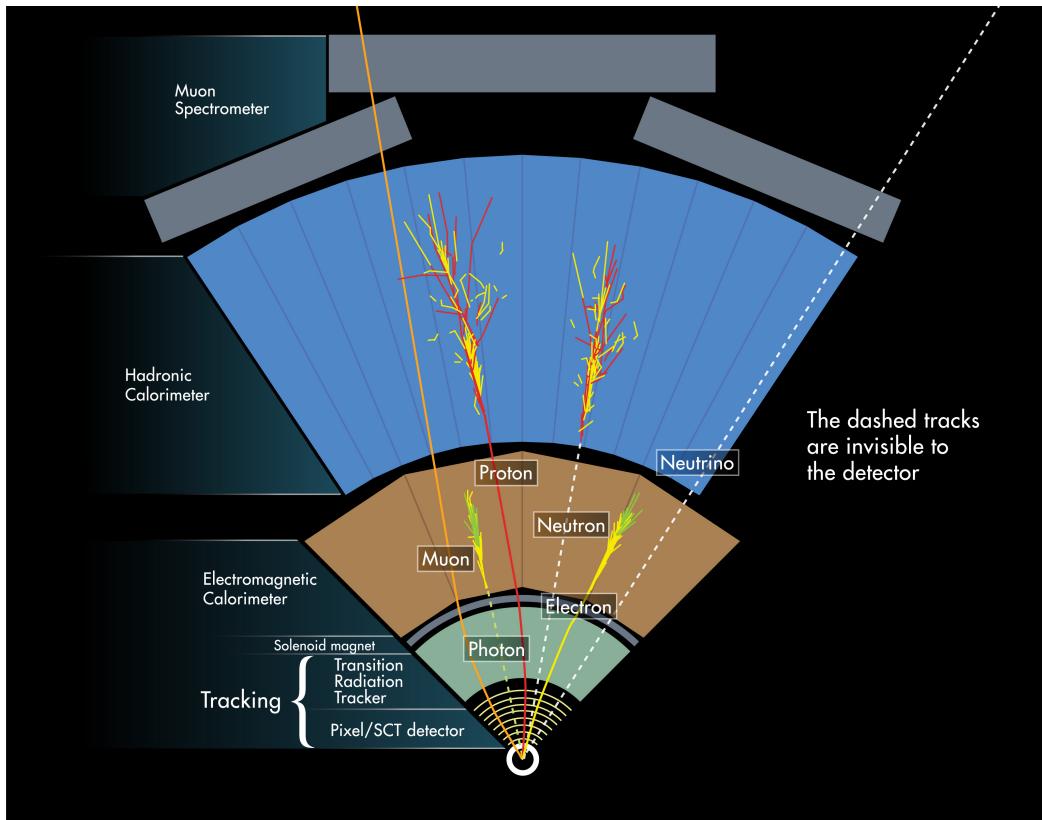


Figure 5.3: An illustration of the main tracking systems in the ATLAS detector, including how some particles behave in the various systems. Credit: ref. [48].

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 an 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 showers¹⁴. 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**, since the distance between nuclear interactions are relatively large. The **HCal** consists of three parts where two of them share some of the same parts as the **ECal**. 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 [49].

5.3.3 Muon Spectrometer

Outside the calorimeters, we find the muon spectrometer. Here high-energy muons are detected. 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

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

be zero. This detector is very large, 11 m radius [50], and consist of three parts as well as a barrel and two end-caps; a magnetic field with three toroidal magnets, a set of 1200 chambers which measure 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 0.5 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 \sim 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 [51]; 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

The Charge Current Drell-Yan Process

The model in this thesis is based on the works of Das et al. [3], Pascoli et al. [4] and CMS Collaboration et al. [52] with a $SU(2)_L \times SU(2)_R \times U(1)_{B-L}$ gauge theory combined with the inverse seesaw mechanism. Here, two protons are accelerated and collided to produce a heavy W_R^\pm boson 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_R boson may decay into a charged lepton l and a heavy pseudo-Dirac neutrino N . The pseudo-Dirac neutrino then decays into another lepton and another W_R boson with opposite sign, which then decays into another lepton and MET/a (light) neutrino:

$$q\bar{q} \rightarrow W_R \rightarrow l_1 N_l \rightarrow l_1 l_2 W_R^* \rightarrow l_1 l_2 l_3 \nu$$

The final state is then three charged leptons (trilepton) plus missing transverse energy which goes undetected as a neutrino in a detector (like ATLAS). This decay process can be seen in Figure 6.1, and is produced through the charged current Drell-Yan (CCDY) process [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 what kind of quarks that collide¹, and which final state particles are produced. With these particle properties we can calculate the angles and angular distances between each produced

¹Remember that we effectively collide the quarks in the protons.

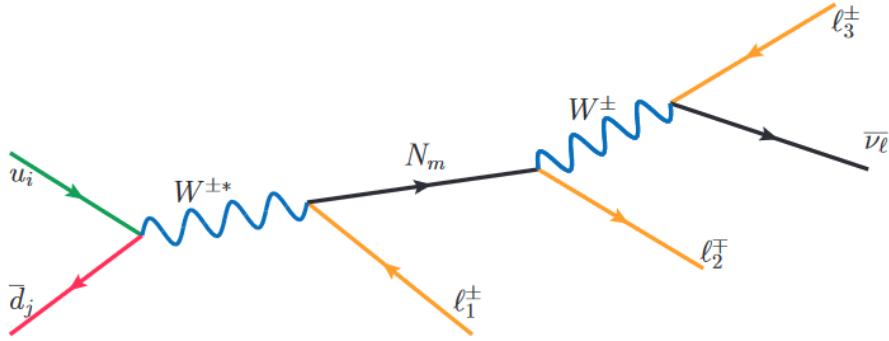


Figure 6.1: The Born diagram for the charged current Drell-Yan process of the proton-proton collision producing a heavy pseudo-Dirac neutrino N in the inverse seesaw mechanism model, leading to a trilepton plus missing transverse energy (a light neutrino) final state. Figure is taken from ref. [4].

particle and the neutrino (MET). 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 6.1 computationally.

The end goal is then to make a program that can automatically identify each lepton, and where it came from, by utilizing these particle properties and using machine learning. We will look more into machine learning in the next chapters. The computational setup and input parameters for the CCDY process are look more into in a later chapter as well.

Chapter 7

Machine Learning

7.1 Introduction

Machine learning (**ML**) and data science are more and more used to study algorithms and statistical models. The meaning of machine learning is to use computational algorithms to automatically determine an outcome from specific patterns in data by using various algorithms that are dependent on some input and hyperparameters. With these inputs and hyperparameters, the algorithms try to learn a general pattern in often large amounts of data to find a solution. **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 that are then used to check the performance of the algorithms on the data. When we have a good enough trained model, we can save it and use it later on similar data to do the same tasks on 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 explains the data the best. 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 the how the model makes new predictions. It is not always that the algorithms can give a very good prediction to all problems, or not at all in some cases depending on the datasets.

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 $f : \mathbf{x} \rightarrow \mathbf{y}$ 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 $\mathcal{C}(\mathbf{y}, f(\mathbf{X}; \theta))$ that judges how well the model performs on the observations. When fitting the model, we want the $\hat{\theta}$ that explains the data the best. When considering a linear regression case with the sum of least squares as the cost function,

$$\mathcal{C}(\mathbf{y}, f(\mathbf{X}; \theta)) = \sum_i^N (y_i - f(\mathbf{x}_i; \theta))^2, \quad (7.1)$$

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

$$\hat{\theta} = \operatorname{argmin}_{\theta} \{\mathcal{C}(\mathbf{y}, f(\mathbf{X}; \theta))\} \quad (7.2)$$

The ML approaches are usually divided into supervised, unsupervised and reinforcement learning¹. **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

¹There 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. [53] for more on deep learning and other possible machine learning tasks.

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 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 we take a closer study at classification and multiclass classification, which is what is used in this thesis. The theory is mostly based on the works of Hastie et al. [54] and Mehta et al. [55].

7.2 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.

7.2.1 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:

$$\mathbb{E}[h] = \int_{X \times Y} \mathcal{C}(h(x), y) \rho(x, y) dx dy. \quad (7.3)$$

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*:

$$\mathbb{E}_E[h] = \frac{1}{n} \sum_i^n \mathcal{C}(h(x_i), y_i). \quad (7.4)$$

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

$$G = \mathbb{E}[h] - \mathbb{E}_E[h]. \quad (7.5)$$

In the limit of the generalization error goes towards zero,

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

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 7.1 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 not small 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.

7.2.2 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

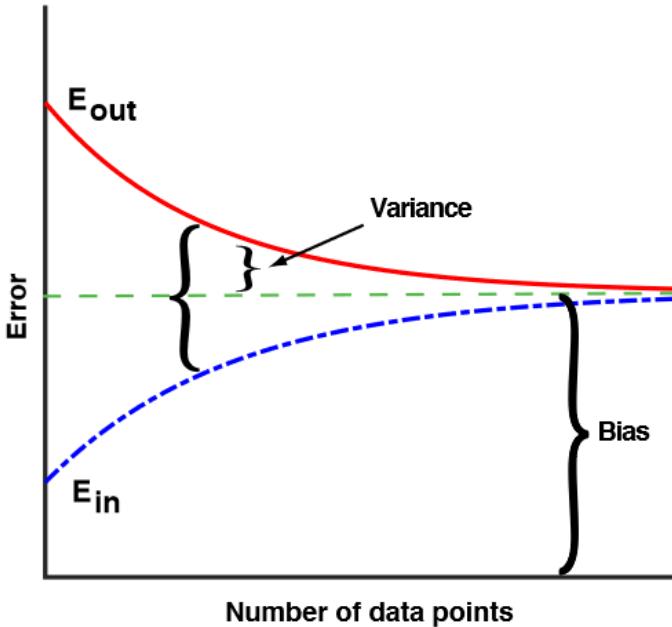


Figure 7.1: Illustration of the in-sample error, E_{in} , out-of-sample error, E_{out} , variance, bias and difference of errors as function of the training set size. It is assumed that the number of data points is not small, and that we cannot exactly fit the true function $f(x)$. The training error increases while the test error decreases as the training set size increases. Figure is taken from ref. Mehta et al. [55].

of independent and dependent variables. We then assume that the true data is created from a noise model

$$y = f(x) + \epsilon, \quad (7.6)$$

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 errors²,

$$\mathcal{C}(\mathbf{y}, f(\mathbf{X}; \theta)) = \sum_i (y_i - f(\mathbf{x}_i; \theta))^2. \quad (7.7)$$

Our best estimates for the model parameters,

$$\hat{\theta}_{\mathcal{D}} = \operatorname{argmin}_{\theta} \{\mathcal{C}(\mathbf{y}, f(\mathbf{X}; \theta))\}, \quad (7.8)$$

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

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

$$\begin{aligned}\mathbb{E}_{\mathcal{D}, \epsilon}[\mathcal{C}(\mathbf{y}, f(\mathbf{X}; \hat{\theta}_{\mathcal{D}}))] &= \sum_i (f(\mathbf{x}_i) - \mathbb{E}_{\mathcal{D}}[f(\mathbf{x}_i; \hat{\theta}_{\mathcal{D}})])^2 \\ &\quad + \sum_i \mathbb{E}_{\mathcal{D}}[\{f(\mathbf{x}_i; \hat{\theta}_{\mathcal{D}}) - \mathbb{E}[f(\mathbf{x}_i; \hat{\theta}_{\mathcal{D}})]\}^2] \\ &\quad + \sum_i \sigma_{\epsilon}^2\end{aligned}\tag{7.9}$$

The first term in equation 7.9 is the bias

$$\text{Bias}^2 = \sum_i (f(\mathbf{x}_i) - \mathbb{E}_{\mathcal{D}}[f(\mathbf{x}_i; \hat{\theta}_{\mathcal{D}})])^2,\tag{7.10}$$

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

$$\text{Var} = \sum_i \mathbb{E}_{\mathcal{D}}[\{f(\mathbf{x}_i; \hat{\theta}_{\mathcal{D}}) - \mathbb{E}[f(\mathbf{x}_i; \hat{\theta}_{\mathcal{D}})]\}^2],\tag{7.11}$$

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

$$E_{\text{out}} = \text{Bias}^2 + \text{Var} + \text{Noise}.\tag{7.12}$$

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.

7.2.3 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 complex³ 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 7.2 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 7.2a 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 7.2a. This optimal line is also where we have a minimum in E_{out} . For the prediction error for test and training samples in Figure 7.2b 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 to big and not to

³When 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.

similar to each other. Often we want to use a more biased model with small variance to minimize E_{out} and maximize the predictions.

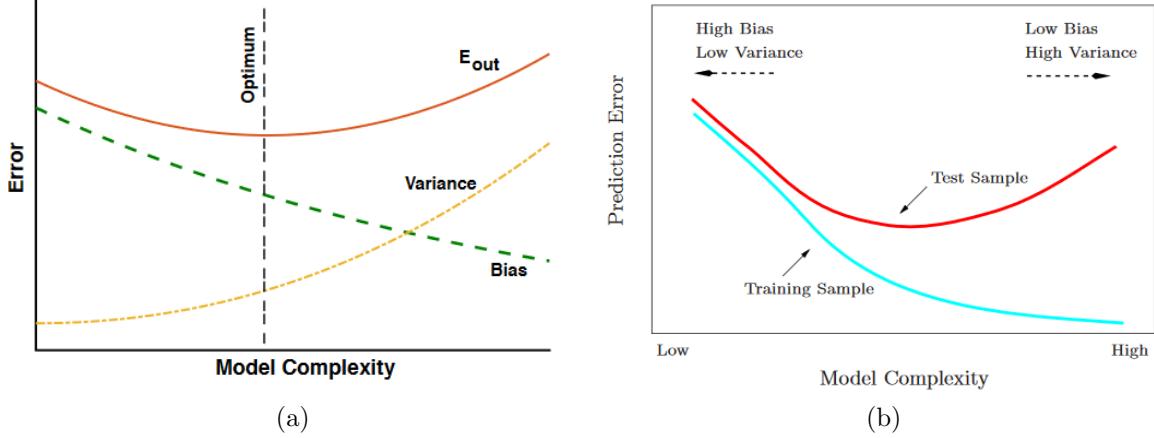


Figure 7.2: Illustrations of the bias-variance tradeoff as function of model complexity. From these two illustrations we see that we want to find the optimal compromise between variance and bias that gives the best model, which does not underfit nor overfit the data. Figures are taken from ref. Hastie et al. [54] and Mehta et al. [55].

7.2.4 Regularization

With increasing data power and an increase in the amount of data collected today, the data sets we can gather can be quite complex. This also gives need for 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 that the models fit 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 that 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:

$$L_{1,\text{norm}} = \sum_i |w_i| \quad (7.13)$$

$$L_{2,\text{norm}} = \sum_i \|w_i\|^2 \quad (7.14)$$

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 that uses *trees*⁴, which affects the splits in the making of trees. *Sampling* and *early stopping* are also other ways to control overfitting by making boosted trees less correlated and stop training when a chosen training metric of a model no longer improves, respectively. All these ways to control overfitting are controlled by various input parameters numerically.

7.2.5 Hyperparameters

As we have already mentioned, hyperparameters are something that have to be manually chosen before fitting a model. The reason for hyperparameters is to tune and optimize the models to do a better fit on the data. They are 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 as well. 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 will be as 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.

⁴We will come back to what this is later.

During the hyperparameters 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 the model and optimize the hyperparameters. The first split of the original data set is into training and test sets. Then the training set can then 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.

7.3 Classification

Classification is one of the most used and most successful tasks in machine learning. Classification uses algorithms to decide which category the input belongs to. The function that produces an output value can instead 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 as they are called, 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 by itself. 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 only 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 two protons together many times like at the [LHC](#). To do the multiclass classification 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 with the best model.

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

$$\mathbf{y} = \mathbf{X}\mathbf{w}. \quad (7.15)$$

The classes are then determined by the sign of the predictions by using sign functions or use 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.

7.4.1 Logistic Regression

A simple "soft" model in statistical analysis for classifying discrete outcomes is logistic regression (**LR**)^[55]. It uses linear regression to fit data and a logistic function⁵, usually the Sigmoid function

$$\sigma(s) = \frac{1}{1 + e^{-s}}, \quad (7.16)$$

to predict the outcomes into categories by 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 the cross-entropy with added L_1 (eq.7.13) and L_2 (eq.7.14) 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. **LR** can also be combined with cross-validation using various optimization solvers supporting the regularization parameters.

⁵It can also be called an activation function.

7.4.2 Ridge Classifier

The Ridge classifier uses ordinary Ridge regression[56] after converting the outcome values into a range of $\{-1,1\}$, treating it like a regression case. The Ridge classifier uses a Least Squares loss for fitting the model by introducing a penalty to the Ordinary Least Squares (**OLS**) normally used in Linear Regression. The penalty uses a complexity coefficient, α , that acts as a shrinkage (regularization) parameter that acts on the linearity of the coefficients.

In the case of a multiclass case, the classifier uses multi-output regression as an extension. It will use a one-versus-all approach, where n , corresponding to how many classes we have, classifiers are trained. Computational advantages in the Ridge classifier can lead to a speed advantage over **LR** for many classes. It can also be combined with cross-validation like **LR**.

7.4.3 Support Vector Machine

Support Vector Machines (**SVMs**)[57] try to construct an optimal hyperplane for the K features in the data, making a K -dimensional space, separating the data points in the target classes. When the maximum distance separating the class data have been reached, we have the optimal hyperplane. The hyperplane is affected by *support vectors*, which are the data points that are closest to the hyperplane. The **SVM** is similar to **LR** in that it takes the output from a linear function, but assigns the values in the range $\{-1,1\}$ like the Ridge classifier. Then a cost function with a regularization parameter is used to maximize the margin between the hyperplane and the data points. The regularization parameter is used to balance the loss and the maximization of the margin. A "soft" margin is introduced to account for misclassification by adding a loss to the regularization parameters. A kernel function is then used to increase the dimensional space and (hopefully) improving the hyperplane.

SVM also supports multiclass classification by changing to several binary classification problems. One method is by implementing a one-versus-one (7.4.13) scheme creating one classifier and trains on two classes at a time, creating many classifiers in total. It can be altered to instead use a one-vs-rest (7.4.13) scheme with a separate classifier for each class.

7.4.4 Multi-Layer Perceptron

A Multi-Layer Perceptron (**MLP**)[58] is an artificial feed-forward neural network (**FFNN**) model, and is similar to **LR** in that it has an *input* and an

output layer, but differs in that between these layers (also called nodes), the **MLP** can have several non-linear layers called *hidden layers*. As 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 7.3a 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 also have some bias terms connected as well. In Figure 7.3b we see a fully connected neural network since all nodes are connected to all nodes in the next layer.

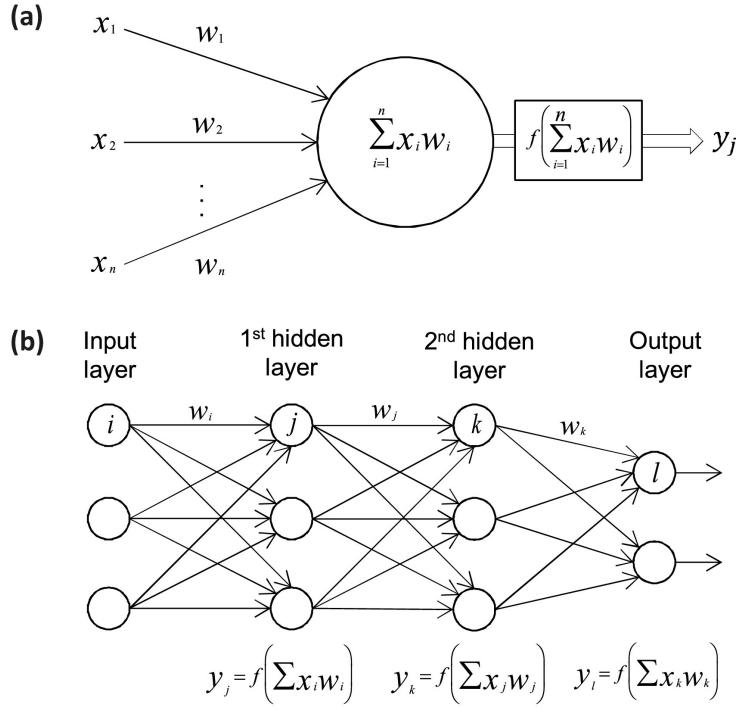


Figure 7.3: (a) Each node in a neural network has some input acted on by some associated weights. Then the weighted inputs are summed together inside the node and passed to a non-linear activation function that transforms it to an output. (b) This is a **FFNN/MLP** with 3 inputs, 2 hidden layers and 2 classes. All the nodes in one layer are 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. [59].

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

There are a few typical choices of activation functions, which include the hyperbolic tangent function, the sigmoid and the rectified linear unit function (**ReLU**). The choice of cost function is also something that has to be considered. The **MLP** library by *Scikit-Learn*, only supports the cross-entropy loss function as the cost function. The logistic loss function is also a good choice of loss function as well.

Neural networks have a large amount of parameters that often leads the network to overfit the data. That is why we also here add a L_2 regularization penalty to the weights, like with Ridge. 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 this case, the weights and biases are the parameters to be adjusted while this is something else in other models. There are a few gradient descent methods we could use like stochastic gradient descent with minibatches to avoid the possibility of interpreting a local minimum as a global minimum. A more modern method is the *adam* solver proposed by Kingma and Ba [60]. It is a stochastic gradient-based optimizer which combines batched data, momentum and adaptive learning rates and adds a few more hyperparameters to be tuned, β_1 , β_2 and ϵ .

For multiclass classification, the softmax function is used as the last hidden layer activation function. It normalizes the output of the network into a probability distribution over the predicted output classes.

7.4.5 Decision Tree

Decision Trees (**DTs**)[61] tries to learn simple decision rules from the data features to construct a tree-like model to predict the target values. The model simply tries to 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 some 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 most homogeneous branch when splitting. The stopping point for splitting is something we can decide as an input parameter to the model by choosing a maximum depth of the tree from the root to the leaves, or set the maximum number of leaves to have at the end. Other parameters for controlling the size and splitting of the tree are to be considered as well since the **DT** is prone to overfitting with many features. This can be fixed by pruning the tree to remove nodes with low importance features, use dimensionality reduction with e.g. principal component analysis (**PCA**)^[62] or decrease some of the controlling parameters.

There are a few different **DT** algorithms to generate the optimal trees. The algorithm that is implemented by 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 as well.

7.4.6 Bagging

Bagging^[63] is an ensemble method that uses several base estimators independently, with a chosen classifier, and takes the average of their predictions. E.g. the base estimator could be a **DT** such that the bagging classifier would make several decision trees and combine them to improve the accuracy score and reduce the variance of the estimator. This is normally better than using only one of the base estimator alone. The base classifiers are fit on random subsets of the training data together with bootstrap aggregation.

There are more bagging methods that depend on the strategy of how they draw the random subsets of the training set. The Bagging strategy is when the samples are drawn with replacement. Other strategies are Pasting, Random Subsets and Random Patches. With bagging we can control the size of the subsets and if we want to draw with replacement or not of samples and features.

7.4.7 Random Forest

Another classification ensemble method is the Random Forest (**RnF**)^[64] 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. Can also choose to bootstrap samples or not.

Like with the **DT** algorithm, we can control the size and splitting of the tree. These two 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.

7.4.8 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 that keeps the weight for each instance and the base estimators are built sequentially. It then builds a combined estimator that reduces the bias. The result is to get a powerful ensemble from several weaker models combined.

One such boosting method is the AdaBoost classifier[63][65]. 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 that 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 at predicting.

The AdaBoost algorithm by 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.

7.4.9 Gradient Boosting

Gradient Boosting Decision Tree (**GBDT**)[66] 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

lets the cost function to be more specified for optimization for fitting the classification problem. The K number of regression trees⁶ 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 to overfitting. When we have 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 have a larger number of classes, a histogram-based gradient estimator can be more useful. Scikit-learn has an experimental implementation of [GBDT](#)s called **HistGradientBoostingClassifier** which is inspired by Ke et al. [67] 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 that 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.

7.4.10 Extreme Gradient Boosting

Another highly efficient, flexible and portable tree boosting method is the Extreme Gradient Boosting ([XGBoost](#))[68]. It is a scalable end-to-end tree boosting system that is an optimized distributed gradient boosting algorithm. It 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. 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 and the potential loss for the possible splits to make a new branch as it looks at distributions of the features for all data points in a leaf to build trees. This decreases the space of possible feature splits search. The algorithm chooses which features and split-points that maximizes the gain. The splits are binary such that it splits according to if a value is bigger or lower than a set threshold

⁶For a binary classification case, only a single tree is fit.

cut by the algorithm. The gain is different depending on the type of loss function that is used. With a small data set (number of rows $\leq 4'194'303$, set by default), 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 this algorithm is more complex than other techniques, 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* that are dependent on the boosting method and *task parameters* that specify learning task parameters and learning objective. We are using a tree booster that has many of the same tree boosting parameters like the **DT** and **RnF** algorithms like regularization terms, hyperparameters for tree controlling, pruning and others. The task parameters consist of the type and size of classes we have, like multiclass classification, and types of evaluation metrics to use.

7.4.11 Light Gradient Boosting Machine

Light Gradient Boosting Machine (**LGBM**)[\[67\]](#) is a distributed gradient boosting framework for machine learning. It is similar to the **XGBoost** algorithm, but it is 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 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. They have very similar input parameters for the algorithms.

7.4.12 Voting Classifier

Most machine learning algorithms have their own weaknesses when modeling and fitting data sets. A method for balancing out these weaknesses is the combination of several different types of (previously unfitted) classifiers that use some voting technique to exploit different traits from the classifiers to even out the weaknesses. The goal is to increase the accuracy score of the models as one better model. This idea is called a Voting classifier[\[69\]](#)[\[70\]](#), and uses two types of voting techniques to predict the classes; a majority (hard) vote and an average predicted probability (soft) vote.

The hard voting uses the predicted class labels for majority rule voting. The classified sample by the model is then the majority class label of the classifiers combined. The soft voting uses the *argmax* of the sums of the

predicted probabilities to predict the class labels. This is most preferred when using an ensemble of equally well performing classifiers. Both have the option to add a sequence of weights to weight the different classifiers. The predicted classes and class probabilities are then multiplied with the corresponding weight and averaged. The class with the highest average is then chosen as the class label. The Voting classifier also supports individual classifier hyperparameter tuning with some hyperparameter search algorithm. It also supports multiclass classification as long as the provided classifiers support multiclass classification.

7.4.13 Multiclass Classification Models

To do multiclass classification, there are several existing techniques. We will look more into two of those techniques⁷; 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 beyond binary though. The classification models we looked at in the last section ?? can either do this automatically, or have input parameters and constraints in the models that have to tell the models to do multiclass classification in some way.

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 that does the fitting, and the classifier fits the single class against the rest of the classes. This means we only need n classifiers 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 binary classifier along with samples and targets as inputs and outputs a list of the classifiers for each class. When doing the predictions, it uses all the classifiers on unseen data where it picks the class with the highest confidence score as prediction.

⁷There is also a third technique, hierarchical classification, that we do not look at.

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 must learn to distinguish them. This happens between all the classes. It then uses a voting scheme to select the class with the most votes when predicting. For n number of classes in the multiclass problem, the **OvO** trains $n(n - 1)/2$ binary classifiers. All the classifiers that are trained here 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 of course slower than the **OvR** since it has a $\mathcal{O}(n^2)$ complexity. Both these two methods do suffer from that there may be regions where the input space can get the same number of votes.

7.5 Evaluation Metrics

To evaluate the performance of the classification models properly and decide which model fits the data the best, we need to have some evaluation metrics that we can use. In this section we will take a look at the evaluation metrics we will use for the classification⁸.

7.5.1 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 by using the probability $P(j)$ of a value j occurring when j is a value that a feature group x_i can take;

$$H(x_i) = - \sum_{j \in x_i} P(j) \log_2 P(j) \quad (7.17)$$

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

$$H(x_i|\mathbf{y}) = - \sum_{y \in \mathbf{y}} P(y) \sum_{j \in x_i} P(j|y) \log_2 P(j|y) \quad (7.18)$$

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

$$I(x_i : \mathbf{y}) = H(x_i) - H(x_i|\mathbf{y}) \quad (7.19)$$

⁸See Scikit-Learn[71] for more details on metrics.

With the information gain we get a measure of the correlation between a feature and the target, which detects non-linearities or the amount of information that one feature provides about another.

7.5.2 Accuracy Score

To measure the performance of the models, we use the accuracy score for classification. This is a measure of 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.

$$\text{Accuracy} = \frac{\sum_{i=1}^n I(\tilde{y}_i = y_i)}{n} \quad (7.20)$$

\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 an indicator function

$$I = \begin{cases} 1, & \text{if } \tilde{y}_i = y_i \\ 0, & \text{if } \tilde{y}_i \neq y_i \end{cases} \quad (7.21)$$

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, training validation and test sets. If there is a big difference between the accuracy score for either validation or test and training, we might under- or overfit the data. When the training score is much better, we most likely overfit the data, and might have to something to change that by regularization or tuning hyperparameters.

Another way to balance out the accuracy scores is to use *cross-validation*. It is a very useful technique against overfitting or to tune hyperparameters. There are several cross-validation techniques, but the main idea of cross-validation is to divide samples into subsets. It will then do the analysis on one subset and compute the accuracy. Then it will do another analysis with another subset to get a different accuracy. After many rounds of 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* that does this, and we can choose how many folds of subsets we want the data to be split into.

7.5.3 Cohen Kappa Score

Another scoring statistic is the Cohen Kappa Score (CKS)[72]. The CKS measures the interrater reliability, which takes into account what the accuracy could have been through random predictions. This comes from that

perfect agreement is not something that usually happens. It accounts for uncertainties in the predictions as from a random classifier vs a more accurate and tuned classifier on the classes. An example is that different human data collectors can interpret data differently from more vague variables, which then affects the data collection and outcomes. As for the accuracy score, 1 is the optimal score, but it ranges from -1 to 1 like many other correlation statistics. It is calculated as

$$\kappa = \frac{\text{Observed Accuracy} - \text{Expected Accuracy}}{1 - \text{Expected Accuracy}} = \frac{p_a - p_e}{1 - p_e} \quad (7.22)$$

7.5.4 Error Evaluation

To evaluate the error as best as we can, we will use a few 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 misclassification cases the model has on a data set:

$$\text{error} = 1 - \text{accuracy} \quad (7.23)$$

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 do this. 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

$$L_{\log} = -\log P(y|p) = -(y \log(p) + (1 - y) \log(1 - p)). \quad (7.24)$$

For a multiclass case, the log loss is taken over a whole set of K labels with a 1-of-K binary indicator matrix \mathbf{Y} and a matrix \mathbf{Pr} of probability estimates as

$$L_{\log}(\mathbf{Y}, \mathbf{Pr}) = -\log P(\mathbf{Y}, \mathbf{Pr}) = -\frac{1}{n} \sum_{i=1}^n \sum_{k=1}^K y_{i,k} \log p_{i,k}. \quad (7.25)$$

Variance

Previously in section 7.2.2 we defined the variance and the bias of a model. These two are used to check for under- and overfitting of the data set by our model(s). 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

$$\text{Var}(\tilde{\mathbf{y}}) = \frac{1}{n} \sum_{i=1}^n (\tilde{y}_i - \frac{1}{n} \sum_{j=1}^n \tilde{y}_j)^2. \quad (7.26)$$

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 7.12, we used the bias squared. This can be calculated as

$$\text{Bias}^2(\mathbf{y}, \tilde{\mathbf{y}}) = \frac{1}{n} \sum_{i=1}^n (y_i - \frac{1}{n} \sum_{j=1}^n \tilde{y}_j)^2. \quad (7.27)$$

7.5.5 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 by using the true targets and predictions by the model.

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

Positive (P) - The observation is positive.

Negative (N) - The observation is negative.

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

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

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

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

With the last four outcomes above, 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).

7.5.6 Confusion Matrix

With the four outcomes in the classification report, we can also compute a *confusion matrix*. For a binary case, the confusion matrix looks like Figure 7.4. Here we see how the observations and predictions let us know what the classes are predicted as versus what they truly are. It gives us a more closer and thorough look at the accuracy of a classification model. The accuracy just shows the overall accuracy, but 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 1. In the optimal case with all predictions correctly guessed, we should have 1's along the diagonal and 0 elsewhere.

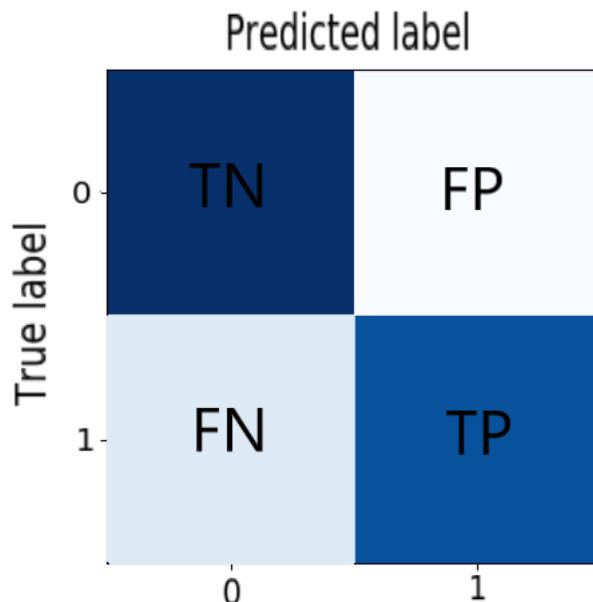


Figure 7.4: The confusion matrix is used to evaluate the accuracy of a classification model by using the four true and false observation and prediction outcomes (TP, TN, FP, FN).

7.5.7 Precision-Recall Curve

Scikit-Learn provides a useful function for plotting precision versus recall. This lets us see how the precision and recall behaves for different thresholds. A high 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. This means that the model can accurately predict the positive results, meaning it is a good model. For a multiclass-case, the precision and recall are computed for each class as binary cases. A high area for each class is the optimal case here as well.

7.5.8 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:

$$\text{Balanced-accuracy} = \frac{1}{2} \left(\frac{TP}{TP + FN} + \frac{TN}{TN + FP} \right) \quad (7.28)$$

Like the accuracy, the balanced accuracy ranges from 0 to 1.

7.5.9 ROC Curve

The Receiver Operating Characteristic (**ROC**) curve also utilize the area under the curve to summarize the overall performance of classification models. It plots the sensitivity versus the specificity for each model, and also shows how a totally random model would perform with area under the curve of 0.5. The optimal model here would show a quick incline in the curve at the beginning before flattening out with area under the curve equal to 1. So a high sensitivity as quickly as possible as function of increasing specificity. Normally a good classifier have an area under the curve value higher than 0.8. It plots for each class as well as the micro-average and macro-average of the classifier.

Part II

***Implementation**

Chapter 8

Methods-Overview

In this part, we will look at the framework for the classification we are doing with different models in this thesis. First we will take a look at the data we are using, how they are made before we use them and how the data will look like after we convert them to be used for our work. Since the data we are using are already produced beforehand as Root files, we will convert the data into data frames with Python and make new additional features as well as the targets from the data file(s). The data will then be preprocessed and analyzed before they are split into training, validation and test sets. We will then go through some of the tuning that is done with the models and the validation set, and use the evaluation metrics from section 7.5, before we use the best fit model on unseen data.

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

8.1 Python packages

Before we look at the data and implementation of the models and codes in the next chapters, we will introduce the most important Python packages we are actively using in this thesis. Many of the packages require other packages to be installed, but they do not have to be explicitly imported in the code itself. The codes and necessary requirements, like package versions¹, we use are found at a GitHub repository², with more explanations on how to setup

¹NB!: Some of the used package versions in this thesis will be updated or changed after some time. Check *requirements.txt* for latest current versions when this thesis was written.

²GitHub repository: <https://github.com/krilangs/ComPhys>—Master

and run the codes in a README file. We also import smaller packages with simple functions that we will use, but we will not explain those.

When we present code snippets in this thesis, we will leave out some parts (noted by ”\\...”) since it will only be used for presentation and visualization of the code used. The full source codes are found at the GitHub repository as mentioned.

- **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 similar.
- **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 that 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, lets us use ROOT with Python. This is a very much used in particles physics for using frameworks with the performance of the ROOT C++ libraries. This also lets us use Python libraries like NumPy and Pandas with ROOT.
- **Uproot**: Uproot is a library for converting ROOT files to be used with machine learning libraries with Python and NumPy.
- **Scikit-Learn**: Scikit-Learn[71] is a library most popularly used for data analysis and machine learning in Python. It contains a lot of useful tools for statistical modeling and machine learning.
- **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.

Chapter 9

Data

What to have here? Thoughts:

1. An explanation of how the data are produced and simulated, and what machine tools are used to produce these data?
2. Mention/explain/refer to input parameters for producing the data?
3. Explanation of how the data set(s) look?

Chapter 10

*Analyzing the Data

We will now use the data for Monte Carlo produced data, several backgrounds and neutrino signals with a few different scenarios for $N1$ mass, masses of 150 GeV and 450 GeV. We will use an already existing Python script¹ 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 using these to make new angular variables for $d\phi$ and dR . We can then 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 original scripts uses ROOT in Python. We will modify them to suite our case and needs. Since most of the implementations are already there, we only have to modify and add some small things like a few new input parameters and such. This mostly includes adding the new variables to plot, different data-taking periods and the signal samples that we can plot. The various backgrounds are already implemented and do not need any changes. The modified scripts can be found at the same GitHub repository as before with a README-file for explaining the usage.

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

¹Made by Eli B. Rye.

Chapter 11

Preparing the Data

11.1 Making New Variables

After downloading a data set with the proton-proton collision events, we start by converting it from ROOT to Python syntax with the uproot library and then to 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 each event and the corresponding four particles in each event, we can use these to 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}), which we talked about in chapter 4. Then we classify the events for making the targets as permutations of the leptons 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 are then neglected here. The function for making the new variables can be seen in Listing 11.1. 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, nlept+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 11.1: Making new variables.

11.2 Plotting New Variables

The new dataframe can now be imported by other scripts using Pandas, and used further. With the *Trilepton_plotter.py* script, we import the dataframe for visualization of the variables in it by using the matplotlib library. We make two different function that plots different features. One function plots the *single* variables for each lepton separately in one plot, so one plot can show the p_T for each lepton in one figure by example, while 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 leptons¹. The figures can then be saved to a desired folder. In Algorithm 1 we see 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)

11.3 Inspect Data

After importing all the necessary libraries in a new script, *Trilepton_classifier.py*, we load in the dataframes and drop unnecessary features we will not be using. This script will be our main script for doing the classification. Then we make the design matrix \mathbf{X} , containing all the particle variables for each event, and the target vector \mathbf{Y} , containing all the targets for each event, from the dataframe. 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} .

¹This solution was found by testing.

With some useful properties of Pandas dataframes and with Seaborn and Scikit-Learn, we can inspect the variable features a bit more before starting the classification. Pandas dataframes lets us easily print a few lines and a summary of the dataframe. We then get a quick overview of how the data looks like. This includes a short look at the features in the dataframe, like the values for a few events, the names and index dtypes of each feature in the dataframe. It also counts and prints the number of non-null values and the memory usage. Another useful thing to print is the individual target counts to check the number of each target in the dataframe. With this we quickly get an overview to see if we have a balanced or imbalanced data set. This can be very important, since it can lead to problems and we might have to do something to change this before doing the classification.

Correlations

An important descriptive statistic for data analysis with multi-variable data is the *correlation matrix* with Seaborn. It is a symmetric table of size $k \times k$, for k features (this includes the target 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. With machine learning, it is also a good preprocessing step that can give some information to whereas dimensionality reduction might come in handy with high-dimensionality data. The closer to 1 the correlation coefficients are, the more correlated are they. We also 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, it has a high significance for predicting the output than a feature close to -1. This helps us exclude features that worsen the prediction. The optimal case is then a high correlation between the independent and dependent variables, while a strong correlation between the independent variables (causing redundancy) is not.

To take a close look at the correlations in the data set, we use the mutual information of the features from section 7.5.1. By using the *mutual_info_classif* function by Scikit-Learn, we can easily compute the information gain of the features and the target. Here, we want the features that maximizes the information gain. This also helps us single out unnecessary features for classification. In the case of decision trees, the information gain is used for the splitting in the trees.

11.4 Preprocessing of the Data

Before we can start with the classification and start choosing the models we will look at, we need to do some preprocessing of the data from the information we got by looking at the data. One aspect we do not have to consider for our data set is NULL values. Since there are no null values in our data, we do not have to do anything about this. We can also check this again quickly by running:

```
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.
```

Listing 11.2: Check NULL values.

Feature Selection

After inspecting the data and the class counts in the target, we utilize feature selection methods to remove unnecessary features in the data that could confuse the classifiers and decrease the accuracy. This should also decrease the dimensionality of the data sets, making it easier for the models to see patterns in the data. With Scikit-Learn, we can use a few different methods:

Univariate feature selection - Here we use some univariate statistical test to select the best features by providing some scoring method. The **SelectKBest** method will remove all features except the k features with the highest scores.

Threshold selection - This method will remove features that are considered unimportant when the feature importance of the feature is below a certain threshold. With the **SelectFromModel** function, we can use linear models with the L_1 norm to select non-zero coefficients, or use impurity-based feature importance with tree-based estimators to remove unimportant features.

Pipeline - We can combine one of the two feature selection methods above to select the most relevant features and let the classifier train on the transformed output.

After training a model and checking the model performance with the evaluation metrics, we can also check the feature importance the models provide with Scikit-Learn. This lets us see which data features that are most

important for the models when they do their training. If some features still show a low importance, we can try to remove these features as well to try and improve the accuracy of the models further if necessary.

Resampling

After the feature selection, we make a function using Imblearn 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 11.3. This can then balance the data by sampling the information we already have and resample the data set.

```
"""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 11.3: Resample data.

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 **X** with the features and the targets **y** into three new sets each. This is done by using a Scikit-Learn function called *train_test_split* as seen in Listing 11.4. First we split **X** and **y** into training and test sets, then we further split the training sets into new smaller training sets and validation sets. 60% of the data are now training, 20% are validation and 20% are test. Remember that 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 fully tuned 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 11.4: Splitting the data.

Scaling

The next technique we will apply is scaling of the data. We will use Standardization 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*. For the training data we will both fit and transform, meaning that it both calculates the mean and standard deviation and standardizes the data set. We also have to transform the validation and test sets as well, but not fit them. In Listing 11.5 we use the *fit_transform* on the training set, while only using *transform* on the validation and test sets.

```

""" 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 11.5: Scaling.

Chapter 12

Model Classification

Part III

Results

Chapter 13

**

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 7.9:

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

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:

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

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

KATRIN Karlsruhe Tritium Neutrino

LEP Large Electron-Positron Collider

LGBM Light Gradient Boosting Machine

LH left-handed

LHC Large Hadron Collider

LHCb LHC-beauty

LR Logistic Regression

MET missing transverse momentum

ML machine learning

MLE Maximum Likelihood Estimation

MLP Multi-Layer Perceptron

MS muon spectrometer

OLS Ordinary Least Squares

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

SVM Support Vector Machine

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] Elizabeth Gibney and Davide Castelvecchi. Neutrino flip wins physics prize, 2015.
 - [11] 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.
 - [12] Alan Kostelecky. The status of cpt. *arXiv preprint hep-ph/9810365*, 1998.
 - [13] Franz Mandl and Graham Shaw. *Quantum field theory*. John Wiley & Sons, 2010.
 - [14] M. Bellac. Quantum and statistical field theory. 1991.
 - [15] 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.
 - [16] Gerhard Ecker. Quantum chromodynamics. *arXiv preprint hep-ph/0604165*, 2006.
 - [17] 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>.
 - [18] Richard Phillips Feynman. *QED: The strange theory of light and matter*. Princeton University Press, 2006.
 - [19] Guido Altarelli. The standard electroweak theory and beyond. *arXiv preprint hep-ph/9811456*, pages 27–93, 2000.

- [20] 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](https://doi.org/10.1016/0029-5582(61)90469-2).
- [21] Steven Weinberg. A model of leptons. *Physical review letters*, 19(21):1264, 1967. doi: <https://doi.org/10.1103/PhysRevLett.19.1264>.
- [22] A Salam. N. svartholm, ed. elementary particle physics: Relativistic groups and analyticity. In *Eighth Nobel Symposium. Stockholm: Almqvist and Wiksell*, page 367, 1968.
- [23] Press release: The Nobel Prize in Physics 1979. Nobel Media AB 2020, 1979. URL <https://www.nobelprize.org/prizes/physics/1979/press-release/>.
- [24] Abdus Salam and Steven Weinberg. Nobel Prize for Physics, 1979. *CERN Courier*, page 395, 1979.
- [25] Rabindra N Mohapatra. Seesaw mechanism and its implications. In *SEESAW 25*, pages 29–44. World Scientific, 2005.
- [26] Darwin Chang and Otto CW Kong. Pseudo-dirac neutrinos. *Physics Letters B*, 477(4):416–423, 2000.
- [27] KRS Balaji, Anna Kalliomäki, and Jukka Maalampi. Revisiting pseudo-dirac neutrinos. *Physics Letters B*, 524(1-2):153–160, 2002.
- [28] R. N. Mohapatra and J. C. Pati. "natural" left-right symmetry. *Phys. Rev. D*, 11:2558–2561, May 1975. doi: 10.1103/PhysRevD.11.2558. URL <https://link.aps.org/doi/10.1103/PhysRevD.11.2558>.
- [29] Jogesh C. Pati and Abdus Salam. Lepton number as the fourth "color". *Phys. Rev. D*, 10:275–289, Jul 1974. doi: 10.1103/PhysRevD.10.275. URL <https://link.aps.org/doi/10.1103/PhysRevD.10.275>.
- [30] Rabindra N. Mohapatra and Jogesh C. Pati. Left-right gauge symmetry and an "isoconjugate" model of CP violation. *Phys. Rev. D*, 11:566–571, Feb 1975. doi: 10.1103/PhysRevD.11.566. URL <https://link.aps.org/doi/10.1103/PhysRevD.11.566>.
- [31] G. Senjanovic and R. N. Mohapatra. Exact left-right symmetry and spontaneous violation of parity. *Phys. Rev. D*, 12:1502–1505, Sep 1975. doi: 10.1103/PhysRevD.12.1502. URL <https://link.aps.org/doi/10.1103/PhysRevD.12.1502>.

- [32] PS Bhupal Dev, Rabindra N Mohapatra, Werner Rodejohann, and Xun-Jie Xu. Vacuum structure of the left-right symmetric model. *Journal of High Energy Physics*, 2019(2):154, 2019.
- [33] Eirik Gramstad. Searches for Supersymmetry in di-Lepton Final States with the ATLAS Detector at $\sqrt{s} = 7$ TeV. 2013.
- [34] Richard D Field. The underlying event in hard scattering processes. *arXiv preprint hep-ph/0201192*, 2002.
- [35] About CERN. CERN, (10.12.20). URL <https://home.cern/about>.
- [36] Stephanie Sammartino McPherson. *Tim Berners-Lee: Inventor of the World Wide Web*. Twenty-First Century Books, 2009.
- [37] Esma Mobs. The CERN accelerator complex - 2019. Complexe des accélérateurs du CERN - 2019. Jul 2019. URL <https://cds.cern.ch/record/2684277>. General Photo.
- [38] XinChou Lou. The Circular Electron Positron Collider. *Nat. Rev. Phys.*, 1:232–234, 2019. doi: <https://doi.org/10.1038/s42254-019-0047-1>.
- [39] Jie Gao. China’s bid for a circular electron-positron collider. *CERN Courier*, 2018.
- [40] Shinichiro Michizono. The International Linear Collider. *Nat. Rev. Phys.*, 1:244–245, 2019. doi: <https://doi.org/10.1038/s42254-019-0044-4>.
- [41] 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.
- [42] The Large Hadron Collider. CERN, (11.12.20). URL <https://home.cern/science/accelerators/large-hadron-collider>.
- [43] Accelerators. CERN, (11.12.20). URL <https://home.cern/science/accelerators>.
- [44] Experiments: LHC experiments. CERN, (11.12.20). URL <https://home.cern/science/experiments>.

- [45] 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.
- [46] ATLAS Collaboration. Measurement of the Inelastic Proton-Proton Cross Section at $\sqrt{s}=13$ TeV with the ATLAS Detector at the LHC. *Phys. Rev. Lett.*, 117:182002, Oct 2016. doi: 10.1103/PhysRevLett.117.182002. URL <https://link.aps.org/doi/10.1103/PhysRevLett.117.182002>.
- [47] Computer generated image of the whole ATLAS detector. CERN Document Server, (01.01.21). URL <https://cds.cern.ch/record/1095924?ln=en>.
- [48] 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>.
- [49] 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.
- [50] CERN. Overall detector concept. *ATLAS Technical Proposal*, 1994.
- [51] 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.
- [52] CMS Collaboration, Vardan Khachatryan, Albert M Sirunyan, Armen Tumasyan, Wolfgang Adam, Thomas Bergauer, Marko Dragicevic, Janos Erö, Christian Fabjan, Markus Friedl, Rudolf Fruehwirth, et al. Search for heavy neutrinos and W bosons with right-handed couplings in proton-proton collisions at $\sqrt{s}= 8$ TeV. *The European Physical Journal C*, 74(11):3149, 2014. doi: <https://doi.org/10.1140/epjc/s10052-014-3149-z>.
- [53] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.

- [54] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.
- [55] 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.
- [56] Jose Manuel Pereira, Mario Basto, and Amelia Ferreira da Silva. The logistic lasso and ridge regression in predicting corporate failure. *Procedia Economics and Finance*, 39:634–641, 2016.
- [57] William S Noble. What is a support vector machine? *Nature biotechnology*, 24(12):1565–1567, 2006.
- [58] 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.
- [59] 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.
- [60] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [61] 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.
- [62] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.
- [63] J Ross Quinlan et al. Bagging, boosting, and c4. 5. In *Aaai/iaai, Vol. 1*, pages 725–730, 1996.
- [64] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.
- [65] Gunnar Rätsch, Takashi Onoda, and K-R Müller. Soft margins for adaboost. *Machine learning*, 42(3):287–320, 2001.

- [66] Jerome H Friedman. Stochastic gradient boosting. *Computational statistics & data analysis*, 38(4):367–378, 2002.
- [67] 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.
- [68] 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.
- [69] Dymitr Ruta and Bogdan Gabrys. Classifier selection for majority voting. *Information fusion*, 6(1):63–81, 2005.
- [70] Jingjing Cao, Sam Kwong, Ran Wang, Xiaodong Li, Ke Li, and Xi-angfei Kong. Class-specific soft voting based multiple extreme learning machines ensemble. *Neurocomputing*, 149:275–284, 2015.
- [71] 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.
- [72] Mary L McHugh. Interrater reliability: the kappa statistic. *Biochemia medica*, 22(3):276–282, 2012.